

Bug in Malware “TSCookie” - Fails to Read Configuration - - JPCERT/CC Eyes

By 朝長 秀誠 (Shusei Tomonaga)

Published: 2018-11-11 · Archived: 2026-04-05 18:17:19 UTC

- [BlackTech](#)

In [a previous article](#) we have introduced malware ‘TSCookie’, which is assumedly used by an attacker group BlackTech. We have been observing continuous attack activities using the malware until now. In the investigation of an attack observed around August 2018, we have confirmed that there was an update in the malware. There are two points meriting attention in this update:

- Communication with C&C server
- Decoding configuration information

This article will introduce the details of the update.

Communication with C&C server

In the previous version, TSCookie included encrypted contents in the Cookie header to communicate to a C&C server.

```
GET /Default.aspx HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Date: Thu, 18 Jan 2018 10:20:55 GMT
Pragma: no-cache
Accept: */*
Cookie: 1405D7CD01C6978E54E86DA9525E1395C4DD2F276DD28EABCC3F6201ADAA66F55C15352D29D0FFE51BC9D431EB23
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Win32)
Host:[host name]:443
```

In the new version, Cookie header is no longer used. Instead, encrypted contents are placed within the URL parameter as below:

```
GET /t3328483620.aspx?m=4132641264&i=44D6CF457ADC27B2AFAAEAA&p=EF4D5069C30D6CAC9 HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Win32)
Host: [host name]:443
```

If received an ack from the server to this HTTP GET request, an HTTP POST request will be sent as a next step. The communication feature is the same as the previous TSCookie.

For encryption, RC4 is still used, but the key is generated differently. Here is an example code for decoding HTTP GET request parameter.

```
data = "&" + sys.argv[1] # sys.argv[1] = URL path
conf_key = sys.argv[2].decode("hex") # sys.argv[2] = Configuration key
field = data.split("&")

url_key = field[1]
i=2
encdata = ""
while i<len(field):
    value = field[i].split("=")
    encdata += value[1]
    i+=1

key1 = 0
for i in range(len(url_key)):
    key1 = ord(url_key[i]) + ROR(key1, 13)
    key1 = key1 & 0xFFFFFFFF

key2 = 0
for i in range(len(conf_key)):
    key2 = ord(conf_key[i]) + ROR(key2, 13)
    key2 = key2 & 0xFFFFFFFF

key = pack("I", key1) + pack("I", key2)

decode_data = rc4(encdata.decode('hex'), key)
```

Decoding configuration information

TSCookie possesses its own configuration information and operates accordingly. The details of the configuration remain the same in the new version. The difference is the decoding method of the configuration. Previously, TSCookie had its 4-byte RC4 key in the beginning of the configuration, which was used for decoding. In the new version, the size is expanded to 0x80 bytes (Figure 1).

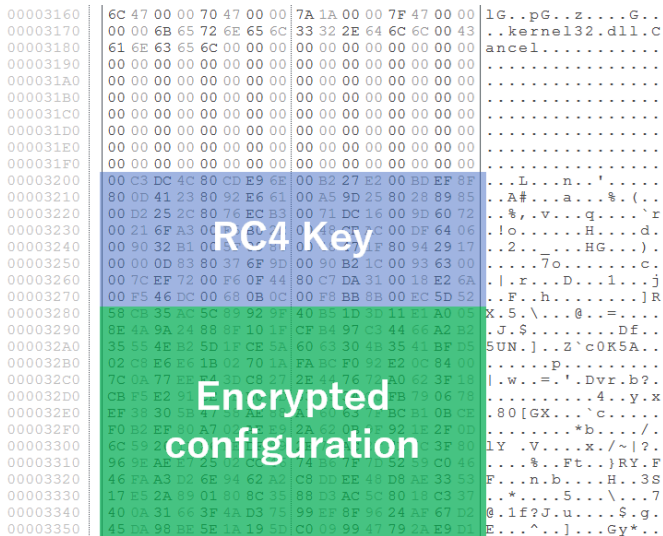


Figure 1: RC4 key and encrypted configuration

We have confirmed that this update made TSCookie fail to read part of the configuration. Figure 2 shows the code copying encrypted configuration (0x8D0 bytes) and RC4 key (0x80 bytes).

