No “Game over” for the Winnti Group

May 21, 2020

The notorious APT group continues to play the video game industry with yet another backdoor

In February 2020, we discovered a new, modular backdoor, which we named PipeMon. Persisting as a Print Processor, it was used by the Winnti Group against several video gaming companies that are based in South Korea and Taiwan and develop MMO (Massively Multiplayer Online) games. Video games developed by these companies are available on popular gaming platforms and have thousands of simultaneous players.

In at least one case, the malware operators compromised a victim’s build system, which could have led to a supply-chain attack, allowing the attackers to trojanize game executables. In another case, the game servers were compromised, which could have allowed the attackers to, for example, manipulate in-game currencies for financial gain.

The Winnti Group, active since at least 2012, is responsible for high-profile supply-chain attacks against the software industry, leading to the distribution of trojanized software (such as CCleaner, ASUS LiveUpdate and multiple video games) that is then used to compromise more victims. Recently, ESET researchers also discovered a campaign of the Winnti Group targeting several Hong Kong universities with ShadowPad and Winnti malware.

About the “Winnti Group” naming:

We have chosen to keep the name “Winnti Group” since it’s the name first used to identify it, in 2013, by Kaspersky. Since Winnti is also a malware family, we always write “Winnti Group” when we refer to the malefactors behind the attacks. Since 2013, it has been demonstrated that Winnti is only one of the many malware families used by the Winnti Group.

Attribution to the Winnti Group

Multiple indicators led us to attribute this campaign to the Winnti Group. Some of the C&C domains used by PipeMon were used by Winnti malware in previous campaigns mentioned in our white paper on the Winnti Group arsenal. Besides, Winnti malware was also found in 2019 at some of the companies that were later compromised with PipeMon.

In addition to Winnti malware, a custom AceHash (a credential harvester) binary found at other victims of the Winnti Group, and signed with a well-known stolen certificate used by the group (Wemade IO), was also used during this campaign.

The certificate used to sign the PipeMon installer, modules and additional tools is linked to a video game company that was compromised in a supply-chain attack in late 2018 by the Winnti Group and was likely stolen at that time.

Interestingly, PipeMon modules are installed in %SYSTEM32%\spool\prtprocs\x64\; this path was also used in the past to drop the second stage of the trojanized CCleaner.

Additionally, compromising a software developer's build environment to subsequently compromise legitimate application is a known modus operandi of the Winnti Group.

Targeted companies

Companies targeted in this campaign are video game developers, producing MMO games and based in South Korea and Taiwan. In at least one case, the attackers were able to compromise the company's build orchestration server, allowing them to take control of the automated build systems. This could have allowed the attackers to include arbitrary code of their choice in the video game executables.

ESET contacted the affected companies and provided the necessary information to remediate the compromise.

Technical analysis

Two different variants of PipeMon were found at the targeted companies. Only for the more recent variant were we able to identify the first stage which is responsible for installing and persisting PipeMon.

First stage

PipeMon's first stage consists of a password-protected RARSFX executable embedded in the .rsrc section of its launcher. The launcher writes the RARSFX to setup0.exe in a directory named with a randomly generated, eight-character, ASCII string located in the directory returned by GetTempPath. Once written to disk, the RARSFX is executed with CreateProcess by providing the decryption password in an argument, as follows:

```
setup0.exe -p*|T/PMR{|T2^LWJ*
```

Note that the password is different with each sample.

The content of the RARSFX is then extracted into %TMP%\RarSFX0 and consists of the following files:

- CrLnc.dat – Encrypted payload
- Duser.dll – Used for UAC bypass
- osksupport.dll – Used for UAC bypass
- PrintDialog.dll – Used for the malicious print processor initialization
- PrintDialog.exe – Legitimate Windows executable used to load PrintDialog.dll
- setup.dll – Installation DLL
- setup.exe – Main executable

Note that in the event of a folder name collision, the number at the end of the RarSFX0 string is incremented until a collision is avoided. Further, not all these files are necessarily present in the archive, depending on the installer.

Once extracted, setup.exe is executed without arguments. Its sole purpose is to load setup.dll using LoadLibraryA. Once loaded, setup.dll checks whether an argument in the format –x:n (where n is an integer) was provided; the mode of operation will be different depending on the presence of n. Supported arguments and their corresponding behavior are shown in Table 1. setup.exe is executed without arguments by the RARSFX, and checks whether it's running with elevated privileges. If not, it will attempt to obtain such privileges using token impersonation if the version of Windows is below Windows 7 build 7601; otherwise it will attempt different UAC bypass techniques, allowing installation of the payload loader into one of:

- C:\Windows\System32\spool\prtprocs\x64\DEment.dll
- C:\Windows\System32\spool\prtprocs\x64\EntAppsvc.dll
- C:\Windows\System32\spool\prtprocs\x64\Interactive.dll

depending on the variant. Note that we weren't able to retrieve samples related to Interactive.dll.

Table 1. setup.exe *supported arguments and their corresponding behavior.*

Command line argument value	Behavior
-x:0	Load the payload loader.

Command line argument value	Behavior
-x:1	Attempt to enable SeLoadDriverPrivilege for the current process. If successful, attempt to install the payload loader; otherwise, restart setup.exe with the –x:2 argument using parent process spoofing.
-x:2	Attempt to enable SeLoadDriverPrivilege for the current process. If successful, attempt to install the payload loader.

This loader is stored encrypted within setup.dll, which will decrypt it before writing it to the aforementioned location.

