securelist.com /apt10-tracking-down-lodeinfo-2022-part-ii/107745/

APT10: Tracking down LODEINFO 2022, part II

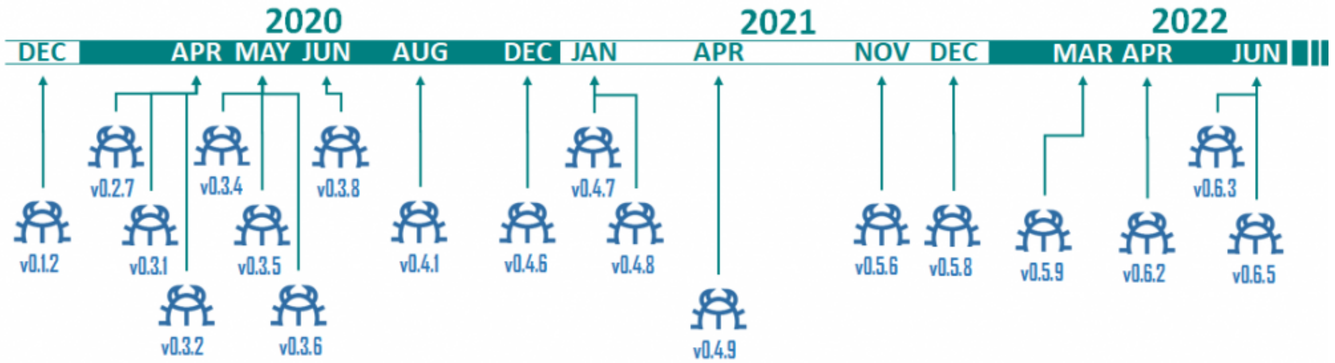
Authors

- Expert Suguru Ishimaru

Evolution of LODEINFO backdoor shellcode

In the previous publication 'Tracking down LODEINFO 2022, part I', we mentioned that the initial infection methods vary in different attack scenarios and that the LODEINFO shellcode was regularly updated for use with each infection vector. In this article, we discuss improvements made to the LODEINFO backdoor shellcode in 2022.

Kaspersky investigated new versions of LODEINFO shellcode, namely v0.5.9, v0.6.2, v0.6.3 and v0.6.5, in March, April and June, respectively. The following chart shows the evolution timeline of this malware since its discovery.

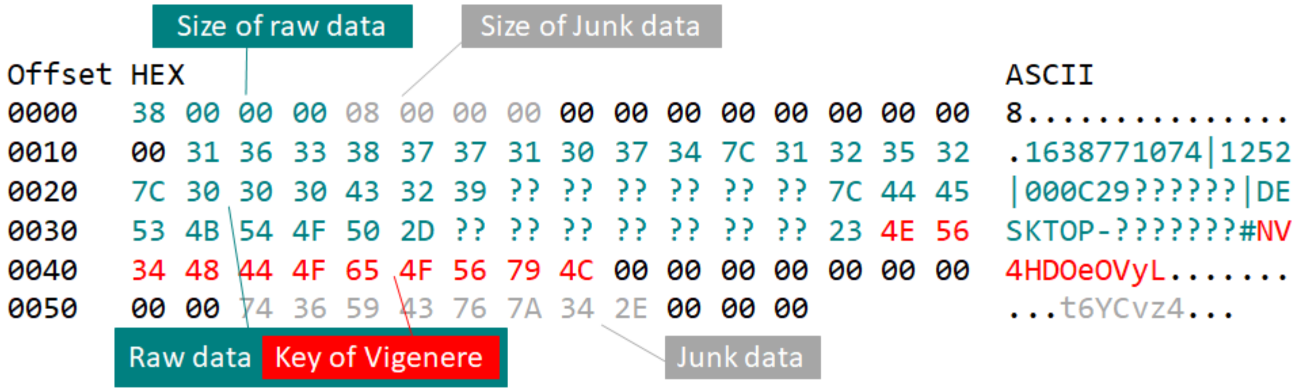


Timeline of LODEINFO releases

LODEINFO v0.5.6: multiple encryption for C2 communication with ancient crypto algorithm

This LODEINFO v0.5.6 shellcode extracted from a loader module demonstrates several enhanced evasion techniques for certain security products, as well as three new backdoor commands implemented by the developer.

After infecting the target machine, the LODEINFO backdoor beacons out machine information to the C2, such as current time, ANSI code page (ACP) identifier, MAC address and hostname. The beacon also contains a hardcoded key (NV4HDOeOVyL) used later by the age-old Vigenere cipher. Furthermore, randomly generated junk data is appended to the end of the data, possibly to evade beaconing detection based on packet size.

