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

x r3mrum.wordpress.com/2017/05/07/loki-bot-atrifacts/

View all posts by R3MRUM

May 7, 2017



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 <u>SANS DFIR Blog</u>:

"Programs use <u>mutex</u> ("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



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 <u>RegOpenKeyEx</u>. At 0x404A81, we see the decoded function ADVAPI32.<u>RegOpenKeyEx</u> being called.

This will open the registry path:

"HKEY_LOCAL_MACHINE\SORTWARE\Microsoft\Cryptograpy\"

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

00404A8A 00404A8C 00404A91 00404A93 00404A98 00404A98 00404A98 00404A98 00404A98 00404A98 00404A98 00404A98	- 53 - 68 1A669FFE - 64 09 - 68 4067FFF - 8040 F8 - 51 - 57 - 53 - 53 - 55 - FFD0	PUSH EBX PUSH EEX PUSH FEPF661A PUSH PEPF661A PUSH ECX. PUSH ECX. PUSH EBX PUSH EBX PUSH EBX PUSH EBX PUSH EBX PUSH SEI CALL 6AX	Arg3 Arg2 = FE9F661A — Hash representing RegQ Arg1 = 9 — Index representing ADVAPI32 FE62c1c28BcF41CA826AA267F5AA6F7.getDL1 Arguments passed to RegQueryValueEx	eryValueEx FunctionFromIDXAndHash
EAX=760A	9480 (ADVAPI32.R	egqueryvalueExA)		
FE62C1C2	83CF41CA826AA267	FSAA6F7.getMachineGUIDFromRegist	try+6A	Handle to open key obtained via RegOpenKeyEx
Address	Hex dump		ASCII ASCII ODDEEDA 00000000	hkey = [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\cryptography]
00292388 00292398 00292388 00292388	CO CO<	00 00<	0013FEB8 004162A8 0013FEB7 00000000 0013FEC0 0000000 0013FEC4 00292388 0013FEC4 00292388	Reserved = 0 pType = NRL pData = 00292388 > 00 ptype = NRL = 520.

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\Cryptography\MachineGuid registry key.

Address	He	x dı	ump														AS	CI	Ι					
00292388	31	39	39	66	62	36	37	33	2D	39	34	62	31	2D	34	63	19	9f	b6	73	3-9)4b)1 -	4c
00292398	61	63	2D	62	31	31	62	2D	65	65	33	63	38	63	64	31	ac	-b	11	b-	ee	230	:8c	d1
002923A8	32	39	63	65	00	00	62	00	31	00	31	00	62	00	2D	00	29	ce		b	1	1	b	-
002923B8	65	00	65	00	33	00	63	00	38	00	63	00	64	00	31	00	e	e	3	С	8	С	d	1
002923C8	32	00	39	00	63	00	65	00	00	00	00	00	00	00	00	00	2	9	С	е				

We can validate this by simply loading up <u>RegEdit</u> on the Windows host that is about to be compromised and navigating to the referenced registry key.



The Machine GUID is supposed to be a value that is unique for each system. This means that your Machine GUID will be different from the Machine GUId depicted here; thus, your mutex will be different from mine.

MD5 Hash Machine GUID

Once the Machine GUID is obtained from the registry, Loki-Bot obtains the MD5 hash of the Machine GUID by making calls to ADVAPI's <u>CryptAcquireContext</u>, <u>CryptCreateHash</u>, <u>CryptHashData</u>, and <u>CryptGetHashParam</u>.