Persistence using Windows Print Processors

The location where the malicious DLL is dropped was not chosen randomly. This is the path where Windows Print Processors are located and setup.dll registers the malicious DLL loader as an alternative Print Processor by setting one of the following registry values:

HKLM\SYSTEM\ControlSet001\Control\Print\Environments\Windows x64\Print Processors\PrintFiiterPipelineSvc\Driver = “DEment.dll”

or

HKLM\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows x64\Print Processors\lltdsvc1\Driver = “EntAppsvc.dll”

depending on the variant. Note the typo in PrintFiiterPipelineSvc (which has no impact on the Print Processor installation since any name can be used).

After having registered the Print Processor, PipeMon restarts the print spooler service (spoolsv.exe). As a result, the malicious print process is loaded when the spooler service starts. Note that the Print Spooler service starts at each PC startup, which ensures persistence across system resets.

This technique is really similar to the Print Monitor persistence technique (being used by DePriMon, for example) and, to our knowledge, has not been documented previously.

Additionally, the encrypted payload, CrLnc.dat, extracted from the RARSFX is written to the registry at the following location, depending on the installer:

- HKLM\SOFTWARE\Microsoft\Print\Components\DC20FD7E-4B1B-4B88-8172-61F0BED7D9E8
- HKLM\SOFTWARE\Microsoft\Print\Components\A66F35-4164-45FF-9CB4-69ACAA10E52D

This encrypted registry payload is then loaded, decrypted and executed by the previously registered Print Processor library. The whole PipeMon staging and persistence is shown in Figure 1.

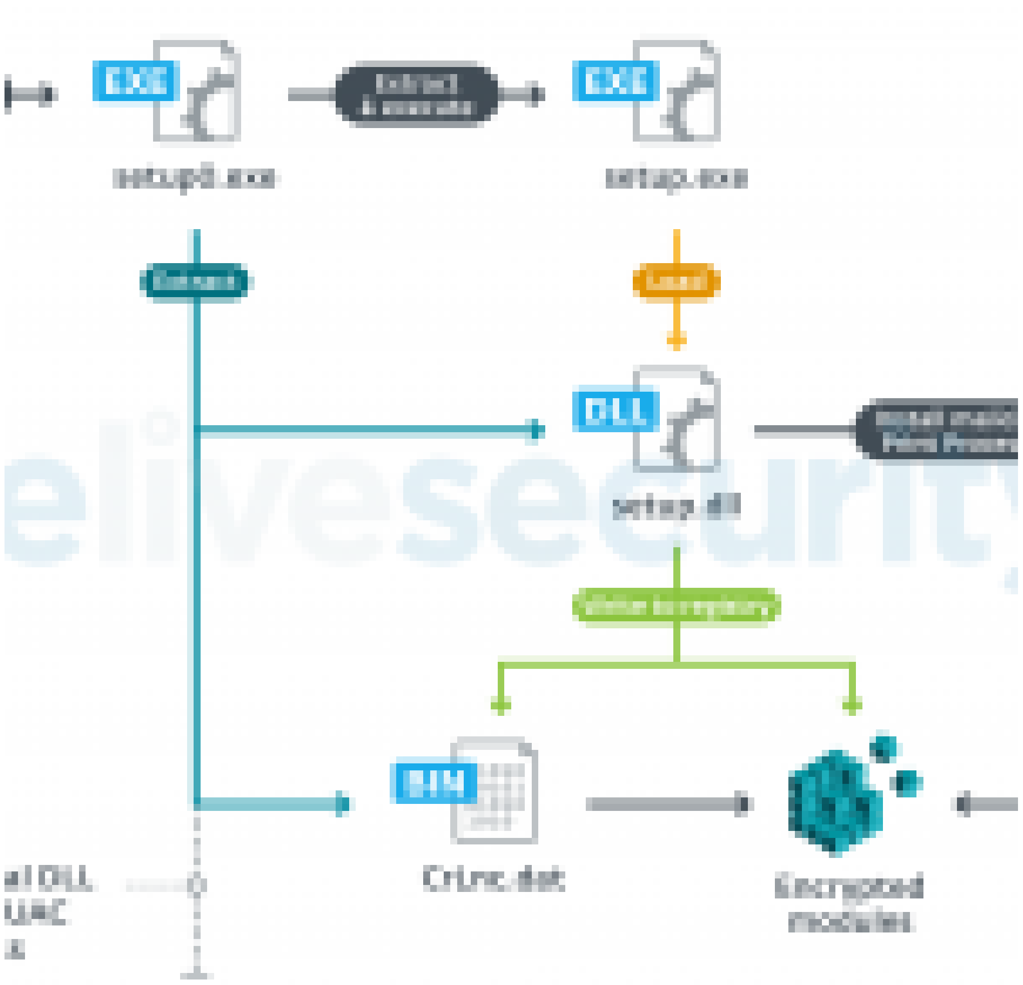


Figure 1. PipeMon staging and persistence

PipeMon

We named this new implant PipeMon because it uses multiple named pipes for inter-module communication and according to its PDB path, the name of the Visual Studio project used by its developer is “Monitor”.

As mentioned previously, two different PipeMon variants were found. Considering the first variant, we couldn’t retrieve the installer; thus, we don’t know for sure the persistence technique that was used. But considering that this first variant of PipeMon was also located in the Print Processor directory, it’s likely that the same persistence mechanism was used.

