ESET Research White papers // October 2019

CONNECTING THE DOTS

Exposing the arsenal and methods of the Winnti Group

Marc-Etienne M.Léveillé
Mathieu Tartare

ENJOY SAFER TECHNOLOGY™

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. EXECUTIVE SUMMARY

It seems the past few years have shown an increase in supply-chain attacks to distribute malware. Is this a reality, or because the industry is getting better at detecting and documenting them? Regardless of the answer, one of the groups that has been shown to be quite effective at conducting supply-chain attacks is the Winnti Group. Not only they have compromised multiple high-profile targets, but in each case, they were able to stay under the radar for many months before they were found and disrupted.

This report describes in detail some of the new malware used by this group in the organizations they target. Another one of their backdoors, ShadowPad, was also updated recently. This report will also expose the new techniques they use to hide their payloads.

By analyzing their tools and techniques, ESET researchers were able to infer some relationships between each reported supply-chain incident.

Key Findings

- One of the goals of the group, or its subgroups, is cryptocurrency mining. The Winnti Group has deployed cryptocurrency mining software using the backdoor they added in *games and software in 2018*.
- There are strong links in the tools and the techniques used in multiple major supply-chain attacks in past years. These links indicate that the following incidents were likely performed by the same group: *CCleaner*, *NetSarang*, *Asus* and *games and software in 2018*.
- This report documents a previously unanalyzed backdoor used by the Winnti Group. Called *PortReuse* by its authors, this Windows backdoor is a passive network implant that injects itself into a process that is already listening on a network port and waits for an incoming magic packet to trigger the malicious code.
- With the help of *Censys*, ESET researchers identified and notified a victim of *PortReuse*: a major Asian mobile hardware and software vendor.
- The ShadowPad malware is still being updated by its author and was updated and used multiple times in 2019. It has the same modular approach and includes additional obfuscation techniques.

2. OVERVIEW OF LATEST WINNTI ACTIVITIES

Since the original report by Kaspersky in 2013 “Winnti. More than just a game” [1], there have been numerous reports on the Winnti Group activities.

2.1 Naming

When reading the multiple reports on the Winnti Group activities, it is sometime difficult to realize that some discuss about the same malware family or component. This report tries to use existing names as much as possible, by using the name as it was first reported. We understand that some of the names are different due to the naming schemes used by different vendors or the fact that the visibility during research couldn’t allow mapping an analysis to a specific existing name. There is also the problem of tools versus the name of the operation, which sometimes bear the same name. It is worth spending some time to define the terms we are going to use.

- **Winnti Group:** For ESET researchers, this is the group that performed the attacks on multiple organizations using the tools and techniques described in this paper, regardless of their intent. Whether they are part of a single group or multiple subgroups is of less importance. The relationships that can be drawn around their different attacks is sufficient to show they were at least in contact.

  Aliases:
  - Winnti Umbrella — 401TRG
  - Axiom — Novetta
  - BARIUM and LEAD — Microsoft. According to Microsoft [2], BARIUM targets the gaming and technology industry while LEAD’s objective is more to steal sensitive data.

    - Group 72 — Cisco Talos
    - Blackfly and Suckfly — Symantec
    - APT41 — FireEye

- **Winnti malware:** A malware family described by Kaspersky in 2013 [3]. This malware uses a rootkit component to hide its activity.

  Aliases:
    - RbDoor (name seen in PDB path of early Winnti variants)
    - RibDoor — Microsoft's detection name
    - HIGHNOON — FireEye

- **ShadowPad:** A malware family used at a later stage in targeted attacks performed by the Winnti Group. It was first described by Kaspersky in the NetSarang incident in 2017 [4].

  Aliases:
    - Barlaiy — Microsoft
    - POISONPLUG — FireEye

- **ShadowHammer:** Malware embedded in Asus Live Update in 2018 [5]. ShadowHammer triggers its malicious behavior only if the computer it is running on has a network adapter with the MAC address whitelisted by the attacker.

2.2 Court cases

There are currently two open court cases in the US against the alleged perpetrator behind this group.

The first one, dated October 26th 2017, not long after the NetSarang incident, is a civil case filed by Microsoft. Without going into attribution, it describes how the ShadowPad malware operates and how it uses legitimate web sites to store the address of its C&C server.

The second one has more details about attribution. The indictment was filed by US federal attorneys, in the District Court of Southern California, in October 2018. It accuses ten Chinese individuals of compromising companies mostly in the aerospace and technology industry. According to the indictment, the intentions were to steal intellectual property from the victims. The incidents they describe go from 2010 to 2015, where they used the Sakula, PlugX, and Winnti malware in the different organizations. Some of their attacks involved compromising the DNS registrar to change the nameservers of their target. We cannot be certain that the same individuals are behind the supply-chain attacks, but given that they were using the same toolset, they probably belong to the same organization.

2.3 Targets

By following their activities over the years, it is intriguing to see that the Winnti Group has had victims in a wide range of industries. They include:

- Aviation
- Gaming
- Pharmaceuticals
- Technology
- Telecommunication
- Software development

This shows that the group may have lot of different intents. For example, there is no explanation we can think of to target the gaming industry to perform espionage operations. This is one of the reasons ESET researchers and researchers for other vendors tend to agree that the Winnti Group may not be monolithic; it is plausible that there are multiple subgroups and that operations are not conducted by the same group that is authoring the malware they use.

Based on ESET's telemetry and public reporting, it seems they focus their efforts to compromise private organizations in Asia, with South Korea being one of the most targeted countries.

3. THE WINNTI GROUP ARSENAL

During the course of our research, we were able to find different malware artifacts using the same techniques or code. To better visualize the full picture, we have created a diagram showing the relationships between all of them.

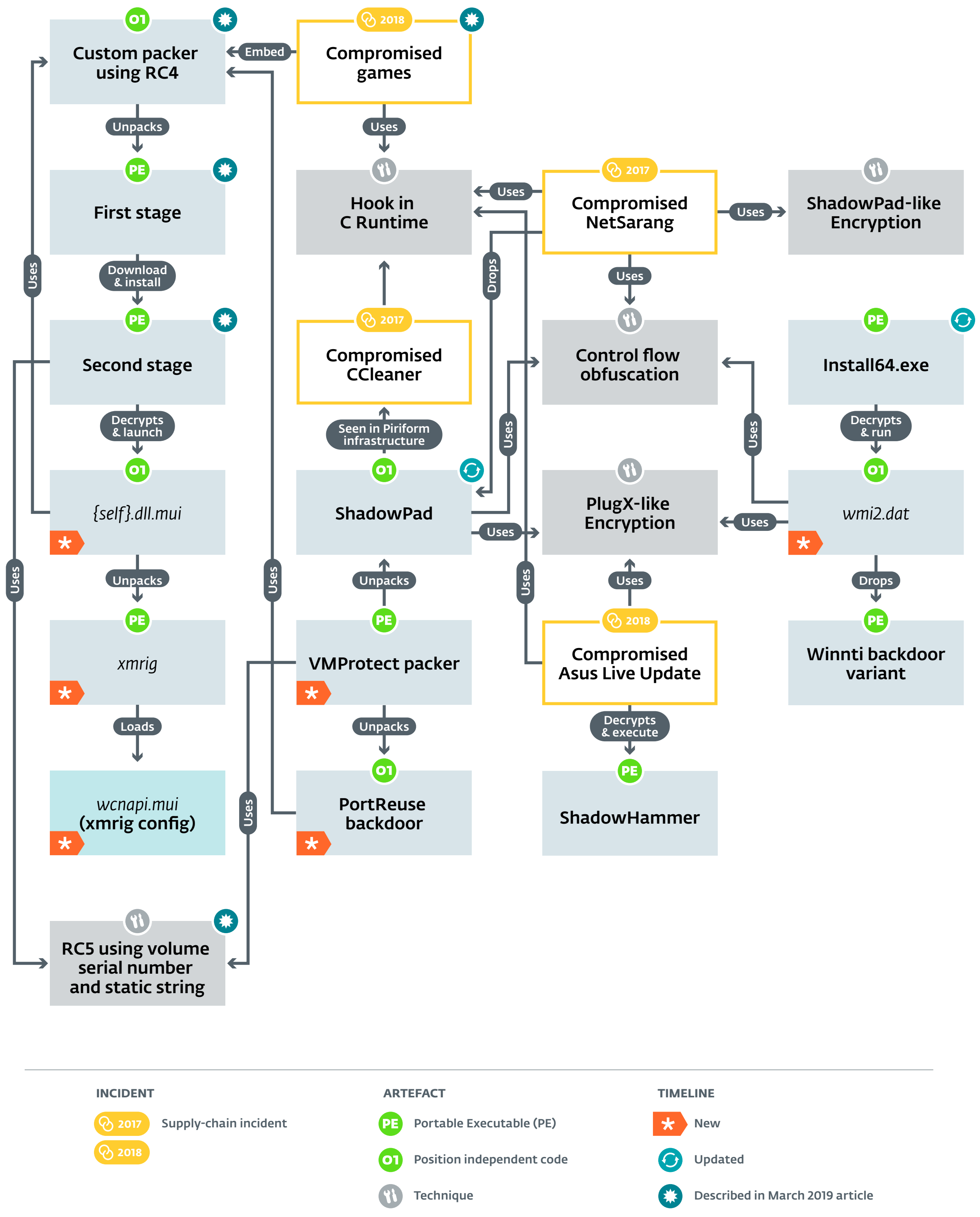
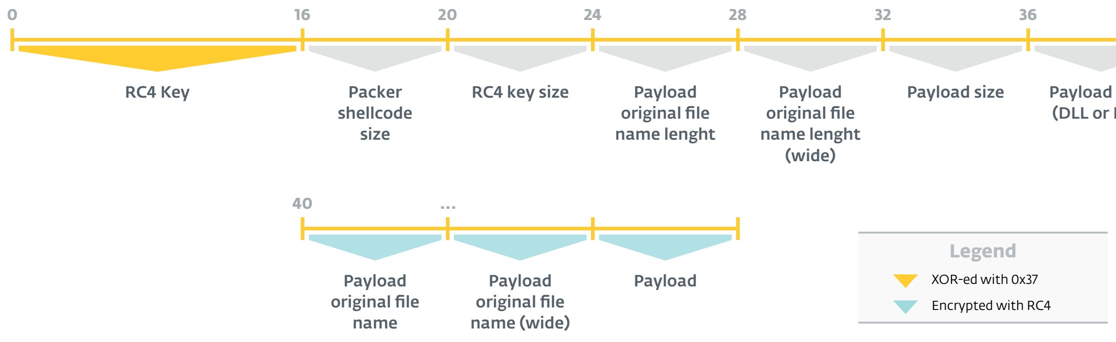


Figure 1 // Overview of artefacts, techniques, events and their relationships used by the Winnti Group

Some of the components shown in this graphic have already been documented before. This report will dig into the new or updated ones. It will also describe techniques that are used and are important because they allow to link several incidents together.