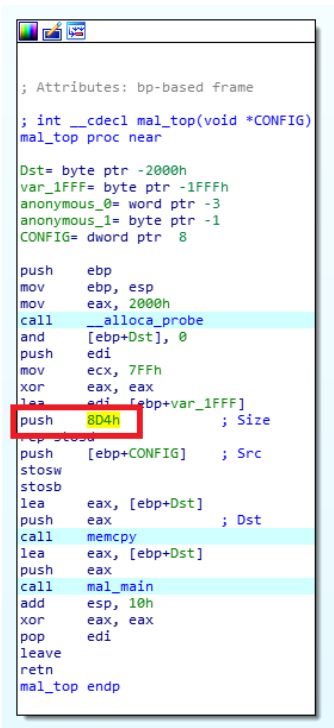


Figure 2: Code copying RC4 Key and encrypted configuration

The code copies data sized 0x8D4 (0x8D0 + 4 bytes), which ignores the updated RC4 key size. To copy the updated RC4 key and configuration correctly, it needs to be set to 0x950 (0x8D0 + 0x80 bytes). With this fault, configuration cannot be decoded properly. Figure 3 describes how TSCookie configuration is decoded.

```

0000790 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 0000790 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00007A0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 00007A0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00007B0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 00007B0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00007C0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 00007C0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00007D0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 00007D0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00007E0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 00007E0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00007F0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 00007F0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000800 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 0000800 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000810 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 0000810 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000820 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 0000820 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000830 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 0000830 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000840 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | 0000840 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000850 | 00 00 00 00 72 F2 C3 F3 | 30 2C B6 CC 4E 5A 01 6A | 0000850 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000860 | 86 A2 52 11 03 BB 06 8F | 8D 66 6F C9 6A 19 4A BC | 0000860 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000870 | 66 94 2B E9 54 62 89 C2 | 4A AD D0 7D AD F9 74 77 | 0000870 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
0000880 | 9C F9 84 47 5A 93 1E 88 | 03 52 6D C7 2A 8A 63 77 | 0000880 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
000089C | 04 D4 38 9B 10 0A 4A AD | 21 46 62 D7 6F 03 31 EB | 000089C | 00 00 00 00 00 00 00 00 | 00 00 00 00 63 00 00 00 |
00008A0 | 63 44 1E AE 51 4F 3F 87 | 36 EE 15 82 0F 0C 9E 09 | 00008A0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00008B0 | 47 2F 22 05 CA 32 83 D3 | E7 8E 2E 4A 2B 00 8C 7A | 00008B0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00008C0 | 73 41 F3 4A E3 A8 C7 05 | 7B 3D C5 AE 28 39 9A E6 | 00008C0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 03 00 00 00 |
00008D0 |

```

Figure 3: Decoded TSCookie configuration
 (Left: Copy size 0x8D4, Right: Copy size 0x950)

Decoded results differ in the left figure (with the wrong, smaller copy size) and right figure (with correct, expanded copy size). Data at 0x89C byte (4 bytes) specifies the waiting time (seconds) before reconnecting to a C&C server. The attackers initially set this to 99 (0x63) seconds (as in the right figure), however, it will not be reconnected for few days since it is not read properly (left figure).

In closing

It is often the case that attackers give an update to their malware based on analysis reports provided from security vendors. We assume that this bug will be fixed sooner or later. We will update when we confirm new malware features.

The malware sample’s hash value is available in Appendix A, and we also list some C&C servers in Appendix B. We hope this is helpful in identifying signs of infection.

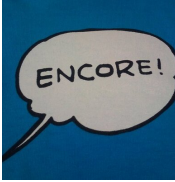
(Translated by Yukako Uchida)

Appendix A SHA-256 Hash Value of a sample

- a5c75f4d882336c670f48f15bf3b3cc3dfe73dba7df36510db0a7c1826d29161

Appendix B C&C server

- mediaplayer.dnset.com
- mediaplayers.ssl443.org
- fashion.androiddatacenter.com
- sakurings.flnet.org



[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.

