Analysis of BlackMoon (Banking Trojan)'s Evolution, And The Possibility of a Latest Version Under Development

peppermalware.com/2019/03/analysis-of-blackmoon-banking-trojans.html

```
int key; // [esp+4h] [ebp-1Ch]
char *v5; // [esp+8h] [ebp-18h]
char *v6; // [esp+Ch] [ebp-14h]
char *v7; // [esp+10h] [ebp-10h]
char *v8; // [esp+14h] [ebp-Ch]
char *v9; // [esp+18h] [ebp-8h]
char *v10; // [esp+1Ch] [ebp-4h]

v10 = sub_41CDD0(edi0, 1, '8');
v9 = sub_41CDD0(edi0, 1, '8');
v8 = sub_41CDD0(edi0, 1, 'a');
v7 = sub_41CDD0(edi0, 1, '1');
v6 = sub_41CDD0(edi0, 1, '9');
v5 = sub_41CDD0(edi0, 1, '5');
key = getfullkey("7ac13b3aa82136afa3090c5137", v10, v9, v8, v7, v6, v5);
if ( v10 )
  sub_41C90C(v10);
```

BlackMoon, also known as KrBanker, is a banking trojan that mainly targets South Korea. I thought this family was dead since time ago (around 2016), however these previous days I got a couple of rencent samples that, after unpacking them and performing a quick analysis, I noticed they were BlackMoon. Virustotal's first submission date for one of these samples is 2018-06-18. First submission date for the other one is 2018-11-01. After digging a bit more into this malware family, my conclussion was that probably there is a latest version of BlackMoon that is under development. I explain it in this post, that I hope you enjoy.

- **Original Packed Sample:** C38E54342CDAE1D9181EC48E94DC5C83
- **Automatic Generated Report:** PepperMalware Report
- **Virustotal First Submission:** 2018-11-01 07:03:51
- **Unpacked Banker Module:** 4634F4EF94D9A3A0E2FCF5078151ADB2
- **Related links:**

Analysis

1. Loader

1.1. Packer

Most of the analyzed samples's packers are wellknown packers such as PeCompact, Aspack, Fsg or Nspack:

Sample	FirstSeen	Packer
09beec989993806345254ca9adcdb034f8649d8a9633bbe8933a52f5093e8be1	2018-11-01	PeCompact
80ea86d195bbc4384a1b9a77a2d477e2c4e6dc6d48f3f80447877dbbe41a4e40	2018-06-18	Aspack
2de1e47c650c0a8865ecc7e7b68379ca071062c0873f46a4addb1aa13b8d48dc	2016-03-03	Fsg
5f17cf9aee107458995c434d21263528132b5d0ab8a20121d3de48478ec6c467	2016-02-28	PeCompact
47434c9c2e887ba6f47a31e757b4ac0c0e648dfee9f93e38bd49e1c17f660dcf	2016-03-05	PeCompact
2012486d87dcc3362745c6f8f178b9be5417c595e79c452a20729d2e60ec814b	2016-03-08	Aspack
05afd7bbf6efa14102f72bad0e3a0686af6522b25228ab760ef57e8d6df36ed1	2016-03-05	Fsg
5e1ca094e11b2dcfdd4c729e2eaf1bdfd0ec84067a39f1c3a233bfff1ff6dcb5	2016-03-20	PeCompact
406c50ed0333d2023de55ce798a4e7d5fa6e45df65c16733ef48961e94277807	2016-04-08	Aspack
4844e92d76b2158be2b5468b70e2d0898f9ba2287a02b2b0aa7af2a2113d4970	2016-03-02	PeCompact
7351373a50acbaa4bb3fa622b0573f473289d745ba717551c82abbe398c1c1ff	2016-03-10	Nspack
09a5dc4f9544f7bbc898d205f1e14518606e158f4a7c7126d7eb604ec9ec5c74	2016-04-09	PeCompact
224ead790d3bab7ede11252728d47e21f0d0274767aa3e6a16628e8970a0149f	2016-02-28	PeCompact
00eae37eaaee93b8155e6bad95564c3d95d71e7397653ffcbae4f95614ffa723	2016-05-05	PeCompact

1.2. Process Injection

Most of the analyzed samples follow the same strategy, they launch an executable (I think it is choosen randomly) from %system32% folder and they inject the new process (hollow process). The unpacked code will be executed in the context of the new process. Some of the executables that we have seen the malware launchs are: wmiprvse.exe, dwwin.exe, comp.exe, cacls.exe, etc...