4. WINNTI'S CUSTOM PACKER

In our previous research [1] on the Winnti Group, we discussed a custom packer used in payloads embedded in compromised videogames and gaming application. It uses a unique structure seen in Figure 1 to embed a PE file. In addition to an RC4 key and the encrypted PE, this structure contains encrypted metadata such as path to the embedded PE file and a launch type value. The packer's configuration structure is shown in Figure 2.



0
16
20
24
28
32
36
40
RC4 Key
Packer shellcode size
RC4 key size
Payload original file name lenght
Payload original file name lenght (wide)
Payload size
Payload type (DLL or EXE)
40
...
Payload original file name
Payload original file name (wide)
Payload
Legend
XOR-ed with 0x37
Encrypted with RC4

Figure 2 // Structure used by the custom packer

The RC4 key (which is XOR-ed with `0x37`) contained in the structure is used to decrypt the PE as well as the encrypted file name and path. The launch type value can be either 1 or 2 and indicates to the unpacking code, to be able to load it properly, whether the PE is an executable or a DLL. It's interesting to note that the RC4 key contains only digits.

It is also worth noting that there are 32- and 64-bits versions of this packer. Examples of the packer's configuration found in video-games and gaming apps discussed in our previous blogpost [1] are shown in Table 1.

Table 1 *Packer configurations found in video-games and gaming apps*

Parent SHA-1	`0f31ed081ccc18816ca1e3c87fe488c9b360d02f`
Payload SHA-1	`dde82093decde6371eb852a5e9a1aa4acf3b56ba`
RC4 key	`17858542`
File name	`111.bin.tmp`
Launch type	`2 (DLL)`
Parent SHA-1	`42f2fc15aa8b9ed896c92fed22a27df9ef9db0ad`
Payload SHA-1	`a260dcf193e747cee49ae83568eea6c04bf93cb3`
RC4 key	`165122939`
File name	`111.bin.tmp`
Launch type	`2 (DLL)`
Parent SHA-1	`7cf41b1acfb05064518a2ad9e4c16fde9185cd4b`
Payload SHA-1	`8272c1f41f7c223316c0d78bd3bd5744e25c2e9f`
RC4 key	`1729131071`
File name	`111.bin.tmp`
Launch type	`1 (PE)`

We developed a standalone unpacking Python script that can be found in our *GitHub repository*.

5. PORTREUSE BACKDOOR

After analyzing the custom packer used by the Winnti Group, we started hunting for more executable files with this packer, in the hope of unearthing other compromised software used in supply-chain attacks. What we've found is not exactly what we were looking for to begin with. Instead of finding compromised software, we discovered a new listening-mode modular backdoor that uses the same packer. We believe its author call it *PortReuse*. This is not a random name: this backdoor injects into a running process already listening on a TCP port, "reusing" an already open port. It hooks the receiving function and waits for a "magic" packet to trigger the malicious behavior. The legitimate traffic is forwarded to the real application, so it is effectively not blocking any legitimate activity on the compromised server. This type of backdoor is sometimes called a passive network implant.

As we mentioned, this backdoor is using the same packing structure as the one used in 2018. It's actually using it in a recursive way: packed components also contain packed components. The metadata of the packed PE files are actually interesting here: the file name field contains meaningful values and even absolute paths in some cases. In this report, we will be describing each component using the names the malware authors have given to these components.

5.1 Modular architecture

The *PortReuse* backdoor exhibits a modular architecture, since all its components are separate processes communicating through named pipes, as shown in Figure 3. This allows reusing existing binary components and replacing only the components that need customization. For instance, we have seen multiple *PortReuse* variants with a different *NetAgent* but using the same *SK3*. *ProcTran* and *UserFunction* exist in 32- and 64-bits versions but can communicate with any SK3 regardless of its version, since they share a common protocol through the named pipe.

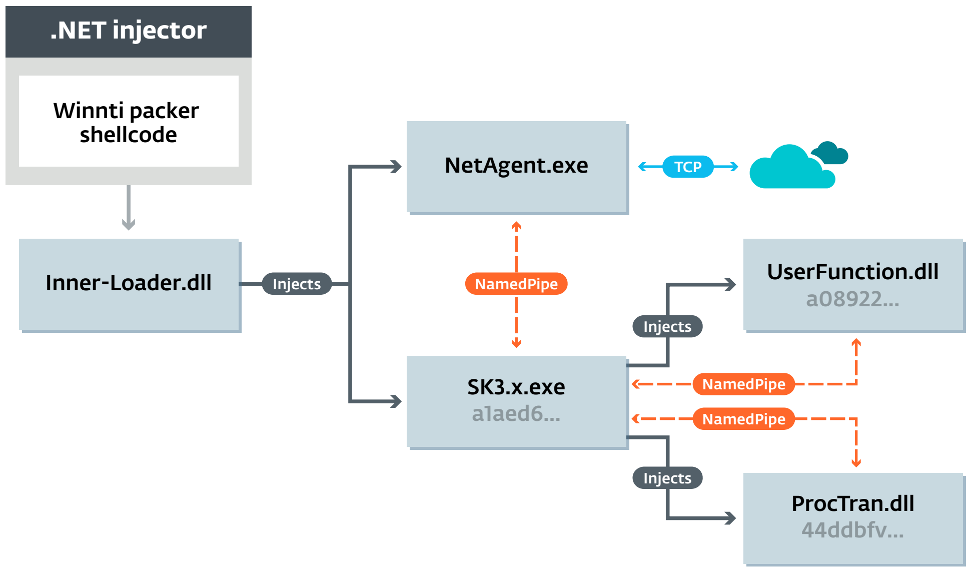


Figure 3 // PortReuse backdoor architecture

There is no C&C server in the backdoor. We have only seen *NetAgent* listening on open sockets. The attacker needs to connect directly to the compromised host.

5.2 Distribution

Only a single file is written to disk to start *PortReuse*. All other components exist in memory only. The initial launch file was found in different formats:

- Embedded in a .NET application launching the initial Winnti packer shellcode
- In a VB script that deserializes and invokes a .NET object that launches the shellcode
- In an executable that has the shellcode directly at the entry point

5.3 InnerLoader

InnerLoader is the first component to be decrypted and launched. As InnerLoader.dll is found packed with the custom packer, we were able to extract the packer metadata. The metadata from the packer, including absolute file path when it was packed, is shown in Table 2.

Table 2 *Inner-Loader packer embedded configuration*

Parent SHA-1	`395e87c5bd00f78bf4c63880c6982a7941a2ecd0`
Payload SHA-1	`97709d62531d12a6994bce5787d519db52435a62`
RC4 key	`761775049`
File name	`E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-WSAAccept\Inner-Loader\x64\Release\Inner-Loader.dll`
Launch type	`2`

Parent SHA-1	`7e9dba96adb34daf2f11d30272d9462bbfc6b321`
Payload SHA-1	`252640016faeff97fa22eb2b736973ed16d73fbe`
RC4 key	`876426830`
File name	`E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-53\Inner-Loader\x64\Release\Inner-Loader.dll`
Launch type	`2`

As shown in the absolute file path, the name of the project from which *InnerLoader* belongs is called *PortReuse*. *InnerLoader* will look for a given process to inject two payloads. In the case of the .NET injector, *InnerLoader* targets a process called `GameServer_NewPoker.exe` and in the case of the VBS injector it will look for a process listening on port 53 (DNS). These payloads are, again, packed using the same packer and are called *NetAgent* and *SK3* according the packer configuration. These packer configurations are shown in Table 3.

Table 3 *NetAgent and SK3 packer embedded configuration*

Parent SHA-1	97709d62531d12a6994bce5787d519db52435a62
Payload SHA-1	e14a6a8447ce1d45494e613d6327430d9025a2e5
RC4 key	761595243
File name	E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-WSAAccept\NetAgent\x64\Release\NetAgent.exe
Launch type	1

Parent SHA-1	97709d62531d12a6994bce5787d519db52435a62
Payload SHA-1	a1aed6fd6990a74590864f9d2a6e714a715fce3e
RC4 key	761595211
File name	E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-WSAAccept\SK3.x\x64\Release\SK3.x.exe
Launch type	1

Parent SHA-1	252640016faeff97fa22eb2b736973ed16d73fbe
Payload SHA-1	74a68dad4bc87eacca93106832f8b4aee82843a2
RC4 key	7125922
File name	E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-53\NetAgent\x64\Release\NetAgent.exe
Launch type	1

Parent SHA-1	252640016faeff97fa22eb2b736973ed16d73fbe
Payload SHA-1	e0f276ed16027ed2953a7b0e5274d3f563a75a9d
RC4 key	250574172
File name	E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-53\SK3.x\x64\Release\SK3.x.exe
Launch type	1

5.4 NetAgent and raw TCP hooking

NetAgent is the module responsible for handling TCP hooking. Depending on the *NetAgent* version, two different hooking techniques are used.