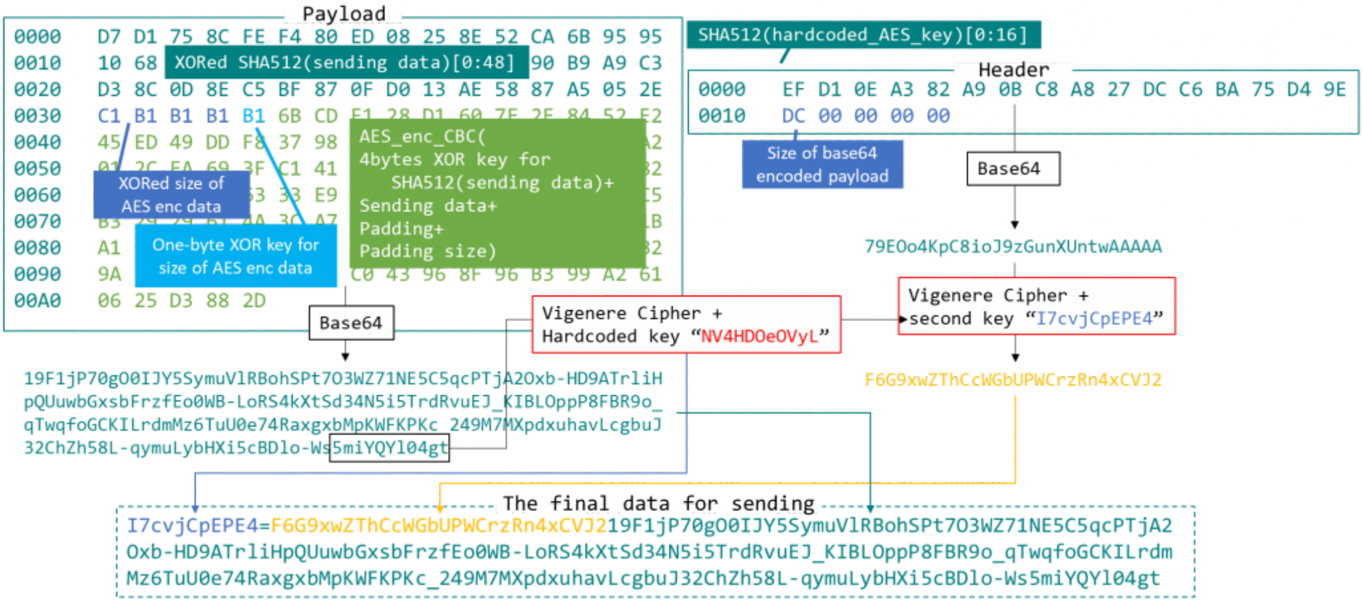


```
Offset HEX                                              ASCII
0000   38 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00  8...............
0010   00 31 36 33 38 37 37 31 30 37 34 7C 31 32 35 32  .1638771074|1252
0020   7C 30 30 30 43 32 39 ?? ?? ?? ?? ?? ?? 7C 44 45  |000C29??????|DE
0030   53 4B 54 4F 50 2D ?? ?? ?? ?? ?? ?? ?? 23 4E 56  SKTOP-???????#NV
0040   34 48 44 4F 65 4F 56 79 4C 00 00 00 00 00 00 00  4HDOeOVyL.......
0050   00 00 74 36 59 43 76 7A 34 2E 00 00 00           ...t6YCvz4...
```

Vigenere cipher key and randomly generated junk data added in LODEINFO v0.5.6

In December 2021, we discovered LODEINFO v0.5.8, with a slight modification that added the LODEINFO implant version number right after the Vigenere cipher key.

The encryption function used to send data was also modified, making it even more complicated. As observed in previous variants, it takes the first 48 bytes of the SHA512 hash value of the data to be sent. Then it XORs the data using a four-byte XOR key that is equal to the elapsed running time, and prepends it before the data. The first 16 bytes to be sent are from another SHA512 hash value, this time taken from the previously mentioned hardcoded AES key (NV4HDOeOVyL). It encrypts 11 bytes at the end of a base64-encoded payload (with replaced padding from “=” to “.”) to dynamically generate the second Vigenere cipher key and the variable of the final generated data. The second key is used by the Vigenere cipher to encrypt the base64 encoded header (url-safe replaced padding from “=” to “.”).


Payload
0000 D7 D1 75 8C FE F4 80 ED 08 25 8E 52 CA 6B 95 95
0010 10 68 XORed SHA512(sending data)[0:48] 90 B9 A9 C3
0020 D3 8C 0D 8E C5 BF 87 0F D0 13 AE 58 87 A5 05 2E
0030 C1 B1 B1 B1 B1 6B CD
0040 45 ED 49 DD F8 37 98
0050
0060
0070
0080 A1
0090 9A C0 43 96 8F 96 B3 99 A2 61
00A0 06 25 D3 88 2D
XORed size of AES enc data
One-byte XOR key for size of AES enc data
AES_enc_CBC(
4bytes XOR key for
SHA512(sending data)+
Sending data+
Padding+
Padding size)
SHA512(hardcoded_AES_key)[0:16]
Header
0000 EF D1 0E A3 82 A9 0B C8 A8 27 DC C6 BA 75 D4 9E
0010 DC 00 00 00 00
Size of base64 encoded payload
Base64
79EOo4KpC8ioJ9zGunXUntwAAAAA
Vigenere Cipher +
second key “I7cvjCpEPE4”
F6G9xwZThCcWGbUPWCrzRn4xCVJ2
Base64
19F1jP70gO0IJY5SymuVlRBohSPt7O3WZ71NE5C5qcPTjA2Oxb-HD9ATrliH
pQUuwbGxsbFrzfEo0WB-LoRS4kXtSd34N5i5TrdRvuEJ_KIBLOppP8FBR9o_
qTwqfoGCKILrdmMz6TuU0e74RaxgxbMpKWFKPKc_249M7MXpdxuhavLcgbuJ
32ChZh58L-qymuLybHXi5cBDlo-Ws5miYQYl04gt
Vigenere Cipher +
Hardcoded key “NV4HDOeOVyL”
The final data for sending
I7cvjCpEPE4=F6G9xwZThCcWGbUPWCrzRn4xCVJ219F1jP70gO0IJY5SymuVlRBohSPt7O3WZ71NE5C5qcPTjA2
Oxb-HD9ATrliHpQUuwbGxsbFrzfEo0WB-LoRS4kXtSd34N5i5TrdRvuEJ_KIBLOppP8FBR9o_qTwqfoGCKILrdm
Mz6TuU0e74RaxgxbMpKWFKPKc_249M7MXpdxuhavLcgbuJ32ChZh58L-qymuLybHXi5cBDlo-Ws5miYQYl04gt