Sample	FirstSeen	Hollowed Process
09beec989993806345254ca9adcdb034f8649d8a9633bbe8933a52f5093e8be1	2018-11-01	system32\wmiprvse.exe
80ea86d195bbc4384a1b9a77a2d477e2c4e6dc6d48f3f80447877dbbe41a4e40	2018-06-18	system32\wmiprvse.exe
2de1e47c650c0a8865ecc7e7b68379ca071062c0873f46a4addb1aa13b8d48dc	2016-03-03	system32\dwwin.exe
5f17cf9aee107458995c434d21263528132b5d0ab8a20121d3de48478ec6c467	2016-02-28	system32\comp.exe
47434c9c2e887ba6f47a31e757b4ac0c0e648dfee9f93e38bd49e1c17f660dcf	2016-03-05	system32\comp.exe
2012486d87dcc3362745c6f8f178b9be5417c595e79c452a20729d2e60ec814b	2016-03-08	system32\cacls.exe
05afd7bbf6efa14102f72bad0e3a0686af6522b25228ab760ef57e8d6df36ed1	2016-03-05	system32\cacls.exe
5e1ca094e11b2dcfdd4c729e2eaf1bdfd0ec84067a39f1c3a233bfff1ff6dcb5	2016-03-20	system32\cacls.exe
406c50ed0333d2023de55ce798a4e7d5fa6e45df65c16733ef48961e94277807	2016-04-08	system32\cacls.exe
4844e92d76b2158be2b5468b70e2d0898f9ba2287a02b2b0aa7af2a2113d4970	2016-03-02	system32\comp.exe
7351373a50acbaa4bb3fa622b0573f473289d745ba717551c82abbe398c1c1ff	2016-03-10	system32\cacls.exe
09a5dc4f9544f7bbc898d205f1e14518606e158f4a7c7126d7eb604ec9ec5c74	2016-04-09	system32\cacls.exe
224ead790d3bab7ede11252728d47e21f0d0274767aa3e6a16628e8970a0149f	2016-02-28	system32\comp.exe

2. Main Module

2.1. Persistence

The malware installs itself under a HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run's subkey. For most of the older samples, the run subkey is a 8-length combination of lowercase and uppercase letters and numbers. However the analyzed samples that date 2018, install themself in the subkey with fixed name 000C29FC2AB3.

Sample	First Seen	Run Subkey
09beec989993806345254ca9adcdb034f8649d8a9633bbe8933a52f5093e8be1	2018-11-01	**000C29FC2AB3**
80ea86d195bbc4384a1b9a77a2d477e2c4e6dc6d48f3f80447877dbbe41a4e40	2018-06-18	**000C29FC2AB3**
2de1e47c650c0a8865ecc7e7b68379ca071062c0873f46a4addb1aa13b8d48dc	2016-03-03	06iSwa6C
5f17cf9aee107458995c434d21263528132b5d0ab8a20121d3de48478ec6c467	2016-02-28	kC6MOsu8
47434c9c2e887ba6f47a31e757b4ac0c0e648dfee9f93e38bd49e1c17f660dcf	2016-03-05	R3tP5nj1
2012486d87dcc3362745c6f8f178b9be5417c595e79c452a20729d2e60ec814b	2016-03-08	66qscw4Q
05afd7bbf6efa14102f72bad0e3a0686af6522b25228ab760ef57e8d6df36ed1	2016-03-05	W60u80qO
5e1ca094e11b2dcfdd4c729e2eaf1bdfd0ec84067a39f1c3a233bfff1ff6dcb5	2016-03-20	uki4Kk2o
406c50ed0333d2023de55ce798a4e7d5fa6e45df65c16733ef48961e94277807	2016-04-08	35V5Bj9b
4844e92d76b2158be2b5468b70e2d0898f9ba2287a02b2b0aa7af2a2113d4970	2016-03-02	AAAC2kY8
7351373a50acbaa4bb3fa622b0573f473289d745ba717551c82abbe398c1c1ff	2016-03-10	1Lf9Tn7B
09a5dc4f9544f7bbc898d205f1e14518606e158f4a7c7126d7eb604ec9ec5c74	2016-04-09	5jNh7p11
224ead790d3bab7ede11252728d47e21f0d0274767aa3e6a16628e8970a0149f	2016-02-28	j3pVbRJ5
00eae37eaaee93b8155e6bad95564c3d95d71e7397653ffcbae4f95614ffa723	2016-05-05	**000C29FC2AB3**

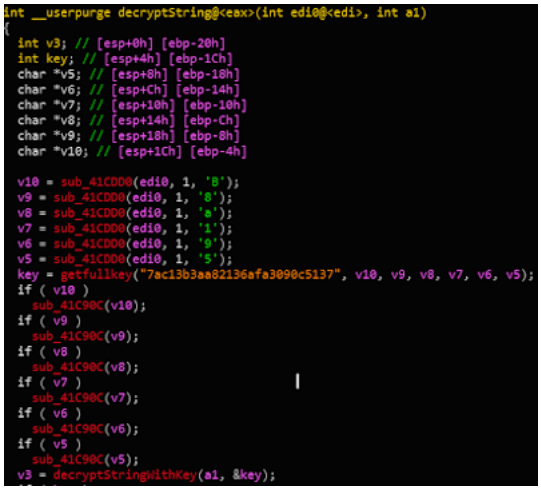
Curiously, the sample 00eae37eaaee93b8155e6bad95564c3d95d71e7397653ffcbae4f95614ffa723 that dates 2016-05-05 (from the older samples, one of the newest), installs itself under the same subkey 000C29FC2AB3.

