LokiBot is Being Distributed by Windows Shortcut Files

Published: 2024-03-13 · Archived: 2026-04-05 18:01:21 UTC

Overview

The SonicWall RTDMI ™ engine has recently detected Windows Shortcut Files (LNKs) inside archives that execute LokiBot malware on the victim's machine. The malicious LNK file is packed inside an archive along with a text file that says, "Find attached March Order" in Spanish, essentially pretending to be a legitimate file. The LNK file executes a PowerShell script to download and execute the LokiBot loader executable from a URL. LokiBot malware has been observed using image steganography, multi-layered packing and Living Off the Land (LOTL) techniques in past campaigns. The malware authors' trend of using a low-profile file type as an initial vector keeps increasing, and they prefer the use of custom packers and protectors to prevent detection rather than updating the core executable code.

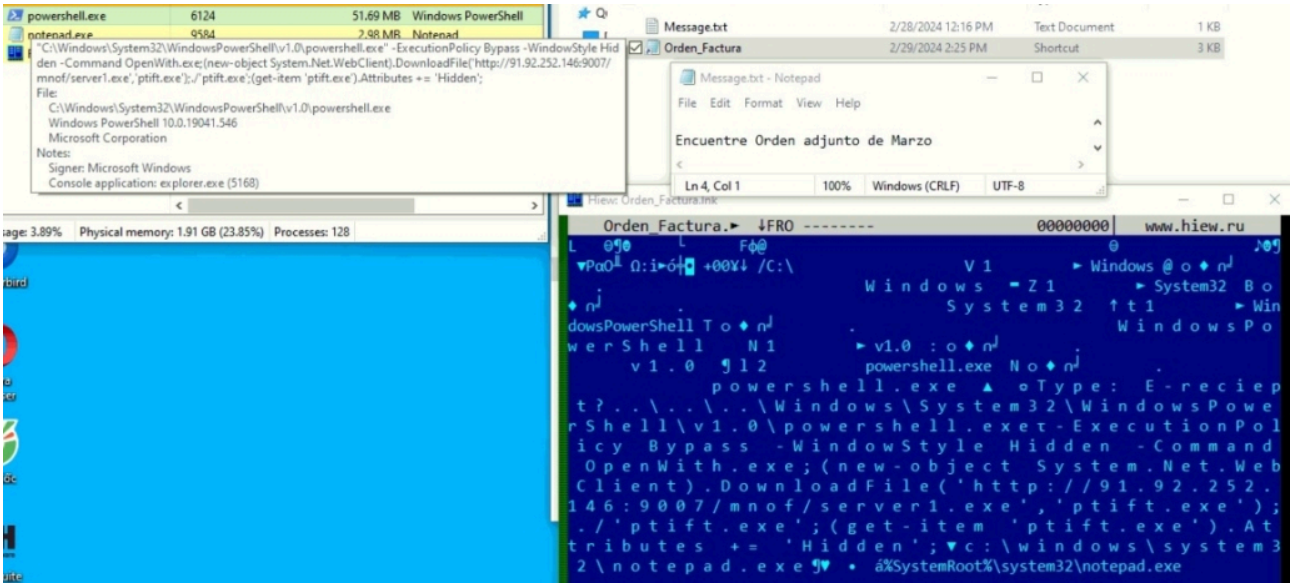
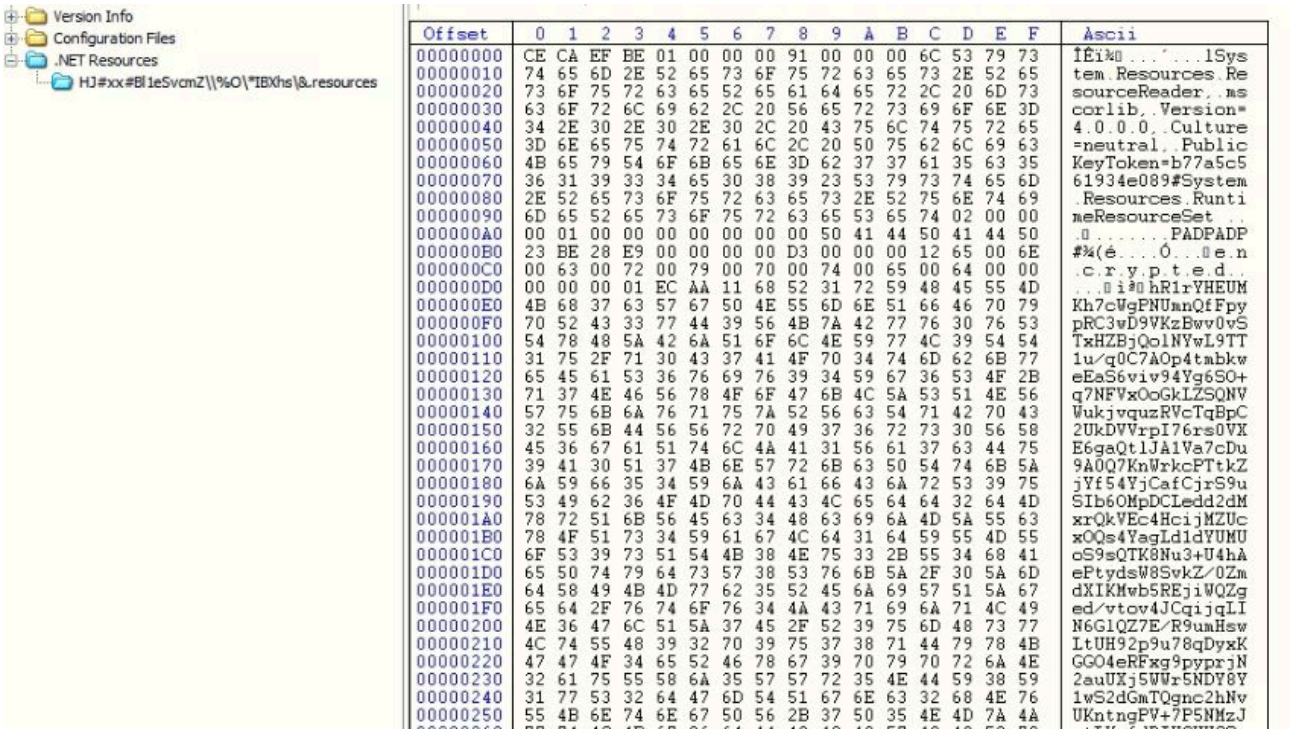


Figure 1: LNK file executing PowerShell to download LokiBot

Loader Executable

The downloaded executable is protected in two layers by the Confuser application to make the analysis difficult for the reversing engineers. After unpacking and de-obfuscating both of the layers, the actual code that is responsible for loading and executing the LokiBot executable is exposed. The LokiBot binary is kept RSA-encrypted and Base64-encoded in the resource of the loader executable, which is only visible after unpacking.