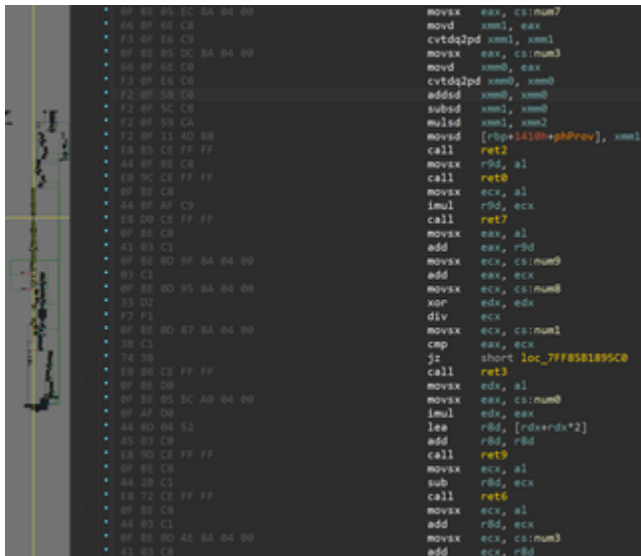
Related articles

```
*key = 0x427c7480;  
*key[4] = 0x015913c2;  
*key[8] = 0x6d472834;  
*key[12] = 0x00007909;  
*key[16] = 0x1379421;  
*key[20] = 0x40003468;  
*key[24] = 0x00798129;  
*key[28] = 0x00000007;  
v4 = m_ret_argOffset0x350(a1 + 3);  
if ( !((v2 = <CryptAcquireContext>)(a1, 0, "Microsoft Enhanced RSA and AES Cryptographic Provider", 0x1, 0xf000000) )  
return 0;  
v5 = m_ret_argOffset0x350(a1 + 3);  
handlehashobj = a1 + 3;  
if ( !((v3 = <CryptCreateHash>)(*a1, 0x0004, 0, 0, a1 + 3) )  
{  
LABEL_0:  
if ( !*a1 )  
return 0;  
v6 = m_ret_argOffset0x350(a1 + 3);  
(v4 = <CryptReleaseContext>)(*a1, 0);  
return 0;  
}  
if ( !<CryptHashData>(*handlehashobj, key, 16, 0) )  
{ (v4 = m_ret_argOffset0x350(a1 + 3));  
v8 = a1 + 3;  
!(v8 = <CryptDeriveKey>(*a1, 0x0004, *handlehashobj, 0x000000, a1 + 2)) // CALS_AES_128  
{  
if ( !*handlehashobj )  
{  
v9 = m_ret_argOffset0x350(a1 + 3);  
(v5 = <CryptDestroyHash>)(*handlehashobj);  
goto LABEL_0;  
}  
v10 = m_ret_argOffset0x350(a1 + 3);  
(v11 = <CryptSetKeyParam>(*v9, 3, 0x0001, 0); // SP_PAD0100 = PKCS017  
v11 = <CryptSetKeyParam>(*v9, 1, 1v, 0); // DV = parameter  
v12 = m_ret_argOffset0x350(a1 + 3);  
(v12 = <CryptSetKeyParam>(*v9, 4, 0x0002, 0); // SP_PAD00 = CBC  
return *v9;  
}
```

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

```
python parse_cross2beacon_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 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 MA0GCSqGSIB3DQEB  
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 GT9UcaLhAkpM0QAG  
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 +IzYpXnmU7pMs1Sd  
0000B0 71 2b 63 52 78 4d 6f 54 4c 6d 68 4e 6f 71 32 55 q+cRxMoLmhNoq2U  
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 fcq6BwMdsMx6gH40  
0000F0 73 6c 42 2f 35 77 6e 63 33 77 51 78 55 62 4f 61 sIB/Swnc3wXub0a  
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 2d 45 4e 44 20 50 55 AQAB.-----END.PU  
000120 42 4c 49 43 20 4b 45 59 2d 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-----  
MIGFMA0GCsqGSIB3DQEBQUAA4GNADCB1QKBgQCNS381HP2V3JD4GT9UcaLhAkpM0QAGRn6Nw6  
RhnVST/1HJ+zHLH82q7XKmo+RU+IzYpXnmU7pMs1Sdq+cRxMoLmhNoq2UTWk9o9RodcZtZxsk  
bM7Tzk7UZjyapTIJfcq6BwMdsMx6gH40s1B/Swnc3wXub0aqEokKorZwmHU3wIDAQAB  
-----END PUBLIC KEY-----
```

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

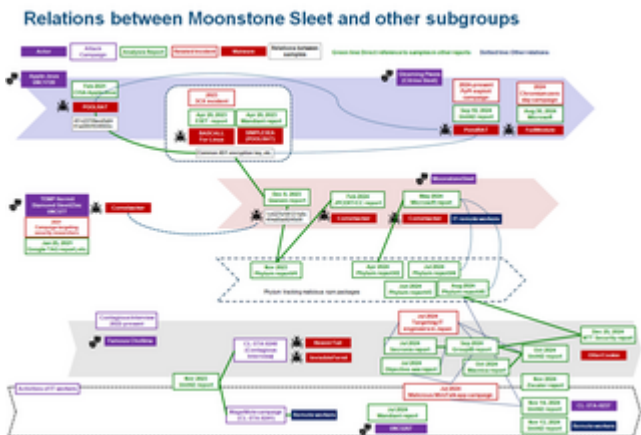


[Malware Identified in Attacks Exploiting Ivanti Connect Secure Vulnerabilities](#)

```
__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.jpccert.or.jp/en/2018/11/tscookie2.html>