Original variant

PipeMon is a modular backdoor where each module is a single DLL exporting a function called IntelLoader and is loaded using a reflective loading technique. Each module exhibits different functionalities that are shown in Table 2.

The loader, responsible for loading the main modules (ManagerMain and GuardClient) is Win32CmdDll.dll and is located in the Print Processors directory. The modules are stored encrypted on disk at the same location with inoffensive-looking names such as:

- banner.bmp
- certificate.cert
- License.hwp
- JSONDIU7c9djE
- D8JNCKS0DJE
- B0SDFUWEkNCj.logN

Note that .hwp is the extension used by Hangul Word Processor from Hangul Office, which is very popular in South Korea.

The modules are RC4 encrypted and the decryption key Com!123Qasdz is hardcoded into each module. Win32CmDll.dll decrypts and injects the ManagerMain and GuardClient modules. The ManagerMain module is responsible for decrypting and injecting the Communication module, while the GuardClient module will ensure that the Communication module is running and reload it if necessary. An overview of how PipeMon operates is shown in Figure 2.

Win32CmDll.dll first tries to inject the ManagerMain and GuardClient modules into a process with one of the following names: lsass.exe, wininit.exe or lsm.exe. If that fails, it tries to inject into one of the registered windows services processes, excluding processes named spoolsv.exe, ekrn.exe (ESET), avp.exe (Kaspersky) or dllhost.exe. As a last option, if everything else failed, it tries to use the processes taskhost.exe, taskhostw.exe or explorer.exe.

The process candidates for Communication module injection must be in the TCP connection table with either 0.0.0.0 as the local address, or an ESTABLISHED connection and owning a LOCAL SERVICE token. These conditions are likely used to hide the Communication module into a process that is already communicating over the network so that the traffic from the Communication module would seem inconspicuous and possibly also whitelisted in the firewall. If no process meets the previous requirements, the ManagerMain module tries to inject the Communication module into explorer.exe. Processes belonging to the Windows Store Apps and processes named egui.exe (ESET) and avpui.exe (Kaspersky) are ignored from the selection.

Table 2. PipeMon module descriptions and their respective PDB paths

Module Name	Description	PDB Path
Win32CmdDll	Decrypts and loads the ManagerMain and GuardClient modules.	S:\Monitor\Monitor_RAW\Launcher\x64\Release\Win32CmdDll.pdb S:\Monitor\Monitor_RAW\libs\x64\Release\Win32CmdDll.pdb
GuardClient	Periodically checks whether the Communication module is running and loads it if not.	S:\Monitor\Monitor_RAW\Client\x64\Release\GuardClient.pdb
ManagerMain	Loads Communication module when executed. Contains encrypted C&C domain which is passed to the Communication module via named pipe. Can execute several commands based on the data received from the Communication module (mostly system information collecting, injecting payloads).	S:\Monitor\Monitor_RAW\Client\x64\Release\ManagerMain.pdb
Communication	Responsible for managing communication between the C&C server and individual modules via named pipes.	S:\Monitor\Monitor_RAW\Client\x64\Release\Communication.pdb F:\PCC\trunk\CommunicationClient\x64\Release\Communication.pdb

Additional modules can be loaded on-demand using dedicated commands (see below), but unfortunately, we weren't able to discover any of them. The names of these modules are an educated guess based on the named pipes used to communicate with them:

- Screen
- Route
- CMD
- InCmd
- File

Inter-module communication

Inter-module communication is performed via named pipes, using two named pipes per communication channel between each individual module, one for sending and one for receiving. Table 3 lists the communication channels and their corresponding named pipes.

Table 3. PipeMon communication channel and their respective named pipes

Communication channel	Named pipe
Communication, Screen	\\.\pipe\ScreenPipeRead%CNC_DEFINED% \\.\pipe\ScreenPipeWrite%CNC_DEFINED%

Communication channel	Named pipe
Communication, Route	\\.\pipe\RoutePipeWriite%B64_TIMESTAMP%
Communication, ManagerMain	\\.\pipe\MainPipeWrite%B64_TIMESTAMP% \\.\pipe\MainPipeRead%B64_TIMESTAMP%
GuardClient, ManagerMain	\\.\pipe\MainHeatPipeRead%B64_TIMESTAMP%
Communication, InCmd	\\.\pipe\InCmdPipeWrite%B64_TIMESTAMP% \\.\pipe\InCmdPipeRead%B64_TIMESTAMP%
Communication, File	\\.\pipe\FilePipeRead%B64_TIMESTAMP% \\.\pipe\FilePipeWrite%B64_TIMESTAMP%
GuardClient, Communication	\\.\pipe\ComHeatPipeRead%B64_TIMESTAMP%
Communication, CMD	\\.\pipe\CMDPipeRead \\.\pipe\CMDPipeWrite

The %CNC_DEFINED% string is received from the C&C server and %B64_TIMESTAMP% variables are base64-encoded timestamps such as the ones shown in Table 4.