In addition, these samples that create the subkey with name **000C29FC2AB3**, they create a mutex named **M_Test** too (the other samples don't create this mutex).

2.2. Encrypted Strings

Most of the important strings of BlackMoon are encrypted.

Here is a capture of the code responsible for decrypting the strings from the sample 09beec989993806345254ca9adcdb034f8649d8a9633bbe8933a52f5093e8be1:

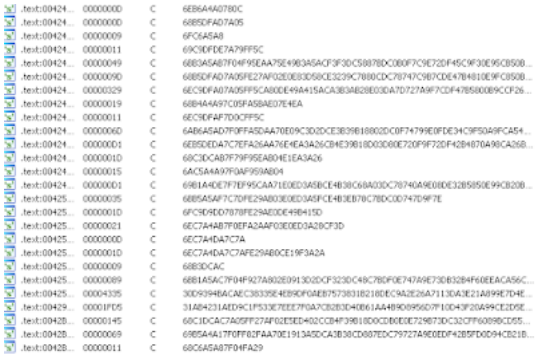


```
int __userpurge decryptString@<eax>(int edi0@<edi>, int a1)
{
  int v3; // [esp+0h] [ebp-20h]
  int key; // [esp+4h] [ebp-1Ch]
  char *v5; // [esp+8h] [ebp-18h]
  char *v6; // [esp+Ch] [ebp-14h]
  char *v7; // [esp+10h] [ebp-10h]
  char *v8; // [esp+14h] [ebp-Ch]
  char *v9; // [esp+18h] [ebp-8h]
  char *v10; // [esp+1Ch] [ebp-4h]

  v10 = sub_41CDD0(edi0, 1, 'B');
  v9 = sub_41CDD0(edi0, 1, '8');
  v8 = sub_41CDD0(edi0, 1, 'a');
  v7 = sub_41CDD0(edi0, 1, '1');
  v6 = sub_41CDD0(edi0, 1, '9');
  v5 = sub_41CDD0(edi0, 1, '5');
  key = getfullkey("7ac13b3aa82136afa3090c5137", v10, v9, v8, v7, v6, v5);
  if ( v10 )
    sub_41C90C(v10);
  if ( v9 )
    sub_41C90C(v9);
  if ( v8 )
    sub_41C90C(v8);
  if ( v7 )
    sub_41C90C(v7);
  if ( v6 )
    sub_41C90C(v6);
  if ( v5 )
    sub_41C90C(v5);
  v3 = decryptStringWithKey(a1, &key);
```

To compose the definitive key that the malware uses to decrypt the strings, it carries an string that is the first part of the key, and then it appends 6 additional characters to that first part of the key. In the capture, the definitive key to be used would be "7ac13b3aa82136afa3090c5137B8a195".

Encrypted strings are like this:



The algorithm used to decrypt each string is rc4(unhexlify(rc4(unhexlify(encrypted_string), key)), key):

```
def decstr(s, k, k2):
    s = binascii.unhexlify(s)
    s = rc4(s, k+k2)
    s = binascii.unhexlify(s)
    s = rc4(s, k+k2)
    return s
```

3. Evolution

3.1. Encrypted Strings Evolution

We have extracted the strings from samples from different dates, to compare them:

Date 2016-02-28:
Sample 5f17cf9aee107458995c434d21263528132b5d0ab8a20121d3de48478ec6c467:

Date 2016-03-03:
Sample 2de1e47c650c0a8865ecc7e7b68379ca071062c0873f46a4addb1aa13b8d48dc:

Date 2016-05-05:
Sample 00eae37eaaee93b8155e6bad95564c3d95d71e7397653ffcbae4f95614ffa723:

Date 2018-06-18:

Sample 80ea86d195bbc4384a1b9a77a2d477e2c4e6dc6d48f3f80447877dbbe41a:

Date 2018-11-01:
Sample 09beec989993806345254ca9adcdb034f8649d8a9633bbe8933a52f5093e:

In the section 2.1 (about persistence), we had already noticed that most of the samples from 2016 create a 8 bytes length subkey under the registry \Run key, with a combination of lowercase and uppercase letters and numbers.