NetAgent will hook either `WSARecv` or `NtDeviceIoControlFile`. In both cases, the hook will first check if the received packet complies to a given magic packet format as already seen in the case of the Winnti malware [6].

The magic packet should contain the following binary data (14 bytes):

```
2c af da 56 16 3b 3a 76 27 73 59 54 96 b9
```

This format was also observed in samples hooking data received on port 53 (DNS) (12 bytes):

```
ff ff 01 00 00 01 00 00 00 00 00 00
```

This packet is actually a valid DNS request header with transaction ID 65535 (0xFFFF), the *recursion desired* flag set (Flags = 0x0100) and one question, as shown in Figure 5.

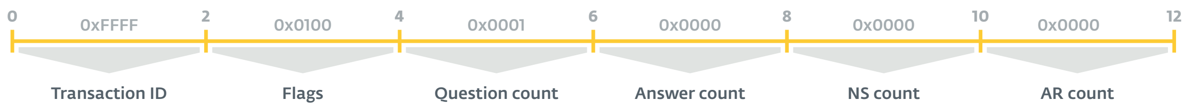
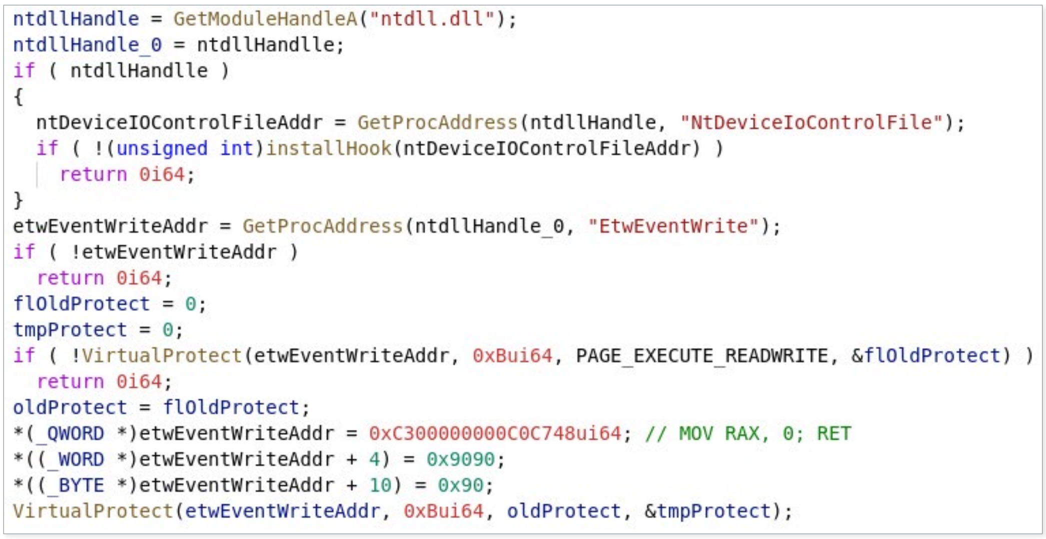


Figure 5 // DNS structure of the magic packet

This allows blending into the network traffic to avoid detection. A request to the backdoor must have the specific transaction ID to trigger which, in normal circumstances, should be chosen at random by DNS clients.

After having received the magic packet, the hook will start forwarding network traffic through the named pipe "`Microsoft Ole Object {30000-7100-12985-00000-00002}`" to the *SK3* module. Until the magic packet is received, the hook passes the execution to the hooked function.

One of the *NetAgent* variants also tries to hide its activity by disabling Event Tracing for Windows (ETW) for the injected process by patching the beginning of `EtwEventWrite` function with "`MOV RAX, 0; RET`" which will then always blindly return zero as shown in Figure 6.



```
ntdllHandle = GetModuleHandleA("ntdll.dll");
ntdllHandle_0 = ntdllHandlle;
if ( ntdllHandlle )
{
  ntDeviceIOControlFileAddr = GetProcAddress(ntdllHandle, "NtDeviceIoControlFile");
  if ( !(unsigned int)installHook(ntDeviceIOControlFileAddr) )
    return 0i64;
}
etwEventWriteAddr = GetProcAddress(ntdllHandle_0, "EtwEventWrite");
if ( !etwEventWriteAddr )
  return 0i64;
flOldProtect = 0;
tmpProtect = 0;
if ( !VirtualProtect(etwEventWriteAddr, 0xBui64, PAGE_EXECUTE_READWRITE, &flOldProtect) )
  return 0i64;
oldProtect = flOldProtect;
*(_QWORD *)etwEventWriteAddr = 0xC300000000C0C748ui64; // MOV RAX, 0; RET
*((_WORD *)etwEventWriteAddr + 4) = 0x9090;
*((_BYTE *)etwEventWriteAddr + 10) = 0x90;
VirtualProtect(etwEventWriteAddr, 0xBui64, oldProtect, &tmpProtect);
```

Figure 6 // Decompiled PortReuse hooking procedure

5.5 SK3

The communication between the backdoor and its client is RC4-encrypted using the key "`CreateThread`" and XOR-encoded with `0x77`. The SK3 module is responsible for decrypting and processing the traffic forwarded by *NetAgent* through the named pipe. The decrypted command format is `CMD_ID CMD_ARGS` where `CMD_ID` is an integer. The list of the supported commands is shown in Table 4.

Table 4 *Commands supported by SK3*

CMD_ID	CMD_ARGS	Behavior
200	`cd path`	Set current working directory
	`dir`	List file in the current directory
	`copy from_path to_path`	Copy file
	`move from_path to_path`	Move file
	`del path`	Remove file
	`systeminfo`	Execute `systeminfo.exe`
201	`PID`	Inject `UserFunction.dll` to the process with the `PID` provided as argument and send the command to it through named pipe, otherwise execute the command directly with `cmd.exe`
202	`FilePath DataToWrite`	Write data to the file provided as argument
203	`FilePath`	Read data from file
207	`N/A`	Close the named pipe with ProcTran and all the connections
210	`Hostname Port`	Connect to Hostname:Port via TCP and forward the received traffic via HTTP. Map the connection with an ID
211	`Id Data`	Send data through the named pipe connected to ProcTran if present; otherwise, use the connection mapped with the provided ID
214	`Id`	Shutdown the connection mapped to the provided ID
250	`N/A`	List processes and their modules
251	`PID`	Kill the process with the given PID
252	`Timestamp`	Change the timestamp of the file
253	`N/A`	Execute the following WQL query: `SELECT Name,ProcessId,Session-Id,CommandLine,ExecutablePath FROM Win32_Process`
261	`PID`	Inject *ProcTran* to the process with the PID provided as argument and send back the traffic received through the named pipe via HTTP

Except in the case of `systeminfo`, the commands with ID 200 are custom implementations of standard Windows commands.

5.6 UserFunction and ProcTran

The SK3 module also contains two packed executables with the following packer configuration:

Table 5 *UserFunction and ProcTran packer configurations*

Parent SHA-1	`A1AED6FD6990A74590864F9D2A6E714A715FCE3E`
Payload SHA-1	`A08922372042B4C3C0FAA120E9DD626823CDB3C7`
RC4 key	852676270
File name	UserFunction.dll
Launch type	2

Parent SHA-1	`A1AED6FD6990A74590864F9D2A6E714A715FCE3E`
Payload SHA-1	`44DDBF7AA256A4B0E25DE585E95EA520BF2C4891`
RC4 key	852679796
File name	ProcTran.dll
Launch type	2

UserFunction and *ProcTran* are started by *SK3* and are responsible for executing commands in other processes (see command 201) and proxying communications. Communication between *SK3* and *ProcTran* and *UserFunction* is done through the named pipes "`Microsoft Ole Object {30000-7100-12985-00000-00001}`" and "`Microsoft Ole Object {30000-7100-12985-00000-00000}`" respectively.

Commands supported by ProcTran to handle network forwarding are shown in Table 6.

Table 6 *Commands supported by ProcTran*

Command ID	Arguments	Decription
`210`	`Hostname Port`	Initiate a new connection to Hostname:Port and start forwarding received data. The connection is associated to an ID
`211`	`ID Data`	Send the data received through the named pipe to the connection corresponding to the given ID
`214`	`ID`	Close the connection corresponding to the given ID

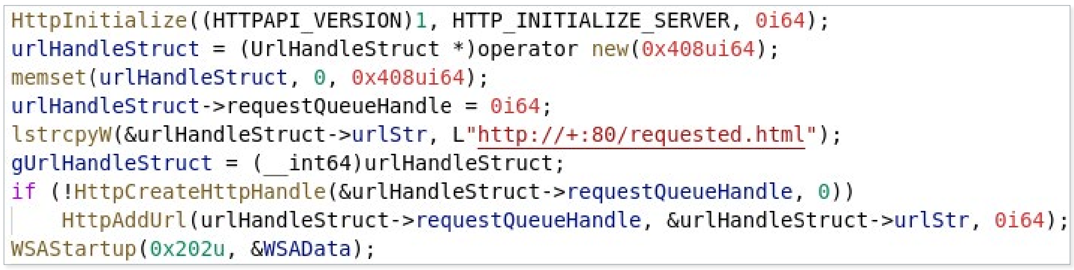
5.7 Passive HTTP listening

In addition to the variants using *NetAgent* to handle the network hook and forwarding the traffic through a named pipe to *SK3*, we also found *PortReuse* variants where *NetAgent* and SK3 were merged in one single module responsible for both, it uses a Windows Server API functionality to route requests for a given URL to the backdoor by using the following *UrlPrefix*:

```
http://+:[port]/requested.html
```

When routing traffic, the strong wildcard (`+`) matches all possible host names and will always be applied first, since it takes precedence over the weak wildcard (`*`) and explicit hostnames [7]. Every connection to this URL will then be redirected to *SK3*.