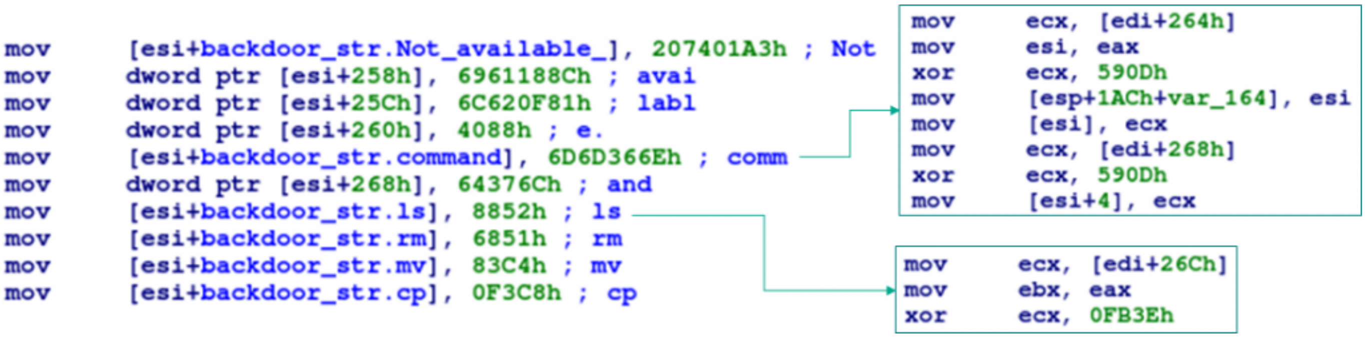
Crypto algorithms and data flow in C2 communications

Finally, the data to be sent to the C2 is produced using the second key, the encrypted header, and the payload through the complex steps described above. The final data packet structure is as follows:

<table>
<tr><th>Offset</th><th>Description</th><th>Crypto algorithm</th></tr>
<tr><td>0x00</td><td>11 bytes from the end of the payload</td><td>Vigenere cipher</td></tr>
<tr><td>0x0C</td><td>A delimiter</td><td>N/A</td></tr>
<tr><td>0x0D</td><td>Message header
<table>
<tr><th>Offset</th><th>Description</th></tr>
<tr><td>0x00</td><td>The first 16 bytes of SHA512 value calculated from the hardcoded AES key.</td></tr>
<tr><td>0x10</td><td>Size of base64 encoded payload</td></tr>
<tr><td>0x15</td><td>A byte of unknown data</td></tr>
</table></td><td>base64 (url-safe and replaced padding from “=” to “.”)
Vigenere cipher</td></tr>
<tr><td>0x29</td><td>Message payload:
<table>
<tr><th>Offset</th><th>Description</th></tr>
<tr><td>0x00</td><td>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.</td></tr>
<tr><td>0x30</td><td>XORed size of encrypted data</td></tr>
<tr><td>0x35</td><td>1 byte XOR key for size of encrypted data (offset 0x30)</td></tr>
<tr><td>0x36</td><td>Encrypted data by AES CBC mode with the hardcoded AES key “88 8C A3 F2 87 36 CC 12 A5 90 18 56 13 B7 C0 A7 E1 07 D4 5C 7D 47 37 AD AB A3 8C C2 12 E3 03 AC” and IV “83 01 36 C9 3A 2D 13 29 23 56 78 A1 F1 0C D1 75”. The data contains elapsed running time, current time, ANSII Code Page, MAC address, host name, etc.</td></tr>
</table></td><td>base64 (url-safe with replaced padding from “=” to “.”)</td></tr>
</table>

LODEINFO v0.5.6: 2-byte XOR obfuscation for backdoor command identifiers

This update included revised crypto algorithms and backdoor command identifiers that were defined as four-byte hardcoded values in previous LODEINFO shellcodes. LODEINFO v0.5.6 backdoor command identifiers are obfuscated with a two-byte XOR operation. Before comparing a command identifier, an XOR operation is applied for each command. The hardcoded XOR key differs for each command as follows:



```
mov     [esi+backdoor_str.Not_available_], 207401A3h ; Not
mov     dword ptr [esi+258h], 6961188Ch ; avai
mov     dword ptr [esi+25Ch], 6C620F81h ; labl
mov     dword ptr [esi+260h], 4088h ; e.
mov     [esi+backdoor_str.command], 6D6D366Eh ; comm
mov     dword ptr [esi+268h], 64376Ch ; and
mov     [esi+backdoor_str.ls], 8852h ; ls
mov     [esi+backdoor_str.rm], 6851h ; rm
mov     [esi+backdoor_str.mv], 83C4h ; mv
mov     [esi+backdoor_str.cp], 0F3C8h ; cp
```

```
mov     ecx, [edi+264h]
mov     esi, eax
xor     ecx, 590Dh
mov     [esp+1ACh+var_164], esi
mov     [esi], ecx
mov     ecx, [edi+268h]
xor     ecx, 590Dh
mov     [esi+4], ecx
```