Table 4. Example timestamps used with named pipes

%BASE64_TIMESTAMP%	Decoded timestamp
MjAxOTAyMjgxMDE1Mzc=	20190228101537
MjAxOTA1MjEyMzU2MjQ=	20190521235624
MjAxOTExMjExMjE2MjY=	20191121121626

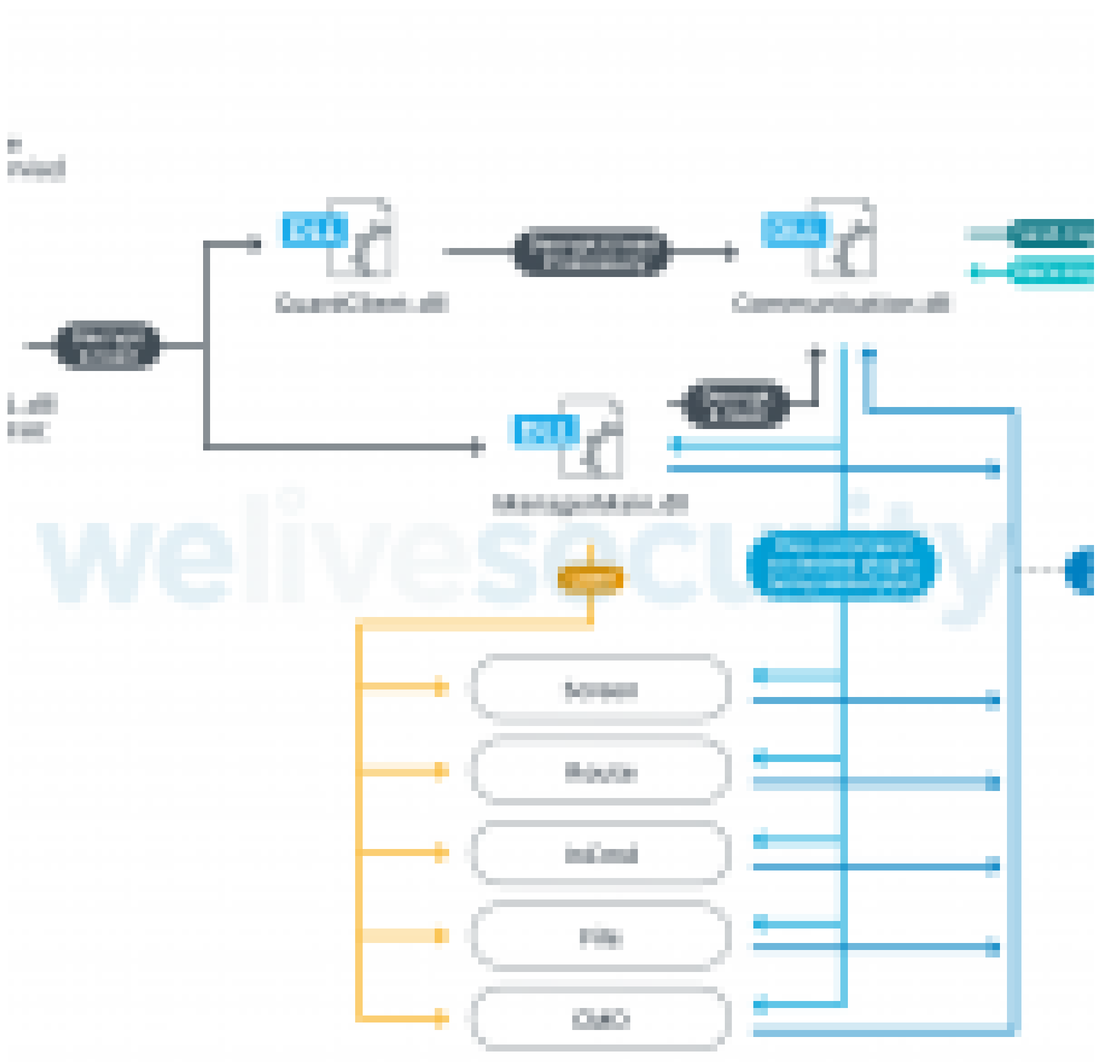


Figure 2. PipeMon IPC scheme (original PipeMon variant)

C&C communication

The Communication module is responsible for managing communications between the C&C server and the other modules via named pipes, similar to the PortReuse backdoor documented in our white paper on the Winnti arsenal.

Its C&C address is hardcoded in the ManagerMain module and encrypted using RC4 with the hardcoded key Com!123Qasdz. It is sent to the Communication module through a named pipe.

A separate communication channel is created for each installed module. The communication protocol used is TLS over TCP. The communication is handled with the HP-Socket library. All the messages are RC4 encrypted using the hardcoded key. If the size of the message to be transferred is greater than or equal to 4KB, it is first compressed using zlib's Deflate implementation.

```
struct CCMSG
{
    BYTE is_compressed;
    CMD cmd;
};

struct CMD
{
    QWORD cmd_type;
    DWORD cmd_size;
    DWORD cmd_arg;
    BYTE data[cmd_size - 16];
};
```

```
struct beacon_msg
{
    BYTE isCompressed = 0;
    CMD cmd_hdr;
    WCHAR win_version[128];
    WCHAR adapters_addrs[128];
    WCHAR adapters_addrs[64];
    WCHAR local_addr[64];
    WCHAR malware_version[64];
    WCHAR computer_name[64];
}
```

Figure 3. C&C message and beacon formats

To initiate communication with the C&C server, a beacon message is first sent that contains the following information:

- OS version
- physical addresses of connected network adapters concatenated with %B64_TIMESTAMP%
- victim's local IP address
- backdoor version/campaign ID; we've observed the following values
    - "1.1.1.4beat"
    - "1.1.1.4Bata"
    - "1.1.1.5"
- Victim computer name

The information about the victim's machine is collected by the ManagerMain module and sent to the Communication module via the named pipe. The backdoor version is hardcoded in the Communication module in cleartext.