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	CE	CA	EF	BE	01	00	00	00	91	00	00	00	6C	53	79	73	ÎÊï¾ . . . ' . . . lSys
00000010	74	65	6D	2E	52	65	73	6F	75	72	63	65	73	2E	52	65	tem.Resources.Re
00000020	73	6F	75	72	63	65	52	65	61	64	65	72	2C	20	6D	73	sourceReader, ms
00000030	63	6F	72	6C	69	62	2C	20	56	65	72	73	69	6F	6E	3D	corlib, Version=
00000040	34	2E	30	2E	30	2E	30	2C	20	43	75	6C	74	75	72	65	4.0.0.0, Culture
00000050	3D	6E	65	75	74	72	61	6C	2C	20	50	75	62	6C	69	63	=neutral, Public
00000060	4B	65	79	54	6F	6B	65	6E	3D	62	37	37	61	35	63	35	KeyToken=b77a5c5
00000070	36	31	39	33	34	65	30	38	39	23	53	79	73	74	65	6D	61934e089#System
00000080	2E	52	65	73	6F	75	72	63	65	73	2E	52	75	6E	74	69	.Resources.Runti
00000090	6D	65	52	65	73	6F	75	72	63	65	53	65	74	02	00	00	meResourceSet ..
000000A0	00	01	00	00	00	00	00	00	00	50	41	44	50	41	44	50	. . . . . PADPADP
000000B0	23	BE	28	E9	00	00	00	00	D3	00	00	00	12	65	00	6E	#¾(é . . . . Ó . . . e . n
000000C0	00	63	00	72	00	79	00	70	00	74	00	65	00	64	00	00	. c . r . y . p . t . e . d . .
000000D0	00	00	00	01	EC	AA	11	68	52	31	72	59	48	45	55	4D	. . . . ì ª . hR1rYHEUM
000000E0	4B	68	37	63	57	67	50	4E	55	6D	6E	51	66	46	70	79	Kh7cWgPNUmnQfFpy
000000F0	70	52	43	33	77	44	39	56	4B	7A	42	77	76	30	76	53	pRC3wD9VKzBwv0vS
00000100	54	78	48	5A	42	6A	51	6F	6C	4E	59	77	4C	39	54	54	TxHZBjQolNYwL9TT
00000110	31	75	2F	71	30	43	37	41	4F	70	34	74	6D	62	6B	77	1u/q0C7AOp4tmbkw
00000120	65	45	61	53	36	76	69	76	39	34	59	67	36	53	4F	2B	eEaS6viv94Yg6SO+
00000130	71	37	4E	46	56	78	4F	6F	47	6B	4C	5A	53	51	4E	56	q7NFVxOoGkLZSQNV
00000140	57	75	6B	6A	76	71	75	7A	52	56	63	54	71	42	70	43	WukjvquzRVcTqBpC
00000150	32	55	6B	44	56	56	72	70	49	37	36	72	73	30	56	58	2UkDVVrpI76rs0VX
00000160	45	36	67	61	51	74	6C	4A	41	31	56	61	37	63	44	75	E6gaQtlJA1Va7cDu
00000170	39	41	30	51	37	4B	6E	57	72	6B	63	50	54	74	6B	5A	9A0Q7KnWrkcPTtkZ
00000180	6A	59	66	35	34	59	6A	43	61	66	43	6A	72	53	39	75	jYf54YjCafCjrS9u
00000190	53	49	62	36	4F	4D	70	44	43	4C	65	64	64	32	64	4D	SIb6OMpDCLedd2dM
000001A0	78	72	51	6B	56	45	63	34	48	63	69	6A	4D	5A	55	63	xrQkVEc4HcijMZUc
000001B0	78	4F	51	73	34	59	61	67	4C	64	31	64	59	55	4D	55	xOQs4YagLd1dYUMU
000001C0	6F	53	39	73	51	54	4B	38	4E	75	33	2B	55	34	68	41	oS9sQTK8Nu3+U4hA
000001D0	65	50	74	79	64	73	57	38	53	76	6B	5A	2F	30	5A	6D	ePtydsW8SvkZ/0Zm
000001E0	64	58	49	4B	4D	77	62	35	52	45	6A	69	57	51	5A	67	dXIKMwb5REjiWQZg
000001F0	65	64	2F	76	74	6F	76	34	4A	43	71	69	6A	71	4C	49	ed/vtov4JCqijqLI
00000200	4E	36	47	6C	51	5A	37	45	2F	52	39	75	6D	48	73	77	N6GlQZ7E/R9umHsw
00000210	4C	74	55	48	39	32	70	39	75	37	38	71	44	79	78	4B	LtUH92p9u78qDyxK
00000220	47	47	4F	34	65	52	46	78	67	39	70	79	70	72	6A	4E	GGO4eRFxg9pyprjN
00000230	32	61	75	55	58	6A	35	57	57	72	35	4E	44	59	38	59	2auUXj5WWr5NDY8Y
00000240	31	77	53	32	64	47	6D	54	51	67	6E	63	32	68	4E	76	1wS2dGmTQgnc2hNv
00000250	55	4B	6E	74	6E	67	50	56	2B	37	50	35	4E	4D	7A	4A	UKntngPV+7P5NMzJ

Figure 2: Encrypted LokiBot executable in the resource of loader executable

To get the RSA key, the malware computes MD5 of the string “UV?vgotlHR?P\Y?LEhgU]CdJIS?W;yBTkDZw??FTEi>Z” and self-concatenates the MD5 value by overwriting the last byte of the MD5. After doing Base64 decoding, the malware decrypts the encrypted executable data using an RSA algorithm with the below parameters:

Initial Vector	:	80 1E 8D 99 B9 3A 9F D2 DA 12 DD 0B 24 D2 6E B1
Key	:	1F C7 9C 09 21 5E 5E A8 52 0B 8C 2A CA 92 ED 1F C7 9C 09 21 5E 5E A8 52 0B 8C 2A CA 92 ED 5D 00
Key Size	:	256
Mode	:	ECB
Padding	:	PKCS7

Figure 3: Parameters

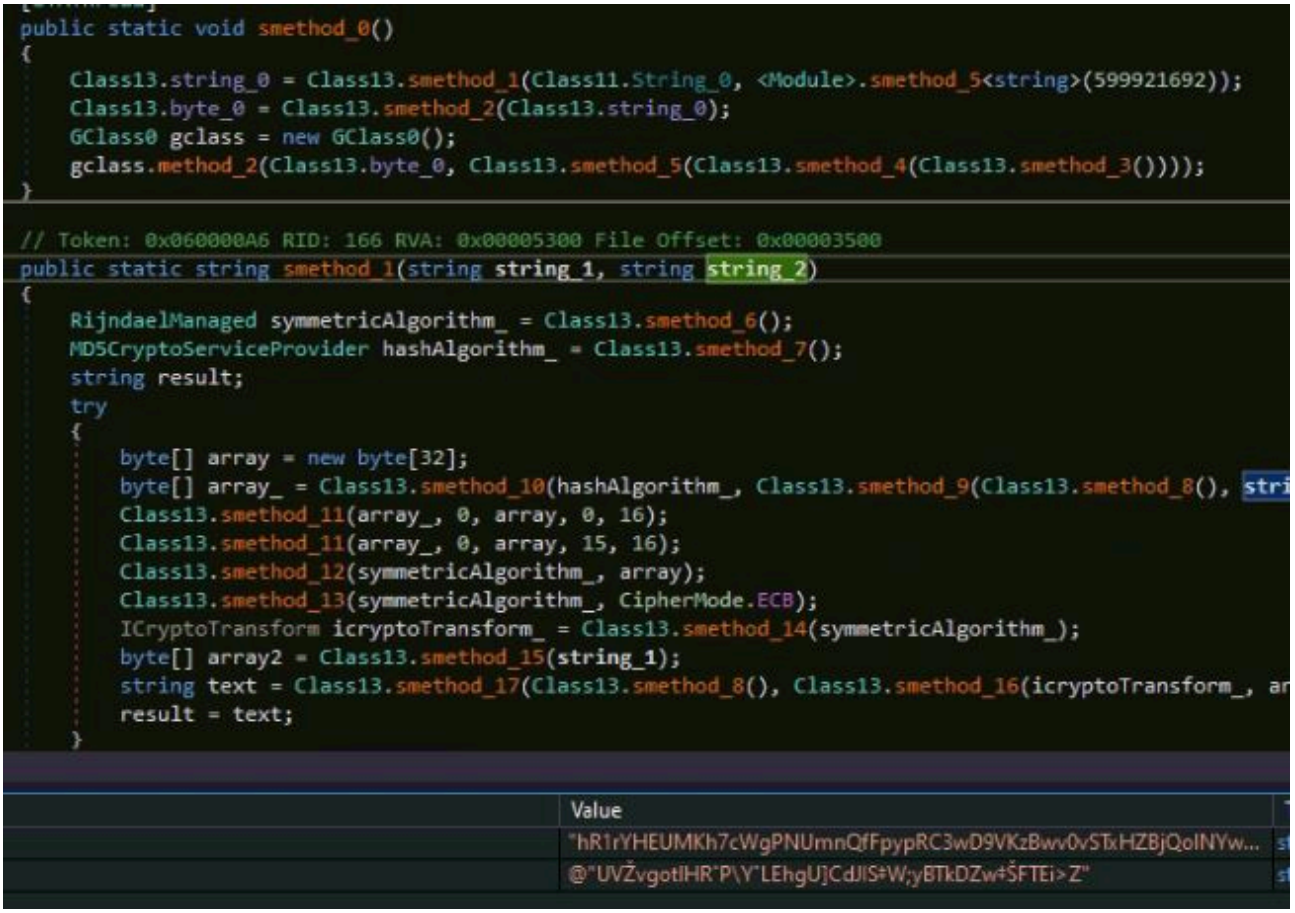


Figure 4: Code performing RSA decryption

The decrypted bytes are the string representation of the executable's hex bytes, and a string-to-hex conversion is performed to finally get the LokiBot executable. The malware executes the LokiBot binary using the CreateProcessW API.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	34	44	35	41	39	30	30	30	30	33	30	30	30	30	30	30	4D5A900003000000
00000010	30	34	30	30	30	30	30	30	46	46	46	46	30	30	30	30	04000000FFFF0000
00000020	42	38	30	30	30	30	30	30	30	30	30	30	30	30	30	30	B8000000000000000
00000030	34	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	4000000000000000
00000040	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
00000050	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
00000060	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0000000000000000
00000070	30	30	30	30	30	30	30	30	46	30	30	30	30	30	30	30	00000000F0000000
00000080	30	45	31	46	42	41	30	45	30	30	42	34	30	39	43	44	0E1FBA0E00B409CD
00000090	32	31	42	38	30	31	34	43	43	44	32	31	35	34	36	38	21B8014CCD215468
000000A0	36	39	37	33	32	30	37	30	37	32	36	46	36	37	37	32	69732070726F6772
000000B0	36	31	36	44	32	30	36	33	36	31	36	45	36	45	36	46	616D2063616E6E6F
000000C0	37	34	32	30	36	32	36	35	32	30	37	32	37	35	36	45	742062652072756E
000000D0	32	30	36	39	36	45	32	30	34	34	34	46	35	33	32	30	20696E20444F5320
000000E0	36	44	36	46	36	34	36	35	32	45	30	44	30	44	30	41	6D6F64652E0D0D0A
000000F0	32	34	30	30	30	30	30	30	30	30	30	30	30	30	30	30	2400000000000000
00000100	43	43	43	44	37	38	46	45	38	38	41	43	31	36	41	44	CCCD78FE88AC16AD

Figure 5: LokiBot binary string representation

LokiBot

LokiBot is an information stealer that has been active in the wild since 2015. It contains a rich list of applications that are used to steal data from the victim's machine. After sending data to its C&C server, the malware receives commands to perform various actions on the victim's machine.

Delayed Execution

The execution of LokiBot starts by examining the argument value for space-separated "-u" occurrences. The malware execution has been delayed by 10000 milliseconds for each occurrence of the value "-u".

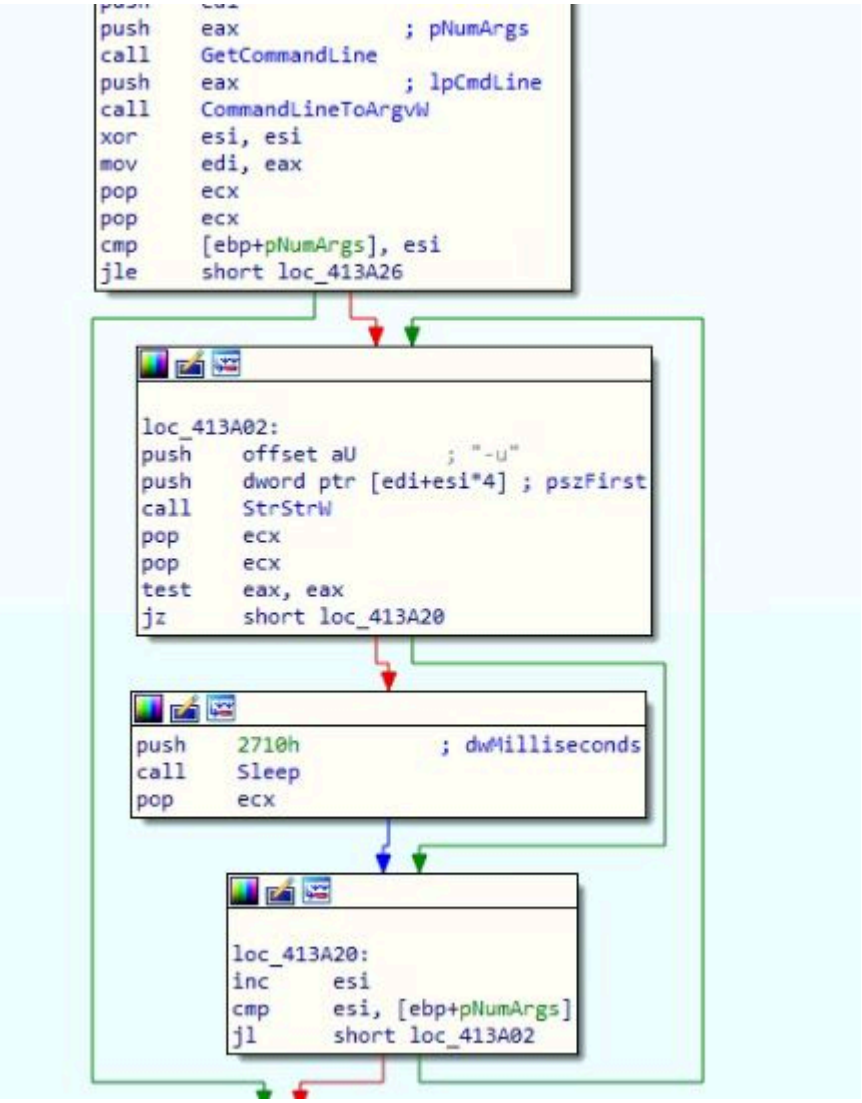


Figure 6: Checks process arguments

API Name Hashing

The malware uses API name hashing and resolves addresses just before invoking the APIs. To make the analysis difficult, the malware does not store the resolved API addresses, instead it loads and executes the API address from the accumulator register. If the malware needs to execute the same API more than once, it needs to resolve the API address again to invoke the API.

Hashing Algorithm

Code representation for the hashing module is in the C language and can be seen below. It generates a DWORD hash for the given array of bytes.

```
DWORD ComputeHash(BYTE* pValue, int iLength)
{
    DWORD dwHash;
    int iIterations = 8; // eax

    dwHash = -1;
    for (int i = 0; i < iLength; i++)
    {
        dwHash = dwHash ^ pValue[i];
        iIterations = 8;
        for (int j = 0; j < iIterations; j++)
        {
            if (0 != (dwHash & 1))
            {
                dwHash = dwHash ^ 0x4358AD54u;
            }
            dwHash = dwHash >> 1;
        }
    }
    dwHash = ~dwHash;
    return dwHash;
}
```

Figure 7: Hashing algorithm

Getting the Load Addresses of DLLs

An array of DLL names with a fixed length of 0x1A for each name is created by the malware. To resolve an API address, the malware calls a function with the index of the DLL names array and API name hash. The name of the DLL is retrieved from the DLL names array using the index value. The malware invokes the *LoadLibraryW* API to get the load address of the DLL, except for the *kernel32* and *ntdll* DLLs. The malware considers *kernel32* and *ntdll* to be already loaded into the memory and does PEB traversal to find the load address. It does this by comparing the DLL name hash with the loaded module name hashes.

▲ sModuleNames	0x0000007f6451f730 {0x0000007f6451f730 "kernel32", 0x0000007f6451f74a "ntdll...		char[0x0000000f][0x0000001a]
▷ [0x00000000]	0x0000007f6451f730 "kernel32"	View	char[0x0000001a]
▷ [0x00000001]	0x0000007f6451f74a "ntdll"	View	char[0x0000001a]
▷ [0x00000002]	0x0000007f6451f764 "shlwapi"	View	char[0x0000001a]
▷ [0x00000003]	0x0000007f6451f77e "CRYPT32"	View	char[0x0000001a]
▷ [0x00000004]	0x0000007f6451f798 "WININET"	View	char[0x0000001a]
▷ [0x00000005]	0x0000007f6451f7b2 "urlmon"	View	char[0x0000001a]
▷ [0x00000006]	0x0000007f6451f7cc "NETAPI32"	View	char[0x0000001a]
▷ [0x00000007]	0x0000007f6451f7e6 "WS2_32"	View	char[0x0000001a]
▷ [0x00000008]	0x0000007f6451f800 "user32"	View	char[0x0000001a]
▷ [0x00000009]	0x0000007f6451f81a "ADVAPI32"	View	char[0x0000001a]
▷ [0x0000000a]	0x0000007f6451f834 "SHELL32"	View	char[0x0000001a]
▷ [0x0000000b]	0x0000007f6451f84e "gdiplus"	View	char[0x0000001a]
▷ [0x0000000c]	0x0000007f6451f868 "gdi32"	View	char[0x0000001a]
▷ [0x0000000d]	0x0000007f6451f882 "ole32"	View	char[0x0000001a]
▷ [0x0000000e]	0x0000007f6451f89c "gdi32"	View	char[0x0000001a]

Figure 8: Array of DLL names

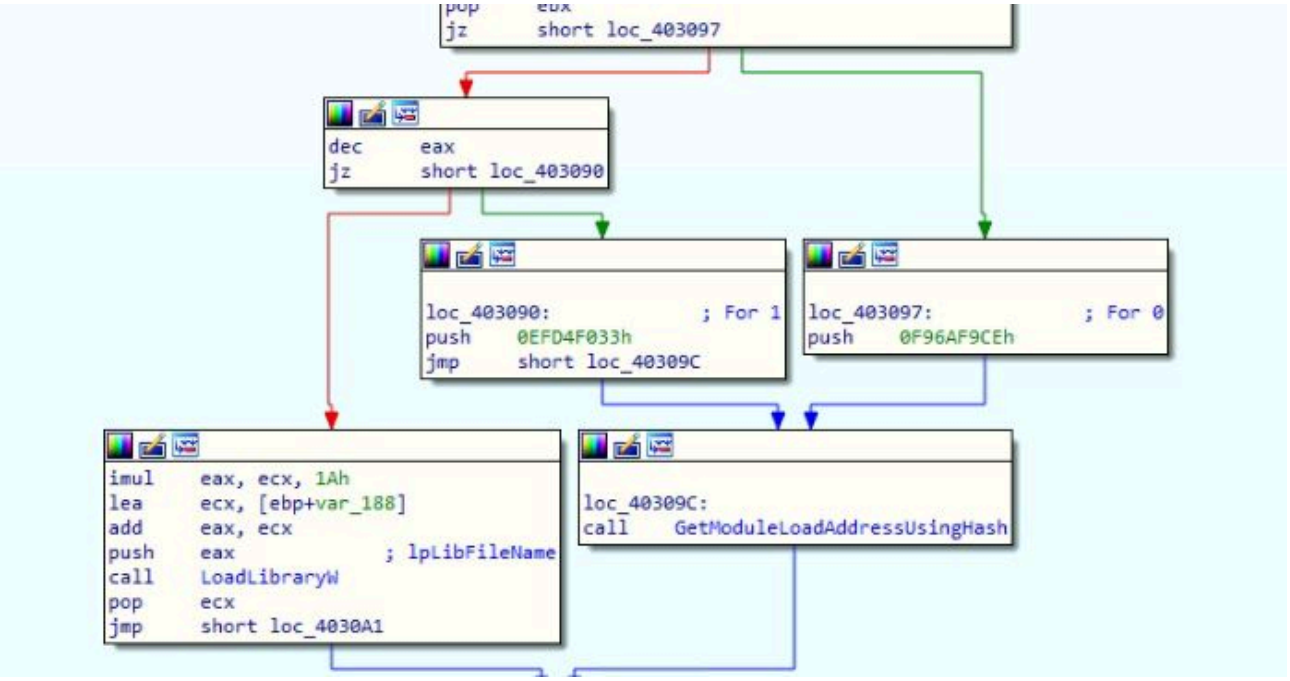


Figure 9: Gets DLL load address

Resolving APIs Addresses

After getting the DLL load address either by calling the *LoadLibraryW* API or by PEB traversal, the malware enumerates the export directory of the loaded DLL and compares the requested API name hash with the exported API name hash to resolve the API address.

Single Instance Execution

The malware retrieves the *MachineGUID* from the registry entry "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography" to compute its MD5 string representation to the 12th byte, which is used as a mutex name. The malware attempts to create the mutex and examines the error value ERROR_ALREADY_EXISTS to terminate the current execution and ensure a single execution of the malware process. The mutex value for my system is "06E1A66DB87D112F02F38F7C" and is expected to always be the same for a given system. It is later sent to the Command and Control (C&C) server. The mutex name can help the malware author identify whether a system is reinfected or has been infected for the first time. Some of the character sequences of the mutex name are used by the malware as dropped file names and a directory name.

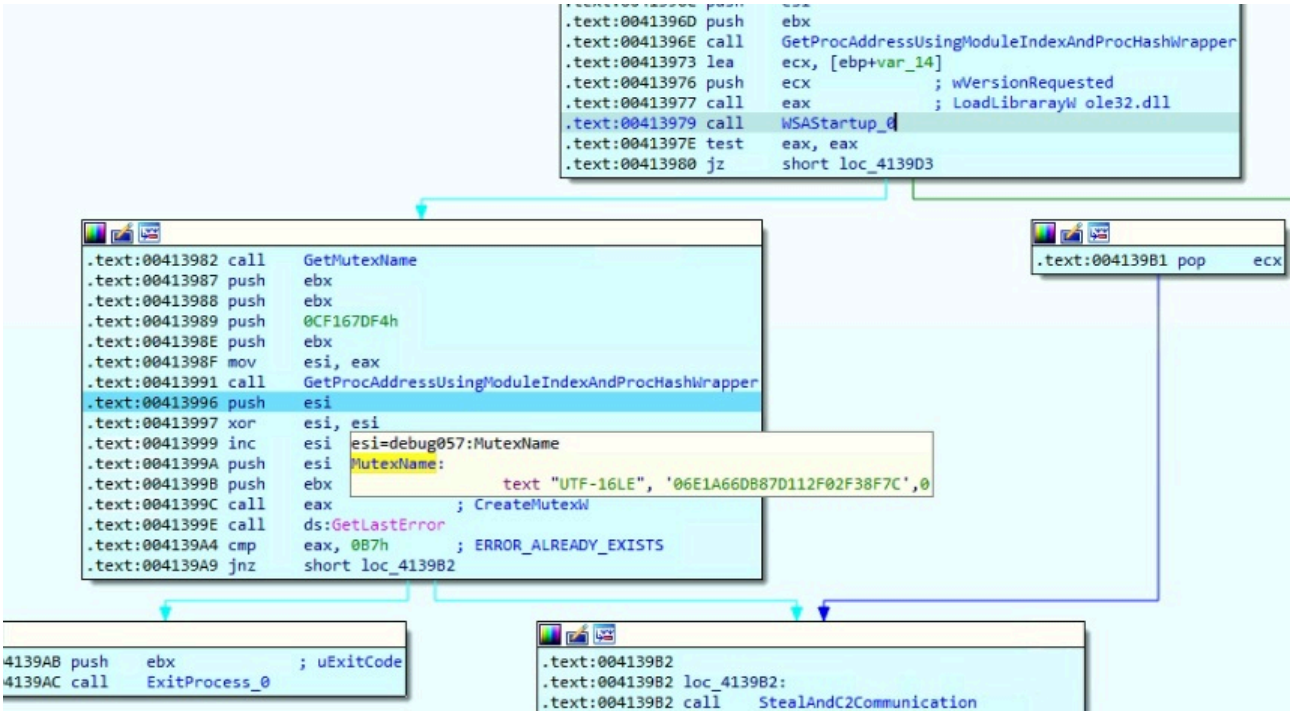


Figure 10: Creates mutex using machine GUID

If the malware fails to get the mutex name using *MachineGUID*, then it executes its backup code to compute a random mutex name using the system time.

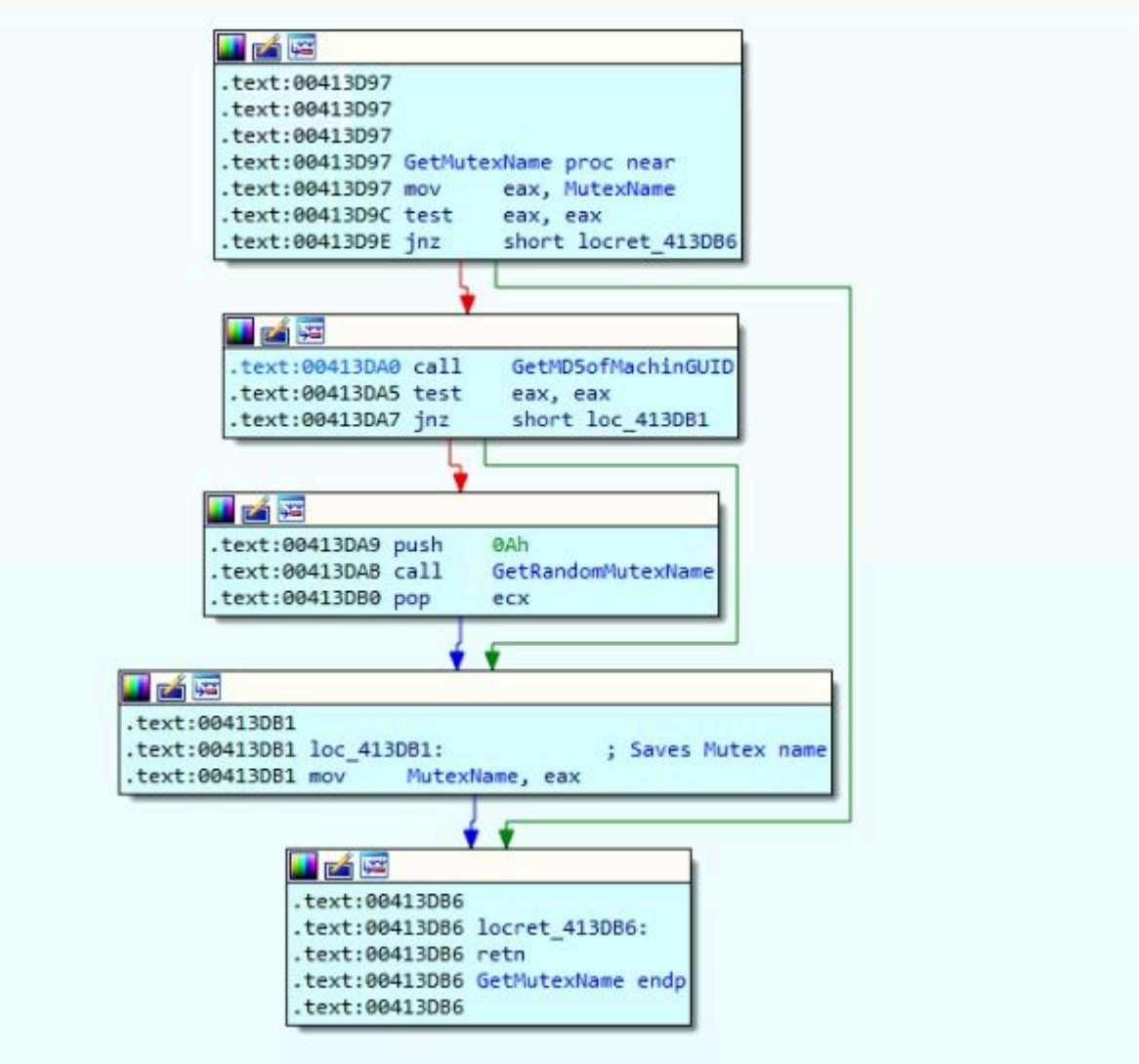


Figure 11: Code to create an alternative random mutex

Stealing Application Data

The malware steals data from installed applications like browsers, SSH clients, document applications, password managers, email clients and FTP clients The data it steals includes login credentials, autofill web forms, document texts and more. The malware contains a total of 101 functions to steal data from the installed applications. The malware initializes the array of function pointers with the address of stealing functions and initializes another array with the respective argument values for the stealing functions. The malware invokes the stealing functions in a loop. It invokes them from the array of function pointers with the respective argument from the arguments array. The stealer functions keep the stolen data in a structured buffer along with their sizes.

Figure 12: Calling stealing functions

Browsers

The malware contains a list of chromium-based browsers, gecko-based browsers and other browsers to steal the stored data. For a gecko-based browser, malware reads the registry entry for the respective browser to get the installation directory and sets the installation directory into the environment path variable to smoothly load the nss3.dll which is used to decrypt the gecko-based browsers data. After decrypting the data, malware restores the environment path variable. The malware reads the profiles.ini file from the browser dedicated application data directory to get the list of profile directories. The malware enumerates each profile directory and uses multiple APIs (*NSS_Init, GetInternalKeySlot, Authenticate, Decrypt, FreeSlot, FreeItem, CheckUserPassword, NSS_Shutdown*) of nss3.dll, to decrypt data from the files mentioned below:

- logins.json
- prefs.js
- signons.sqlite
- signons.txt
- signons2.txt

For chromium-based browsers, the malware steals data from the files “Web Data” and “Login Data” placing them into respective application data folders for the browsers. For Edge and Internet Explorer, the malware steals data from Windows Password Manager and registry entry “Software\Microsoft\Internet Explorer\IntelliForms\Storage2”. The malware enumerates and retrieves credentials from the Windows Password Manager using multiple APIs (VaultEnumerateVaults, VaultOpenVault. VaultEnumerateItems, VaultGetItem, VaultFree, VaultCloseVault) of library vaultcli.dll.

Chromium-based	Gecko-based	Others
Dragon	Mozilla Firefox	Safari
ChromePlus	SeaMonkey	Opera Next
Chrome	Flock	Opera Stable
Nichrome	Black Hawk	QtWeb
RockMelt	Cyberfox	QupZilla
Spark	IceDragon	Internet Explorer
Chromium	K-Meleon	Opera
Titan Browser	Lunascape	
Torch	Pale Moon	
YandexBrowser	Waterfox	
Epic Privacy Browser		
CocCoc		
Vivaldi		
Chromodo		
Superbird		
Coowon		
Mustang Browser		
360Browser		
Citrio		
Chrome SxS		
Orbitum		

Iridium		

Figure 13: List of browsers

Apart from the browsers, the malware steals data from other installed applications which include SSH clients, FTP clients, email applications, document applications, password managers and more, which are mentioned in the table below:

SSH	FTP		Email	Document	Password Manager	Others
SuperPutty	FTPShell	WS_FTP	Mozilla	WordPerfect	KeePass	Pidgin
KiTTY	NppFTP (Notepad++ plugin)	ExpanDrive	Thunderbird	To-Do DeskList	Enpass	Full Tilt Poker
PuTTY	MyFTP	Steed	Foxmail	Stickies	RoboForm	WinChips
	FTPBox	FlashFXP	Pocomail	NoteFly	1Password	PokerStars
	Sherrod FTP	NovaFTP	IncrediMail	Notezilla	mSecure	
	NetSarang XFTP	NetDrive	GmailNotifierPro	Sticky Notes		
	FTP Now	SmartFTP	CheckMail			
	EasyFTP	FarFTPPlugin	SoftwarenetzMailing			
	SFTP Net Drive	Bitvise SSH Client	Opera Mail			
	AbleFTP	VNC software	Postbox			
	JaSFtp	Syncovery	FossaMail			
	Automize	FreshFTP	Mailbox			
	Cyberduck	BitKinex	Outlook			
	FullSync	UltraFXP	yMail			
	FTPInfo	FTP Now	Trojitá			
	LinasFTP	SecureFX	TrulyMail			
	FileZilla	Odin Secure FTP Expert				
	Staff-FTP	Fling				
	BlazeFtp	ClassicFTP				
	Fastream NETFile	WinFtp Client				
	GoFTP	WinSCP				
	ALFTP	32bit FTP				

Figure 14: List of applications to steal data

C&C Communication

The malware has an array to contain four C&C server addresses, however, in current variants, only one of the unique and valid C&C server addresses is found. After decrypting the C&C URL, the malware separates the IP address, port number, path and protocol to establish a connection.

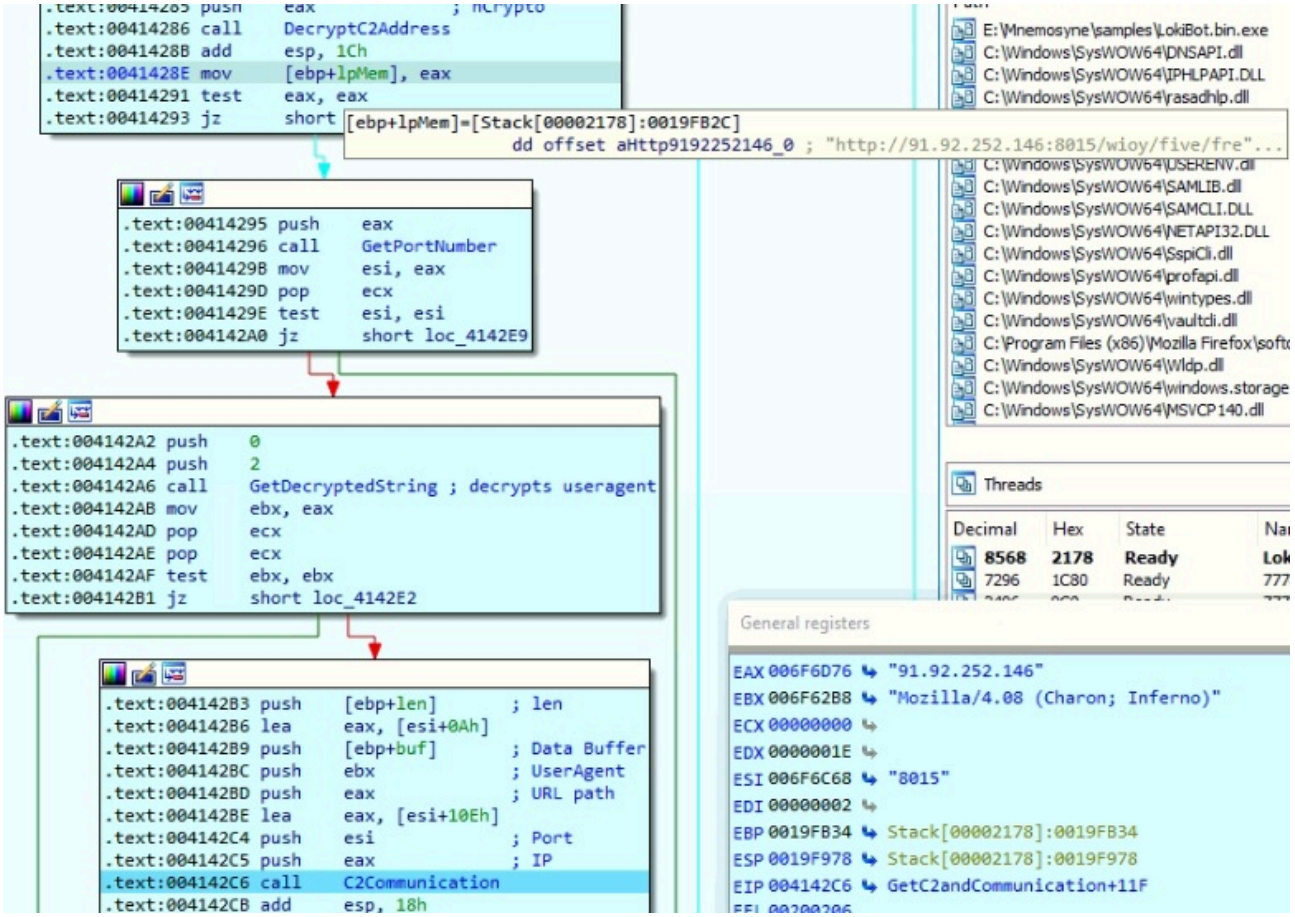


Figure 15: Code for C&C communication

The malware compresses the stolen application data using aPLib compression library and appends into the structured buffer. The malware then prepares the HTTP post request with the user-agent “Mozilla/4.08 (Charon; Inferno)” and sends the stolen data to the C&C server. After sending the HTTP post request, the malware keeps the check sum of the data it sent into a file “C:\Users\Deepak\AppData\Roaming\DB87D1\112F02.hdb”. The directory name is taken from the mutex name’s character offsets from seven to 12, and the file name is taken from offsets 12 to 17.

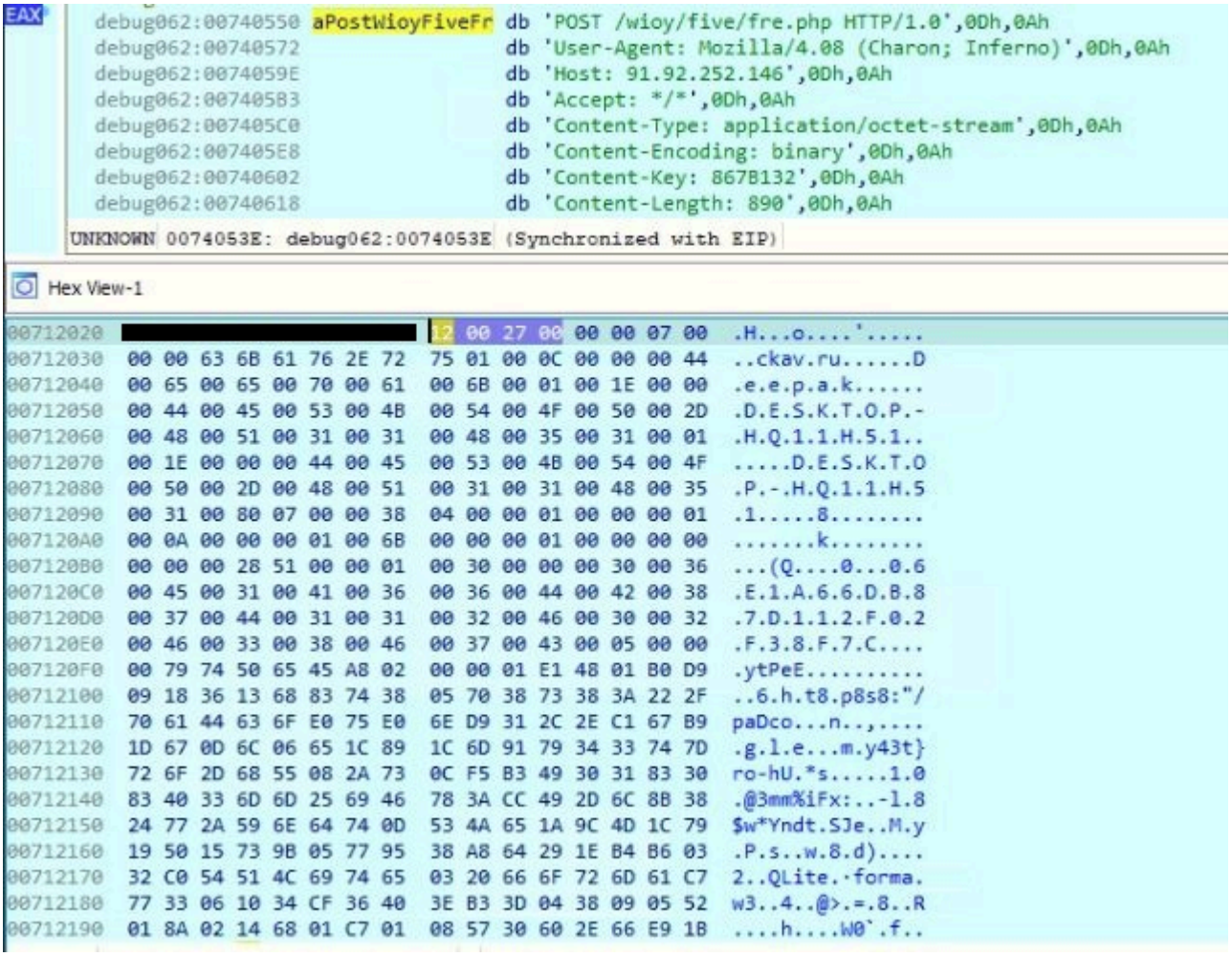


Figure 16: Stolen data sent to the C&C server

An explanation of values sent by the malware to the C&C server is mentioned in the table below:

Offset	Size	Field Description	Value
00	2	Version	12
02	2	Information type	27
04	2	IsUnicodeFlag	0
06	4	Size of binary ID	07
10	7	BinaryID	ckav.ru
17	2	IsUnicodeFlag	1
19	4	Size of username	12
23	12	Username	Deepak (Unicode)
35	2	IsUnicodeFlag	01

37	4	Size of computer name	30
41	30	Computer name	DESKTOP-HQ11H51 (unicode)
71	2	IsUnicodeFlag	01
73	4	Size of computer name	30
77	30	Computer name	DESKTOP-HQ11H51 (unicode)
107	4	Screen width	780
111	4	Screen height	438
115	2	IsUserAdmin	01
117	2	IsBuiltInAdministrator	00
119	2	PROCESSOR_ARCHITECTURE_AMD64	01
121	2	OS major version	10
123	2	OS minor version	00
125	2	Left over	6B
127	2	Related to service pack version	01
129	2	IsFirstPacketSent	00
131	2	Word value 1	00
133	2	Word value 0	00
135	2	Word value 0	00
137	2	Word value 0	00
139	4	Size of stolen data	20776
141	2	IsUnicodeFlag	01
145	4	Size of mutex name	30
149	48	Mutex name	06E1A66DB87D112F02F38F7C
197	4	Size of random string	5
201	5	Random string	ytPeE
206	4	Size of compressed data	680
210	680	Compressed data	01 E1 48 01 B0 D9…

Figure 17: Description of the data sent to the C&C

After sending the stolen application data, the malware checks to see if the process is running in administrator mode to steal credentials stored in Windows Credential Manager. The malware enumerates files from the directory "%AppData%\Roaming\Microsoft\Credentials" using the APIs *FindFirstFileW* and *FindNextFileW* to read the encrypted data which is decrypted by accessing the process memory of lsass.exe.

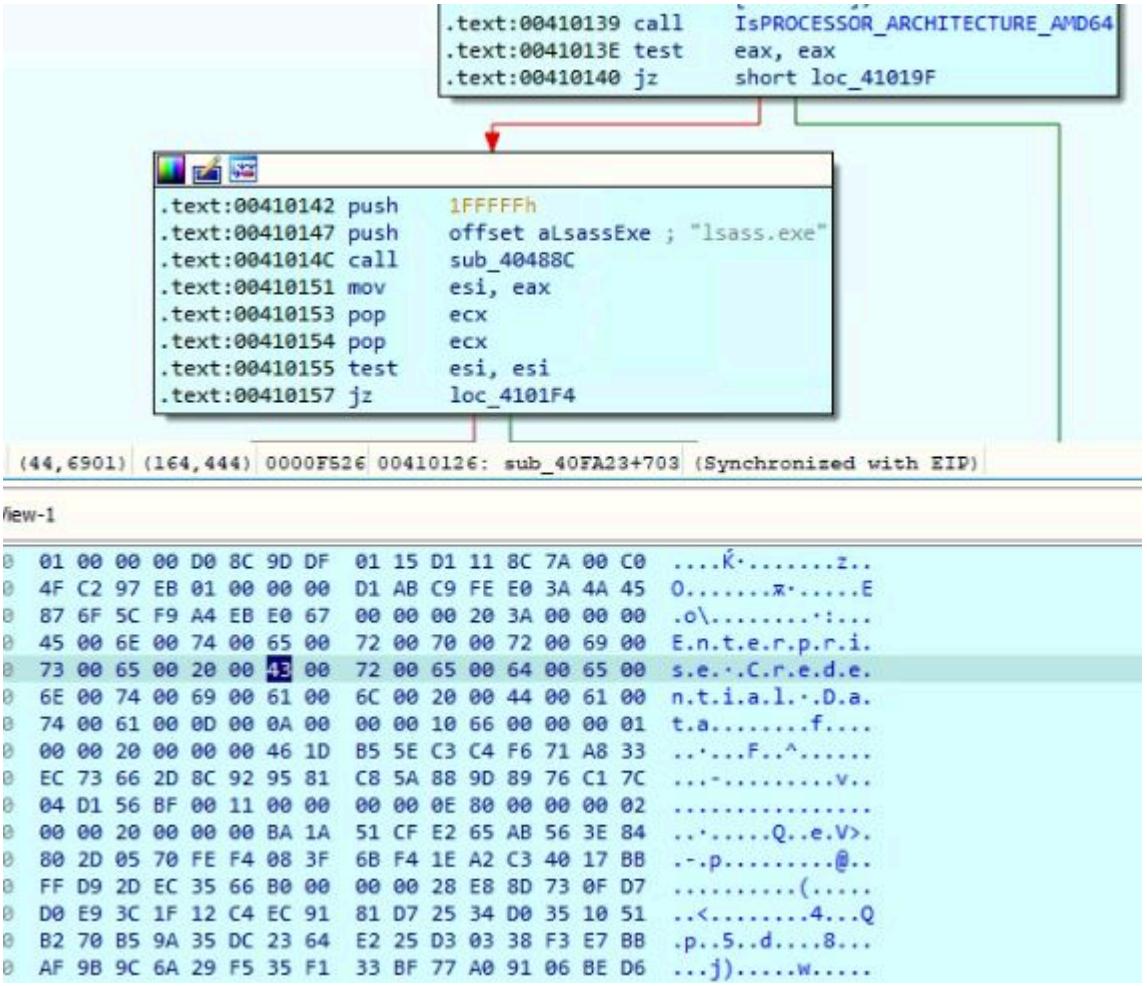


Figure 18: Encrypted credentials in Windows Credential Manager

Decrypted credentials are compressed using aPLib compression library and appended into the structured buffer after malware variant-specific information and collected system information. It is then sent to the C&C server.

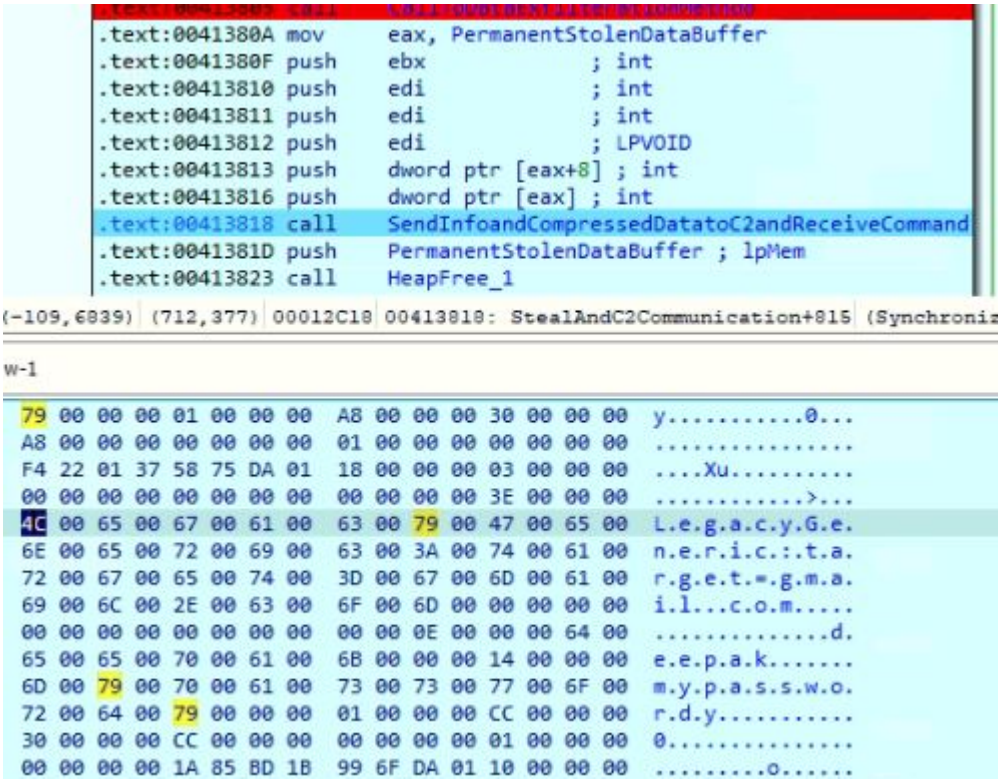


Figure 19: Decrypted credential from Windows Credential Manager

The malware creates the file “%APPDATA%\Roaming\DB87D1\112F02.lck” before attempting to decrypt credentials from Windows Credential Manager and deletes the files once the decryption routine is completed.

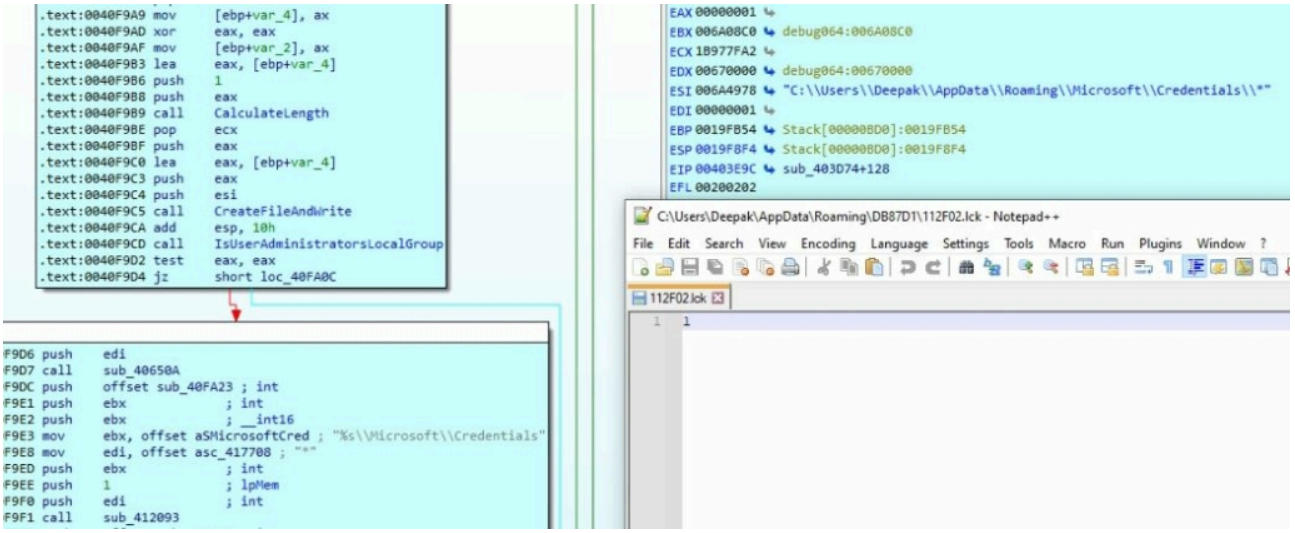


Figure 20: Creating the lock file

After sending the stolen data, the malware expects commands from the C&C server and spawns a separate thread to perform actions based on the received command.

Registry Entry

The malware creates a self-copy in the application data folder using a file move operation. If it fails in moving the file, then the malware uses the file copy operation. The malware creates a registry entry “HKCU\http://91.92.252.146:8015/wioy/five/fre.php\DB87D1” and sets the values as the path of the dropped file.

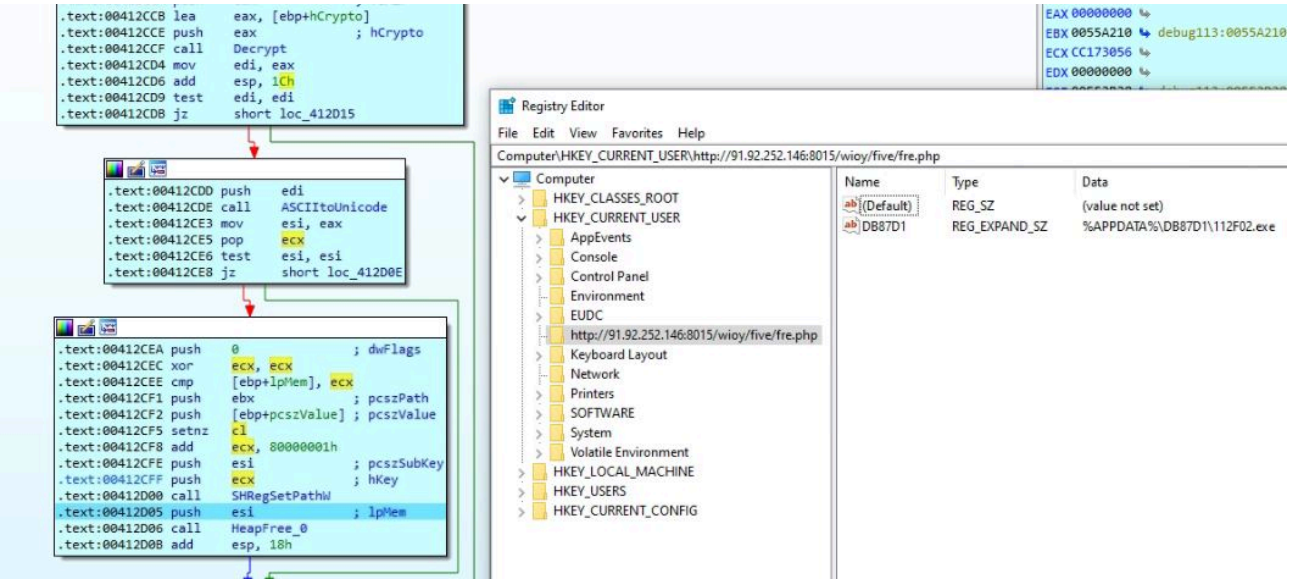


Figure 21: Registry entry

The malware modifies the attributes of the directory to the following values to lower the visibility:

- FILE_ATTRIBUTE_HIDDEN
- FILE_ATTRIBUTE_SYSTEM
- FILE_ATTRIBUTE_NOT_CONTENT_INDEXED

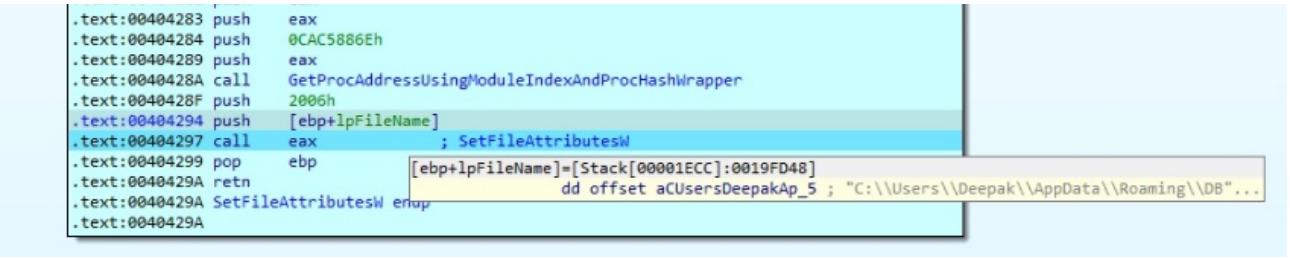


Figure 22: Modifies directory attributes

The archive file cannot be found in any of the popular threat intelligence sharing portals like VirusTotal and ReversingLabs at the time of writing this blog, which indicates its uniqueness and limited distribution.

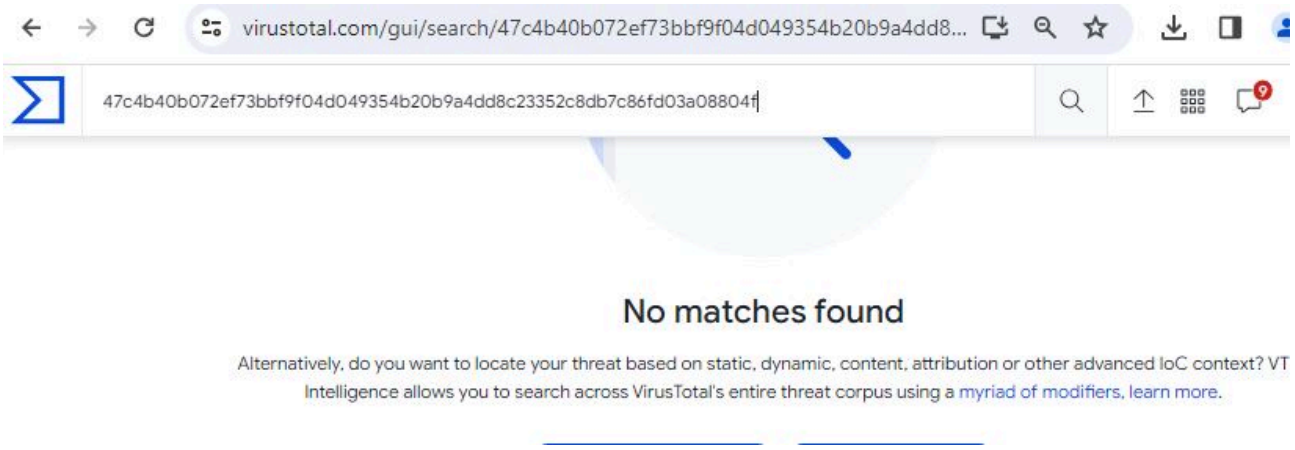


Figure 23: File is not available on Virus Total

Evidence of the detection by our RTDMI engine can be seen below in the Capture ATP report for this file.

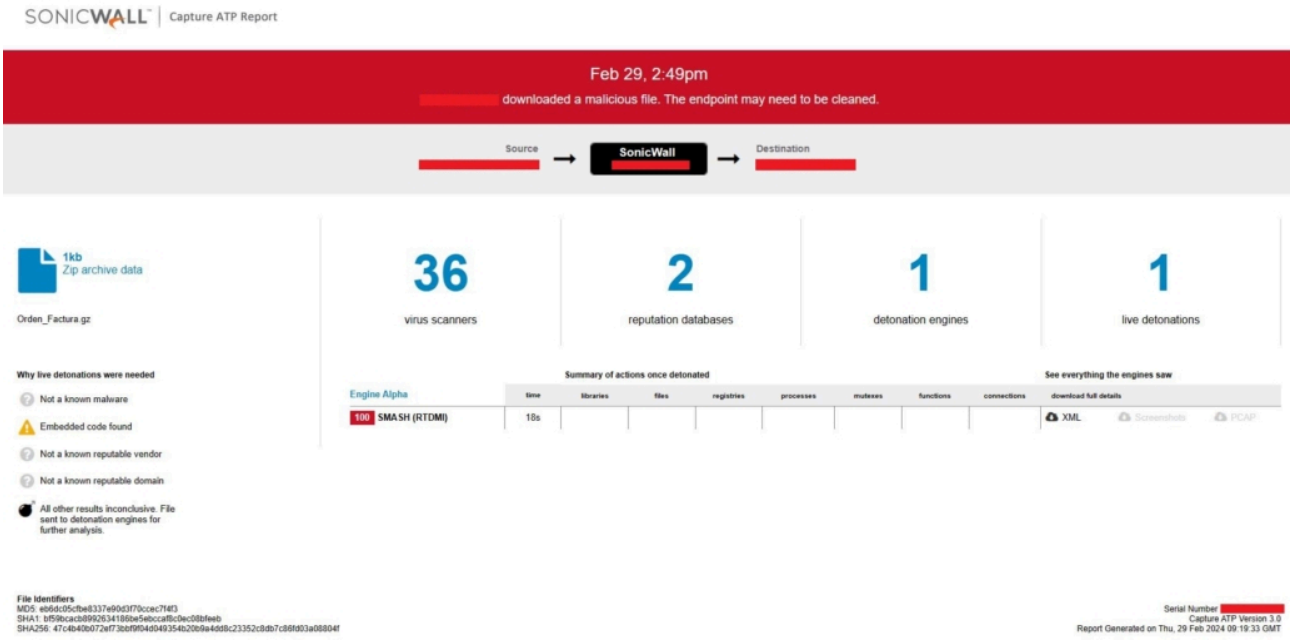


Figure 24: Capture report

Source: https://blog.sonicwall.com/en-us/2024/03/lokibot-is-being-distributed-by-windows-shortcut-files/