

## Decrypting Strings in Emdivi

Hello, this is You 'Tsuru' Nakatsuru at Analysis Center.

As introduced in the previous blog post, my colleagues presented on the attacks arising in Japan at [CODE BLUE 2015](#), entitled "Revealing the Attack Operations Targeting Japan".

In this entry, I will introduce the details of an IDAPython script "emdivi\_string\_decryptor.py", which JPCERT/CC developed to analyse Emdivi, a remote control malware. The script was also introduced in our presentation at CODE BLUE 2015. Please utilize the script along with the codes, etc., that are already published on JPCERT/CC's GitHub account.

JPCERTCC/aa-tools · GitHub

<https://github.com/JPCERTCC/aa-tools>

### Encrypted Strings within Emdivi

Emdivi encrypts strings such as URLs that they connect to and stores them in itself. Depending on the sample, encrypted strings are Base64-encoded and stored as in Figure 1, or in other cases, just saved in an encrypted binary format.

Address	Length	Type	String
.idata:00449F8C	0000000A	C	REG_QWORD
.idata:00449F98	0000000F	C	pad_allocation
.idata:00449FA8	00000059	C	pM3ZyT1X5kYNNbLBagOBcTmqHdcDvZfNL0ol/sW7raRcyhZuw5Mhlm/3MAP9v4CPQ9XPueRnoRnFTegjwv uP87xg5TtrbqfFeQ9EHlSUMCv+dxARUkQwvLkHQIMYVo8QyFdlmH5Bbhp8mMGEFz2TIES0MS1kMCDCAg==
.idata:0044A068	00000041	C	H8RGdqlXELDHQkrdYvdG3/NFZHQ/FESTpFax0Nw7UKITSgRWj1od6bThQAZa/
.idata:0044A080	00000041	C	jANdDayM+REnzYQZpNqEIJb3dHLt5tGB4C1KwMMWtRxaCG+pkLm8wK7VWVGfdWE
.idata:0044A0F8	00000041	C	kdRQeljaHq9vORIMKULVRT/dc4s5eTG2bed3KlydNVCNteieM7R3eev83rjhbT
.idata:0044A140	00000041	C	kwAOmjTIL0vzzuZnq8IZ3/Zb2n7QOjugFVygNgmdkKsqScDnbZdkHuleGMKNU
.idata:0044A188	00000041	C	79VMZU0KUPzuzZnq8IYc+/bkDP/20BTFUFX4BAbhZz1UX7/Wv64290q
.idata:0044A1D0	00000025	C	Software\Microsoft\Internet Explorer
.idata:0044A1F8	00000008	C	Version
.idata:0044A210	00000005	C	.exe

Figure 1: Encrypted strings encoded with Base64

In the incident analysis phase, these encrypted strings need to be decrypted in order to identify information such as URLs that the malware connects to, etc. For this purpose, JPCERT/CC developed emdivi\_string\_decryptor.py.

### Analysis on Emdivi's Decryption Process

Emdivi runs the following process in order to decrypt the strings in itself.

Upon its startup, it calculates the key for decryption, based on the following string information:

- The sample's version strings  
e.g. t17.08.26..3340.4444
- Random long strings in the sample  
e.g. jp5cQEhSR7xMEdv1JOjh5eKGsMxSCAE5M57CijC8VgN1KMbBvP9 (Omitted hereafter)

It then combines these strings with Base64 encode, MD5 hash value calculation and others to calculate the decryption key as in Figure 2. Depending on Emdivi's version, it sometimes combines addition and other arithmetic process for this calculation.

```

push    8
pop     ecx
mov     edi, offset encryption_key ; dd0cffb45ea86a40f43f98025f4a0d6c
push    ebx ; maxcount
rep movsd
push    1 ; char
lea     ecx, [ebp+var_2C]
call    sub_401C6A
call    __EH_epilog3_GS
retn
aa_make_encryption_key endp

```

Figure 2: Calculated decryption key

Using the decryption key calculated here, Emdivi performs decryption just before it uses the strings. Processes related to the encryption are implemented as classes. Figure 3 shows information of those classes.

```

dd offset const CryptoLibs::Base::RTTI Complete Object Locator'
const Base::vftable' dd offset __purecall ; DATA XREF: XxTea::XxTea(void)+193fo
; sub_43D62D:loc_42738Cfo
dd offset __purecall
dd offset __purecall
dd offset const CryptoLibs::XxTea::RTTI Complete Object Locator'
const XxTea::vftable' dd offset aa_XxTea_convert ; DATA XREF: XxTea::XxTea(void)+107fo
dd offset aa_XxTea_decrypt
dd offset aa_XxTea_encrypt

```

Figure 3: Encryption related classes defined within Emdivi

Many samples of Emdivi use XxTEA as in Figure 3, but we have confirmed that some versions use AES. Also, there are some versions that

switch the encryption and decryption process, and we have seen that some use XxTEA encryption process for decryption.

After analysing various samples of Emdivi, we were able to summarise the method for the decryption key calculation and the decryption process as in Chart 1 below. For details of the decryption key generation process, please refer to the source codes of `emdivi_string_decryptor.py`.

Table 1: Decryption process for each Emdivi version

	t17	t19 and t20 mid versions	t20 early and late versions
Decryption process	XxTEA Decryption	XxTEA Encryption	AES Encryption
Method to calculate decryption key	MD5( MD5(base64(ver)) + MD5(key_string))	scanf( "%x%x%x%x", Inc_Add(ver17_key) )	Inc_Add(ver17_key)[:24]

### Before Using `emdivi_string_decryptor.py`

Since `emdivi_string_decryptor.py` is an IDAPython script, it requires disassembler IDA for execution. Also, the version string used when calculating the decryption key is required for string decryption.

