LeoUncia and OrcaRat

The PWC-named malware OrcaRat is presented as a new piece of malware but looking at the URI used for C&C communication, it could be an updated version of a well-known and kind of old piece of malware: LeoUncia.

Status

Let's face it:

> px~NFEHrGXF9QA=2/5mGabiSKSCIqbiJwAKjf+Z81pOurL1xeCaw=1/xXiPyUqR/hBL9DW2nbQQEDwNXIYD3l5EkpfyrdVpVC8kp/4WeCaArZAnd+QEYVSY9QMw=2

URI taken from an OrcaRat sample.

It looks a lot like:

> qFUtb6Sw/TytLfLsy/HnqI8QCX/ZRfFP9KL/_2yA9GIK/iufEXR2r/e6ZFBfoN/fcgL04f7/ZBzUuV5T/Balrp2Wm

URI taken from a LeoUncia sample.

What about it? Could it be the same kind of things, huh? Let's dig a little deeper inside the code to check if it is just some sort of coincidence or if it is indeed the same code that is behind these two pieces of malware.

PWC explain it pretty well: the URI is made of some sort of Base64-encoded strings with the middle one being the seed to be associated to the master key to decrypt the whole thing. Actually:

```
URI = E1/E2/E3/E4/E5
```

and to obtain Di (the original data that gives us Ei once encrypted), we must perform the following operation:

```
Di = rc4(md5(custom_debase64(E3)+master_key)).decrypt(Ei)
```

where master_key is “OrcaKiller” for the OrcaRat sample.

What can we find in LeoUncia that is to be found in OrcaRat too?

URI decryption

First, let's have a look at the URI decryption routine.

Dealing with OrcaRat, we have seen the following algorithm:

```
Di = rc4(md5(custom_debase64(E3)+master_key)).decrypt(Ei)
```

When we talk about LeoUncia, we can have a look at the blog posts made by FireEye back in December 2010, especially the second one, where some assembly code has been screenshot from IDA without ever giving the name of the underlying algorithm: yes, it is RC4!

Once decoded from Base64, the binary data we obtain from the URI is comprised of two parts: the first 16 bytes are the decryption key, and the rest of the data is the information to be decrypted. Putting back pieces together, we have the following algorithm for LeoUncia:

```
D = rc4(custom_debase64(E)[:16]).decrypt(custom_debase64(E)[16:])
```

The two samples both share a "custom" Base64 encoding with the use of RC4; nothing fabulous, but it is a start.

Encoding

We dig further with the encoding algorithm: the so-called “custom” Base64.

In both case, the first goal of the customization is to avoid the presence of some **“/”** in any encoded data, because it would break down the process of cutting the URI along with the **“/”** separator. For LeoUncia, the Base64 being used is the Base64-URI that replaces **“+”** and **“/”** by **“.”** and **“_”**, while for OrcaRat, **“+”** are kept and **“/”** are replaced by **“~”**.

Additionally, OrcaRat authors thought it would be great if the URI was a little less obviously Base64-related. So, rather than splitting every eight characters to avoid having **"="** in the URI, they decided that replacing the endings **"="** in **"=1"** and **"=="** in **"=2"** would be a great improvement.

Hibernation feature

Let's have a look at one of the feature of LeoUncia: the hibernate feature.

The feature does the same in OrcaRat: check for some date and time written in a file, and sleep for as long as needed before deleting the aforementioned file. (We would also notice that an useless call to FileTimeToSystemTime has been removed meanwhile.)

The real difference lies in the obfuscation of the filename: LeoUncia was using a plain-text filename (“readx”), whereas OrcaRat is obfuscating (just the same way it obfuscates the Campaign ID) this data: the filename is “wbt.dat” (obfuscated string XORed character-by-character with the XOR key “product”) and it is located in the “App Data” folder of the user OrcaRat is running with.

```
push    offset FileName ; "readx"
call    ds:CreateFileA
```

Code seen in a very old LeoUncia sample: plain-text hibernation filename.

```
mov     al, byte_1000D098
mov     cl, byte_1000D099
mov     dl, byte_1000D09A
xor     al, 'h'
xor     cl, 'x'
mov     byte ptr [esp+938h+var_938+1], al
mov     al, byte_1000D09B
mov     byte ptr [esp+938h+var_938+2], cl
mov     cl, byte_1000D09C
push    ebp
xor     dl, 'i'
xor     al, 'n'
xor     cl, 'g'
push    esi
push    edi
mov     byte ptr [esp+944h+var_938+3], dl
mov     [esp+944h+var_934], al
mov     [esp+944h+var_933], cl
mov     ecx, 41h
xor     eax, eax
lea     edi, [esp+944h+Buffer]
lea     edx, [esp+944h+Buffer]
push    104h            ; uSize
push    edx             ; lpBuffer
rep stosd
mov     [esp+94Ch+var_930], 20h
mov     byte ptr [esp+94Ch+var_938], '\'
mov     [esp+94Ch+var_932], 0
mov     [esp+94Ch+NumberOfBytesWritten], 0
call    ds:GetSystemDirectoryA
```

Code seen in a more recent LeoUncia sample: XORing with “hxing” the hibernation filename.

```
call    ds:SHGetFolderPathA
mov     ebp, ds:lstrlenA
test    eax, eax
jnz     loc_403FC7
mov     al, byte_40E086
mov     cl, byte_40E084
mov     dl, byte_40E085
xor     al, 'o'
xor     cl, 'p'
mov     [esp+518h+var_502], al
mov     eax, dword_40E087
mov     [esp+518h+String2], cl
mov     cl, al
mov     al, ah
xor     al, 'u'
xor     dl, 'r'
mov     [esp+518h+var_500], al
mov     ax, word ptr dword_40E087+2
mov     [esp+518h+var_503], dl
mov     dl, al
mov     esi, ds:lstrcatA
mov     al, ah
xor     al, 't'
xor     cl, 'd'
mov     [esp+518h+var_4FE], al
lea     eax, [esp+518h+pszPath]
xor     dl, 'c'
push    offset String2  ; "\\"
push    eax             ; lpString1
mov     [esp+520h+var_501], cl
mov     [esp+520h+var_4FF], dl
mov     [esp+520h+var_4FD], 0
call    esi ; lstrcatA
```

Code seen in an OrcaRat sample: XORing with “product” the hibernation filename.

Debug strings

Finally, let's look at the debug strings we can find in the binaries.

The LeoUncia sample studied by FireEye includes a perfect English string:

- “\r\nThe Remote Shell Execute: %s completed!\r\n”

Unfortunately, we cannot find this string in the OrcaRat sample. Bad luck...

But when we look at a more recent sample of LeoUncia, we have one with the above string and two other interesting strings:

- “\r\nThe Remote Shell Execute: %s completed!\r\n”
- “\r\nReturnTime Set Error!\r\n”
- “\r\nReturnTime set success!\r\n”

These two strings are linked to the writing in the hibernation file, and indicates to the C&C manager that its command either succeeded or failed.

That is very interesting because the OrcaRat sample is also using some very similar debug strings to notify its C&C about the hibernate command:

- “\r\nSet return time error = %d!\r\n”
- “\r\nSet return time success!\r\n”

And yes, it is always easier to debug your code when you know the error code; that's an improvement!

Conclusion

These two families are most likely linked in the sense that OrcaRat is a nicely updated version of LeoUncia.