The decompiled procedure used to register a UrlPrefix is shown in Figure 7:



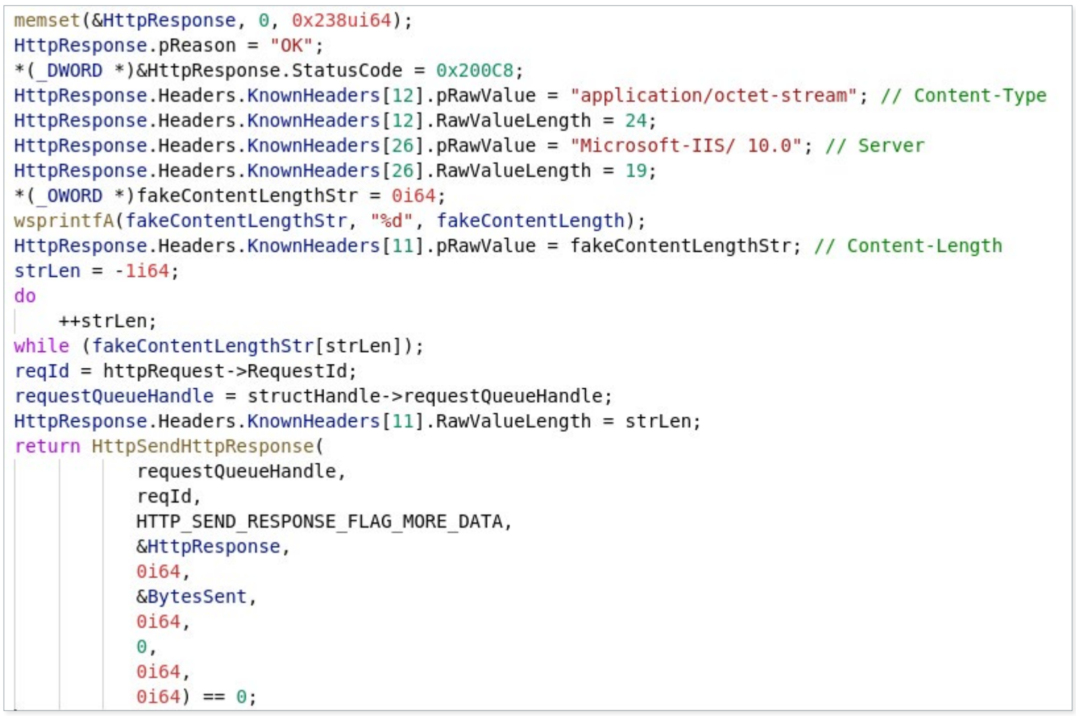
```
HttpInitialize((HTTPAPI_VERSION)1, HTTP_INITIALIZE_SERVER, 0i64);
urlHandleStruct = (UrlHandleStruct *)operator new(0x408ui64);
memset(urlHandleStruct, 0, 0x408ui64);
urlHandleStruct->requestQueueHandle = 0i64;
lstrcpyW(&urlHandleStruct->urlStr, L"http://+:80/requested.html");
gUrlHandleStruct = (__int64)urlHandleStruct;
if (!HttpCreateHttpHandle(&urlHandleStruct->requestQueueHandle, 0))
    HttpAddUrl(urlHandleStruct->requestQueueHandle, &urlHandleStruct->urlStr, 0i64);
WSAStartup(0x202u, &WSAData);
```

Figure 7 // Decompiled UrlPrefix registration procedure

In the case where a GET request matching the `UrlPrefix` is received, the backdoor will send back a custom HTTP response faking a Microsoft IIS 10.0 response—`Content-Length` header being the result of `(GetTickCount() % 700 + 600) << 20`:

```
HTTP/1.1 200 OK
Content-Length: 1034944512
Content-Type: application/octet-stream
Server: Microsoft-IIS/ 10.0 Microsoft-HTTPAPI/2.0
Date: Wed, 20 Mar 2019 20:02:49 GMT
```

Note that there is a space between `Microsoft-IIS/` and `10.0` and that `Microsoft-HTTPAPI/2.0` is appended to the server header—in Figure 8.



```
memset(&HttpResponse, 0, 0x238ui64);
HttpResponse.pReason = "OK";
*(_DWORD *)&HttpResponse.StatusCode = 0x200C8;
HttpResponse.Headers.KnownHeaders[12].pRawValue = "application/octet-stream"; // Content-Type
HttpResponse.Headers.KnownHeaders[12].RawValueLength = 24;
HttpResponse.Headers.KnownHeaders[26].pRawValue = "Microsoft-IIS/ 10.0"; // Server
HttpResponse.Headers.KnownHeaders[26].RawValueLength = 19;
*(_OWORD *)fakeContentLengthStr = 0i64;
wsprintfA(fakeContentLengthStr, "%d", fakeContentLength);
HttpResponse.Headers.KnownHeaders[11].pRawValue = fakeContentLengthStr; // Content-Length
strLen = -1i64;
do
    ++strLen;
while (fakeContentLengthStr[strLen]);
reqId = httpRequest->RequestId;
requestQueueHandle = structHandle->requestQueueHandle;
HttpResponse.Headers.KnownHeaders[11].RawValueLength = strLen;
return HttpSendHttpResponse(
           requestQueueHandle,
           reqId,
           HTTP_SEND_RESPONSE_FLAG_MORE_DATA,
           &HttpResponse,
           0i64,
           &BytesSent,
           0i64,
           0,
           0i64,
           0i64) == 0;
```

Figure 8 // Decompiled procedure for handling the HTTP response on GET requests

In the case where a `POST` request matching the `UrlPrefix` is received, the backdoor will initiate communication and execute commands.

5.8 Multiple variants to target different ports

The *PortReuse* backdoor is targeting various commonly used ports such as 53 (DNS over TCP), 80, 443, 3389 (Remote Desktop Protocol), and 5985 (Windows Remote Management). One of the variants we found is also port-agnostic: it parses the TCP header and triggers only if the source port is less than 22. The backdoor must first be injected into a running process that, in order to install the networking hook, is already listening on the targeted port. In Table 7 are shown various targets of the *PortReuse* backdoor along with their filenames from the custom packer and the port reuse technique used:

Table 7 *Hooking techniques and targeted ports*

SHA-1 and file name	Target	Port reuse technique
SHA-1 `395e87c5bd00f78bf4c63880c6982a7941a2ecd0` **File name** `E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-WSAAccept\Inner-Loader\x64\Release\Inner-Loader.dll`	TCP with source port less than 22 Inject into a process named `GameServer_NewPoker.exe`	`WSARecv` hook
SHA-1 `252640016faeff97fa22eb2b736973ed16d73fbe` **File name** `E:\code\PortReuse\3389-share\DeviceIOContrl-Hook\v1.3-53\Inner-Loader\x64\Release\Inner-Loader.dll`	Port 53 Inject into a process listening on port 53	`DeviceIOControl` hook

SHA-1 and file name	Target	Port reuse technique
SHA-1 `f5ba05240b1609d4131d5dca7f5e6e90b5748004` **File name** `Inner-Loader.dll`	Port 3389 Inject into a process that have loaded `termsrv.dll` RDP server	`DeviceIOControl` hook
SHA-1 `52a8c38890360d0b32993a44c9e94e660f3fa8f4` **File name** `E:\code\PortReuse\iis-share\2.5\IIS_Share\x64\Release\IIS_Share.dll`	Port 80 Posing as Microsoft IIS 10.0	`UrlPrefix` `http://+:80/requested.html`
SHA-1 `dbe3eece00c255a3fdf924b82621394377b0e865` **File name** `80.dll`	Port 80 Posing as Microsoft IIS 10.0	`UrlPrefix` `http://+:80/requested.html`
SHA-1 `A5B756F1EC956A00934D68940D4559694FAA8ED6` **File name** N/A	Port 443 Posing as Microsoft IIS 10.0	`UrlPrefix` `http://+:443/requested.html`
SHA-1 `1AECD365F5D0DEBA62026D84189BD180814D7292` **File name** N/A	Port 5985 Posing as Microsoft IIS 10.0	`UrlPrefix` `http://+:5985/requested.html`

It is worth noting that the Winnti malware used a kernel driver to hijack network communications while in that case the port reuse technique allows the backdoor to operate from userland.

5.9 Finding victims of PortReuse

Since the “magic” used to trigger the *PortReuse* code is now known, we can use it to find hosts connected to the Internet that could be compromised by this backdoor. In the case of the variants injecting in IIS it can be achieved by performing a GET request and inspecting the Server and Content-Length headers.

Since, based on all the samples we have analyzed, performing such a request does not produce any side effects, we asked the help of Censys to perform an Internet-wide scan so to identify potential victims.

Thanks to the Censys team, we were able to identify eight IP addresses that replied with an HTTP response matching the signature of *PortReuse*. We found that all eight of these IP addresses belonged to a single organization: a major mobile hardware and software manufacturer based in Asia. We notified the company and are working with the victim to remediate. It is possible that the Winnti Group was planning a devastating supply-chain attack by compromising this organization.

6. VMPROTECTED PACKER USED IN TARGETED ORGANIZATIONS

In addition to the Winnti malware, organizations targeted by Winnti were found to be compromised with a VMProtected DLL. Kaspersky mentions this VMProtected technique in the detailed article about ShadowHammer [5].

These VMProtected samples are actually responsible for decrypting a payload that is either embedded in its PE file overlay or read from `[Drive]:\$Recycle.Bin\COM1:NULL.DAT`. This file (`NULL.DAT`) is an NTFS Alternate Data Stream (ADS) on a file named “`COM1`”, which has the special meaning of being the first serial port it is a reserved name not to be used as a filename [8]. This makes working with the file a bit more difficult than usual.

To be able to read COM1 from the command prompt, its path needs to be prefixed with "`\\.\`".