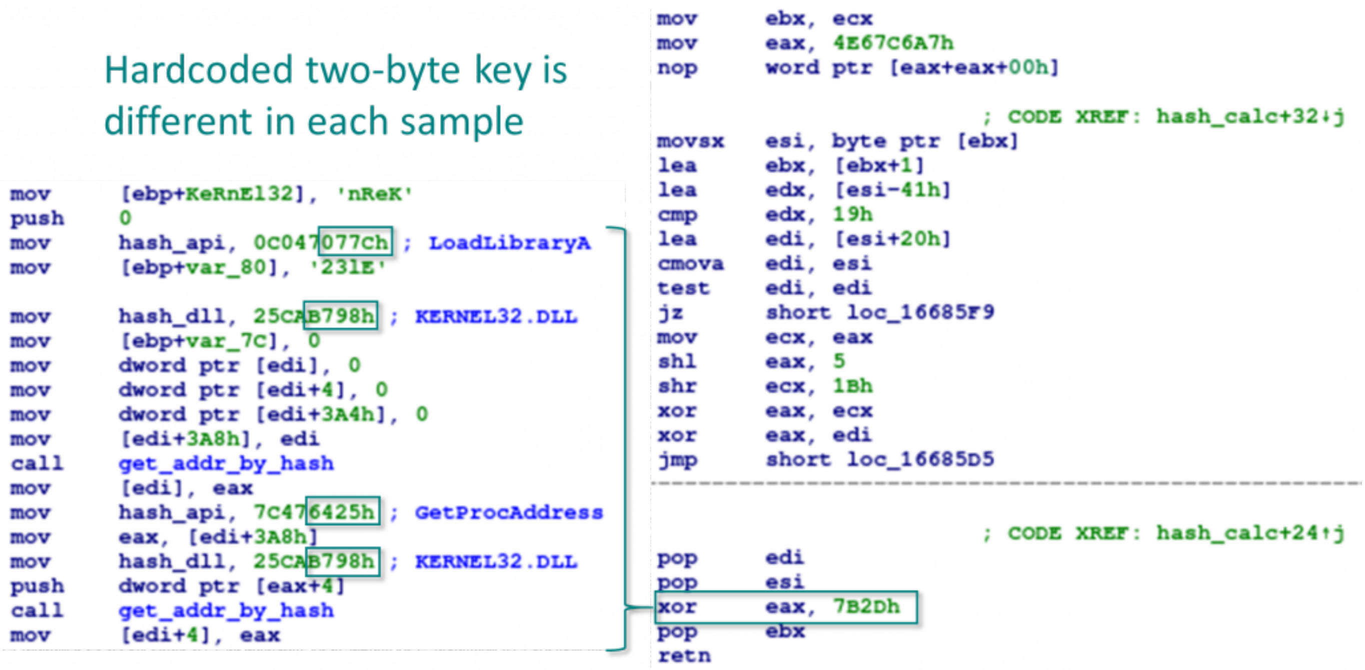
```
mov     ecx, [edi+26Ch]
mov     ebx, eax
xor     ecx, 0FB3Eh
```

Two-byte XOR for four-byte stack strings of backdoor command identifiers

We also observed the actor implementing new backdoor commands such as “comc”, “autorun”, and “config” in LODEINFO v0.5.6 and later versions. Twenty-one backdoor commands, including three new commands, are embedded in the LODEINFO backdoor to control the victim host.

LODEINFO v0.5.9: hashing algorithm to get API functions

Version 0.5.9 has a new hash calculation algorithm compared to v0.5.8. The hashing algorithm is used by the malware to calculate hashes for API function names, to resolve the function addresses. In this case it seems to be a custom algorithm developed by the actor. The logic of the hash calculation has an XOR operation with a two-byte key at the end and the hardcoded XOR key, which is different in each sample.



Hardcoded two-byte key is different in each sample

```
mov     [ebp+KeRnEl32], 'nReK'
push    0
mov     hash_api, 0C047077Ch ; LoadLibraryA
mov     [ebp+var_80], '231E'

mov     hash_dll, 25CAB798h ; KERNEL32.DLL
mov     [ebp+var_7C], 0
mov     dword ptr [edi], 0
mov     dword ptr [edi+4], 0
mov     dword ptr [edi+3A4h], 0
mov     [edi+3A8h], edi
call    get_addr_by_hash
mov     [edi], eax
mov     hash_api, 7C476425h ; GetProcAddress
mov     eax, [edi+3A8h]
mov     hash_dll, 25CAB798h ; KERNEL32.DLL
push    dword ptr [eax+4]
call    get_addr_by_hash
mov     [edi+4], eax
```