00403874	· ·	56	PUSH ESI	rarg4
00402975		E.E.	DUCH CCT	4507
00403873	· ·	30	PUSH EST	Args
00403876	· ·	68 D10741D3	PUSH D34107D1	I Ang2 = D34107D1 — Hash representing CryptAcquireContext
00403978		64 00	DUCH O	Angl - O leden and a ADMANIA
0040387B	· ·	0A 09	PUSH 9	Argi = 9—Index representing ADVAPI32
0040387D	· ·	E8 63F9FFFF	CALL getDLLFunctionFromIDXAnd	LFE62C1C283CF41CA826AA267F5AA6F7.getDLLFunctionFromIDXAndHash
00403883		68 00000000	BUIEU 6000000	
00403882	· ·	68 00000F0	PUSH FOODOOO	
00403887	Ι.	6A 01	PUSH 1	
00403007	L .	56		
00403889	· ·	56	PUSH ESI	
00403884	Ι.	56	DUEN EET	
00403004	L .	30	FUSH EST	
00403888	· ·	8D4D F8	LEA ECX, [LOCAL.2]	
00402995	L .	51	DUCH CCV	
0040300E	L .	31	PUSH ECX	
0040388F	· ·	FFDO	CALL EAX	ADVAPI32.CrvptAcquireContext
00402801		8500	TEET EAV EAV	
00403891	· ·	0000	TEST EAX, EAX	
00403893	I	OF84 8400000	17 00403910	
00403000	× I	0075 00	HOLL FET DUDDD DTD FET LOCUL D	4
00403899	· ·	88/3 F8	MOV ESI, DWORD PTR 55: [LOCAL.2	
0040389c	Ι.	64 00	PUSH 0	raro4 = 0
00103030	L .	00 00		
0040389E	· ·	6A 00	PUSM 0	Arg3 = 0
00403840	Ι.	68 E34680ED	DUSH EDROAGE3	Anno = EDBOA6E3 - Hash representing CryptCreateHash
00403040	L .	CO PORODOLD	FUSH EDUDADES	A ge - cooders - there are a get the second second
004038A5		6A 09	PUSH 9	AFg1 = 9 - index representing ADVAPI32
00403847		E8 30E0EEEE	CALL Get DLL EugetionEcomTDVAnd	EE62C1C283CE41CA826A4267E5A46E7. getDLLEunctionEconTDVAndHash
00403047	L .	EO SSESFFEF	CALL GEODELFUNCTIONFLOMILD/AND	Gredzelezoser+leadzoaazo/risaadr/.getbeeruneeruneeruneerune
004038AC		8D4D FC	LEA ECX, [LOCAL.1]	
00403845		51	DUCH ECY	
004038AF	· ·	21	PUSH ECX	
00403860	· ·	6A 00	PUSH 0	
00402882		64 00	DUEN O	
00403662	· ·	GA UU	PUSH U	
00403884	I .	68 03800000	PUSH 8003 - ALG ID	
00403880		66	DUCH CCT	
00403869	· ·	20	PUSH ESI	
004038BA	I .	FFDO	CALL EAX	ADVAPI32.CrvptCreateHash
00403886		8500	TEET EAV EAV	norro aparte gran encontration
004038BC	· ·	8500	TEST EAX, EAX	
004038BE	I	74 51	17 SHORT 00403911	
00103002	- Y			
004038C0	· ·	6A 00	PUSH 0	Γ Arg4 = 0
00403802	Ι.	53	DUSH ERV	4603
00403062	L .	33	FUSH EDA	
004038C3	· ·	FF75 08	PUSH DWORD PTR SS:[ARG.1]	[] Arg2 => [ARG.1] — Machine GUID
00402866	L .	EE75 EC	DUSH DWORD DTD SSILLOCAL 11	Ardi -> LOCAL 1]
00403000	L .	FF/J FC	PUSH DWORD PIK 55.[LUCAL.1]	A gi => [LOCAL.1]
004038C9	· ·	E8 98FDFFFF	CALL CryptHashData	LFE62C1C283CF41CA826AA267F5AA6F7.crvptHashData
00402965		9264 10	ADD CCD 10	
004030CE	· ·	0304 10	ADD ESP, IO	
004038D1	· ·	85C0	TEST EAX.EAX	
00402802		74 30	17 SHORT 00402011	
00403003	* ¥	/ 4 3%	37 BHOKT 00403911	1
004038p5		8875 FC	MOV ESI, DWORD PTR 55: LOCAL, 1	1
00402808		64 00	DUCH A	Tearad - 0
00403608	· ·	04 00	PUSH 0	I FAI 94 = 0
004038DA		6A 00	PUSH 0	Ard3 = 0
00402805		69 EDDRAGEE	DUCH FEASOPED	Ana2 - FEASDED - Hack concepting CountCatHackParam
004036DC	· ·	00 FUUDAOFE	PUSH FEABUBED	I Ar gz = FEAODED - hash representing cryptoethashraram
004038E1		6A 09	PUSH 9	Arg1 = 9—Index representing ADVAPI32
00403953			CALL GOT DULL FUNCTION FOR CONTRACTOR	Frédeledente Allender (1997) and the free free free free free free free fr
004030E3	· ·	EO FUFOFFFF	CALL GEODELFUNCTIONFROMIDICAND	=FE02C1C203CF41CA020AA20/F3AA0F/.getULLFUNCt10NFF0m1DXANDMaSN
004038F8		6A 00	PUSH 0	
00403851		0040 54	LEA DOV FLOCAL 31	
004038EA		0040 14	LEA ECA, [LOCAL, 5]	
004038ED		51	PUSH ECX	
00403855	L .	6.7	BUCH CDT - Address where the second	ting MDE back of the Machine CUID will be stored
004038EE		37	PUSH EDI — Address where the result	ping Muo nash of the Machine GUID will be stored
004038EE		6A 02	PUSH 2	-
00403064	- T	5.0		
004038F1		20	PUSH ESI	
004038E2		FEDO	CALL EAX	ADVAPT32. CryptGetHashParam
00403054		0500		norre aseres predecidatinen an
004038F4		8500	TEST EAX, EAX	
00403866	1.1.1	74 19	17 SHORT 00403911	
00403050				- Inst - Freeze 13
004038F8		FF/5 FC	PUSH DWORD PTR 55:[LOCAL.1]	[rargi => [LOCAL.1]
004038FB		E8 2EEDEEEE	CALL COVIDESTCOVHASH	FE62C1C283CE41C482644267E5446E7, cryptDestroyHash
00403010	L .	C. OO	chile of ypercent oynash	- concare over the owned of a protect of the owned of the
00403900		6A 00	PUSH 0	
00403902		EE75 E8	PUSH DWORD PTP SS: [LOCAL 2]	
00403902	L .	FF73 F0	FUSH DWORD FIR SSILLOCALIZ	
00403905		E8 3FFDFFFF	CALL CryptReleaseContext	

After <u>CryptGetHashParam</u> executes, the MD5 hash of the Machine GUID is returned.