However a sample dated 2016-05-05 and the newer samples dated 2018 create a subkey under \Run with name **000C29FC2AB3**. In addition these samples create a mutex with name **M_Test** (this mutex is not created by the 2016's samples).

If we take a look at the lists of strings, the sample dated 2016-05-05 and the samples dated 2018, all of them have similar lists of encrypted strings, where strings are ordered in similar order (thought they are not totally identicals).

The other samples dated 2016 contain another lists of strings, identical between them, but different from the lists of the samples dated 2018.

3.2. BlackMoon Versions: Latest Version Under Development?

Having in mind the IoCs collected in the previous sections, we can conclude that there is a first version of BlackMoon malware, whose samples are dated around 2016, and other version that could be under development, whose samples we have one of them dated 2016-05-05, and other two dated 2018-06 and 2018-11.

Version 1:

- Persistence: 8 bytes length subkey under registry \Run key, with a combination of lowercase and uppercase letters and numbers
- Encrypted strings: "http://", "/ca.php", "?m=", "&h;=", "GET", "?p", "POST", "users.qzone.qq.com", "GET /fcg-bin/cgi_get_portrait.fcg?uins=", etc...
- Samples dated 2016

Version 2 - probably under development version:

- Persistence: subkey under \Run with name **000C29FC2AB3**
- Mutex: **M_Test**
- Encrypted strings: "ScriptControl", "Language", "VBScript", "ExecuteStatement", "Function MACAddress()", "Dim mc,mo", "Set mc=GetObject(\"Winmgmts:\").InstancesOf(\"Win32_NetworkAdapterConfiguration\"), "For Each mo In mc", etc...
- A sample dated 2016-05-05, other 2 samples dated 2018

We have only 3 samples that we have classified as version 2. Probably they are quite similar, but we must have in mind that the lists of encrypted strings for these samples are not totally identical. However, the Run key 000C29FC2AB3 and the mutex M_Test, make us to think these 3 samples are the same version.

From my point of view, these 3 newer samples could be a version that is under development. Because of that, each version 2's sample is a bit different from the others. And because of that, the name M_Test for the mutex and the non-random name for the \Run subkey.

3.3. BinDiff

Lets compare with BinDiff the following samples (once they are already unpacked) trying to understand the evolution of this malware:

Version1:

- 2de1e47c650c0a8865ecc7e7b68379ca071062c0873f46a4addb1aa13b8d48dc
- 2016-03-03
- Original sample packed with Fsg

Version2:

- 00eae37eaaee93b8155e6bad95564c3d95d71e7397653ffcbae4f95614ffa723
- 2016-05-05
- Original sample packed with PeCompact

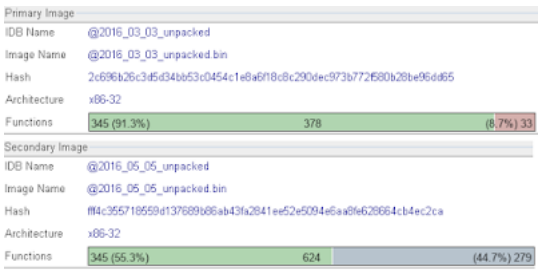
Version2:

- 80ea86d195bbc4384a1b9a77a2d477e2c4e6dc6d48f3f80447877dbbe41a
- 2018-06-18
- Original sample packed with AsPack

Version2:

- 09beec989993806345254ca9adcdb034f8649d8a9633bbe8933a52f5093e
- 2018-11-01
- Original sample packed with PeCompact

3.3.1. 2016-03-03 -> 2016-05-05 Statistics: 345 matching functions



3.3.2. 2016-05-05 -> 2018-06-18 Statistics: 591 matching functions

Primary Image			
IDB Name	@2016_05_05_unpacked		
Image Name	@2016_05_05_unpacked.bin		
Hash	ff4c355718559d137689b66ab43fa2841ee52e5094e6aa8fe626664cb4ec2ca		
Architecture	x86-32		
Functions	591 (94.7%)	624	(5.3%) 33
Secondary Image			
IDB Name	@2018_06_18_unpacked		
Image Name	@2018_06_18_unpacked.bin		
Hash	dd9aac81ddb433138a01d338c062255e105203eedec88262b7c5f166895a0093		
Architecture	x86-32		
Functions	591 (26.8%)	2204	(73.2%) 1613

