

Loki-Bot: Come out, come out, wherever you are!

By Published by R3MRUM View all posts by R3MRUM

Published: 2017-05-07 · Archived: 2026-04-05 19:07:25 UTC

Intro

I'm going to make my first post an easy one. I'm currently in the middle of writing up my GREM Gold paper, which focuses on the reverse engineering of a Loki-Bot v1.8 sample. This post is going to focus on how Loki-Bot creates its mutex and the folders, files, and registry keys that are created as a result.

Per [PhishMe](#):

Loki Bot is a commodity **malware** sold on underground sites which is designed to steal private data from infected machines, and **then** submit that info to a command and control host via HTTP POST. This private data includes stored passwords, login credential information from Web browsers, and a variety of cryptocurrency wallets.

What is a Mutex?

Understanding what a Mutex is can be a bit difficult to understand for those with little-to-no programming background. I found it best described on the [SANS DFIR Blog](#):

“Programs use mutex (“mutual exclusion”) objects as a locking mechanism to serialize access to a resource on the system.” ... “Furthermore, malware might use a mutex to avoid reinfecting the host. For instance, the specimen might attempt to open a handle to a mutex with a specific name. The specimen might exit if the mutex exists, because the host is already infected.”

Creating the Mutex

So, based on the mutex description, Loki-Bot uses a mutex to ensure that multiple versions of Loki-Bot cant be running at the same time. In order for this to happen, both versions of Loki-Bot need to have the same logic for naming the mutex. What we are going to talk about next is said logic.

Obtaining the Machine GUID

```

00404A63 | . 53          PUSH EBX
00404A64 | . 53          PUSH EBX
00404A65 | . 68 DCACB4F4 PUSH F4B4ACDC
00404A66 | . 6A 09       PUSH 9
00404A6C | . E8 74E7FFF CALL getDLLFunctionFromIDXAndHash
00404A71 | . 8D4D FC     LEA ECX, [LOCAL.1]
00404A74 | . 51         PUSH ECX
00404A75 | . 68 19010200 PUSH 20119
00404A7A | . 53         PUSH EBX
00404A7B | . FF75 0C     PUSH DWORD PTR SS:[ARG.2]
00404A7E | . FF75 08     PUSH DWORD PTR SS:[ARG.1]
00404A81 | . FFD0       CALL EAX

```

Arg4 => 0
 Arg3 => 0
 Arg2 = F4B4ACDC — Hash representing RegOpenKey
 Arg1 = 9 — Index representing ADVAPI32
 FE62C1C283CF41CA826AA267F5AA6F7.getDLLFunctionFromIDXAndHash

Hex value representing desired access. In this case it's requesting read access
 ARG.2 ["SOFTWARE\Microsoft\Cryptography"] passed from getMachineGUIDFromRegistry
 ARG.1 [8000002] passed from getMachineGUIDFromRegistry
 ADVAPI32.RegopenKey

Address	Hex dump	ASCII	Actual Value	Translated Value
AE22D263	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0013FEB8	80000002
AE22D273	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0013FEC0	004162B4
AE22D283	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0013FEC4	00000000
AE22D293	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		0013FEC8	00020119

Hkey = HKEY_LOCAL_MACHINE
 Subkey = "SOFTWARE\Microsoft\Cryptography"
 Reserved = 0
 DesiredAccess = KEY_READ|KEY_WOW64_64KEY
 Result = 0013FEDC -> AE22D263

First and foremost, know that Loki-Bot employs function hashing to thwart analysis. This is what you are seeing from 0x404A63 to 0x404A6C. Two important arguments passed to the function labeled `getDLLFunctionFromIDXAndHash` are `Arg1` (DLL Index) and `Arg2` (Function Hash). In this instance, these values are set to 9 and 'F4B4ACDC'. Without diving too deep into this, know that the DLL Index of 9 equates to `ADVAPI32` and the hash 'F4B4ACDC' decodes to [RegOpenKeyEx](#). At 0x404A81, we see the decoded function `ADVAPI32.RegOpenKeyEx` being called.

