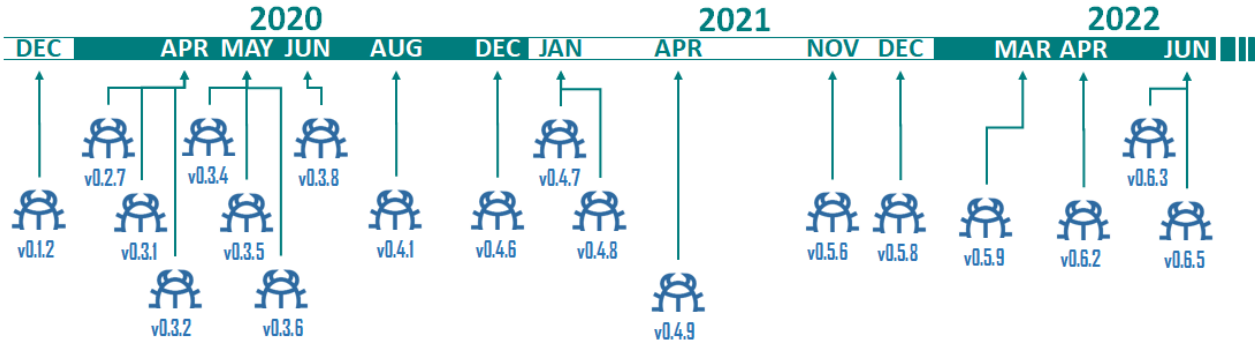
APT10: Tracking down LODEINFO 2022, part II

By Suguru Ishimaru

Published: 2022-10-31 · Archived: 2026-04-05 17:53:29 UTC

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.

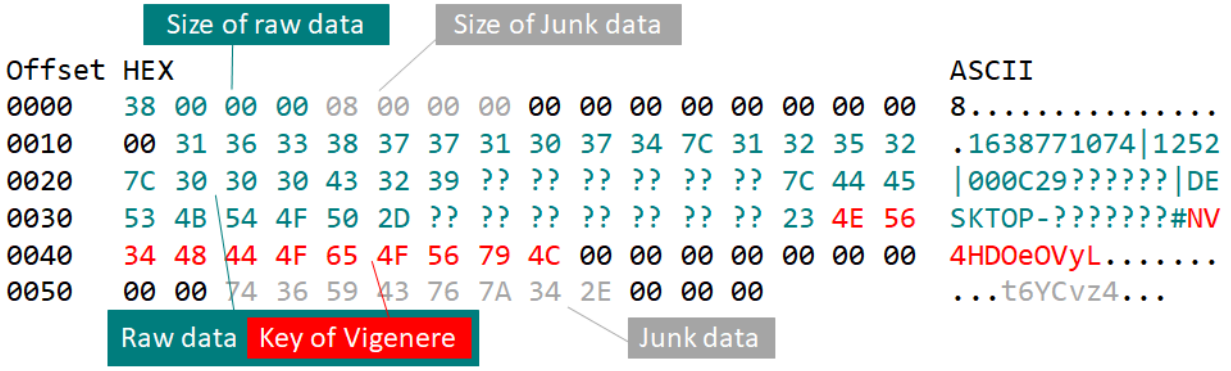


Timeline of LODEINFO releases

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.

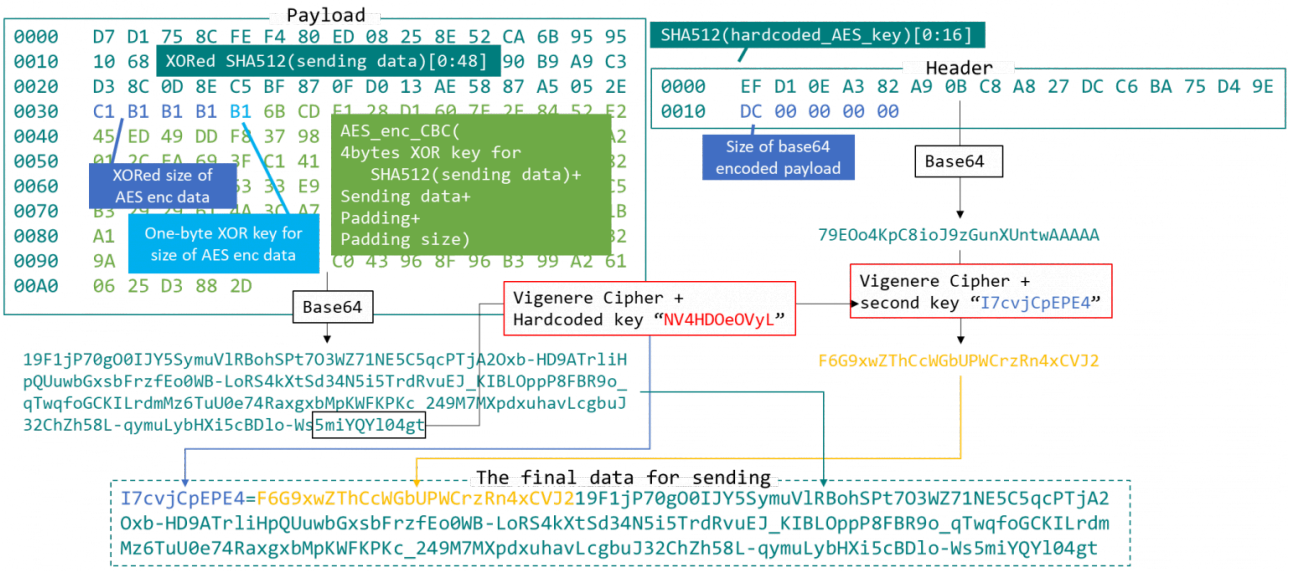
After infecting the target machine, the LODEINFO backdoor beacons out machine information to the C2, such as current time, ANSI code page (ACP) identifier, MAC address and hostname. The beacon also contains a hardcoded key (NV4HDOeOVyL) used later by the age-old Vigenere cipher. Furthermore, randomly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



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 ".").