Before actually using the tool, you have to obtain the version string from the memory or communication data.

If you are obtaining the string from the memory, execute Emdivi in an analysis environment and then search for the string in the memory by using debugger, etc. You do not have to worry about virtual environment detection, since the version string is generated before the detection process.

If you are obtaining the version string from the communication data, you would need to decode that data. For decoding, you can use `emdivi_postdata_decoder.py` which is also made public together with `emdivi_string_decryptor.py`. Here below is what you will see when executing by giving the data you want to decode to `emdivi_postdata_decoder.py`'s argument.

```
> python emdivi_postdata_decoder.py
"r13ftV=C%5DZ%03k%07%06%7Edkgd%05%19%7Dq%05%05%1E%0D%02%0C;yhmsuRvo=%00;date=-
b%27f.4%60%25%23%3A%24%2C%3A%26%22%3A%3A%27%20%24%3A%20%20%20%1Dh%1DZ%40.4%22%3A%25%3A%23%22%24%25%1D0%7Eu9%5EDI%1Dh%1DYQY.4%26%24%20%2CY%1Dh%1DSY%40%3G-
%3D"
[*] 3 field(s) found in postdata
"r13ftV" -> "win7_32JP_SP1-IE11*968"
"yhmsuRvo" -> "1"
"date" -> "9v3r: t17. 08. 26. . 3340. 4444 | NT: 6. 1. 7601 [ja-JP]
| MEM: 2048M | GMT (9)"
```

Please note that the version string included in the communication data may be different from the string required for the decryption. Therefore, we recommend obtaining the string from the memory.

### Executing `emdivi_string_decryptor.py`

If you have obtained the version string, you are all set. Load the target Emdivi into IDA, and execute `emdivi_string_decryptor.py`. Then, it will process as follows:

1. Input version string
2. Calculate decryption key
3. Search for encrypted string
4. Decrypt string
5. Output results and insert comments in the corresponding section

Upon execution of `emdivi_string_decryptor.py`, the following dialog box appears to input the version string.

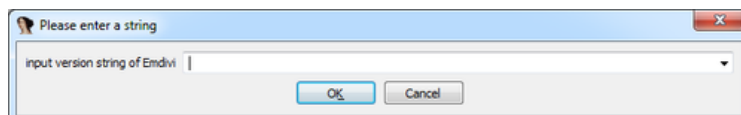
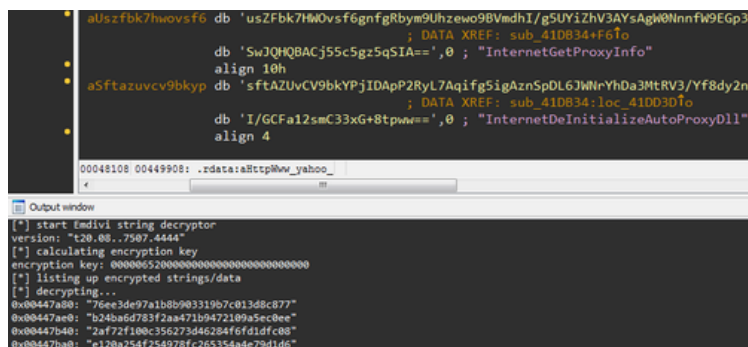


Figure 4: Dialog box to input version string

After you input the version string, the tool will display the decrypted string in the console, and it will be inserted as a comment to where the encrypted string is stored. This is shown in Figure 5.



```

aUszfbk7hwovsf6 db 'usZFbk7HwOvsf6gnfGRbym9Uhzewo9BvmdhI/g5UYiZhV3AYsAgW0Nnnfw9EGp3'
; DATA XREF: sub_410B34+F6To
db 'SwJQHQBACj55c5gz5qSIA==',0 ; "InternetGetProxyInfo"
align 10h
aSftazuv9bkyp db 'sftAZUvCV9bkYPjIDAp2RyL7Aqifg5igAznSpDL6JWlrYhDa3MtrV3/Yf8dy2n'
; DATA XREF: sub_410B34:loc_410D30To
db 'I/GCFa12smC33xG+8tpwv==',0 ; "InternetDeInitializeAutoProxyDll"
align 4
00048108 00449908: .xdata:aBtctplWw_yahoo_

```

```

[*] start Emdivi string decryptor
version: "1.0.0..7507.4444"
[*] calculating encryption key
encryption key: 00000052000000000000000000000000
[*] listing up encrypted strings/data
[*] decrypting...
0x00447a80: "75e3de97a1b0b00331067c013d8c877"
0x00447ae0: "b240a6d783f2a471b9472189a5ec0ee"
0x00447b40: "2af72f100c356273d46284f6d1dfc08"
0x00447ba0: "c170a75df754978fc765354dc79d1d6"

```

Figure 5: Screenshot after executing emdivi\_string\_decryptor.py

Now you can obtain various information including URLs that the malware connects to. Based on these and other related pieces of information, JPCERT/CC coordinates and handles incidents caused by attack operations involving Emdivi.

## In Summary

We hope that the scripts that we made public will contribute in dealing with the attacks relating to Emdivi, and in improving malware analysis techniques.

A blog entry by Kaspersky (see Reference) has revealed that a few versions of Emdivi use the infected users' SID. Unfortunately, the current version of emdivi\_string\_decryptor.py is not yet adapted to input SID. Furthermore, it is possible that new versions of Emdivi with other encryption methods may emerge in the future. We welcome any pull requests on GitHub.

Thanks for reading.

-You Nakatsuru

## Reference

New activity of The Blue Termite APT - Securelist

<https://securelist.com/blog/research/71876/new-activity-of-the-blue-termite-apt/>