3.3.3. 2018-06-18 -> 2018-11-01 Statistics: 1743 matching functions

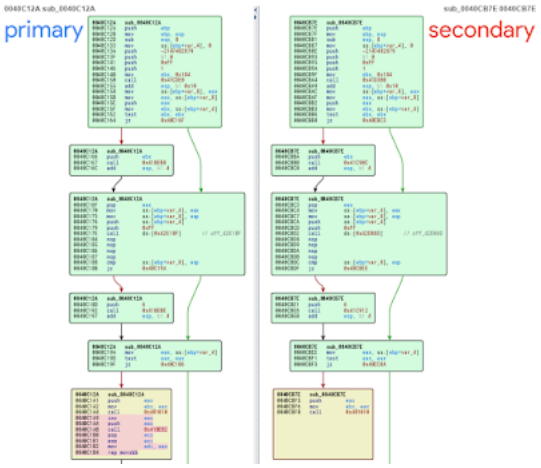
Primary Image			
IDB Name	@2018_06_18_unpacked		
Image Name	@2018_06_18_unpacked.bin		
Hash	dd9aac81ddb433138a01d338c062255e105203eedec88262b7c5f166895a0093		
Architecture	x86-32		
Functions	1743 (79.1%)	2204	(20.9%) 461
Secondary Image			
IDB Name	@2018_11_01_unpacked		
Image Name	@2018_11_01_unpacked.bin		
Hash	f0d57176f8a27f89a3ca2d14a80d5b672b0590a1e449cbfa653a8091cdfc9cd7		
Architecture	x86-32		
Functions	1743 (87.2%)	1999	(12.8%) 256

I think the most interesting indicator about similarity, at least in this case, is the number of matching functions because the unpacked modules were dumped with Volatility's procdump command, with --memory --unsafe modificators. Probably most of the primary and secondary unmatched functions are due to residual parts of the code of the packer in memory and maybe due to recompilations of the code with newer versions of the runtime.

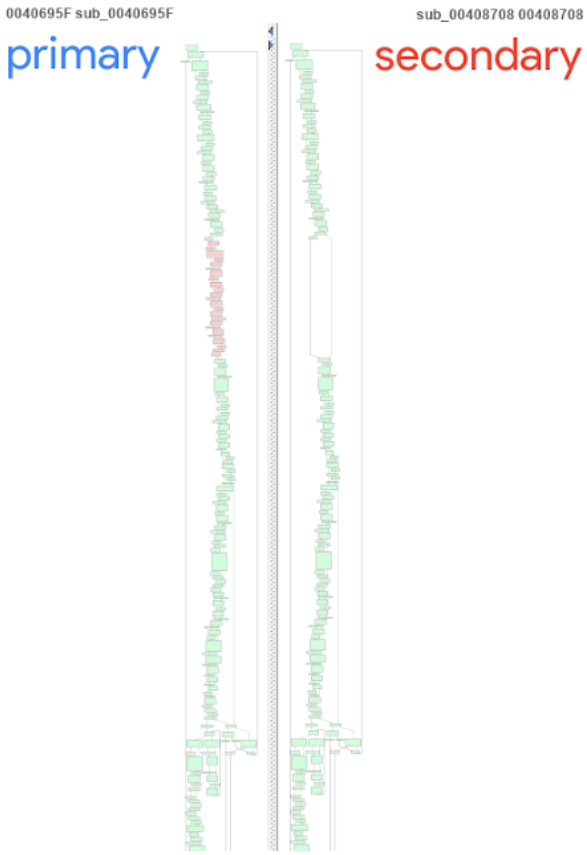
If we compare the paired functions, we find that most of the changes between versions are due to ligth modifications, small fixes, etc... as we will see in the following sections.

3.3.4. 2016-03-03 -> 2016-05-05 Differences:

For example, here is a function from the sample dated 2016-03-03 compared to the same function from the sample dated 2016-05-05, where we can see that small changes were done in this function:



Another function. In this case a larger part of code was removed from the function in the newer version:



Btw, in the case of 2016-03-03 -> 2016-05-05, most of the matching functions are ubicated in totally different addresses:

Si...	Con...	Address	Primary Name	Type	Address	Secondary Name
1.00	0.62	0041BEA6	sub_0041BEA6	Normal	0041AC15	sub_0041AC15
1.00	0.64	00421090	sub_00421090	Normal	00421B60	sub_00421B60
1.00	0.82	0041AA47	sub_0041AA47	Normal	0041B54B	sub_0041B54B
1.00	0.82	0041AA73	sub_0041AA73	Normal	0041B577	sub_0041B577
1.00	0.82	0041AA9F	sub_0041AA9F	Normal	0041B5A3	sub_0041B5A3
1.00	0.82	0041ADC9	sub_0041ADC9	Normal	0041C52B	sub_0041C52B
1.00	0.82	0041BF08	sub_0041BF08	Normal	0041CA22	sub_0041CA22
1.00	0.82	0041BFB8	sub_0041BFB8	Normal	0041CA4E	sub_0041CA4E
1.00	0.82	0041BFE4	sub_0041BFE4	Normal	0041CA64	sub_0041CA64
1.00	0.82	0041BFFA	sub_0041BFFA	Normal	0041CA7A	sub_0041CA7A
1.00	0.88	0040828D	sub_0040828D	Normal	00409F05	sub_00409F05
1.00	0.90	00420A60	sub_00420A60	Normal	00421B20	sub_00421B20
1.00	0.93	0041CB50	sub_0041CB50	Normal	0041FC40	sub_0041FC40
1.00	0.95	00421758	sub_00421758	Normal	004221B8	sub_004221B8
1.00	0.95	00421778	sub_00421778	Normal	004221D8	sub_004221D8
1.00	0.96	0041C050	sub_0041C050	Normal	0041CAB0	sub_0041CAB0
1.00	0.96	0041BF1E	sub_0041BF1E	Normal	0041C988	sub_0041C988
1.00	0.96	0041BF34	sub_0041BF34	Normal	0041C99E	sub_0041C99E
1.00	0.96	0041BF4A	sub_0041BF4A	Normal	0041C9B4	sub_0041C9B4
1.00	0.96	0041BF60	sub_0041BF60	Normal	0041C9CA	sub_0041C9CA
1.00	0.96	0041BF76	sub_0041BF76	Normal	0041C9E0	sub_0041C9E0
1.00	0.96	0041BF8C	sub_0041BF8C	Normal	0041C9F6	sub_0041C9F6
1.00	0.96	0041BFA2	sub_0041BFA2	Normal	0041CA0C	sub_0041CA0C
1.00	0.96	0041BFCE	sub_0041BFCE	Normal	0041CA38	sub_0041CA38

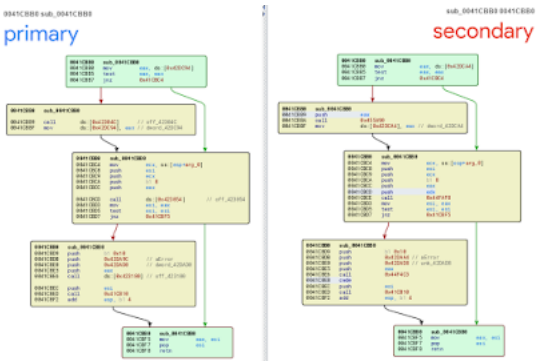
Probably, in spite of the fact that the code doesn't change a lot and there are a lot of matching functions, **a code refactorization was done from version 1 to the first samples of version 2** (around 2016-05).

3.3.5. 2016-05-05 -> 2018-06-18 Differences:

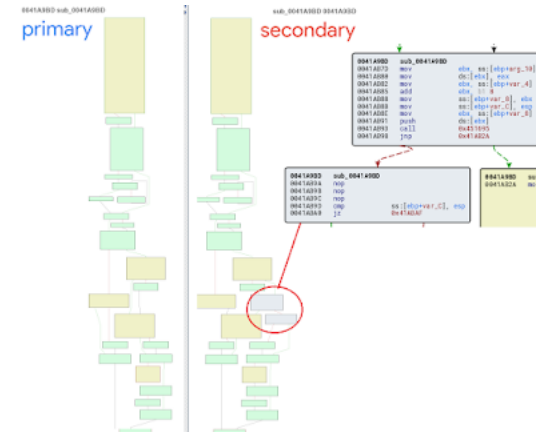
In this case, in addition to the similarity between functions pairs, lot of the matching functions are ubicated in the same offset into the unpacked sample:

Si...	Co...	Address	Primary Name	Type	Address	Secondary Name
0.96	0.97	004083D7	sub_004083D7	Normal	004083D7	sub_004083D7
0.96	0.98	00405C14	sub_00405C14	Normal	00405C14	sub_00405C14
0.95	0.95	0041C906	sub_0041C906	Normal	0041C906	sub_0041C906
0.94	0.94	0041B5CF	sub_0041B5CF	Normal	0041B5CF	sub_0041B5CF
0.94	0.99	0041DFA0	sub_0041DFA0	Normal	0041DFA0	sub_0041DFA0
0.93	0.94	00403A60	sub_00403A60	Normal	00403A60	sub_00403A60
0.93	0.94	0040373E	sub_0040373E	Normal	0040373E	sub_0040373E
0.93	0.97	00403C2A	sub_00403C2A	Normal	00403C2A	sub_00403C2A
0.93	0.94	004038A4	sub_004038A4	Normal	004038A4	sub_004038A4
0.93	0.94	00403982	sub_00403982	Normal	00403982	sub_00403982
0.92	0.97	0041E790	sub_0041E790	Normal	0041E790	sub_0041E790
0.91	0.99	00421520	sub_00421520	Normal	00421520	sub_00421520
0.89	0.89	0041CBB0	sub_0041CBB0	Normal	0041CBB0	sub_0041CBB0
0.88	0.90	00401010	sub_00401010	Normal	00401010	sub_00401010
0.87	0.96	0041A9BD	sub_0041A9BD	Normal	0041A9BD	sub_0041A9BD
0.87	0.92	00441E32	sub_00441E32	Normal	0044AAD3	sub_0044AAD3
0.87	0.92	0043EB85	sub_0043EB85	Normal	0044A828	sub_0044A828
0.86	0.92	0044133E	sub_0044133E	Normal	0044A391	sub_0044A391
0.85	0.91	0043FBE2	sub_0043FBE2	Normal	00451437	sub_00451437
0.85	0.90	00440468	sub_00440468	Normal	0044A6FA	sub_0044A6FA
0.85	0.85	00419E04	sub_00419E04	Normal	0041A5A4	sub_0041A5A4
0.84	0.90	00440525	sub_00440525	Normal	004450EA	sub_004450EA
0.84	0.96	00420FE0	sub_00420FE0	Normal	00420FE0	sub_00420FE0
0.84	0.85	0041C918	sub_0041C918	Normal	0041C918	sub_0041C918
0.84	0.99	00407BD9	sub_00407BD9	Normal	00407BD9	sub_00407BD9