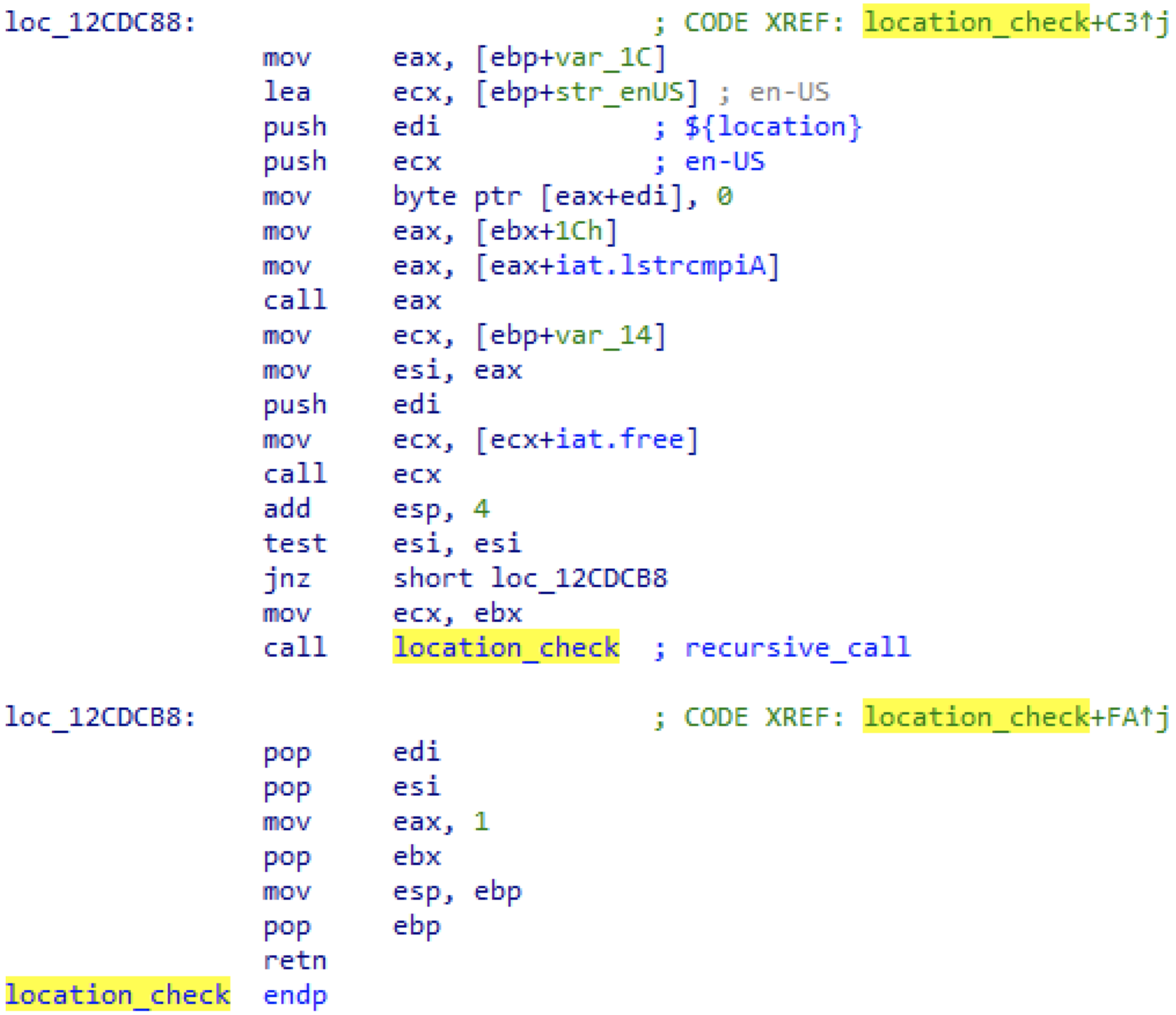
```
mov     ebx, ecx
mov     eax, 4E67C6A7h
nop     word ptr [eax+eax+00h]

                                ; CODE XREF: hash_calc+32↓j
movsx   esi, byte ptr [ebx]
lea     ebx, [ebx+1]
lea     edx, [esi-41h]
cmp     edx, 19h
lea     edi, [esi+20h]
cmova   edi, esi
test    edi, edi
jz      short loc_16685F9
mov     ecx, eax
shl     eax, 5
shr     ecx, 1Bh
xor     eax, ecx
xor     eax, edi
jmp     short loc_16685D5
-------------------------------------------------------

                                ; CODE XREF: hash_calc+24↑j
pop     edi
pop     esi
xor     eax, 7B2Dh
pop     ebx
retn
```

Changed hash calculation algorithm and additional two-byte XOR key in v0.5.9

This modification suggests the attacker's goal was to evade signature-based detections and make the reverse engineering process more difficult for security researchers.

LODEINFO v0.6.2: evasion of en_US environment

In LODEINFO v0.6.2 and later versions, the shellcode has a new feature that looks for the "en_US" locale on the victim's machine in a recursive function and halts execution if that locale is found.



```
loc_12CDC88:                            ; CODE XREF: location_check+C3↑j
                mov     eax, [ebp+var_1C]
                lea     ecx, [ebp+str_enUS] ; en-US
                push    edi             ; ${location}
                push    ecx             ; en-US
                mov     byte ptr [eax+edi], 0
                mov     eax, [ebx+1Ch]
                mov     eax, [eax+iat.lstrcmpiA]
                call    eax
                mov     ecx, [ebp+var_14]
                mov     esi, eax
                push    edi
                mov     ecx, [ecx+iat.free]
                call    ecx
                add     esp, 4
                test    esi, esi
                jnz     short loc_12CDCB8
                mov     ecx, ebx
                call    location_check  ; recursive_call

loc_12CDCB8:                            ; CODE XREF: location_check+FA↑j
                pop     edi
                pop     esi
                mov     eax, 1
                pop     ebx
                mov     esp, ebp
                pop     ebp
                retn
location_check  endp
```

Recursive call if the "en-US" locale is found

According to our own investigations, as well as open-source intelligence collected on this malware, the main targets of these attacks are Japanese entities. The aim of this feature, therefore, is to evade execution in sandboxes and on researcher machines, something that occurs most commonly in an English-language locale.

LODEINFO v0.6.2: generating user agent for C2 communications

The function responsible for generating the user agent for C2 communication has also been updated from v0.6.2. The malware generates the user agent string using the following hardcoded formatted string, where the **%s** is substituted with the version number of the installed chrome.exe application:

“Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/**%s** Safari/537.36″.