The format of the beacon message is shown in Figure 3 and the supported commands are shown in Table 5.

Table 5. List of commands

Command type	Command argument	Description
0x02	0x03	Install the File module
0x03	0x03	Install the CMD module
0x03	0x0B	Install the InCmd module
0x04	0x02	Queue command for the Route module
0x04	0x03	Install the Route module
0x05	*	Send victim’s RDP information to the C&C server
0x06	0x05	Send OS, CPU, PC and time zone information to the C&C server
0x06	0x06	Send network information to the C&C server
0x06	0x07	Send disk drive information to the C&C server
0x07	*	Send running processes information to the C&C server
0x09	*	DLL injection
0x0C	0x15	Send names of "InCmd" pipes and events to the C&C server
0x0C	0x16	Send name of "Route" pipe to the C&C server
0x0C	0x17	Send names of "File" pipes to the C&C server

** The argument supplied for this command type is ignored*

Updated variant

As mentioned earlier, the attackers also use an updated version of PipeMon for which we were able to retrieve the first stage described above. While exhibiting an architecture highly similar to the original variant, its code was likely rewritten from scratch.

The RC4 code used to decrypt the modules and strings was replaced by a simple XOR with 0x75E8EEAF as the key and all the hardcoded strings were removed. The named pipes used for inter-module communication are now named using random values instead of explicit names and conform to the format \\.\pipe\%rand%, where %rand% is a pseudorandomly generated string of 31 characters containing only mixed case alphabetic characters.

Here, only the main loader (i.e. the malicious DLL installed as a Print Processor) is stored as a file on disk; the modules are stored in the registry by the installer (from the CrLnc.dat file) and are described in Table 6.

Table 6. Updated modules

Module name	Description
CoreLnc.dll	Loaded by the malicious Print Processor. Responsible only for loading the Core.dll module embedded in its .data section.
Core.dll	Loads the Net.dll module embedded in its .data section. Handles commands from the C&C server and communications between individual modules and the C&C server through named pipes.
Net.dll	New Communication module. Handles the networking.

Module injection is not performed using the reflective loading technique with an export function anymore; custom loader shellcode is used instead and is injected along with the module to be loaded.

The C&C message format was changed as well, and is shown in Figure 4.

```
struct CCMSG
{
    BYTE is_compressed;
    CMD cmd;
};

struct CMD
{
    QWORD cmd_type;
    DWORD cmd_size;
    DWORD cmd_arg;
    BYTE data[cmd_size - 16];
};
```

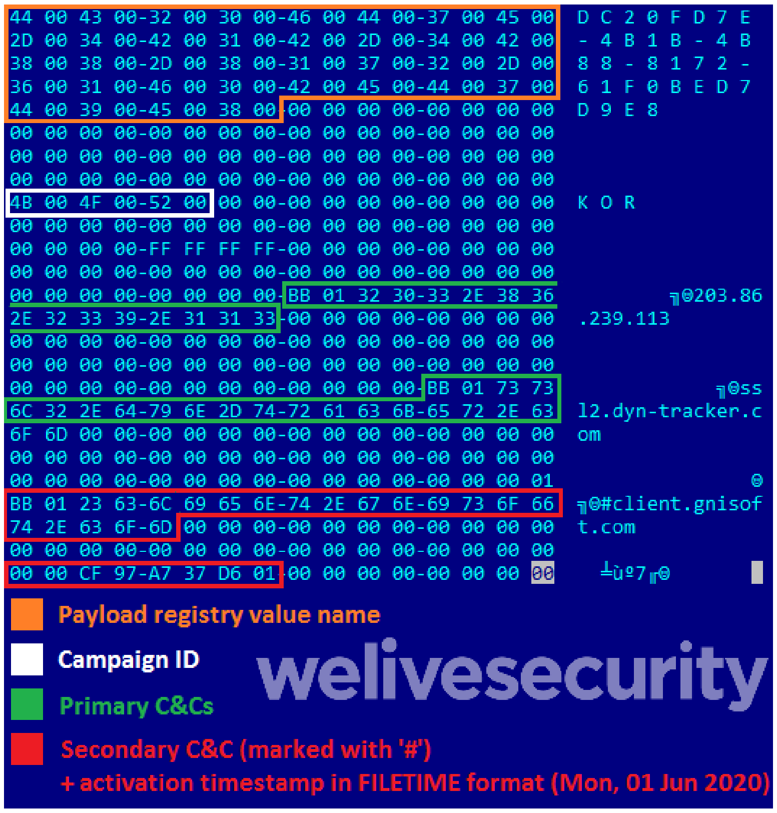
```
struct CCMSG
{
    DWORD signature = 0xFA149DEB;
    DWORD not_used;
    WORD buff_size;
    WORD msgcode;
    BYTE buff[buff_size];
};
```

Figure 4. Previous (top) and updated (bottom) C&C message format

Interestingly, the backdoor's configuration is encrypted and embedded in the loader DLL. The configuration contains:

- Name of the registry value
- Campaign identifier
- C&C IP addresses or domain names
- Timestamp (in FILETIME format) corresponding to the date from which to start using a second C&C domain marked with '#' in the configuration.