Crypto algorithms and data flow in C2 communications

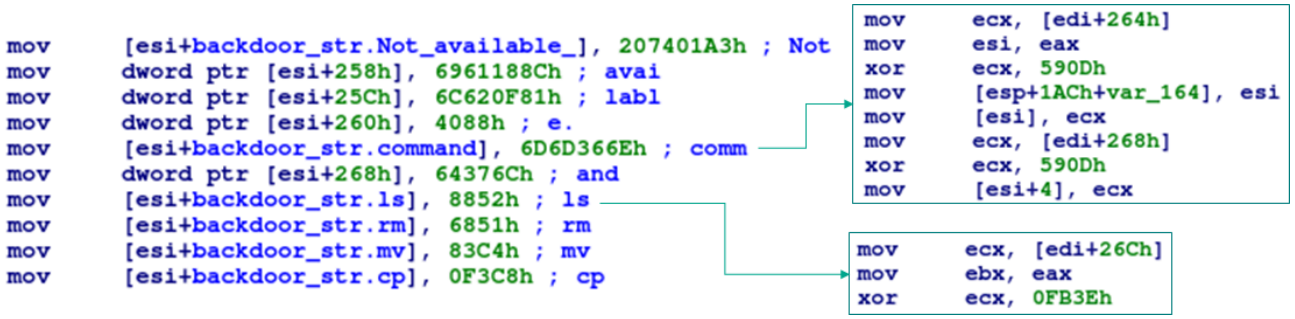
Finally, the data to be sent to the C2 is produced using the second key, the encrypted header, and the payload through the complex steps described above. The final data packet structure is as follows:

<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:

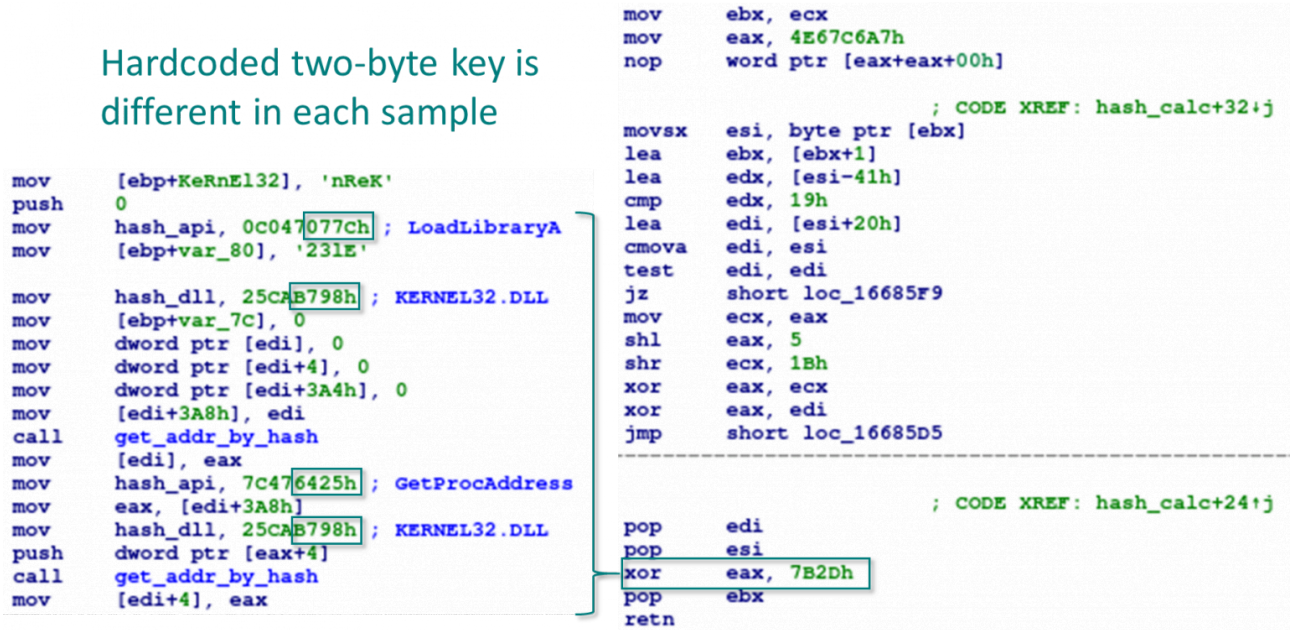


Two-byte XOR for four-byte stack strings of backdoor command identifiers

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.



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. (Shows a message “Not available.” in v0.3.5.)	v0.4.1	Removed
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

LOADERINFO loader without a BLOB

Binary of LODEINFO with a one-byte XORed shellcode

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