This will open the registry path:

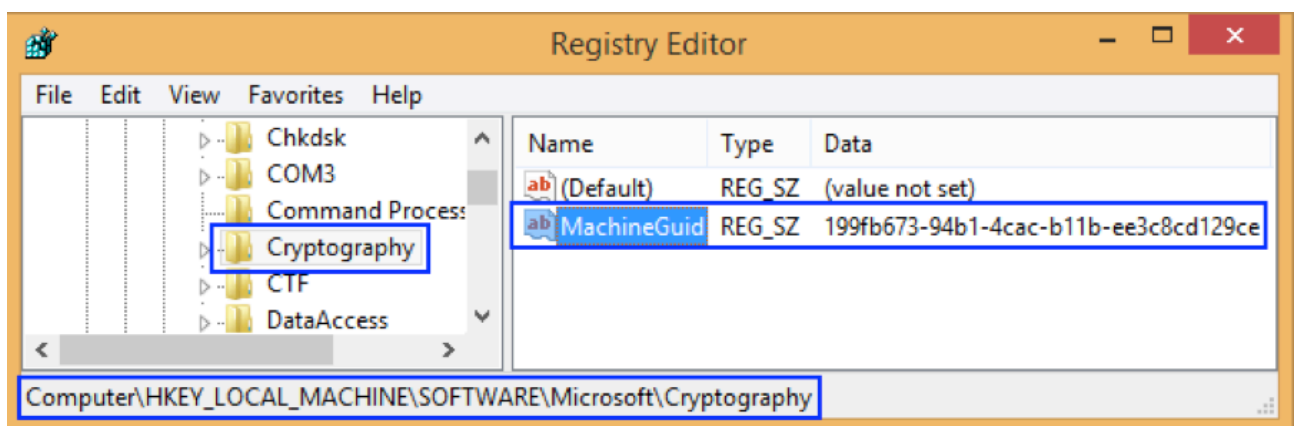
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptograpy\"

But it doesn't actually *read* the value contained within the key it needs. For this to happen, `ADVAPI32's RegQueryValueEx` function needs to be called.

After successful execution, the value stored in the memory address referenced in the `pData` argument (0x292388) now contains the value that was in the `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptograpy\MachineGuid` registry key.

Address	Hex dump	ASCII
00292388	31 39 39 66 62 36 37 33 2D 39 34 62 31 2D 34 63	199fb673-94b1-4c
00292398	61 63 2D 62 31 31 62 2D 65 65 33 63 38 63 64 31	ac-b11b-ee3c8cd1
002923A8	32 39 63 65 00 00 62 00 31 00 31 00 62 00 2D 00	29ce b 1 1 b -
002923B8	65 00 65 00 33 00 63 00 38 00 63 00 64 00 31 00	e e 3 c 8 c d 1
002923C8	32 00 39 00 63 00 65 00 00 00 00 00 00 00 00 00	2 9 c e

We can validate this by simply loading up [RegEdit](#) on the Windows host that is about to be compromised and navigating to the referenced registry key.




```
remnux@remnux: ~
File Edit Tabs Help
remnux@remnux:~$ echo -n '199fb673-94b1-4cac-b11b-ee3c8cd129ce' |md5sum | tr [a-z] [A-Z]
9BD0BA527DFA20AB1F4A05B8D0D4E04B -
remnux@remnux:~$
```

Trim Hash & Create Mutex

Finally, Loki-Bot trims the MD5 hash of the Machine GUID to 24-characters:

“9BD0BA527DFA20AB1F4A05B8”.

004141B3	. E8 10040000	CALL getMutexName	
004141B8	. 53	PUSH EBX	
004141B9	. 53	PUSH EBX	
004141BA	. 68 F47D16CF	PUSH CF167DF4	Arg4
004141BF	. 53	PUSH EBX	Arg3
004141C0	. 8BF0	MOV ESI, EAX — Move MD5 hash from EAX to ESI	Arg2 = CF167DF4 — Hash referring to CreateMutex
004141C2	. E8 1EF0FEFF	CALL getDLLFunctionFromIDXAndHash	Arg1 — EBX is 0, the index value for Kernel32
004141C7	. 56	PUSH ESI — Set lpName argument of CreateMutex to the 24-character trimmed MD5 hash	FE62C1C283CF41CA826AA267F5AA6F7.getDLLFunctionFromIDXAndHash
004141C8	. 33F6	XOR ESI, ESI	
004141CA	. 46	INC ESI	
004141CB	. 56	PUSH ESI	
004141CC	. 53	PUSH EBX	
004141CD	. FFD0	CALL EAX	KERNEL32.CreateMutexw
004141CF	. FF15 10604100	CALL DWORD PTR DS:[<&KERNEL32.GetLastError>]	KERNEL32.GetLastError
004141D5	. 3D B7000000	CMP EAX, 0B7	CONST B7 => ERROR_ALREADY_EXISTS
004141DA	. 75 07	JNE SHORT 004141E3 — Jump if mutex does not exist	
004141DC	. 53	PUSH EBX	Arg1
004141DD	. E8 D0010000	CALL exitProcess — Exit the process if it does	FE62C1C283CF41CA826AA267F5AA6F7.exitProcess
004141E2	. 59	POP ECX	
004141E3	> E8 4CF6FFFF	CALL MineAndStealData	