The malware gets the version number of the installed chrome.exe from the EXE file present at one of the following file paths:

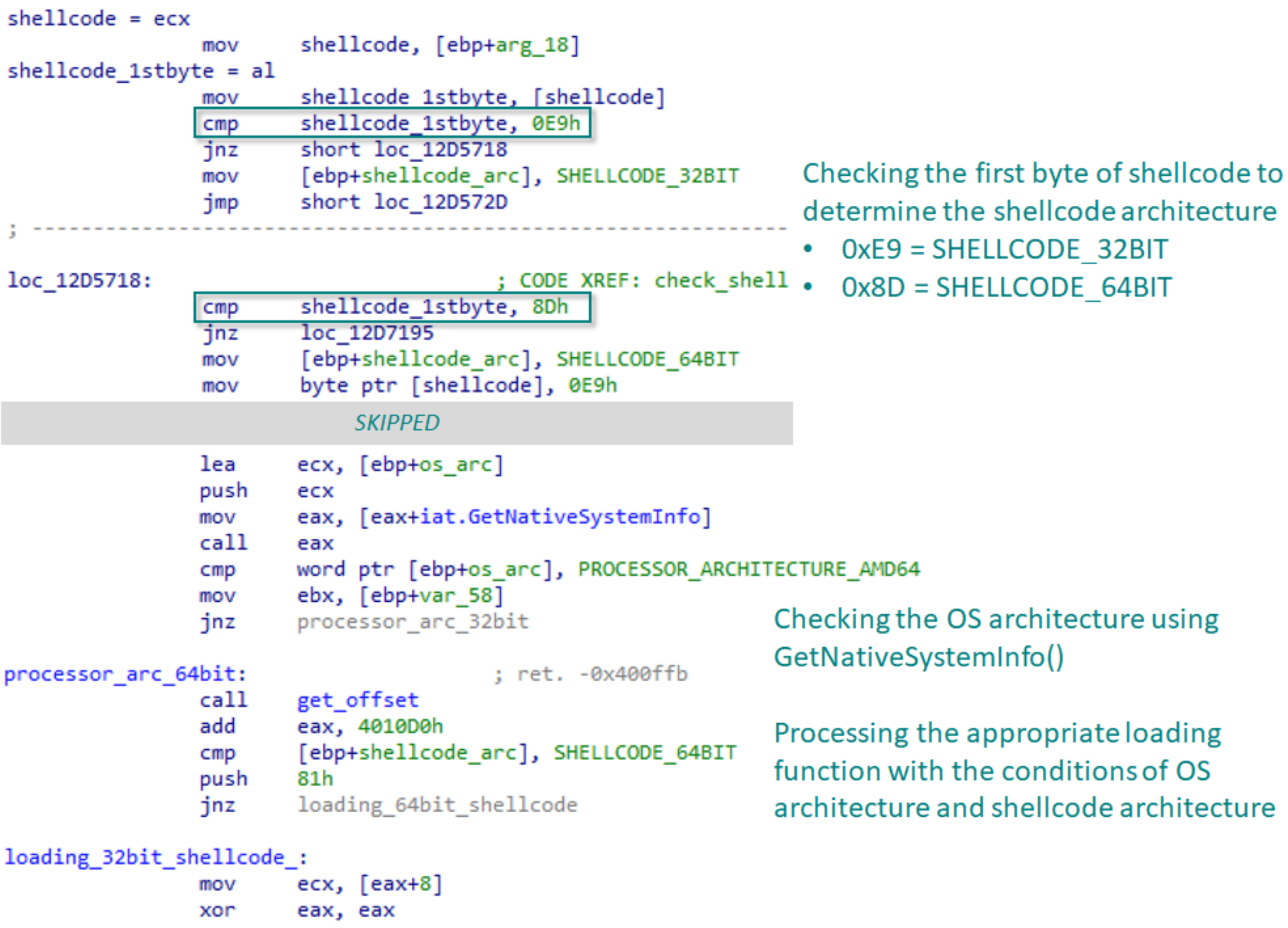
- C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
- C:\Program Files\Google\Chrome\Application\chrome.exe
- C:\Users\Administrator\AppData\Local\Google\Chrome\Application\chrome.exe

Otherwise, if none of these files exists on the system, the malware uses the hardcoded version 98.0.4758.102 to create the following user agent string:

- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36

LODEINFO v0.6.2: supporting the injection of the 64-bit shellcode in ‘memory’ command

Based on our deep analysis of this version, we discovered a very interesting update in the shellcode loading scheme implemented from version v0.6.2, in the function that handles the ‘memory’ command.



```
shellcode = ecx
                mov     shellcode, [ebp+arg_18]
shellcode_1stbyte = al
                mov     shellcode_1stbyte, [shellcode]
                cmp     shellcode_1stbyte, 0E9h
                jnz     short loc_12D5718
                mov     [ebp+shellcode_arc], SHELLCODE_32BIT
                jmp     short loc_12D572D
; ---------------------------------------------------------------

loc_12D5718:                            ; CODE XREF: check_shell
                cmp     shellcode_1stbyte, 8Dh
                jnz     loc_12D7195
                mov     [ebp+shellcode_arc], SHELLCODE_64BIT
                mov     byte ptr [shellcode], 0E9h
                              SKIPPED
                lea     ecx, [ebp+os_arc]
                push    ecx
                mov     eax, [eax+iat.GetNativeSystemInfo]
                call    eax
                cmp     word ptr [ebp+os_arc], PROCESSOR_ARCHITECTURE_AMD64
                mov     ebx, [ebp+var_58]
                jnz     processor_arc_32bit

processor_arc_64bit:                    ; ret. -0x400ffb
                call    get_offset
                add     eax, 4010D0h
                cmp     [ebp+shellcode_arc], SHELLCODE_64BIT
                push    81h
                jnz     loading_64bit_shellcode

loading_32bit_shellcode_:
                mov     ecx, [eax+8]
                xor     eax, eax
```