Regardless of the location of the payload, the key derivation and decryption algorithms are always the same. The payloads are encrypted using RC5 in ECB mode and the decryption key is derived from the volume ID of the targeted machine as follows:

```
# Generate RC5 key from volume serial number
import os

volume_serial_number = os.stat("C:").st_dev
secret_string = "f@Ukd!rCto R$."
rc5_key = ""
pos = 0
for char in secret_string:
        byte_from_serial = (volume_serial_number >> ((pos & 3)*8)) & 0xff
        xored_value = (ord(char) ^ byte_from_serial)
        if pos % 4 == 0:
                rc5_key += "%02X" % xored_value
        else:
                rc5_key += "%02x" % xored_value
        pos += 1

print rc5_key
```

In most of the VMProtected samples we found, the string used to derive the decryption key from the volume ID is the same ("`f@Ukd!rCto R$.`"). In some more recent cases the derivation string was "`d37lo{r`". Once decrypted, the payload is position-independent codes executed in a separate thread.

The derivation string, key derivation from the volume ID and RC5 implementation are the same as used in Win64/Winnti.BN, which is the known second stage of the compromised videogames and gaming applications and are used to decrypt the third stage. Unlike the samples we are looking at here, Win64/Winnti.BN is not packed with VMProtect, but uses exactly the same key derivation and encryption algorithm.

ESET researchers were able to decrypt several payloads packed using this custom VMProtected packer. We found that the payload was either the *PortReuse* backdoor or the ShadowPad malware [4].

6.1 PortReuse backdoor

The *PortReuse* backdoors dropped by the VMProtected samples were all using the `UrlPrefix` technique on various ports and the `Server` header of the backdoor response was `Microsoft-IIS/ 10.0 Microsoft-HTTPAPI/2.0`. The backdoor was observed to be in use at least two organizations.

Considering that the PortReuse backdoor was found in VMProtected droppers similar to what was previously described by Kaspersky with Operation ShadowHammer [5], in addition to the fact that the same decryption algorithm is used by the 2nd stage from the compromised video-games and also by the gaming application uncovered by ESET, strongly suggests that Operation ShadowHammer and these supply-chain attacks are connected and that the PortReuse backdoor is part of the Winnti Group arsenal.

6.2 ShadowPad

As mentioned previously, some of the payloads dropped by the VMProtected samples were similar to the ShadowPad malware, using the same plugin architecture with identical module ID, similar embedded configurations and encryption schemes.

Interestingly, if we look at the modules timestamps, we can see that public webpages to retrieve the C&C server instead of DGA (Domain Generation Algorithm) started to be used from at least the 25th August 2017, less than two weeks after the publication on the NetSarang compromise by Kaspersy [4]. The Module IDs along with their names and timestamps from various samples are shown in Table 8, Table 9 and Table 10.

Table 8 *Module ID and their respective names and timestamps*

Module ID	Module name	Module timestamp
100	Root	Fri 25 Aug 2017 04:40:54 AM UTC
101	Plugins	Fri 25 Aug 2017 04:39:02 AM UTC
102	Config	Fri 25 Aug 2017 04:39:10 AM UTC
103	Install	Fri 25 Aug 2017 04:40:15 AM UTC
104	Online	Fri 25 Aug 2017 04:39:27 AM UTC
200	TCP	Fri 25 Aug 2017 04:40:15 AM UTC
201	HTTP	Fri 25 Aug 2017 04:35:45 AM UTC
202	UDP	Fri 25 Aug 2017 04:35:54 AM UTC

Table 9 *Module ID and their respective names and timestamps*

Module ID	Module name	Module timestamp
100	Root	Wed 21 Mar 2018 11:09:32 AM UTC
101	Plugins	Wed 21 Mar 2018 10:53:13 AM UTC
102	Config	Wed 21 Mar 2018 10:53:17 AM UTC
103	Install	Wed 21 Mar 2018 10:54:07 AM UTC
104	Online	Wed 21 Mar 2018 10:53:30 AM UTC
200	TCP	Wed 21 Mar 2018 10:50:44 AM UTC
201	HTTP	Wed 21 Mar 2018 10:50:52 AM UTC
202	HTTPS	Wed 21 Mar 2018 10:50:58 AM UTC

In addition to these ShadowPad variants using the same IDs as described in [4], we found an updated version with modules that, according to their timestamps, were compiled in 2019 according to their timestamps, with additional obfuscations and using random module IDs.

Table 10 *Updated module ID and their respective names and timestamps*

Module ID	Module name	Module timestamp
58338	DNS	Fri 15 Mar 2019 05:22:19 PM UTC
3331	TCP	Fri 15 Mar 2019 05:21:40 PM UTC
22707	UDP	Fri 15 Mar 2019 05:22:13 PM UTC
48503	HTTP	Fri 15 Mar 2019 05:21:48 PM UTC
33173	HTTPS	Fri 15 Mar 2019 05:21:56 PM UTC
4626	Root	Fri 15 Mar 2019 05:23:15 PM UTC
12996	Config	Fri 15 Mar 2019 05:23:26 PM UTC
61013	Plugins	Fri 15 Mar 2019 05:23:21 PM UTC
5176	Online	Fri 15 Mar 2019 05:23:49 PM UTC
35573	Install	Fri 15 Mar 2019 05:23:43 PM UTC

The Config module maintains an encrypted string pool that contains two URLs of a public profile or publicly posted document on legitimate websites, which are used to retrieve and decrypt the real C&C server URL.

In addition to these URL, the string pool contains a string similar to a campaign identifier that was related to the targeted victims. In previous versions of the backdoor, the string pool starts at offset 0x56 relative to the beginning of the configuration block, while for the updated version, the offset is 0x5c.

The algorithm to decrypt the C&C address is similar to the one used in older versions of ShadowPad. First, the string enclosed between consecutive '`$`' delimiters from the publicly posted document is extracted. Then each character is decremented by 0x61 ('a') and each resulting pair of four bits is then concatenated into a byte and the resulting buffer decrypted using the decryption routine that is also used to decrypt the strings from rest of the code.

The string decryption algorithms used differ between samples, but follows a pattern similar to the following Python transcription:

```
decryption_buf = []
xor_key = (0xff & cyphertext[0]) | (0xff00 & (cyphertext[1] << 8))

for byte in cyphertext[2:]:
    decryption_buf.append((0xff & xor_key) ^ byte)
    xor_key = (0xffffffff & (0xffffffff & (0x8CC70000 * xor_key))
               - (0xffffffff & (0x1B507339 * ((xor_key >> 16))))
               - 0x70A927AC)

print "".join(map(chr, decryption_buf))
```

The pages used to retrieve the C&C from the payloads we decrypted are shown in Table 11.

Table 11 *URL of pages containing encrypted C&C and their respective decrypted C&C*

URL	Decrypted C&C

`URL://https://docs.google[.]com/document/d/1jcRsFZM59x_4AKJabmz8sPFsKOZArV4bTn3WsYonUns`	From document history:
	`HTTPS://154.223.131[.]237:443`
	`UDP://117.16.142[.]9:443`
	`UDP://103.19.3.109:443`

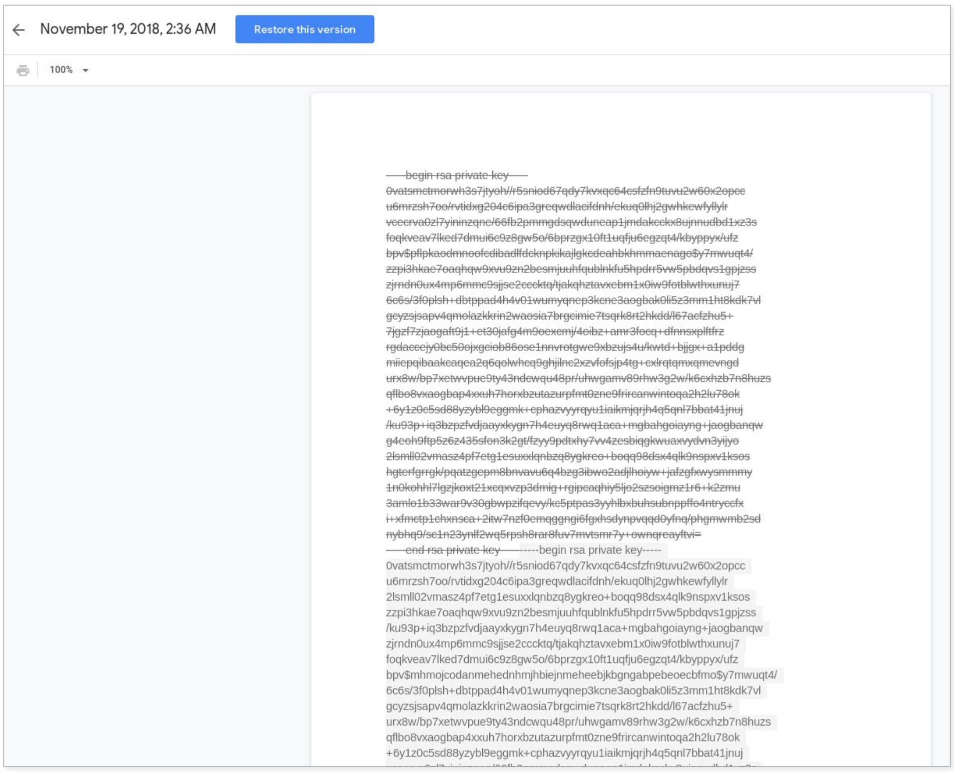
`URL://https://docs.google[.]com/document/d/1KJ_RJRtkKhcuJjXOCKtEOLuwH3sRi72PUhtfukncyRc`	Access denied

`URL://https://docs.google[.]com/document/d/1T5P3SS-QTO1nOS6IlKFA_chimnMPmhon8E_kuRSodWw`	From document history:
	`UDP://110.45.146.253:443`
	`UDP://110.45.146.254:443`
	`TCP://110.45.146.254:443`
	`UDP://117.16.142.69:443`
	`UDP://122.10.117.206:443`
	`UDP://207.148.125.56:443`
	`UDP://118.193.236.206:443`
	`UDP://167.88.176.205:443`
	`UDP://167.88.176.205:443`
	`UDP://103.224.83.95:443`
	`UDP://103.19.3.21:443`
	`TCP://103.19.3.21:443`