This makes us to think both binaries are quite similiar, in spite of the fact that we find minimal changes like in this function:

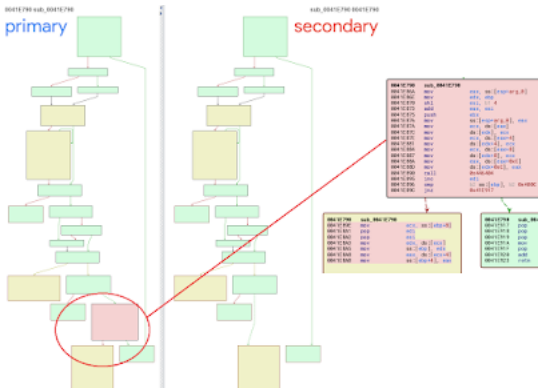


However, there are other functions with more important changes that make us to think that there have been at least a minimal development between both samples (manual modifications on the code: improvements or fixes, not only recompilation + repacking):



3.3.5. 2018-06-18 -> 2018-11-01 Differences:

Again, lot of the matching functions ubicated in the same offset, and minimal changes between paired functions. And again, some parts of the code with more important changes that suggest a minimal development by the authors between the first and the second sample:



4. Conclusions

From my point of view, there are **two main versions** of BlackMoon family.

Samples from the first version date first half-year of 2016.

Around May-2016, a new version was started. In the sample that dates 2016-05-05 we can appreciate a **code refactorization and more important changes in the code**. In addition, we can find changes in the behavior, such as the **non-random subkey under the \Run registry key, named 000C29FC2AB3, and the non-random mutex created by the malware with name M_Test**.

There are minimal changes between the sample that dates 2018-06-18 and the samples that dates 2016-05-05, and again minimal changes between the samples that dates 2016-11-01 and the sample that dates 2018-06-18. However, there are **enough changes between these version 2's samples to appreciate that a development was done by the authors,** there must be modifications of the source code between them (not only recompilation + repacking).

My conclussion is, there is a version of the BlackMoon that is under development. We can find quite recent samples (based on the VirusTotal first seen date) of this version under development. I can't say totally sure if the code of that recent samples were modified and compiled in 2018 or previously (in spite of the fact that I think the code was recently modified and it is currently evolving, maybe that samples were only repacked or their bytes lightly modified, or maybe VirusTotal didn't see these samples before).

In addition to the larger changes from the first version to the second version, we can appreciate an evolution of the code of the second version: from the sample 00eae37eaaee93b8155e6bad95564c3d95d71e7397653ffcbae4f95614ffa723 (May-2016), to the sample 80ea86d195bbc4384a1b9a77a2d477e2c4e6dc6d48f3f80447877dbbe41a (June-2018), and to the sample 09beec989993806345254ca9adcdb034f8649d8a9633bbe8933a52f5093e (November-2018). So, from my point of view,it seems **there are enough evidences to think that there is a BlackMoon version that is under development and currently evolving.**

5. Yara Rules and Scripts

5.1. BlackMoon Yara Rule

Unpacked module:

```
rule blackmoon_unpacked {
strings:
        $code1 = { 89 45 ?? 68 01 01 00 80 6A 00 68 ?? 00 00 00 68 01 00 00 00 BB ?? ?? 00 00 E8 ?? ?? ?? ?? 83 C4 10 }
        $code2 = { FF 75 ?? FF 75 ?? FF 75 ?? FF 75 ?? FF 75 ?? FF 75 ?? FF 75 ?? B9 ?? ?? 00 00 E8 }
condition:
        (all of them)
}
```