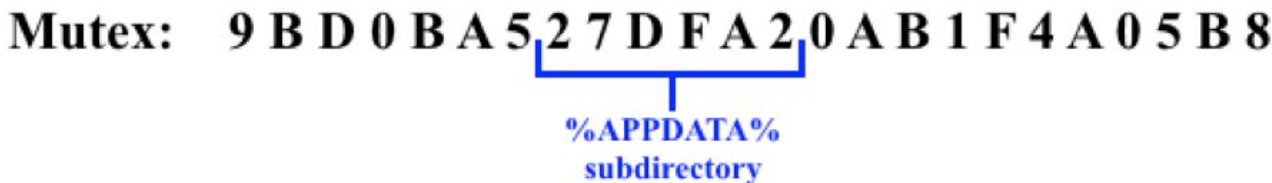
EAX=755E44D0 (KERNEL32.CreateMutexw) - jumps to KERNELBASE.CreateMutexw
FE62C1C283CF41CA826AA267F5AA6F7.main+136

Address	Hex dump	ASCII	
00183650	39 00 42 00 44 00 30 00 42 00 41 00 35 00 32 00	9 B D 0 B A 5 2	0013FF08 00000000 pSecurity = NULL
00183660	37 00 44 00 46 00 41 00 32 00 30 00 41 00 42 00	7 D F A 2 0 A B	0013FF0C 00000001 InitialOwner = TRUE
00183670	31 00 46 00 34 00 41 00 30 00 35 00 42 00 38 00	1 F 4 A 0 5 B 8	0013FF10 00183650 P6: LName = "9BD0BA527DFA20AB1F4A05B8"

It then passes this trimmed value to Kernel32’s [CreateMutexW](#) function as the *lpName* attribute. If the function succeeds, it means that no other version of Loki-Bot is running on the system at that time and execution continues on. If it fails, it means another version of Loki-Bot is running, so Loki-Bot quietly exits.

Identify Folder/Files

Now that we know the mutex, we can identify the folders and files that are related to Loki-Bot. As part of setting up persistence, Loki-Bot will create a hidden folder within your %APPDATA% path whose name set by the 8th thru 13th characters of the mutex.



Once the hidden folder “%APPDATA%\27DFA2\” has been created, Loki-Bot will store several different types of files within it; all with the same filename but with different extensions. The filename used for the different files is also extracted from the mutex.

Mutex: 9 B D 0 B A 5 2 7 D F A 2 0 A B 1 F 4 A 0 5 B 8



With the filename known, we can then identify the following files:

- **%APPDATA%\27DFA2\20AB1F.exe** – A copy of the malware that will execute every time the user account is logged into.
- **%APPDATA%\27DFA2\20AB1F.hdb** – A database of hashes for data that has already been exfiltrated to the C2 server.
- **%APPDATA%\27DFA2\20AB1F.lck** – A lock file created when either decrypting Windows Credentials or Keylogging to prevent resource conflicts.
- **%APPDATA%\27DFA2\20AB1F.kdb** – A database of keylogger data that has yet to be sent to the C2 server.

Identify Registry Key

The path for the specific persistence registry key used is encrypted within the binary using [Triple DES](#) encryption, which is why static analysis wont yield much. Once decrypted, my sample returned the following registry path used for persistence:

“HKEY_LOCAL_MACHINE\ Software\Microsoft\Windows\CurrentVersion\Run”

The registry key within this path is then derived from the Mutex exactly how our %APPDATA% subfolder was:

“HKEY_LOCAL_MACHINE\ Software\Microsoft\Windows\CurrentVersion\Run\27DFA2”

The value assigned to this key is the executable that is stored within the %APPDATA% subfolder:

“%APPDATA%\27DFA2\20AB1F.exe”

Conclusion

That pretty much covers all artifacts related to Loki-Bot that could be present on a compromised system. First step is to identify your system’s Machine GUID. Once you do that, MD5 hash and then trim that value. The result will help you identify all the different folders, files, and registry keys associated with the malware.