0040388 0040388 0040388	2 • FFD0 4 • 85C0 6 • 74 19 8 • FE75 FC	CALL EAX TEST EAX, EAX — Just executed Cr JZ SHORT 00403911 PUSH DWORD PTR SS:[LOCAL_1]	ADVAPI32.CryptGetHashParam
EAX=000 FE62C10	00001 283CF41CA826AA267F	5AA6F7.getMD5Hash+0CA	
Address	Hex dump		ASCII ASCII
0020270	8 98 D0 BA 52 7D F	A 20 AB 1F 4A 05 B8 D0 D4 E0	4B ∎аR}ú « J∣,ĐÔàK 0013FE 0013FE

The MD5 hash of our Machine GUID appears to be

"9BD0BA527DFA20AB1F4A05B8D0D4E04B". There are a number of different ways that we could validate this result but I find that it's easiest using the linux command line.

remnux@remnux: ~	- 0	×
ile <u>E</u> dit <u>T</u> abs <u>H</u> elp		
emnux@remnux:~\$ echo -n '199fb673-94b1-4cac-b11b-ee3c8cd129ce' md5sum tr [a-z] BD0BA527DFA20AB1F4A05B8D0D4E04B -	[A-Z]	
emnux@remnux:~\$		~

Trim Hash & Create Mutex

Finally, Loki-Bot trims the MD5 hash of the Machine GUID to 24-characters: "9BD0BA527DFA20AB1F4A05B8".

00414185 • • • • • • • • • • • • • • • • • • •
00414188 • 53 PUSH EBX AF 94 00414189 • 53 PUSH EBX AF 93 0041418F • 68 F47D16CF PUSH CF167DF4 AF 93 0041418F • 53 PUSH EBX AF 93 0041418F • 53 PUSH EBX AF 93 0041418F • 88F0 MOV ES1 EAX AF 91 — EBX is 0, the index value for Kernel32
00414189 • 53 PUSH_EBX Ar 03 00414184 • 68 F47D16CF PUSH_EBX Ar 03 00414184 • 68 F47D16CF PUSH_EBX Ar 03 00414186 • 53 PUSH_EBX Ar 02 00414100 • 88F0 MOV_EST, EAX — Move MD5 hash from EAX to ESI Ar 01 — EBX is 0, the index value for Kernel32
0041418A 68 F47D16CF PUSH CF167DF4 AF 02 CF167DF4—Hash referring to CreateMutex 0041418F 53 PUSH EBX AF 01—EBX is 0, the index value for Kernel32 0041410 88F0 MOV ES1, EAX — Move MD5 hash from EAX to ESI AF 01—EBX is 0, the index value for Kernel32
0041418F · 53 PUSH EBX 004141C0 · 88F0 MOV EST, EAX — Move MD5 hash from EAX to ESI
00434100 00434100 88F0 Nov ESI EAX — Move MD5 hash from EAX to ESI
004141C01 · 88F0 MOV EST, EAX — MOVE MUS hash from EAX to EST
004141C2 • E8 1EF0FEFF CALL getDLLFunctionFromIDXAndHash LFE62C1C283CF41CA826A4267F5A46F7.getDLLFunctionFromIDXAndHash
004141C7 • 56 PUSH ESI Set IpName argument of CreateMutex to the 24-character trimmed MD5 hash
004141C8 • 33F6 XOR ESI,ESI
004141CA - 46 INC EST
004141C8 . 56 PUSH EST
CALL LAX KERNELSZ CFEATEMUTEXW
004141CF • FF15_1060410(CALL DWORD PTR DS: dkennel.32.getLasterror
004141D5 · 3D 87000000 CMP EAX,087 CONST 87 => ERROR_ALREADY_EXISTS
004141DA 75 07 JNE SHORT 004141E3 Jump if mutex does not exist
004141pc - 53 PUSH EBX CAPOL
00414100 . E8 00010000 CALL avit Process I it does
VOLATES - 39 FOR ECX
U04141E3 > E8 4CF0FFFF CALL MINEANOSTERIDATA
EAX=755E44D0 (KERNEL32.CreateMutexw) - jumps to KERNELBASE.CreateMutexw
FE62C1C283CF41CA826AA267F5AA6F7.main+136
Address Hex dump ASCII A 00131608 0000000 PSecurity = NULL
D0183650 30 00 42 00 44 00 30 00 42 00 41 00 35 00 32 00 8 80 0 8 4 5 2 0003FF0C 00000001 101tialowner = TRUE
00183650 22 00 44 00 46 00 41 00 32 00 30 00 41 00 42 00 2 D E A 2 0 A B 0013FF10 00183650 P6: Name = "98D08A5270FA20A81F4A0588"

It then passes this trimmed value to Kernel32's <u>CreateMutexW</u> function as the *IpName* 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.

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



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 <u>Triple DES</u> 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.