An example of a configuration dump embedded in the loader DLL is shown in Figure 5. Configurations extracted from several loader DLLs are shown in Table 7.



```
44 00 43 00-32 00 30 00-46 00 44 00-37 00 45 00  D C 2 0 F D 7 E
2D 00 34 00-42 00 31 00-42 00 2D 00-34 00 42 00  - 4 B 1 B - 4 B
38 00 38 00-2D 00 38 00-31 00 37 00-32 00 2D 00  8 8 - 8 1 7 2 -
36 00 31 00-46 00 30 00-42 00 45 00-44 00 37 00  6 1 F 0 B E D 7
44 00 39 00-45 00 38 00-00 00 00 00-00 00 00 00  D 9 E 8
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
4B 00 4F 00-52 00 00 00-00 00 00 00-00 00 00 00  K O R
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-FF FF FF FF-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-BB 01 32 30-33 2E 38 36          ╗☺203.86
2E 32 33 39-2E 31 31 33-00 00 00 00-00 00 00 00  .239.113
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-BB 01 73 73              ╗☺ss
6C 32 2E 64-79 6E 2D 74-72 61 63 6B-65 72 2E 63  l2.dyn-tracker.c
6F 6D 00 00-00 00 00 00-00 00 00 00-00 00 00 00  om
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 01                 ☺
BB 01 23 63-6C 69 65 6E-74 2E 67 6E-69 73 6F 66  ╗☺#client.gnisof
74 2E 63 6F-6D 00 00 00-00 00 00 00-00 00 00 00  t.com
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 CF 97-A7 37 D6 01-00 00 00 00-00 00 00 00    ╧ùº7╓☺
```

🟧 Payload registry value name
⬜ Campaign ID
🟩 Primary C&Cs
🟥 Secondary C&C (marked with '#')
+ activation timestamp in FILETIME format (Mon, 01 Jun 2020)

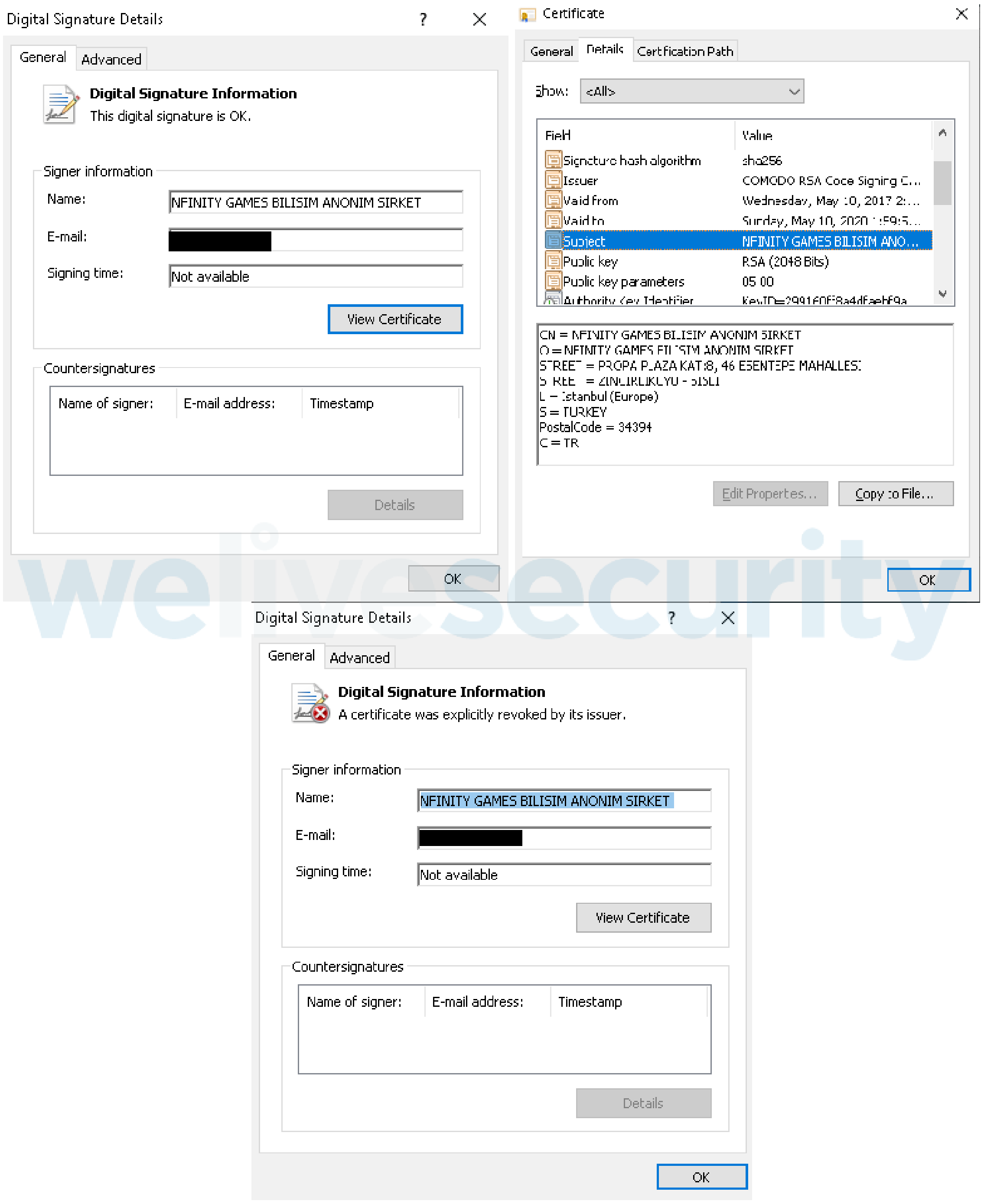
Figure 5. Example of decrypted configuration (with few zero-bytes removed because of image size)