Checking the first byte of shellcode to determine the shellcode architecture

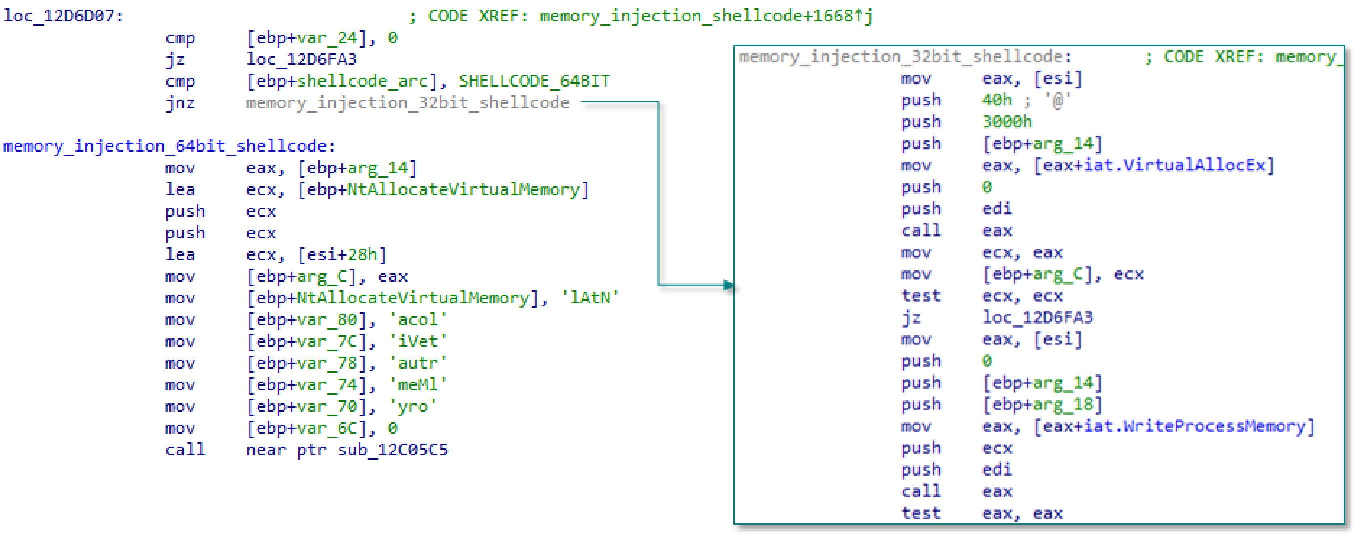
- 0xE9 = SHELLCODE_32BIT
- 0x8D = SHELLCODE_64BIT

Checking the OS architecture using GetNativeSystemInfo()

Processing the appropriate loading function with the conditions of OS architecture and shellcode architecture

Checking the OS architecture and the next shellcode architecture

During the memory injection process, performed using the function responsible for the memory command, the malware checks the first byte of the second stage shellcode to determine the shellcode architecture using a magic hex value. If the first byte is 0xE9, the architecture is 32-bit, and if it is 0x8D, the architecture is 64-bit. After the check is completed, if the first byte was 0x8D, it gets replaced with 0xE9 in order for the shellcode to execute properly. In the function shown below, the malware checks the OS architecture of the infected machine and handles the appropriate loading scheme according to OS architecture and shellcode architecture.



```
loc_12D6D07:                            ; CODE XREF: memory_injection_shellcode+1668↑j
                cmp     [ebp+var_24], 0
                jz      loc_12D6FA3
                cmp     [ebp+shellcode_arc], SHELLCODE_64BIT
                jnz     memory_injection_32bit_shellcode

memory_injection_64bit_shellcode:
                mov     eax, [ebp+arg_14]
                lea     ecx, [ebp+NtAllocateVirtualMemory]
                push    ecx
                push    ecx
                lea     ecx, [esi+28h]
                mov     [ebp+arg_C], eax
                mov     [ebp+NtAllocateVirtualMemory], 'lAtN'
                mov     [ebp+var_80], 'acol'
                mov     [ebp+var_7C], 'iVet'
                mov     [ebp+var_78], 'autr'
                mov     [ebp+var_74], 'meMl'
                mov     [ebp+var_70], 'yro'
                mov     [ebp+var_6C], 0
                call    near ptr sub_12C05C5
```

```
memory_injection_32bit_shellcode:       ; CODE XREF: memory_
                mov     eax, [esi]
                push    40h ; '@'
                push    3000h
                push    [ebp+arg_14]
                mov     eax, [eax+iat.VirtualAllocEx]
                push    0
                push    edi
                call    eax
                mov     ecx, eax
                mov     [ebp+arg_C], ecx
                test    ecx, ecx
                jz      loc_12D6FA3
                mov     eax, [esi]
                push    0
                push    [ebp+arg_14]
                push    [ebp+arg_18]
                mov     eax, [eax+iat.WriteProcessMemory]
                push    ecx
                push    edi
                call    eax
                test    eax, eax
```

Memory injection of the 64-bit shellcode was supported in v0.6.2

In the shellcode injection process, it uses the basic Windows APIs such as VirtualAllocEx(), WriteProcessMemory() and CreateRemoteThread() for memory injection of the 32-bit shellcode and NtAllocateVirtualMemory(), NtWriteVirtualMemory() and RtlCreateUserThread() for supporting the memory injection of the 64-bit shellcode.

LODEINFO v0.6.3: reducing backdoor commands

As for updates implemented in the LODEINFO backdoor commands, the obfuscation method using two-byte XOR encryption for backdoor command identifiers as well as the debug strings remained untouched up to version 0.5.6. However, in version 0.6.3, the actor removed some of the unnecessary backdoor commands to improve the efficiency of the backdoor. The number of backdoor commands was reduced from 21 in v0.6.2 to 11 in v0.6.3. The modifications to the C2 command list are shown in the table below.