5.2. Script to Extract BlackMoon Encrypted Strings

The following script extracts and decrypts the encrypted strings from a BlackMoon unpacked sample:

python strings_decryptor.py <path to unpacked blackmoon>

```
import os
import sys
import binascii
import traceback

################################################

def rc4(data, key):
    x = 0
    box = range(256)
    for i in range(256):
        x = (x + box[i] + ord(key[i % len(key)])) % 256
        box[i], box[x] = box[x], box[i]
    x = 0
    y = 0
    out = []
    for char in data:
        x = (x + 1) % 256
        y = (y + box[x]) % 256
        box[x], box[y] = box[y], box[x]
        out.append(chr(ord(char) ^ box[(box[x] + box[y]) % 256]))
    return ''.join(out)

################################################

def findencstrings(s):
    l = []
    laststr = ""
    for i in range(0, len(s)):
        if s[i] in "0123456789ABCDEF":
            laststr += s[i]
        else:
            if ord(s[i])==0 and len(laststr)>=6: l.append(laststr)
            laststr = ""
    return l

################################################

def decstr(s, k, k2):
    sorig=s
    try:
        if len(s)%2: s = s[0:-1]
        s = binascii.unhexlify(s)
        s = rc4(s, k+k2)
        step1 = s
        if len(s)%2: s = s[0:-1]
        s = binascii.unhexlify(s)
        s = rc4(s, k+k2)
        return True, s
    except Exception as e:
        return False, "ERROR:" + repr(e) + ", string:" + sorig

################################################

def findkey1(s):
    l = []
    laststr = ""
    for i in range(0, len(s)):
        if s[i] in "0123456789abcdefABCDEF":
            laststr += s[i]
        else:
            if ord(s[i])==0 and len(laststr)>=20 and len(laststr)<=30 and not len(laststr)%2 and laststr not in l:
l.append(laststr)
            laststr = ""
    if len(l): return l
    return None

################################################

def findkey2(s):
    key=""
    for i in range(0x0, len(s)-0x100):
        if s[i:i+8]=="\x68\x01\x01\x00\x80\x6a\x00\x68" and s[i+8] in "0123456789abcdefABCDEF" and s[i+9:i+12]=="\x00\x00\x00":
            key+=s[i+8]
    return key

################################################

def get_strings_from_pe(s):
    ldecs = []
    lenc = findencstrings(s)
```

```
    lk1 = findkey1(s)
    k2 = findkey2(s)
    if lk1 and k2 and lenc:
        for k1 in lk1:
            for i in range(0,len(k2)-6):
                for senc in lenc:
                    decs = decstr(senc, k1, k2[i:i+6])
                    if decs[0]: ldecs.append(decs[1])
    return ldecs

################################################

def analexe(s):
    decrypted_string_list = []
    try: decrypted_string_list = get_strings_from_pe(s)
    except Exception as e:
        print "blackmoon exception in get_strings_from_pe"
        print traceback.format_exc()
    for e in decrypted_string_list:
        print "blackmoon decrypted string:", e

################################################

if __name__ == "__main__":
    if os.path.exists(sys.argv[1]):
        f = open(sys.argv[1], "rb")
        s = f.read()
        f.close()
        analexe(s)
    else:
        print "Incorrect path"
```

6. Other Notes

6.1. Another sample dated 2018 suspicious of being BlackMoon

Once I started to investigate a bit more and to search information about BlackMoon family, I found a tweet talking about another sample that could be BlackMoon and whose first submission is 2018-08-08.

I toke a quick look at this sample, here you can find the unpacked module. Some interesting strings from this unpacked module:

- http://aa.mrmr11[.]cn:8000/fdeee.dll
- yPBfy0A4q1Y3gvgmREe0r1UR0fZVidMd4V8CB3oKTzNaOYCyPaSVz48Sw5mVifR3sVxYgeM7EyVu6DwnrfAG/AxGgDr+9GIP3cQ59d/eLtP
- C:\Program Files\AppPatch\lpDllName
- 360zipUpdate.EXE

The original sample was packed with Aspack, as other recent BlackMoon samples. However this first unpacked module doesn't look like BlackMoon: the code, the strings, etc... are totally different.

Anyway, when I have analyzed this sample, the dll that it tries to download (http://aa.mrmr11[.]cn:8000/fdeee.dll) had been already removed. Maybe that dll was the BlackMoon module. I can't be sure if this sample is BlackMoon or not (this other any.run analysis contains this same IoC: "C:\Program Files\AppPatch\lpDllName", it downloads a dll too, and the second process name is similar format. Maybe same family. This other one was tagged as #trojan #dupzom).