Table 7. Configuration extracted from several loaders

Loader SHA-1	Campaign ID	Payload registry name	C&C IP/domains	Alternative domain activation timestamp
6c97039605f93ccf1afccbab8174d26a43f91b20	KOR2	DC20FD7E-4B1B-4B88-8172-61F0BED7D9E8	154.223.215.116 ssl2.dyn-tracker.com #client.gnisoft.com	0x01d637a797cf0000 (Monday, June 1, 2020 12:00:00am)
97da4f938166007ce365c29e1d685a1b850c5bb0	KOR	DC20FD7E-4B1B-4B88-8172-61F0BED7D9E8	203.86.239.113 ssl2.dyn-tracker.com #client.gnisoft.com	0x01d637a797cf0000 (Monday, June 1, 2020 12:00:00am)
7ca43f3612db0891b2c4c8ccab1543f581d0d10c	kor1	DC20FD7E-4B1B-4B88-8172-61F0BED7D9E8	203.86.239.113 www2.dyn.tracker.com (note the typo here: dyn.tracker instead of dyn-tracker) #nmn.nhndesk.com	0x01d61f4b7500c000 (Friday, May 1, 2020 12:00:00am)
b02ad3e8b1cf0b78ad9239374d535a0ac57bf27e	tw1	A66F35-4164-45FF-9CB4-69ACAA10E52D	ssl.lcrest.com	-

Stolen code-signing certificate

PipeMon modules and installers are all signed with the same valid code-signing certificate that was likely stolen during a previous campaign of the Winnti Group. The certificate's owner revoked it as soon as they were notified of the issue.


Digital Signature Details

Digital Signature Information
This digital signature is OK.

Signer information

Name:	NFINITY GAMES BILISIM ANONIM SIRKET
E-mail:	[redacted]
Signing time:	Not available

Countersignatures

Name of signer:	E-mail address:	Timestamp

Certificate — Details

Show: <All>

Field	Value
Signature hash algorithm	sha256
Issuer	COMODO RSA Code Signing C...
Valid from	Wednesday, May 10, 2017 2:...
Valid to	Sunday, May 10, 2020 1:59:5...
Subject	NFINITY GAMES BILISIM ANO...
Public key	RSA (2048 Bits)
Public key parameters	05 00
Authority Key Identifier	KeyID=29916?F8a4dfaebf9a

```
CN = NFINITY GAMES BILISIM ANONIM SIRKET
O = NFINITY GAMES BILISIM ANONIM SIRKET
STREET = PROPA PLAZA KAT:8, 46 ESENTEPE MAHALLESI
STREET = ZINCIRLIKUYU - SISLI
L = Istanbul (Europe)
S = TURKEY
PostalCode = 34394
C = TR
```

Digital Signature Details

Digital Signature Information
A certificate was explicitly revoked by its issuer.

Signer information

Name:	NFINITY GAMES BILISIM ANONIM SIRKET
E-mail:	[redacted]
Signing time:	Not available

Countersignatures

Name of signer:	E-mail address:	Timestamp

Figure 6. Code-signing certificate used to sign PipeMon first stage and modules before (top) and after (bottom) revocation.

We found on a sample sharing platform other tools signed with this certificate, such as HTRan, a connection bouncer, the WinEggDrop port scanner, Netcat, and Mimikatz which may have been used by the attackers as well.

Furthermore, a custom AceHash build signed with a Wemade IO stolen certificate already mentioned in our previous white paper and usually used by the Winnti Group was found on some machines compromised with PipeMon.

Conclusion

Once again, the Winnti Group has targeted video game developers in Asia with a new modular backdoor signed with a code-signing certificate likely stolen during a previous campaign and sharing some similarities with the PortReuse backdoor. This new implant shows that the Winnti Group is still actively developing new tools using multiple open source projects; they don't rely solely on their flagship backdoors, ShadowPad and the Winnti malware.

We will continue to monitor new activities of the Winnti Group and will publish relevant information on our blog. For any inquiries, contact us at *threatintel@eset.com*. The IoCs are also available at our GitHub repository.

Indicators of Compromise

ESET detection names

Win64/PipeMon.A
Win64/PipeMon.B
Win64/PipeMon.C
Win64/PipeMon.D
Win64/PipeMon.E

Filenames

100.exe
103.exe
Slack.exe
setup.exe
%SYSTEM32%\spool\prtprocs\x64\DEment.dll
%SYSTEM32%\spool\prtprocs\x64\EntAppsvc.dll
%SYSTEM32%\spool\prtprocs\x64\Interactive.dll
%SYSTEM32%\spool\prtprocs\x64\banner.bmp
%SYSTEM32%\spool\prtprocs\x64\certificate.cert
%SYSTEM32%\spool\prtprocs\x64\banner.bmp
%SYSTEM32%\spool\prtprocs\x64\License.hwp
%SYSTEM32%\spool\prtprocs\x64\D8JNCKS0DJE
%SYSTEM32%\spool\prtprocs\x64\B0SDFUWEkNCj.log
%SYSTEM32%\spool\prtprocs\x64\K9ds0fhNCisdjf
%SYSTEM32%\spool\prtprocs\x64\JSONDIU7c9djE
%SYSTEM32%\spool\prtprocs\x64\NTFSSSE.log
AceHash64.exe
mz64x.exe