`URL://https://steamcommunity[.]com/id/869406565`	C&C removed
`URL://https://steamcommunity[.]com/id/61198869528`	C&C altered
`URL://https://raw.githubusercontent[.]com/Enterprise-Backup/windows/master/Readme.html`	C&C removed

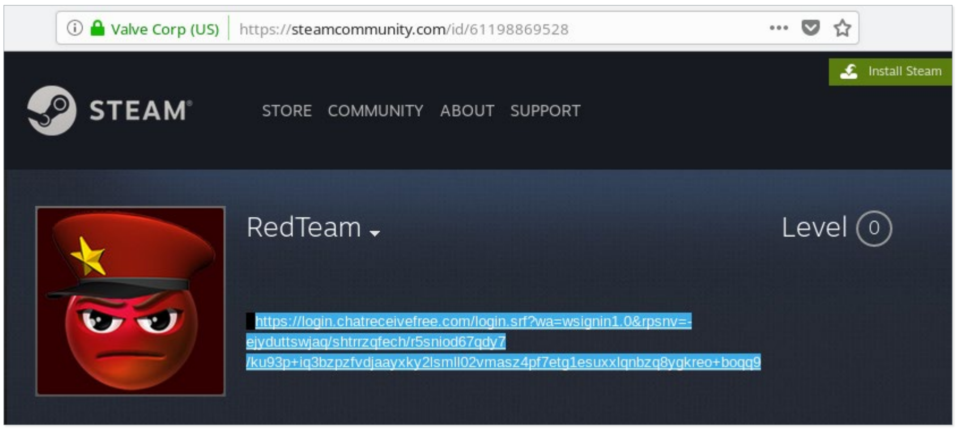
`URL://https://pastebin[.]com/JgduT7NH`	C&C removed
`URL://https://docs.google[.]com/document/d/1-vFbL5nw85uJeS-X9sYEJ0CAsUzJE3kidJg6Gg_vZ7s`	C&C removed
`URL://https://social.msdn.microsoft[.]com/profile/Pf9Je@`	C&C removed

Unfortunately, the attackers cleaned all the page content before we had access to the malware samples. However, two documents hosted on Google Docs contained information in the document history that enabled us to list the C&C servers they used for that campaign. The supported protocols to contact the C&C server are HTTP, HTTPS, UDP, TCP and DNS.



~~-----begin rsa private key-----~~
~~0vatsmctmorwh3s7jtyoh//r5sniod67qdy7kvxqc64csfzfn9tuvu2w60x2opcc~~
~~u6mrzsh7oo/rvtidxg204c6ipa3greqwdlacifdnh/ekuq0lhj2gwhkewfyllylr~~
~~vccerva0zl7yininzqnc/66fb2pmmgdsqwduncap1jmdakcckx8ujnnudbd1xz3s~~
~~foqkveav7lked7dmui6c9z8gw5o/6bprzgx10ft1uqfju6egzqt4/kbyppyx/ufz~~
~~bpv$pflpkaodmnoofcdibadlfdcknpkikajlgkcdeahbkhmmacnago$y7mwuqt4/~~
~~zzpi3hkae7oaqhqw9xvu9zn2besmjuuhfqublnkfu5hpdrr5vw5pbdqvs1gpjzss~~
~~zjrndn0ux4mp6mmc9sjjse2cccktq/tjakqhztavxebm1x0iw9fotblwthxunuj7~~
~~6c6s/3f0plsh+dbtppad4h4v01wumyqnep3kcne3aogbak0li5z3mm1ht8kdk7vl~~
~~gcyzsjsapv4qmolazkkrin2waosia7brgcimie7tsqrk8rt2hkdd/l67acfzhu5+~~
~~7jgzf7zjaogaft9j1+ct30jafg4m9ocxcmj/4oibz+amr3focq+dfnnsxplftfrz~~
~~rgdacccjy0bc50ojxgciob86osc1nnvrotgwe9xbzujs4u/kwtd+bjjgx+a1pddg~~
~~miicpqibaakcaqca2q6qolwhcq9ghjilnc2xzvfofsjp4tg+cxlrqtqmxqmcvngd~~
~~urx8w/bp7xetwvpue9ty43ndcwqu48pr/uhwgamv89rhw3g2w/k6cxhzb7n8huzs~~
~~qflbo8vxaogbap4xxuh7horxbzutazurpfmt0zne9frircanwintoqa2h2lu78ok~~
~~+6y1z0c5sd88yzybl9eggmk+cphazvyyrqyu1iaikmjqrjh4q5qnl7bbat41jnuj~~
~~/ku93p+iq3bzpzfvdjaayxkygn7h4euyq8rwq1aca+mgbahgoiayng+jaogbanqw~~
~~g4coh9ftp5z6z435sfon3k2gt/fzyy9pdtxhy7vv4zesbiqgkwuaxvydvn3yijyo~~
~~2lsmll02vmasz4pf7etg1esuxxlqnbzq8ygkreo+boqq98dsx4qlk9nspxv1ksos~~
~~hgterfgrrgk/pqatzgepm8bnvavu6q4bzg3ibwo2adjlhoiyw+jafzgfxwysmmmy~~
~~1n0kohhl7lgzjkoxt21xcqxvzp3dmig+rgipcaqhiy5ljo2szsoigmz1r6+k2zmu~~
~~3amlo1b33war9v30gbwpzifqevy/kc5ptpas3yyhlbxbuhsubnppffo4ntryccfx~~
~~i+xfmctp1chxnsca+2itw7nzf0cmqggngi6fgxhsdynpvqqd0yfnq/phgmwmb2sd~~
~~nybhq9/sc1n23ynlf2wq5rpsh8rar8fuv7mvtsmr7y+ownqreayftvi=~~
~~-----end rsa private key-----~~-----begin rsa private key-----
0vatsmctmorwh3s7jtyoh//r5sniod67qdy7kvxqc64csfzfn9tuvu2w60x2opcc
u6mrzsh7oo/rvtidxg204c6ipa3greqwdlacifdnh/ekuq0lhj2gwhkewfyllylr
2lsmll02vmasz4pf7etg1esuxxlqnbzq8ygkreo+boqq98dsx4qlk9nspxv1ksos
zzpi3hkae7oaqhqw9xvu9zn2besmjuuhfqublnkfu5hpdrr5vw5pbdqvs1gpjzss
/ku93p+iq3bzpzfvdjaayxkygn7h4euyq8rwq1aca+mgbahgoiayng+jaogbanqw
zjrndn0ux4mp6mmc9sjjse2cccktq/tjakqhztavxebm1x0iw9fotblwthxunuj7
foqkveav7lked7dmui6c9z8gw5o/6bprzgx10ft1uqfju6egzqt4/kbyppyx/ufz
bpv$mhmojcodanmehednhmjhbiejnmeheebjkbgngabpebeoecbfmo$y7mwuqt4/
6c6s/3f0plsh+dbtppad4h4v01wumyqnep3kcne3aogbak0li5z3mm1ht8kdk7vl
gcyzsjsapv4qmolazkkrin2waosia7brgcimie7tsqrk8rt2hkdd/l67acfzhu5+
urx8w/bp7xetwvpue9ty43ndcwqu48pr/uhwgamv89rhw3g2w/k6cxhzb7n8huzs
qflbo8vxaogbap4xxuh7horxbzutazurpfmt0zne9frircanwintoqa2h2lu78ok
+6y1z0c5sd88yzybl9eggmk+cphazvyyrqyu1iaikmjqrjh4q5qnl7bbat41jnuj

Figure 9 // History of a public Google Docs document containing encrypted C&C URL



https://login.chatreceivefree.com/login.srf?wa=wsignin1.0&rpsnv=-
ejyduttswjaq/shtrrzqfech/r5sniod67qdy7
/ku93p+iq3bzpzfvdjaayxky2lsmll02vmasz4pf7etg1esuxxlqnbzq8ygkreo+boqq9

Figure 10 // Steam profile controlled by the operators

6.3 Persistence

These VMProtected samples persist on the system by using a DLL hijacking technique [7] similar to what was used by the 2nd stage of the CCleaner backdoor as described by CrowdStrike [8]. The DLL has an exported function that is used to copy itself to `C:\Windows\System32\TSVIPSrv.DLL`. It will then be loaded when the `SessionEnv` service which tries to load `TSVIPSrv.DLL` from the `System32` directory.

7. WINNTI VARIANTS

7.1 Several variants, same dropper

Some of the victims in the gaming industry were found to be compromised by different Winnti variants installed using the same dropper named `Install.exe` and `Install64.exe` for the 32- and 64-bits versions respectively.

7.2 Dropper (Install.exe)

This dropper uses a decryption key from the command line argument to decrypt the payload as follows:

```
c:\Install.exe {AES key} {path to encrypted payload}
```

The decryption key was always the same regardless of the Winnti variant dropped or the victim, and the decryption algorithm used by `Install.exe` is AES-256-CTR. This dropper has been used since at least mid-2017 and is still in use as of the time of writing.

7.3 Payloads

Several different Winnti versions are dropped by `Install.exe` and exhibit similarities shared with other variants previously uncovered such as:

- Decryption password of the dropper component is passed from command line argument [6]
- AES-encrypted payload [6].
- Use of DPAPI encryption [9] .
- Use of PlugX-like encryption [9].
- 32- and 64-bit versions of the rootkit are embedded in the loader [10].
- Copies itself to `%WINDIR%\system32\wbem\wbemcomn.dll` and replaces the original Windows DLL [9].
- Leverages the WMI performance adapter service (`wmiAPSrv`) [9].
- C&C URL and campaign ID are embedded in the configuration [6].

7.4 C&C servers and campaign ID

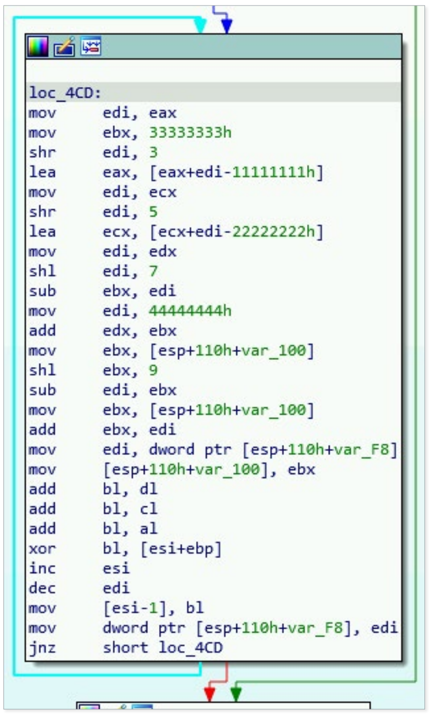
Winnti samples contain a configuration containing the C&C domain and a campaign ID. From the various samples we collected that were dropped by `Install.exe`, we were able to extract the following C&Cs and campaign IDs:

Table 12 *Winnti C&C servers and their corresponding campaign IDs*

C&C	Campaign ID
xp101.dyn-dns[.]co:443	`AA TT 0926`
svn-dns.ahnlabinc[.]com:443	`GRA KR 0629`
dns1-1.7release[.]com:443	`ENA KR2 0629`
dns1-1.7release[.]com:443	`EOS TW 0629`
ssl.dyn-dns[.]co:80	`gx DE 0705`

7.5 PlugX-like encryption

In some cases, the payload decrypted by `Install.exe` is a position-independent code with an embedded encrypted payload. This embedded payload is decrypted using the PlugX decryption algorithm before executing it, as shown in Figure 11.



```
loc_4CD:
mov     edi, eax
mov     ebx, 33333333h
shr     edi, 3
lea     eax, [eax+edi-11111111h]
mov     edi, ecx
shr     edi, 5
lea     ecx, [ecx+edi-22222222h]
mov     edi, edx
shl     edi, 7
sub     ebx, edi
mov     edi, 44444444h
add     edx, ebx
mov     ebx, [esp+110h+var_100]
shl     ebx, 9
sub     edi, ebx
mov     ebx, [esp+110h+var_100]
add     ebx, edi
mov     edi, dword ptr [esp+110h+var_F8]
mov     [esp+110h+var_100], ebx
add     bl, dl
add     bl, cl
add     bl, al
xor     bl, [esi+ebp]
inc     esi
dec     edi
mov     [esi-1], bl
mov     dword ptr [esp+110h+var_F8], edi
jnz     short loc_4CD
```

Figure 11 // PlugX-like decryption used by a decrypted wmi2.dat shellcode

The PlugX-like encryption algorithm is similar to the one mentioned in the Operation ShadowHammer blogpost, which suggests another link between the ShadowHammer campaign and the campaigns against in the video game industry previously uncovered by ESET. Moreover, Trend Micro also mentioned the use of PlugX-like encryption for the Winnti GitHub abuse campaign [9].

7.6 Use of custom AceHash builds

Winnti is also known to use custom AceHash builds [11]. 32- and 64-bit signed builds of this password dumper were also found among several victims compromised by Winnti variants installed from the `Install.exe` dropper, sometimes in conjunction with Mimikatz.

8. 3RD STAGE OF THE 2018 SUPPLY-CHAIN ATTACKS: WIN64/WINNTI.BN'S ENCRYPTED PAYLOAD

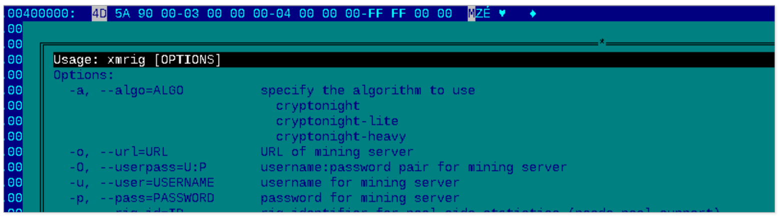
In our previous research [1], we mentioned that the 2nd stage delivered to the victims appends the extension `.mui` to its DLL path, reads that file and decrypts it using RC5 by deriving the decryption key from the volume ID of the victim's hard drive.

8.1 Same cryptography implementation

The RC5 and key derivation implementation are the same as the one used in the ShadowPad VMProtected launcher. The string "`f@Ukd!rCto R$.`" is used to derive the decryption key from the volume ID ID – the same key used by the PortReuse and ShadowPad VMProtected launcher. These findings add another connection between operation ShadowHammer and compromise campaign in the video-game industry.

8.2 Monero miner

We were able to obtain and decrypt the `.mui` files loaded by Win64/Winnti.BN. They were actually XMRig executables, again packed using the same custom packer as the one used to pack the 1st stage of compromised games as well as the PortReuse backdoor.



```
00400000:  4D 5A 90 00-03 00 00 00-04 00 00 00-FF FF 00 00  MZÉ ♥   ♦
Usage: xmrig [OPTIONS]
Options:
  -a, --algo=ALGO          specify the algorithm to use
                             cryptonight
                             cryptonight-lite
                             cryptonight-heavy
  -o, --url=URL            URL of mining server
  -O, --userpass=U:P       username:password pair for mining server
  -u, --user=USERNAME      username for mining server
  -p, --pass=PASSWORD      password for mining server
```

Figure 12 // Strings from a XMRig decrypted sample

XMRig is an open-source Monero (XMR) CPU miner and is launched with the following command line:

```
x -c wcnapi.mui
```

where `wcnapi.mui` is the configuration file provided to XMRig. Unfortunately, we do not have a sample of the configuration file. In any event, this sheds some light on the financial motivation behind this campaign.

Even though we observed only XMRig as final payload, we cannot exclude the possibility that attackers may have sent different payload to victims of interest based on the MAC addresses collected by the 1st stage.

9. CONCLUSION

The Winnti group is capable of breaching large organizations via many different means. Given how complex the backdoor and methods they use are, we think they have significant resources and time allocated to malicious activities.

There are a lot of discussions among malware researchers regarding attribution. More specifically, whether or not the different attacks, both espionage and for financial gain, are performed by the same people or organization. Given the amount of work required, we know these attacks aren't performed by a single individual. The perpetrators running the different operations may or may not be the authors of the actual malware we see. They may share tools among different teams and what we see is only a representation of what they accomplish as a team. Or is it multiple teams?

The fact that clustering is hard here doesn't mean we can't link the different incidents. Given the code and techniques they reuse, we can make strong correlations between incidents. However, by looking only at samples and events, it would be speculation to try to find out how the organization actually works. Who profits from financial gain? Do they mine cryptocurrencies to finance their own activities or do they use the same malware outside of their work hours? There is currently no evidence that would help answer these questions with certainty.

The Winnti Group is still very active in 2019 and continues to target both gaming and other industries. The update to the ShadowPad malware shows they are still developing and using it. The relatively new PortReuse malware also shows they update their arsenal and give themselves an additional way to compromise their victims for a long period of time.

10. REFERENCES

1 M-E. M.Léveillé, ESET, "Gaming industry still in the scope of attackers in Asia," 11 March 2019. [Online]. Available: *https://www.welivesecurity.com/2019/03/11/gaming-industry-scope-attackers-asia/*.

2 Microsoft Defender ATP Research Team, "Detecting threat actors in recent German industrial attacks with Windows Defender ATP," 25 January 2017. [Online]. Available: *https://www.microsoft.com/security/blog/2017/01/25/detecting-threat-actors-in-recent-german-industrial-attacks-with-windows-defender-atp/*.

3 GReAT, "Winnti. More than just a game," 11 April 2013. [Online]. Available: *https://securelist.com/winnti-more-than-just-a-game/37029/*.

4 GReAT, "ShadowPad: popular server management software hit in supply chain attack," 15 August 2017. [Online]. Available: *https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2017/08/07172148/ShadowPad_technical_description_PDF.pdf*.

5 GReAT, "Operation ShadowHammer: a high-profile supply chain attack," 23 April 2019. [Online]. Available: *https://securelist.com/operation-shadowhammer-a-high-profile-supply-chain-attack/90380/*.

6 T. Haruyama, "Winnti Polymorphism," 2016. [Online]. Available: *https://hitcon.org/2016/pacific/0composition/pdf/1201/1201%20R2%201610%20winnti%20polymorphism.pdf*.

7 D. Hohnstein, "Lateral Movement — SCM and DLL Hijacking Primer," 18 April 2019. [Online]. Available: *https://posts.specterops.io/lateral-movement-scm-and-dll-hijacking-primer-d2f61e8ab992*.

8 K. Sood, "CCleaner Stage 2: In-Depth Analysis of the Payload," 10 November 2017. [Online]. Available: *https://www.crowdstrike.com/blog/in-depth-analysis-of-the-ccleaner-backdoor-stage-2-dropper-and-its-payload/*.

9 C. Pernet, "Winnti Abuses GitHub for C&C Communications," 22 March 2017. [Online]. Available: *https://blog.trendmicro.com/trendlabs-security-intelligence/winnti-abuses-github/*.

10 Macnica Networks, "Japan Security Analyst Conference 2018," 25 January 2018. [Online]. Available: *https://www.jpcert.or.jp/present/2018/JSAC2018_09_yanagishita-takeuchi.pdf*.

11 Hiroshi Takeuchi, Hajime Yanagishita - Macnica Networks, "Catch Painful TTPs for Adversaries," 2018. [Online]. Available: *https://hitcon.org/2018/pacific/downloads/1214-R2/1330-1400.pdf*.

12 Novetta, "Winnti Analysis," April 2015. [Online]. Available: *https://www.novetta.com/wp-content/uploads/2015/04/novetta_winntianalysis.pdf*.

13 T. Hegel, "Burning Umbrella: An Intelligence Report on the Winnti Umbrella and Associated State-Sponsored Attackers," 3 May 2018. [Online]. Available: *https://401trg.com/burning-umbrella/*.

11. INDICATORS OF COMPROMISE

11.1 ESET detection names

```
Win32/Winnti trojan
Win64/Winnti trojan
MSIL/Injector.UNL trojan
Win32/CoinMiner.DV potentially unwanted application
Win32/Spy.Agent.ORQ trojan
Win64/Agent.HE trojan
Win64/Agent.HH trojan
Win64/Agent.NM trojan
Win64/CoinMiner.DN potentially unwanted application
Win64/Injector.BS trojan
Win64/Injector.BT trojan
Win64/Packed.VMProtect.FH trojan
Win64/Packed.VMProtect.FI trojan
Win64/Spy.Agent.F trojan
Win64/TrojanDropper.Agent.AM trojan
Win64/TrojanDropper.Agent.CJ trojan
```

11.2 File names

```
[Drive]:\$Recycle.Bin\Com1:NULL.DAT
wcnapi.mui
```

11.3 C&C servers

```
154.223.131[.]237
117.16.142[.]9
103.19.3[.]109
110.45.146[.]253
117.16.142[.]69
122.10.117[.]206
207.148.125[.]56
118.193.236[.]206
167.88.176[.]205
103.224.83[.]95
103.19.3[.]21
xp101.dyn-dns[.]com
svn-dns.ahnlabinc[.]com
dns1-1.7release[.]com
ssl.dyn-dns[.]com
```

11.4 PortReuse HTTP response

```
Server: Microsoft-IIS/ 10.0 Microsoft-HTTPAPI/2.0
```

11.5 PortReuse backdoor

.NET injector

```
395E87C5BD00F78BF4C63880C6982A7941A2ECD0
```

VBS injector

```
08B825C87171500E694798527E17A849160B0A72
```

InnerLoader

```
97709D62531D12A6994BCE5787D519DB52435A62
252640016FAEFF97FA22EB2B736973ED16D73FBE
F5BA05240B1609D4131D5DCA7F5E6E90B5748004
```

NetAgent

```
E14A6A8447CE1D45494E613D6327430D9025A2E5
74A68DAD4BC87EACCA93106832F8B4AEE82843A2
5AB3461B17EE3806ABBB06B8966F6B0011F3D8F2
```

SK3

```
A1AED6FD6990A74590864F9D2A6E714A715FCE3E
E0F276ED16027ED2953A7B0E5274D3F563A75A9D
14C32D0C0346EF4A2B1993FDA9AAB670806B9284
```

Merged NetAgent & ProcTran

```
52A8C38890360D0B32993A44C9E94E660F3FA8F4
20CA6EAE9D6CF2275F9BFD24A0E07F75BEE119BA
DBE3EECE00C255A3FDF924B82621394377B0E865
```

UserFunction

```
A08922372042B4C3C0FAA120E9DD626823CDB3C7
93F623C91F579D33788F84A9A83478CD2E9646AA
```

ProcTran

```
44DDBF7AA256A4B0E25DE585E95EA520BF2C4891
75B7A4B7E01CECC9AFBDAB01C49E9D7FCCACFDC0
```

11.6 VMProtected samples

11.7 ShadowPad backdoor

Payload in Overlay

```
82072CB53416C89BFEE95B239F9A90677A0848DF
634344FAFD6E16F171B0857962149659639FDF41
ED0C9354D34D6E9F09B7038D391E846CDD9E0EAE
E6D43344A354EB17E0E0E76AD391FBCAF9C34119
22B82AE0819DA2FD887BE55A8508FFB46D02CA99
F14694BDDE921B31030300CC9BDC5574BA3D9F74
971BB08196BBA400B07CF213345F55CE0A6EEDC8
438178A5816D3EF6AC02D4DB929A48FA558E514C
4DC5FADECE500CCD8CC49CFCF8A1B59BAEE3382A
C44D06F79E5E42B08BE17A8A7DBAF61400F1DE28
672BB391B92681ADCFCFB4F2F728EDF32F2FB8FE
```

11.8 PortReuse backdoor

Payload in ADS

```
9E8883A6DE72D338E2C0C1A0E291D013A0CE9058
B09ADDDE1523C223C4F8FBF0E541C627E4A04400
```

Payload in Overlay

```
BD1F1494B8D18DAF07DE7D47549A7E27FF3FFD05
757FF5EC3DC53ABBB62391B14883EF460F6FD404
BDBADB2E3EEDD72DD6F8D9235699A139CAB69AAE
4D090E6B749D4D3D8E413F44EB2DE6925C78CD82
B4446480813D3BFC8DE4049A32A72CC0EB0D8094
```

11.9 Winnti droppers (Install.exe)

```
95A41FDDDC8CAF097902B484F8440BDDAD0C5B32
D9A54F79CA15C7E363DBE62B4D1C5C8D103103A2
DAF1CD345F44CB2BF1CFA8D68EECAF1961CBD51F
3DF753F56BB53F72D3DF735A898D7221C3B5272E
6C10C9D46531FBC5F0C2372A116AB31C730ED4B7
D74F1C8257409AD964DB22087A559609C2D0D978
E6677E5E2D68BC544B210E69D9C8DF6A2752C20A
EC0E4A6E2E630267C13B449ED4CF3F04598E40DF
F61403E7730D17B967DA3143BC7CB33EEBE826C0
FD9DED44C47585541B89FFD25907A9A2ED41A995
E0B1005DA5B35E31F09FC82A694F188A92CCA85D
CD36CAF7F7CD9F161743348D2EA69A9E0254C3B5
2C35E28FBA5D05F10430C4D70E4938426F38E228
1AE6FBAD7AF15FB7E60DBBFEA964F0E49372AE53
1EC1B5A902869ED5D51012826A34FFA9225853CB
```

11.10 Winnti

```
B08D72576B93687DFC61ABFA740DD39490D6A262
DE197A5DC5B38E4B72BC37C14CF38E577DDEB8B5
4EA2ED895111A70B9A59DF37343440E4A3A97A47
DE197A5DC5B38E4B72BC37C14CF38E577DDEB8B5
C452BDF6FF99243A12789FF4B99AC71A5DA5F696
B08D72576B93687DFC61ABFA740DD39490D6A262
24AA07A0B3665BF97A1545B0F2749CD509F1B4CA
E26B59789029D23BD9232FA6B1C90EC9379B9066
C262D297EAEC622E3FB8E1FC2A0017E28168879A
645720EC88C993B28D982C0AD89A5ACA79CE7E16
B6819C870DF88A973EB48B572AD1CFEAEB6A655A
8DF84B01B08EE983C66BECC59C0F361D246A96ED
723B27ABA08CBB3A9CA42F7E8350451D00829E5A
55155C3A7B993584A07ACDBF92F2200804C00E02
5105F3020B5E680FA66D664C7F8C811F072933CF
D62A0BD08C5B435D1B8A0505E8018D58A9667B2C
C262D297EAEC622E3FB8E1FC2A0017E28168879A
7B0AAE2AA17BD5712DD682F35C7A8E3E1CDCC57C
```

11.11 AceHash

```
47A262BAE22BB77850A1E3E38F8E529189D291F6
35C026F8C35BFCEECD23EACE19F09D3DF2FD72DA
43FF18CEB3814F1DAE940AD977C59A96BB016E76
D24BBB898A4A301870CAB85F836090B0FC968163
```

11.12 XMRig

```
70B21E3AC69F0220784228375BA6BEF37FE0C488
9BFB1C92489DA812DBE53B2A8E2CC2724CF74B4E
EE5FEB8E9428A04C454966F6E19E202CCB33545F
```

12. MITRE ATT&CK TECHNIQUES

Tactic	ID	Name	Description
Initial Access	T1195	Supply Chain Compromise	The Winnti Group has compromised multiple software packages.
Persistence	T1038	DLL Search Order Hijacking	VMProtected samples persist on the system by using a DLL hijacking technique similar to what was used by the 2nd stage of the CCleaner backdoor.
	T1179	Hooking	PortReuse backdoor performs raw TCP hooking.
Defense Evasion	T1116	Code Signing	The Winnti Group signs its malware using stolen code-signing certificates.
	T1140	Deobfuscate/Decode Files or Information	The victim's volume ID is used to decrypt payloads.
	T1158	Hidden Files and Directories	Some VMProtected samples load their payloads from Alternate Data Streams.
	T1027	Obfuscated Files or Information	Several obfuscation techniques are used by the Winnti Group, such as the use of VMProtect and a custom packer.
	T1055	Process Injection	PortReuse backdoor is injected into a process listening on a given port.
	T1045	Software Packing	The Winnti Group uses a custom packer for the PortReuse backdoor and compromised software.
	T1089	Disabling Security Tools	PortReuse backdoor disables Event Tracing for Windows.
Discovery	T1057	Process Discovery	PortReuse backdoor searches for processes listening on a particular port.
Command And Control	T1043	Commonly Used Port	PortReuse backdoor communicates through commonly used ports.
	T1024	Custom Cryptographic Protocol	Winnti malware and ShadowPad use custom cryptographic protocols such as ZXShell-like encryption.
	T1001	Data Obfuscation	ShadowPad C&C server URL is hidden in public webpages.
	T1104	Multi-Stage Channels	ShadowPad uses public web pages to retrieve its second stage C&C server.
	T1071	Standard Application Layer Protocol	Standard Application Layer Protocols are used by PortReuse and ShadowPad.
	T1032	Standard Cryptographic Protocol	The Winnti Group uses standard cryptographic protocols such as RC4, RC5 and AES.
	T1041	Exfiltration Over Command and Control Channel	PortReuse backdoor exfiltrates data over the Command and Control Channel.
Impact	T1496	Resource Hijacking	The final stage of hijacked video games is a Monero miner (XMRig).
	T1492	Stored Data Manipulation	The Winnti Group introduces malicious code into legitimate software.