Command	Description and updates	Implemented since version	Presence of commands in v0.6.3 – v0.6.5
command	Show embedded backdoor command list.	v0.1.2	Available
send	Download a file from C2.	v0.1.2	Available
recv	Upload a file to C2.	v0.1.2	Available
memory	Inject the shellcode in memory. This command has been updated to support the 64-bit shellcode in v0.6.2 and later versions.	v0.1.2	Available
kill	Kill a process using process ID.	v0.1.2	Available
cd	Change directory.	v0.1.2	Available
ver	Send malware and system information including current OS version, malware version, process ID, EXE file path, system username, current directory, C2 and Mutex name.	v0.1.2	Available
print	Make a screenshot.	v0.3.1	Available
ransom	Encrypt files by a generated AES key, which is also encrypted with RSA using the hardcoded RSA key. (Shows a “Not available.” message in v0.3.5)	v0.3.8	Available
comc	Execute command using WMI.	v0.5.6	Available
config	Just shows a “Not available.” message from v0.5.6 until v0.6.5.	v0.5.6	Available
ls	Get a file list.	v0.1.2	Removed
rm	Delete a file.	v0.3.1	Removed
mv	Move a file.	v0.4.8	Removed
cp	Copy a file.	v0.4.8	Removed
cat	Upload a file to C2.	v0.1.2	Removed
mkdir	Make a directory.	v0.4.8	Removed
keylog	Check for Japanese keyboard layout. Save keystrokes, datetime and active window name. Uses 1-byte XOR encryption and a file %temp%\%hostname%.tmp.	v0.4.1	Removed

	(Shows a message “Not available.” in v0.3.5.)		
ps	Show process list.	v0.4.6	Removed
pkill	Terminate a process.	v0.4.6	Removed
autorun	Set/delete persistence.	v0.5.6	Removed

Conclusions

LODEINFO malware is updated very frequently and continues to actively target Japanese organizations. At the time of writing this report, in September 2022, we detected v0.6.6 and v0.6.7 with new TTPs.

One of the core modifications of the LODEINFO shellcode was support for Intel 64-bit architecture, to expand the targeted victim environments. The updated TTPs and improvements in LODEINFO and related malware, such as the implementation of the Vigenere cipher, complex infection flow with fileless malware, partial XOR encryption, C2 communication packets with a unique data structure and variable length, and password-protected documents, indicate that the attacker is particularly focused on making detection, analysis and investigation harder for security researchers.

For this reason, it becomes more and more difficult to keep track of this actor. That is why we believe it is important to emphasize collaboration within the security research community, to share our results and findings about LODEINFO and related malware attacks.

Indicators of compromise

Malicious document

da20ff8988198063b56680833c298113

LODEINFO zip implant

89bd9cf51f8e01bc3b6ec025ed5775fc

LODEINFO loader with an embedded BLOB

cb2fcd4fd44a7b98af37c6542b198f8d	LODEINFO v0.5.9
a0828f194d3835ea218609dd93d87d16	LODEINFO v0.5.9
16cd587529c230b1a6b47b66d3c84fcf	LODEINFO v0.5.9
de4c87a05becc78ab2e3f568cd46272c	LODEINFO v0.5.9
9066bec5834279ffcb8876f2fdb8752c	LODEINFO v0.5.9
016a974e70bbce6161862e0ac01a0211	LODEINFO v0.6.2
d3cae3b6d948ffd17c5a165bad94f857	LODEINFO v0.6.2
16f0b02bf9676d066d245fe0c717ba52	LODEINFO v0.6.2
ff71fadc33b883de934e632ddb4c6b78	LODEINFO v0.6.2
1a5a74453ebb9747b433342d1ba242cc	LODEINFO v0.6.2
013ef386b1c792faec51fc550fef063a	LODEINFO v0.6.2
da1c9006b493d7e95db4d354c5f0e99f	LODEINFO v0.6.2
a8220a76c2fe3f505a7561c3adba5d4a	LODEINFO v0.6.3

LOADERINFO loader without a BLOB

26892038ab19c44ba55c84b20083cdbd	loads a809231cf901bad9d643494d0eb5a630
c5bdf14982543b71fb419df3b43fbf07	loads c9d724c2c5ae9653045396deaf7e3417
db0bfce29c7c2f076f711cdde2898227	loads ad206315afaa0cd5b42f0fc7b537fefd

Binary of LODEINFO with a one-byte XORed shellcode

a809231cf901bad9d643494d0eb5a630	LODEINFO v0.6.3
0fcf90fe2f5165286814ab858d6d4f2a	LODEINFO v0.6.5
ad206315afaa0cd5b42f0fc7b537fefd	LODEINFO v0.6.5
c9d724c2c5ae9653045396deaf7e3417	LODEINFO v0.6.5
f7de43a56bbb271f045851b77656d6bd	LODEINFO v0.6.5

Implants that contain LODEINFO loader and a one-byte XORed shellcode

15b80c5e86b8fd08440fe1a9ca9706c9
6780d9241ad4d8de6e78d936fbf5a922

SFX file

76cdb7fe189845a0bc243969dba4e7a3
edc27b958c36b3af5ebc3f775ce0bcc7

Hardcoded C2s

103.175.16[.]39
172.104.72[.]4
172.104.112[.]218
172.105.223[.]216
202.182.108[.]127
45.77.28[.]124
5.8.95[.]174
www.dvdsesso[.]com