Named pipes

\\.\pipe\ScreenPipeRead%CNC_DEFINED%
\\.\pipe\ScreenPipeWrite%CNC_DEFINED%
\\.\pipe\RoutePipeWriite%B64_TIMESTAMP%
\\.\pipe\MainPipeWrite%B64_TIMESTAMP%
\\.\pipe\MainPipeRead%B64_TIMESTAMP%
\\.\pipe\MainHeatPipeRead%B64_TIMESTAMP%
\\.\pipe\InCmdPipeWrite%B64_TIMESTAMP%
\\.\pipe\InCmdPipeRead%B64_TIMESTAMP%
\\.\pipe\FilePipeRead%B64_TIMESTAMP%
\\.\pipe\FilePipeWrite%B64_TIMESTAMP%
\\.\pipe\ComHeatPipeRead%B64_TIMESTAMP%
\\.\pipe\CMDPipeRead
\\.\pipe\CMDPipeWrite

Registry

HKLM\SYSTEM\ControlSet001\Control\Print\Environments\Windows x64\Print Processors\PrintFiiterPipelineSvc\Driver = "DEment.dll"

HKLM\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows x64\Print Processors\lltdsvc1\Driver = "EntAppsvc.dll"

HKLM\SOFTWARE\Microsoft\Print\Components\DC20FD7E-4B1B-4B88-8172-61F0BED7D9E8
HKLM\SOFTWARE\Microsoft\Print\Components\A66F35-4164-45FF-9CB4-69ACAA10E52D

Samples

First stage

4B90E2E2D1DEA7889DC15059E11E11353FA621A6
C7A9DCD4F9B2F26F50E8DD7F96352AEC7C4123FE
3508EB2857E279E0165DE5AD7BBF811422959158
729D526E75462AA8D33A1493B5A77CB28DD654BC
5663AF9295F171FDD41A6D819094A5196920AA4B

PipeMon

23789B2C9F831E385B22942DBC22F085D62B48C7
53C5AE2655808365F1030E1E06982A7A6141E47F
E422CC1D7B2958A59F44EE6D1B4E10B524893E9D
5BB96743FEB1C3375A6E2660B8397C68BEF4AAC2
78F4ACD69DC8F9477CAB9C732C91A92374ADCACD
B56D8F826FA8E073E6AD1B99B433EAF7501F129E
534CD47EB38FEE7093D24BAC66C2CF8DF24C7D03

PipeMon encrypted binaries

168101B9B3B512583B3CE6531CFCE6E5FB581409
C887B35EA883F8622F7C48EC9D0427AFE833BF46
44D0A2A43ECC8619DE8DB99C1465DB4E3C8FF995
E17972F1A3C667EEBB155A228278AA3B5F89F560
C03BE8BB8D03BE24A6C5CF2ED14EDFCEFA8E8429
2B0481C61F367A99987B7EC0ADE4B6995425151C

Additional tools

WinEggDrop

AF9C220D177B0B54A790C6CC135824E7C829B681

Mimikatz

4A240EDEF042AE3CE47E8E42C2395DB43190909D

Netcat

751A9CBFFEC28B22105CDCAF073A371DE255F176

HTran

48230228B69D764F71A7BF8C08C85436B503109E

AceHash

D24BBB898A4A301870CAB85F836090B0FC968163

Code-signing certificate SHA-1 thumbprints

745EAC99E03232763F98FB6099F575DFC7BDFAA3
2830DE648BF0A521320036B96CE0D82BEF05994C

C&C domains

n8.ahnlabinc[.]com
owa.ahnlabinc[.]com
ssl2.ahnlabinc[.]com
www2.dyn.tracker[.]com
ssl2.dyn-tracker[.]com
client.gnisoft[.]com
nmn.nhndesk[.]com
ssl.lcrest[.]com

C&C IP addresses

154.223.215[.]116
203.86.239[.]113

MITRE ATT&CK techniques

Tactic	ID	Name	Description
Persistence	T1013	Port Monitor	PipeMon uses a persistence technique similar to Port Monitor based on Print Processors.
Privilege Escalation	T1134	Access Token Manipulation	The PipeMon installer tries to gain administrative privileges using token impersonation.
T1088	Bypass User Account Control	The PipeMon installer uses UAC bypass techniques to install the payload.	
T1502	Parent PID Spoofing	The PipeMon installer uses parent PID spoofing to elevate privileges.	
Defense Evasion	T1116	Code Signing	PipeMon, its installer and additional tools are signed with stolen code-signing certificates.
T1027	Obfuscate Files or Information	PipeMon modules are stored encrypted on disk.	
T1112	Modify Registry	The PipeMon installer modifies the registry to install PipeMon as a Print Processor.	
T1055	Process Injection	PipeMon injects its modules into various processes using reflective loading.	
Discovery	T1057	Process Discovery	PipeMon iterates over the running processes to find a suitable injection target.
T1063	Security Software discovery	PipeMon checks for the presence of ESET and Kaspersky software.	
Collection	T1113	Screen Capture	One of the PipeMon modules is likely a screenshotter.
Command and Control	T1043	Commonly Used Ports	PipeMon communicates through port 443.
T1095	Custom Command and Control Protocol	PipeMon communication module uses a custom protocol based on TLS over TCP.	
T1032	Standard Cryptographic Protocol	PipeMon communication is RC4 encrypted.	
T1008	Fallback Channels	The updated PipeMon version uses a fallback channel once a particular date is reached.	

21 May 2020 - 11:30AM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion