Malware Forensics

REPORT

Content

Summary

I conducted this malware analysis report on authorisation from Longtext. This report analyses malware that has affected the company's system security due to a rogue employee. Longtext provided me with a virtual machine and the malware's executable. There are three main goals of this report:

- Investigate how the malware infects and works on the system
- Investigate associated malware infrastructure
- Analysation of the malware statically and dynamically
- Provide recommendations for Longtext

Summary of results

Results	Risk	Recommendation
Botnet connectivity	severe	Implement network security, such as network segmentation, firewalls, incident response plans, network monitoring, and maybe even an intrusion detection system. These can all help make sure you can see what is happening on the network
Persistence generation	severe	Have audit logs of systems to see what has been added to potential added to the registry
Command execution	severe	Implement some form of privilege to make sure users have the minimum required rights for their role. Implement input validation; this can help prevent command injection risks Application sandbox can make it so processes and applications are not a part of the main system
Company compromisation	severe	Implement staff training to help staff understand what they are looking out for and what to be suspicious of
Data leak	Severe	Implement encryption to stop data from being leaked

Malware details

Malware packed				
Link	https://www.virustotal.com/gui/file/f3883f8a9bceb7a00b36eac80c042623f87080cbd6ed017878c5ad7f3f0a0ce3/behavior			
Name/file type	malfor-cw-sample.exe	Win32 EXE		
Size	464.79 KB			
Hash value	f3883f8a9bceb7a00b36eac80c042623f87080cbd6ed017878c5ad7f3f0a0ce3			
Behaviour	Bot net			
Indicator of compromise	Registry (persistence), IP Address, file path, Domain names			
Packer	Enigma virtual box			
Threat intelligence	Pavel Chekov			
Activity summary				
Mitre signatures	**IDS Rules**	**Sigma Rules**	**Dropped files**	**Network comms**
• 1 Low	• 3 High • 4 low	• 1 Medium	• 41 other	• 2 HTTP • 5 IP

Table 1: Malware packed information

Malware unpacked		
Link	https://www.virustotal.com/gui/file/3b0422bff4061fa53574d3724f58cf9e695069ece1b8c9ad69a2b69fe62a3dcb/behavior	
Name / file type	malfor-cw-sample_unpacked.exe	Win32 EXE
Size	56.79 KB	
Hash value	3b0422bff4061fa53574d3724f58cf9e695069ece1b8c9ad69a2b69fe62a3dcb	
Behaviour	Trojan, backdoor, Bot net	
Indicator of compromise	Registry (persistence), IP Address, file path, Domain names	
Unpacker	https://lifeinhex.com/tag/unpacker/	
Threat intelligence	Pavel Chekov, Equation group	

Activity summary				
Mitre signatures	**IDS rules**	**Sigma rules**	**Dropped files**	**Network comms**
• 1 low • 24 Info	• 3 High • 2 Medium • 7 Low	• 1 High • 1 Medium • 1 Low	• 14 Other • 1 Text	• 5 HTTP • 6 IP

Table 2: Malware unpacked information

Static analysis

Unpacking

I ran a program called PEid; this allowed me to check if the malware was packed (a form of compressing to make the malware unanalysable). In Figure 1, we can see malware is unpacked.

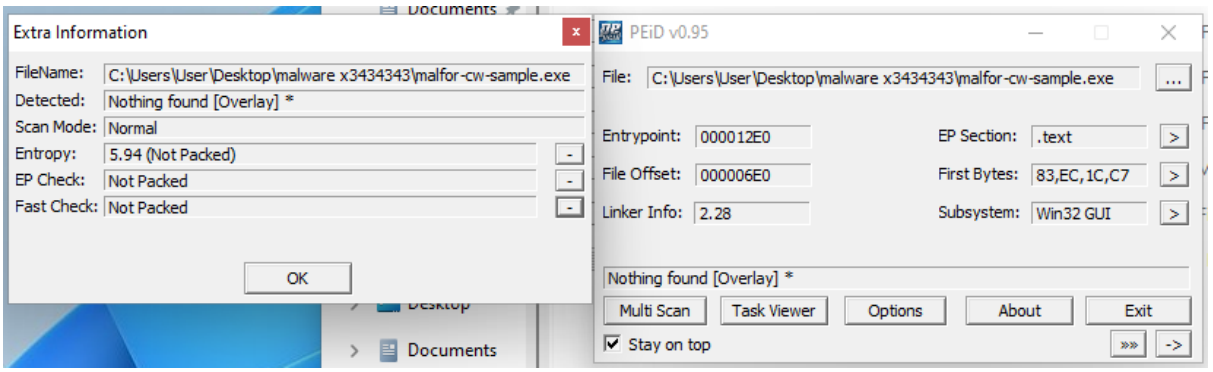


Figure 1: unpacked malware.

I loaded the malware into an interactive disassembler (IDA). IDA allows me to view the malware's functionality and see the different functions it uses to perform actions. In IDA, I noticed that all functions were called "sub_*retrospective number*" (figure 2), with over 1600+ functions. Some functions are hidden under "byte_**retrospective number*". IDA led me to believe that this malware is packed.

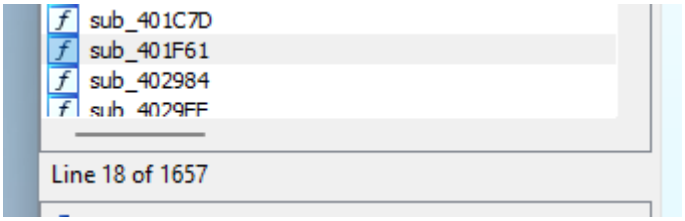


Figure 2: 1600+ functions

I ran the "strings" program (figure 3) on the malware to see if it exposed any functions that would help me prove it was packed. This program grabs all strings referenced in the malware. Figure 3 shows this program's result and tells me something is packing this malware.

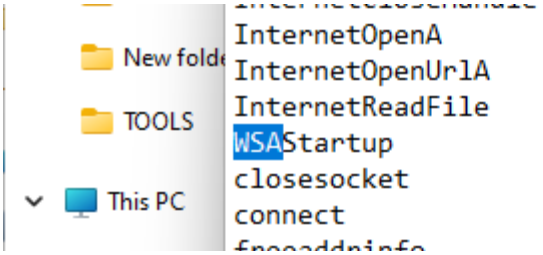


Figure 3: Strings on the malware

I checked Virustotal to see if it could find any details, which I could not find using the programs I have on my system. With this in the details section of the malware (Figure 4), I can see the malware is using a protector/packer called "Enigma Virtual Box" (Enigma Virtual Box consolidates application files and registry into a single executable file).

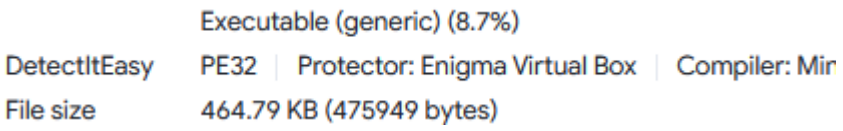


Figure 4: The protector/packer

I found an unpacker known as “EnigmaVMUnpacker.” Allowing me to unpack the malware. Figure 5 shows the unpacker application.

Figure 5: The enigma unpacker

Having the malware unpacked allows me to understand better what they are trying to achieve since now I can see the functions, what they connect to, and how they affect each other. IDA led me down a rabbit hole of different functions where I explained them all in disassembling the malware. Figure 6 shows the new function table of the malware, now changed from “sub_*retrospective number*” to WinMain(x,x,x,x).

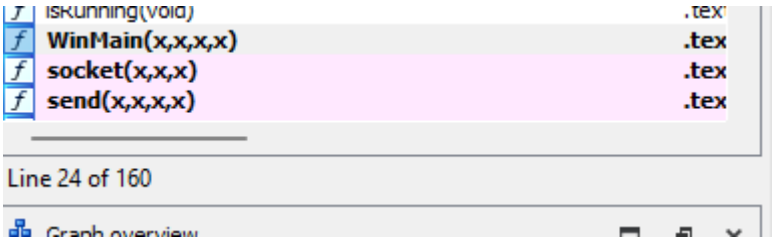


Figure 6: The new functions

Malware Functionality

Malfor-cw-sample.exe is a multifaceted malware designed to compromise the system's security and manipulate a target device's integrity. Its main objective is to connect the user's device to a botnet, which an attacker can then exploit to perform tasks sent to the user's device via an internet relay chat (IRC) client; it also does this by using persistence to allow for prolonged presence on the machine.

Botnets, which are networks of malware-infected machines controlled by a malicious actor, are the primary cause of many Internet security issues (Stone-Gross., 2010). Demonstrates how damaging these types of malware can be. They are the primary means by which cybercriminals carry out their nefarious activities, such as sending spam, launching denial-of-service attacks, or stealing personal information (Stone-Gross., 2010).

```
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
malfor
PRIVMSG %s :Shutdown password entered - botnet shutting down
Botnet shutdown by user %s.  Would you like to restart it?
Botnet shutdown
PRIVMSG %s :not that one, sorry.. wrong pw!
```

Figure 7: Botnet in strings

Figure 7 shows how this is a botnet; I can see command responses. In IDA, we can see this more in-depth and explain how this malware works and infects the system.

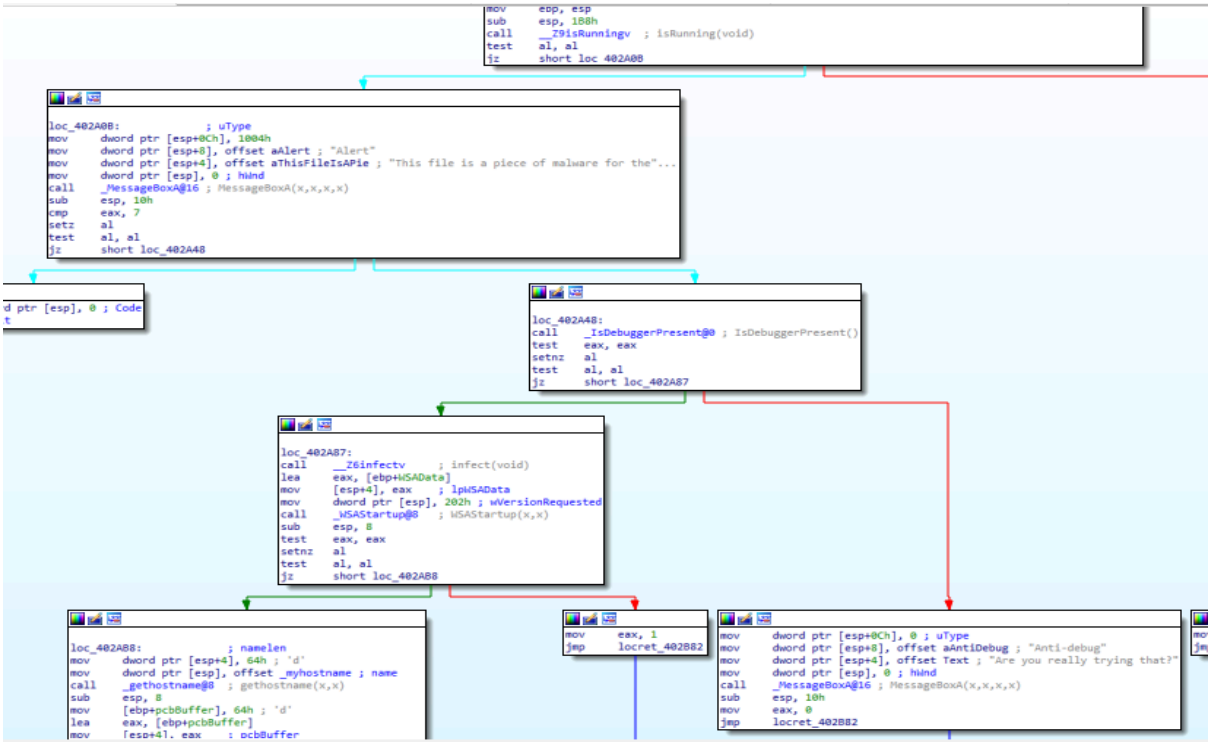


Figure 8: Malware WinMain function

In Figure 8, presents many different functions that control how the malware begins to infect the system. In my disassembly report, I go into more detail about all of these other functions.

```
var_C= dword ptr -0Ch

; __unwind {
push    ebp
mov     ebp, esp
sub     esp, 148h
mov     dword ptr [esp+20h], 0 ; lpdwDisposition
mov     dword ptr [esp+1Ch], offset _hKey ; phkResult
mov     dword ptr [esp+18h], 0 ; lpSecurityAttributes
mov     dword ptr [esp+14h], 0F003Fh ; samDesired
mov     dword ptr [esp+10h], 0 ; dwOptions
mov     dword ptr [esp+0Ch], 0 ; lpClass
mov     dword ptr [esp+8], 0 ; Reserved
mov     dword ptr [esp+4], offset SubKey ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe"...
mov     dword ptr [esp], 80000001h ; hKey
call    _RegCreateKeyExA@36 ; RegCreateKeyExA(x,x,x,x,x,x,x,x,x)
sub     esp, 24h
```

Figure 9: Persistence generation

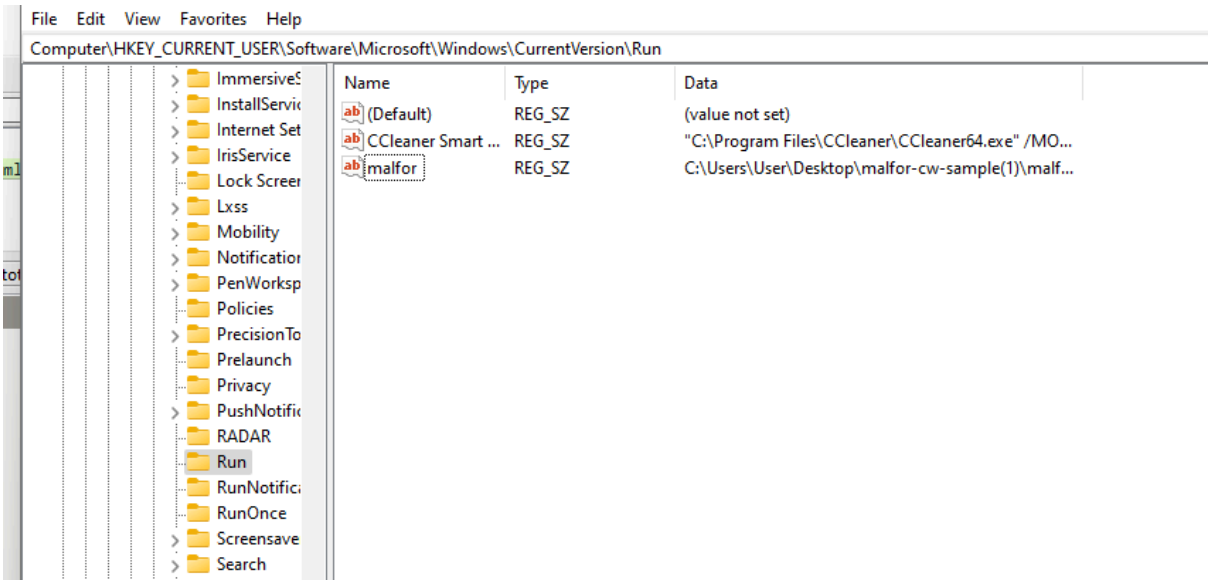


Figure 10: The persistence:

The infection function begins by generating persistence. It adds the malware executable to the registry (Figure 10). This means the malware will instantly start whenever the user loads their computer, creating a constant loop of connecting to the botnet.

Figure 11: OpenURl and ReadFile Channame

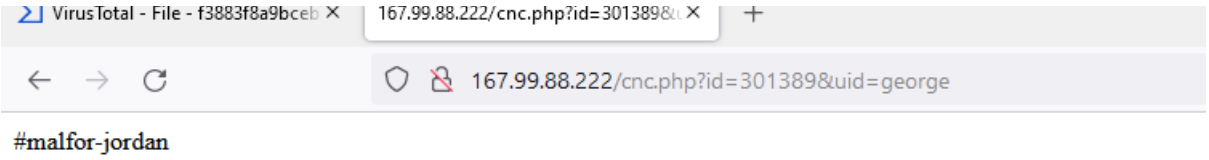


Figure 11: website for the channel

After the infection, it uses Winsock dll processes by calling the WSAstartup function. After the "__Z14updateChanNamev" function, this function goes to a malware webpage "http://167.99.88.222/cnc.php?id=John-PC&uid=John" (figure 11) is one, and this has a possible malware channel name, which then connects to the botnet, by grabbing the channel name. From what we know about the IRC client, this can be a channel name used to run commands on the botnet.

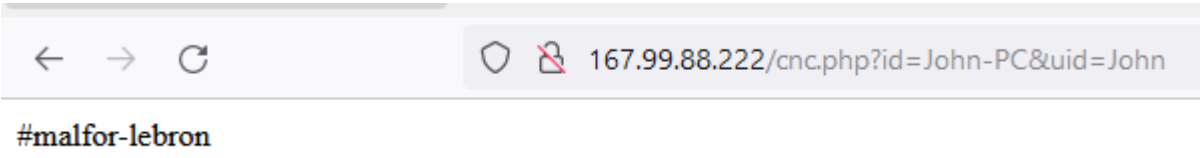


Figure 12: Updated channel

Observing the channel website for some time, I noticed the change would change every so many days (figure 12). From what I know, a botnet might do this to help evade detection by changing known patterns; this also helps when avoiding blacklisted channels. People can blacklist specific channels to stop potential forms of attack. Since it changes channels a lot, it bypasses that security measure.

The malware grabs the connection specifications, e.g. port number "6667". Confirming the IRC client connectivity since this port is a TCP/UDP port for the IRC client. Which then grabs the socket information and connects to this section of the server to control the bots. These all relate to a final function, "__Z9ircclientv". This function controls the bots.

```
mov     [esp+0Ch], eax
mov     dword ptr [esp+8], offset aBot ; "bot"
mov     dword ptr [esp+4], offset aSD ; "%s%d"
mov     dword ptr [esp], offset _nick ; Buffer
call    _sprintf
```

Figure 12: name generation

This function uses the name “bot” and generates a random number character string using the _nick and buffer (figure 13). The overall primary function in this section is the IRC control it has, shown by all the executables.

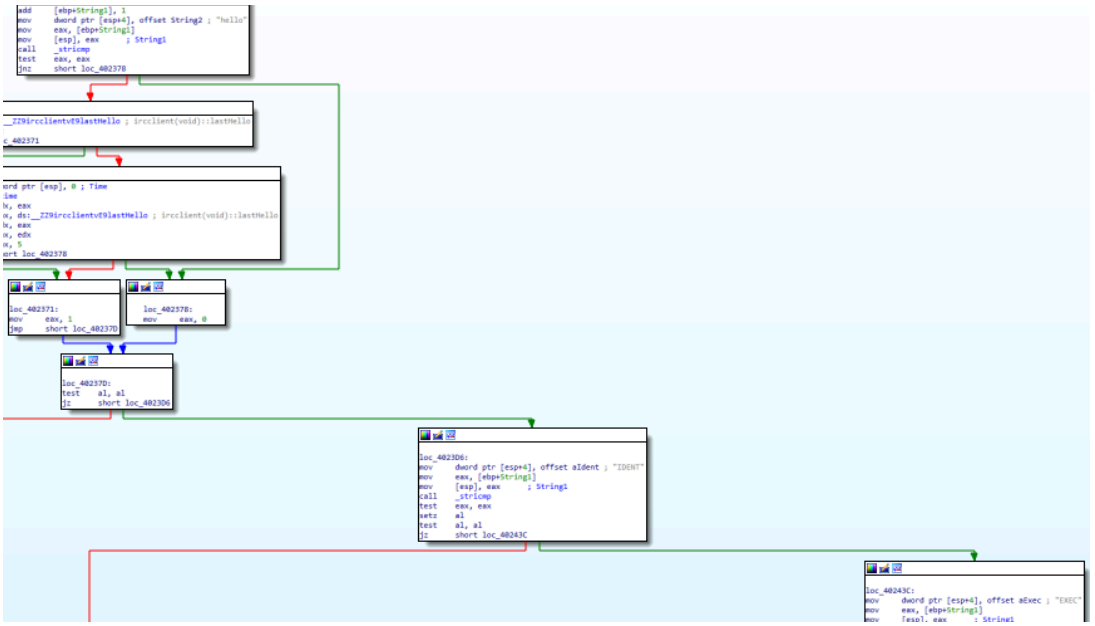
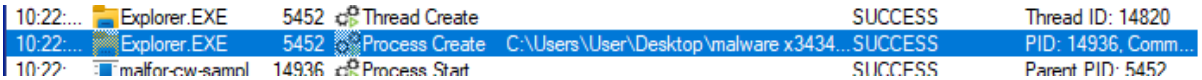


Figure 13: executable functions in the malware

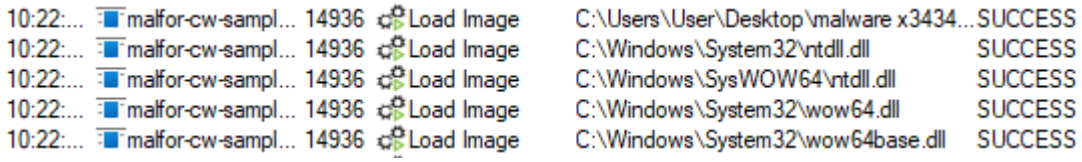
The later sections of the malware, we can see different commands and functionalities. Figure 13 shows the beginning of the command section for this botnet. In these sections, some commands include exec, open, pic, DDoS, and many more. These commands run in the IRC client. Their commands perform a specific function, such as the “exec” allowing the user to open particular programs. “exec #notepad.exe” will open the notepad, and “DDoS” will take down a network.

Dynamic analysis

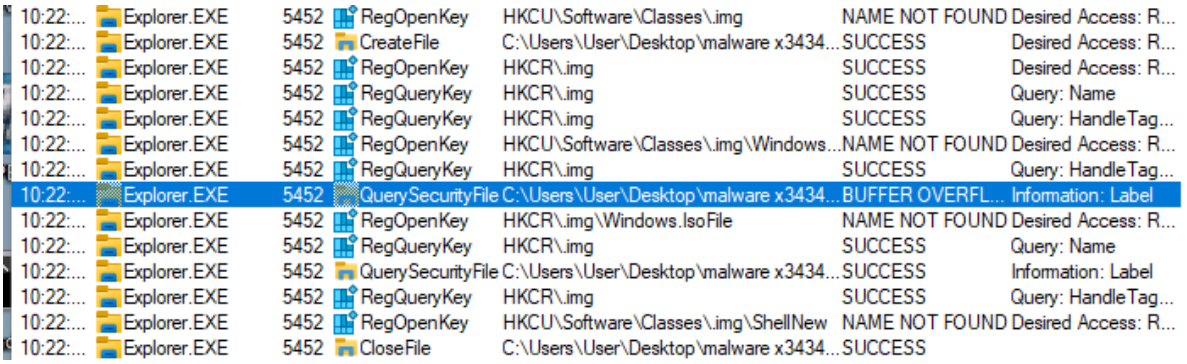
Process monitoring



The image above shows how, after running the executable, we can see the creation of a process

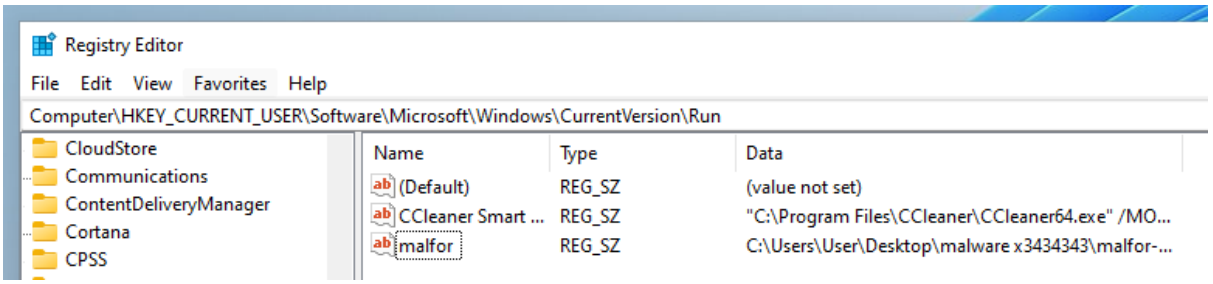


I can then see the loading of ".dll" these different types of API calls throughout the malware.



In this image we can see the creation of a file, which it then goes and queries the file, this could mean it reading the attributes of the location it has been run in.

Persistence generation



Immediately after running the malware, I went and checked the registry. As we can see in the image above, the malware has planted itself in the "Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run". The malware will run immediately on the device's start-up.

IRC client

I downloaded an IRC client before starting the malware and seeing what I could find. For this report, I will be using HexChat: https://hexchat.github.io/. This client would allow me to connect to the botnet master.

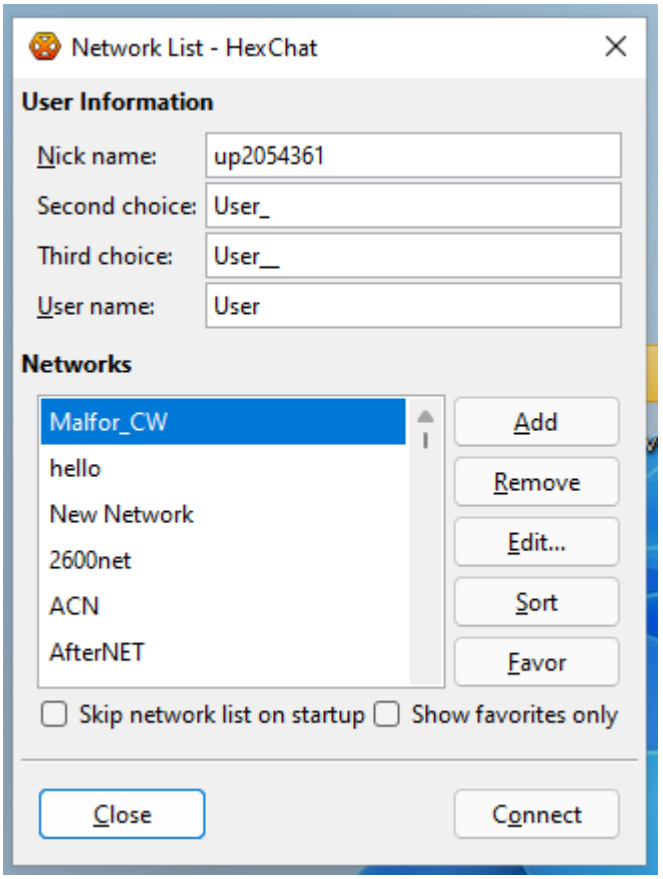


Figure 14: IRC client account name and network

Figure 14 shows the IRC client running. I went and set my username to UP2054361 and created a new network. I will add the connection details soon. At this point, I need the details for the network to connect. Knowing this is a network-based malware, I decided to run wireshark while running the malware to see what I could see for connections Figure 15.

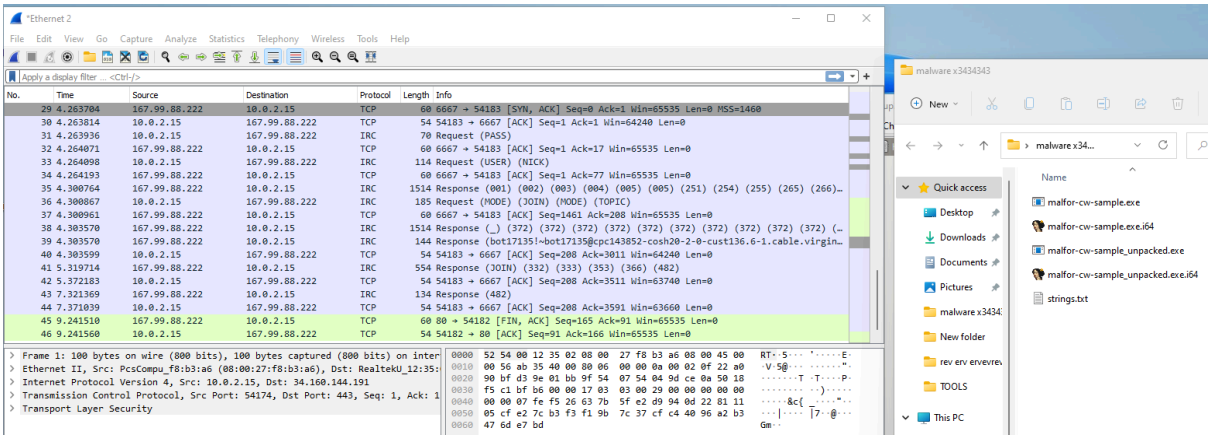


Figure 15: Wireshark responses

Figure 15 shows a bunch of network responses from the IP address "167.99.88.222", from what we know is the attacker's IP. Seeing this, I noticed two critical responses.

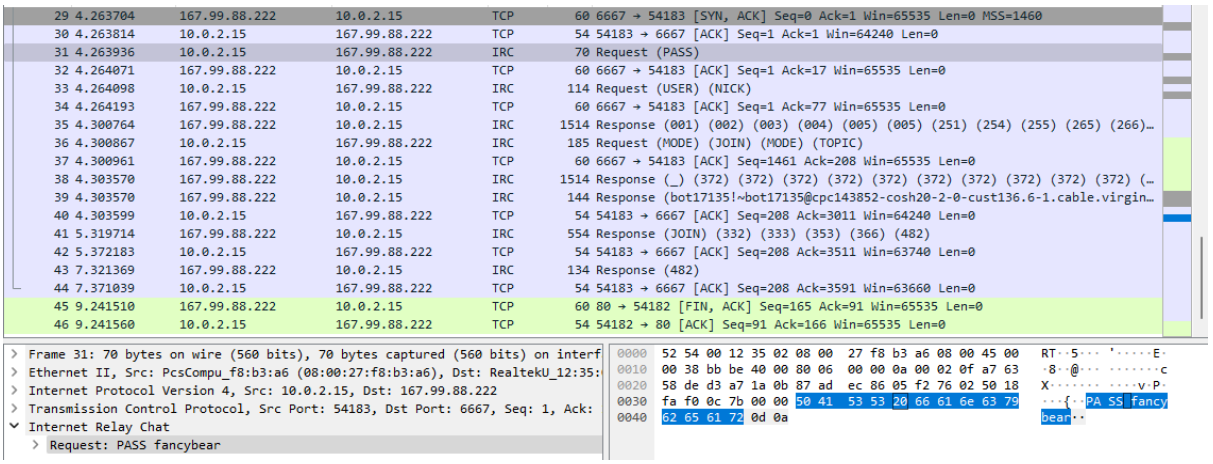


Figure 16: IRC PASS request

I put this password into the IRC network settings password; this allowed me to connect to the IRC host server, as shown in Figure 17.

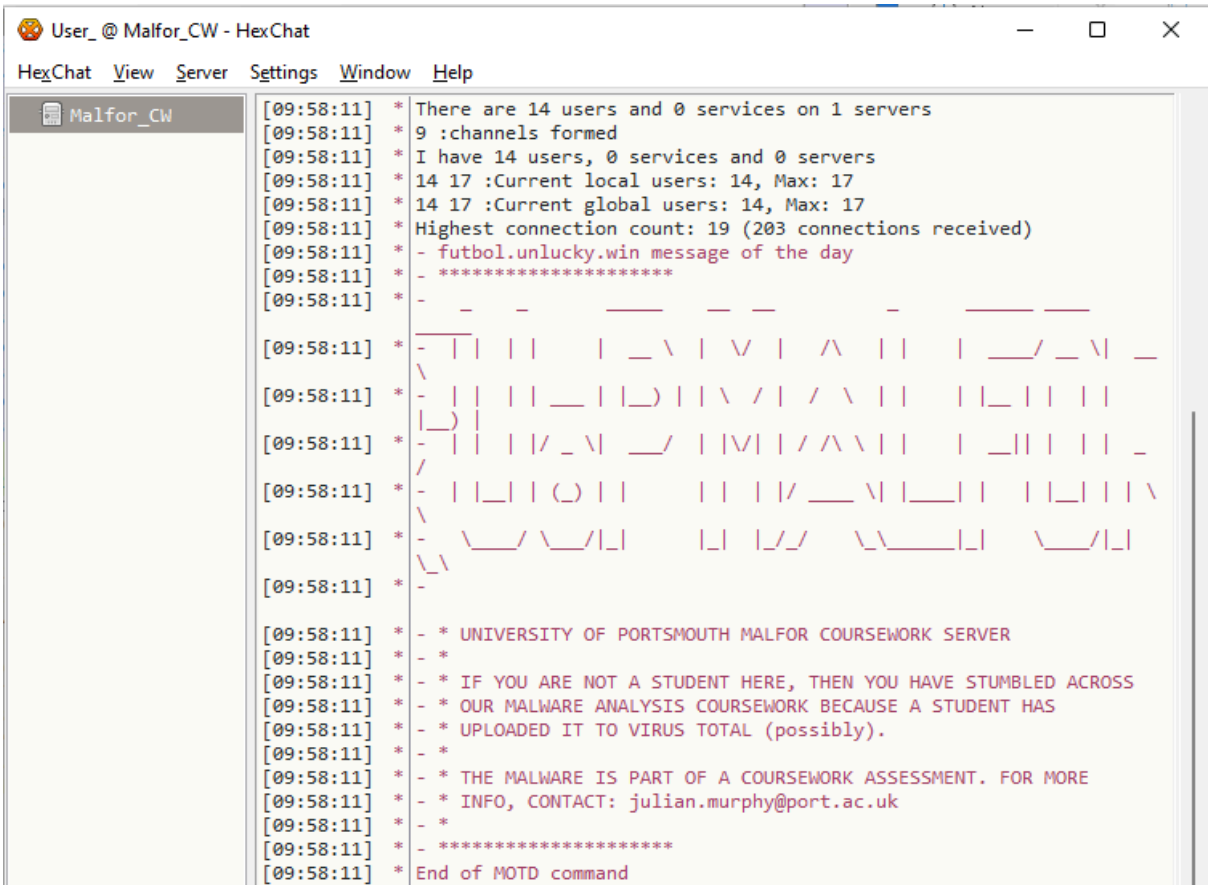


Figure 17: The IRC host server

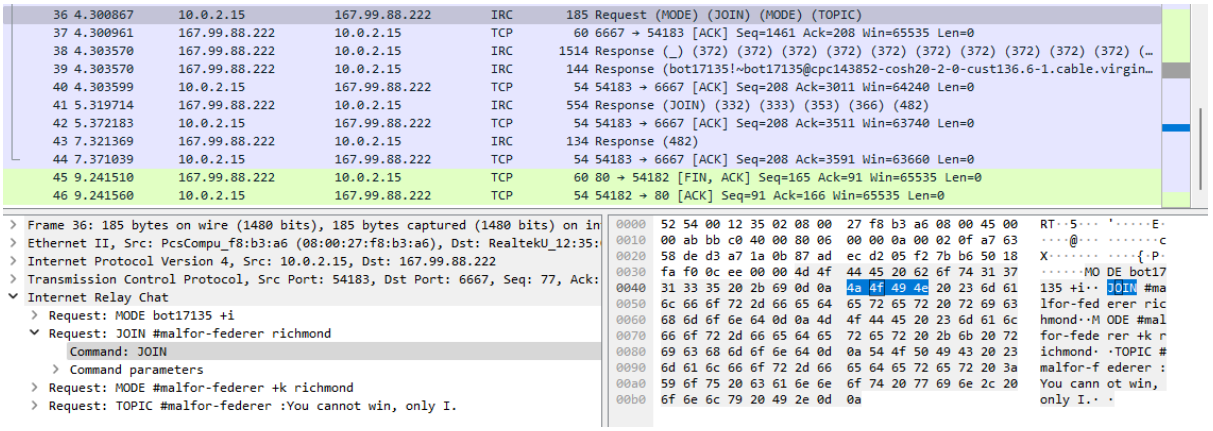


Figure 18: IRC JOIN response

Since I connected to the host server, I needed to connect to the channel controlling the bots. In Wireshark, a response gave me the join channel and the password for it. I could have gotten the channel name from the website, but this was the only way to get the password. The full command to join is in Figure 19.

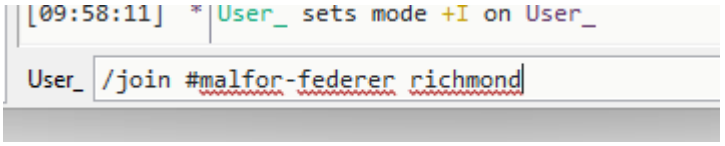


Figure 19: The IRC join command

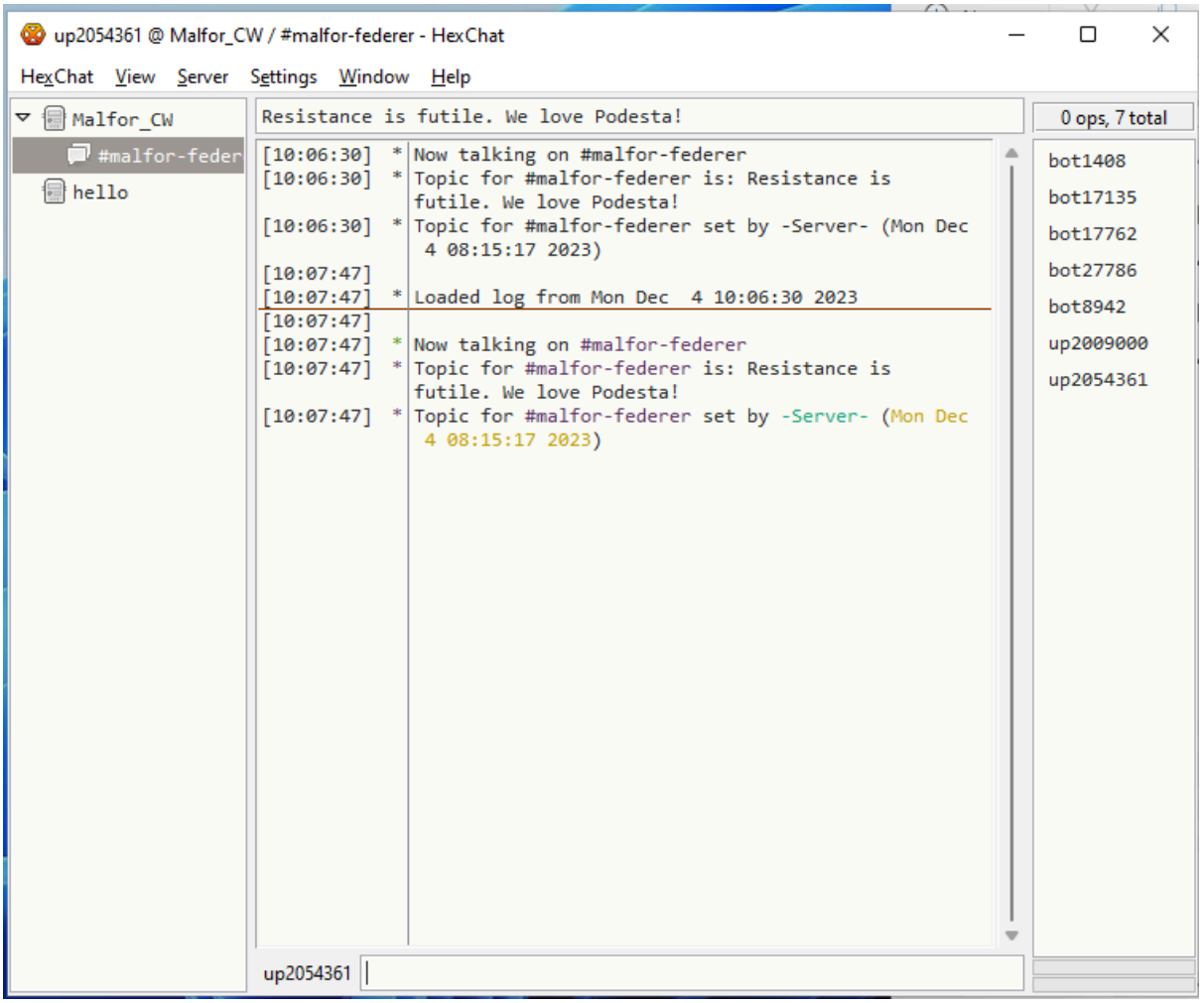
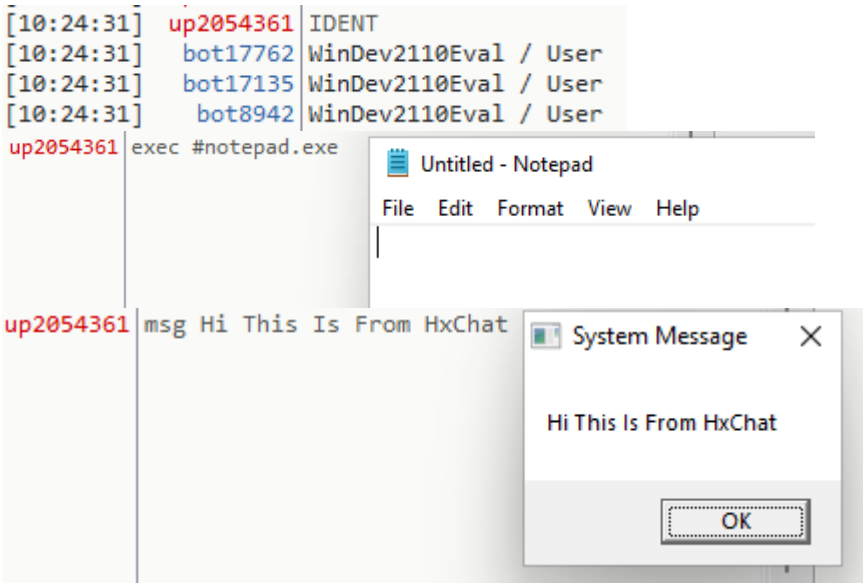
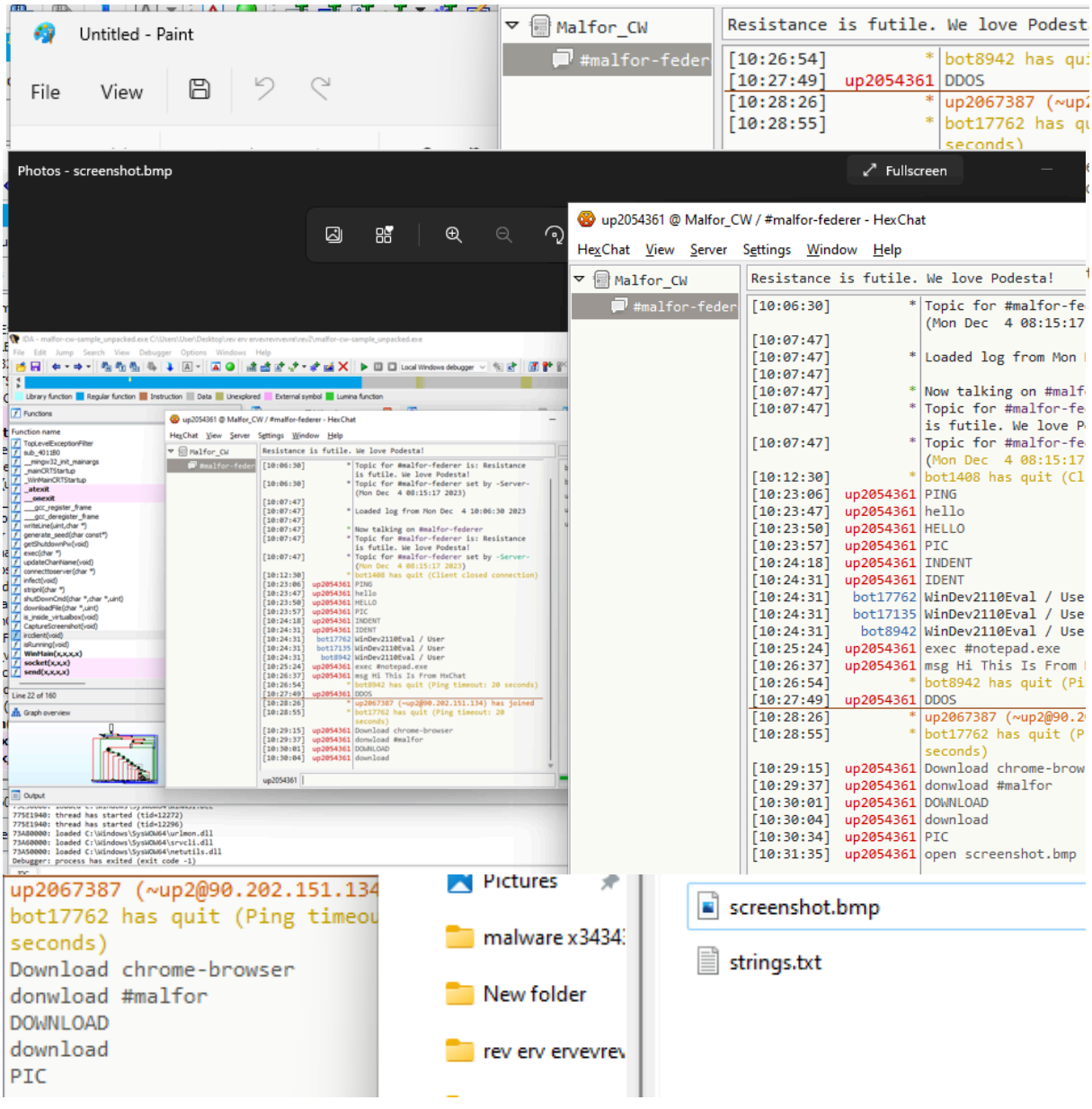


Figure 20: Bot master access

After entering the command to join the channel, I accessed the bot master channel (figure 20). Allowing me to test the command I saw in IDA (figure 12) and see what they can do and how they function. I have referenced the shutdown command in the reverse engineering section.


[10:24:31] up2054361 IDENT
[10:24:31] bot17762 WinDev2110Eval / User
[10:24:31] bot17135 WinDev2110Eval / User
[10:24:31] bot8942 WinDev2110Eval / User
up2054361 exec #notepad.exe
Untitled - Notepad
File Edit Format View Help
up2054361 msg Hi This Is From HxChat
System Message
Hi This Is From HxChat
OK


Untitled - Paint
File View
Malfor_CW
#malfor-feder
Resistance is futile. We love Podest
[10:26:54] * bot8942 has qu
[10:27:49] up2054361 DDOS
[10:28:26] * up2067387 (~up
[10:28:55] * bot17762 has q
seconds)
Photos - screenshot.bmp
Fullscreen
up2054361 @ Malfor_CW / #malfor-federer - HexChat
HexChat View Server Settings Window Help
Malfor_CW
#malfor-feder
Resistance is futile. We love Podesta!
[10:06:30] * Topic for #malfor-fe
(Mon Dec 4 08:15:17
[10:07:47]
[10:07:47] * Loaded log from Mon
[10:07:47]
[10:07:47] * Now talking on #malf
[10:07:47] * Topic for #malfor-fe
is futile. We love P
[10:07:47] * Topic for #malfor-fe
(Mon Dec 4 08:15:17
[10:12:30] * bot1408 has quit (Cl
[10:23:06] up2054361 PING
[10:23:47] up2054361 hello
[10:23:50] up2054361 HELLO
[10:23:57] up2054361 PIC
[10:24:18] up2054361 INDENT
[10:24:31] up2054361 IDENT
[10:24:31] bot17762 WinDev2110Eval / Use
[10:24:31] bot17135 WinDev2110Eval / Use
[10:24:31] bot8942 WinDev2110Eval / Use
[10:25:24] up2054361 exec #notepad.exe
[10:26:37] up2054361 msg Hi This Is From
[10:26:54] * bot8942 has quit (Pi
[10:27:49] up2054361 DDOS
[10:28:26] * up2067387 (~up2@90.2
[10:28:55] * bot17762 has quit (P
seconds)
[10:29:15] up2054361 Download chrome-brow
[10:29:37] up2054361 donwload #malfor
[10:30:01] up2054361 DOWNLOAD
[10:30:04] up2054361 download
[10:30:34] up2054361 PIC
[10:31:35] up2054361 open screenshot.bmp
up2067387 (~up2@90.202.151.134
bot17762 has quit (Ping timeou
seconds)
Download chrome-browser
donwload #malfor
DOWNLOAD
download
PIC
Pictures
malware x3434
New folder
rev erv ervevrev
screenshot.bmp
strings.txt

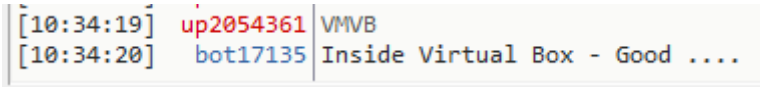


Figure 21: all commands and responses

Figure 21 shows most of the commands I could run and the response. I was demonstrating how a user could manipulate a botnet to be able to perform more malicious functions for them. This malware was not programmed to perform a specific malicious task, but it can be.

Reverse engineering

I will be using reverse engineering tactics to gain an understanding of the shutdown password. I need access to the shutdown function later in the IRC function (figure 29). I assume the password is stored there since there are presents of comparing strings. Having a string comparison could mean that the password is stored in the memory of the malware.

To begin reverse engineering on this malware, we need to be able to use it in a debugger. However, this malware has functions that check for debugger applications to avoid it running in them. So, the first thing we need to do is remove that functionality. Luckily, I can do this by changing the js. JZ is an if function but stands for just zero, meaning it will carry on if the result is a 0 (Figure 22 in red). If it is 1, it will stop you from running the debugger.

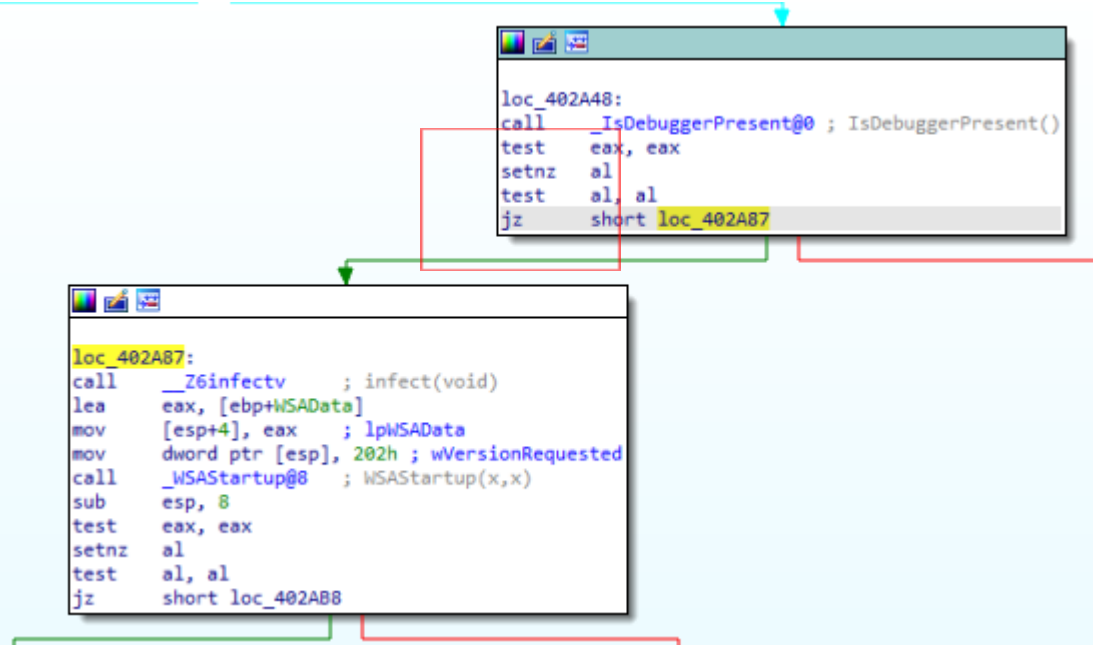


Figure 22: JZ debugger function

The way we change the jz call is by changing the hex value of the function. Figure 23 shows the hex value of jz "74 31". I only need the 74 since that's the value of JZ.

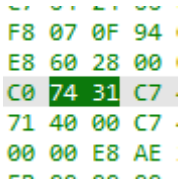


Figure 23: JZ hex value

I will be changing it to “EB”; this hex value will change the call of “JZ” to “jmp”. Having jump (jmp) be the call instead of JZ means it will go straight to the following function without checking for a debugger (Figure 24), allowing me to use it to study memory in the code.

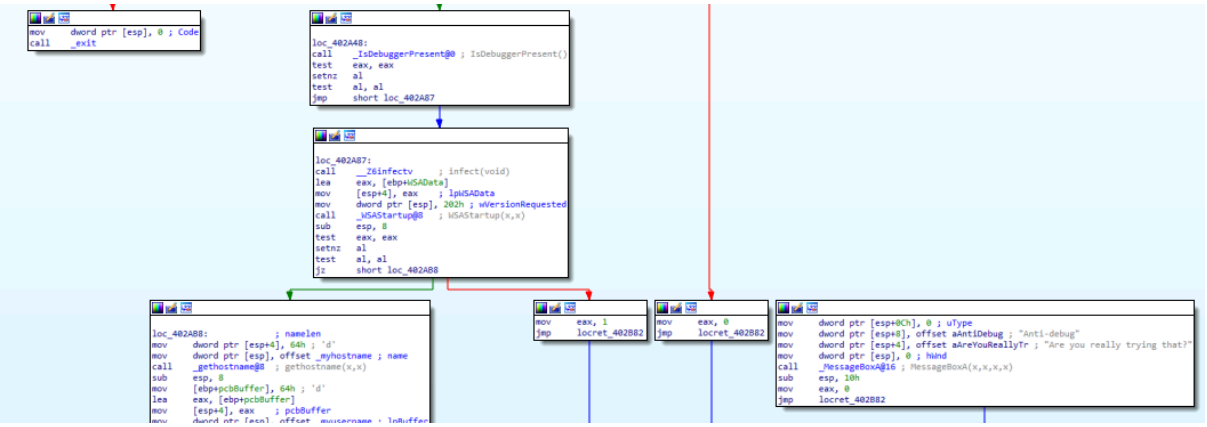


Figure 24: No longer checking for debugger

There was another anti-debugger check in the IRC client I went to, and again, I did the same thing to allow me to use the debugger.

At this point, I could run the malware in the debugger. I started using breakpoint to understand which route the malware would take (Figure 25), which let me study the paths and move myself to the designated function I wanted to reach.

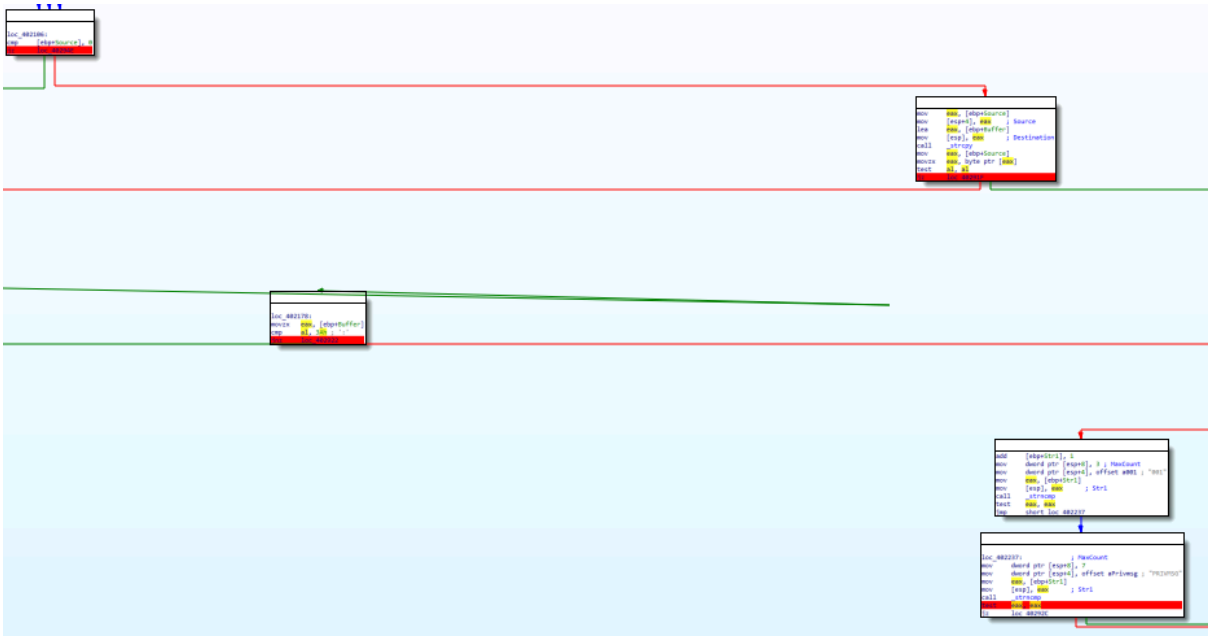


Figure 25: Breakpoint path finding

I then reached a section where the malware was checking for an integer to be equal to “001 (Figure 26). I needed to change the “jnz” (just not zero) since I needed to go to the next max count function. I could not change the integers of this path, so I went and changed the call to “jmp” this made it just directly to the following function (figure 26). We now had two functions that connected and went straight on (Figure 27)

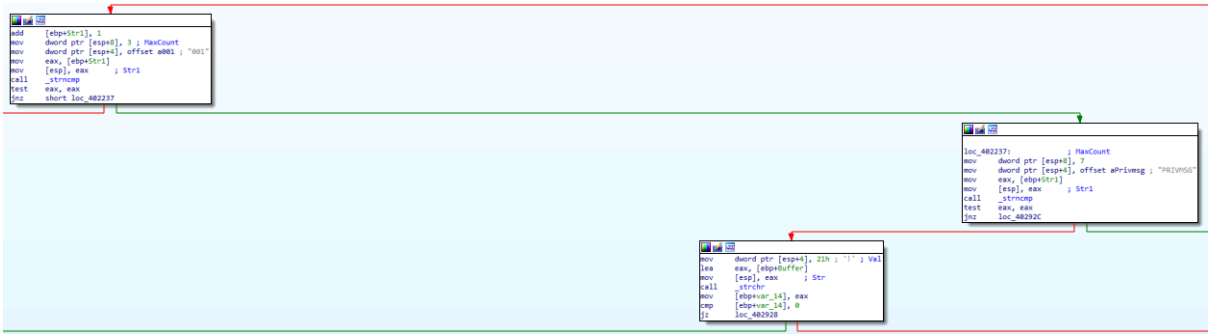


Figure 26: Max count function and PRIVMSG

Figure 27: The connected functions

Doing this allowed me to finally reach and traverse the commands until we got the shutdown command section.

When reaching this function, we needed to change one call to move down into the string comparing/encryption (Figure 28).

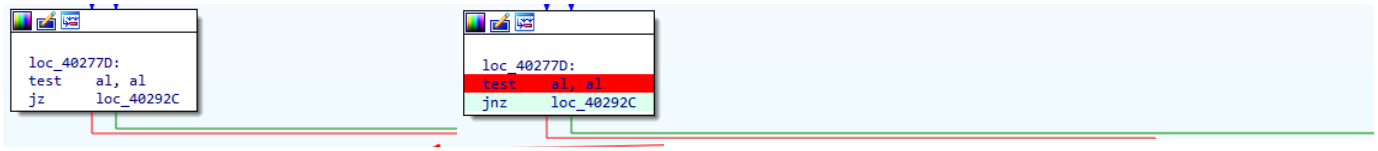


Figure 28: Call change

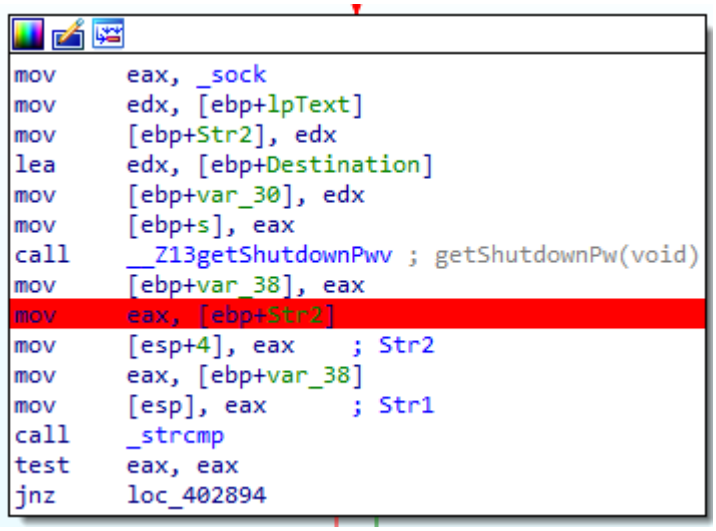


Figure 29: String storage and compare

This function is the comparison of strings, and I know this because there is "Str1" and "Str2" (Figure 29). Figure 30 shows that "str1" is an incorrect string; it is checking to see if the user has inputted "xtybtgp", which is wrong. Since we know this, that means "Str2" is the correct string, which stores as an "eax", so if we use the debugger and go to that section, we can see where the "eax" is stored in memory.

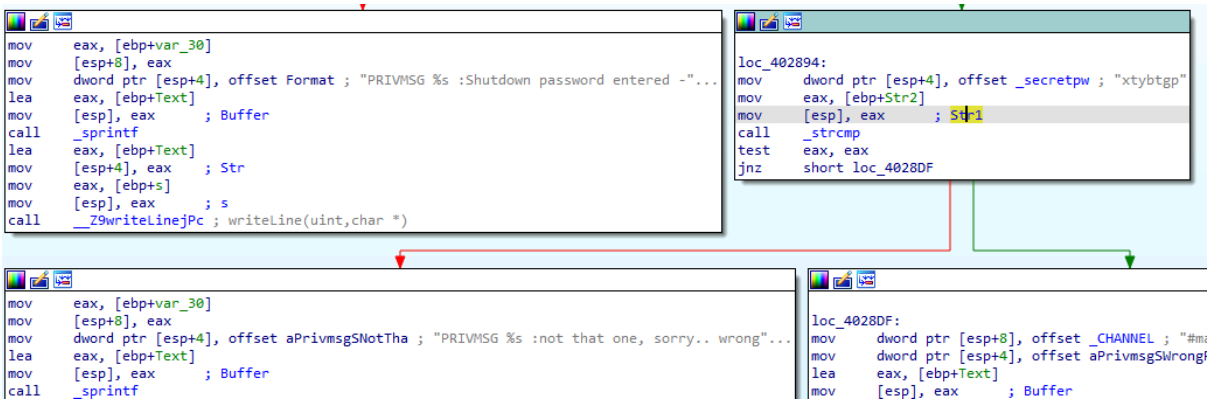


Figure 30: incorrect and correct string

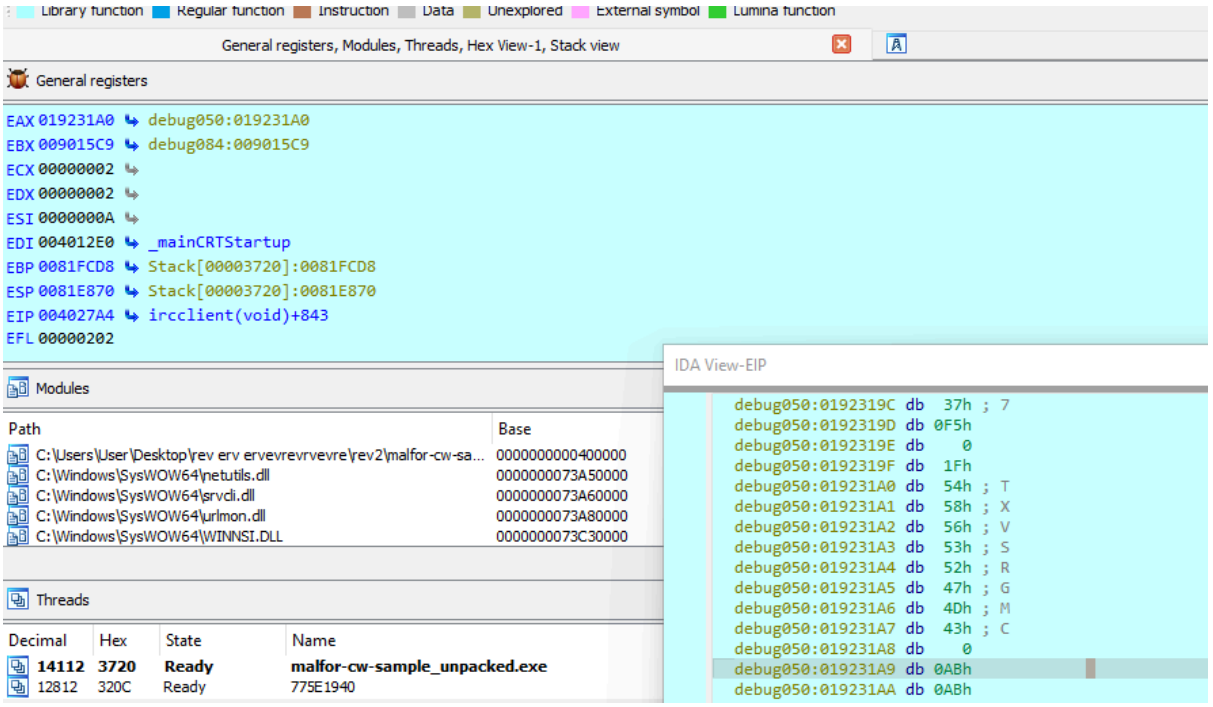


Figure 31: Eax and stored password

After reaching that section in the debugger, I can see in the general registers it gives the location in memory of the “eax”, going to this location in IDA EIP view, I can see the password for the shutdown is “TXVSRGMC”, which allows me to shut down the botnet shown in Figure 32.

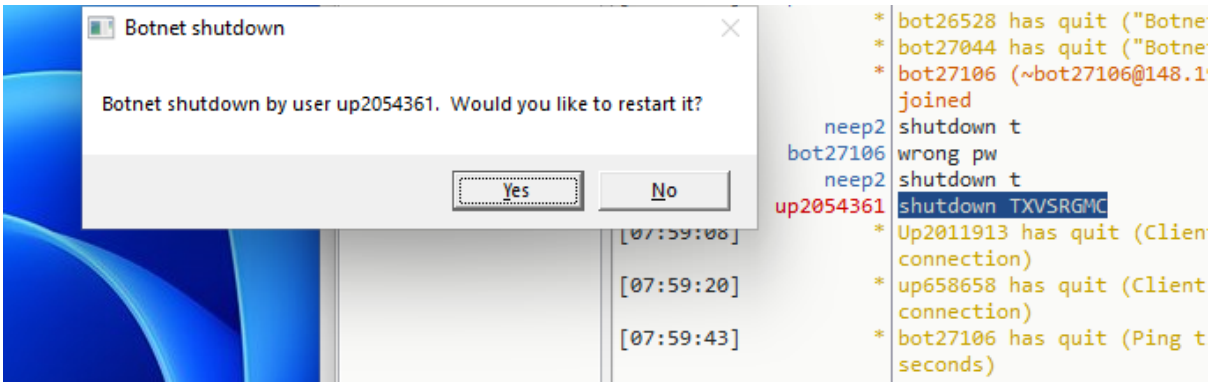


Figure 32: Botnet shutdown

Origin/removal

Origin

From information from Longtext, I can identify the origin of this malware as a rogue employee "Pavel Chekov". However, I identified something linked to the server's IP address when scanning through this malware. There was a webpage (Figure 33). This webpage states the name of a known hacking group, "Equation Group", with potential ties to the Tailored Access Operations unit with the National Security Agency. This can point to possible nation-state involvement for this attack.

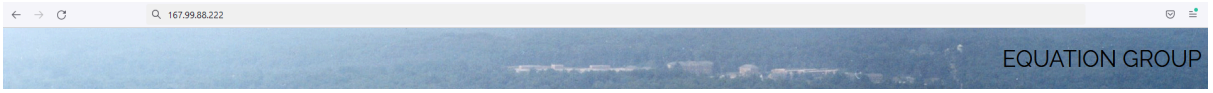


Figure 33: Webpage

Pavel Chekov's involvement facilitates the introduction of malware since he can exploit internal resources. Indicating a solid understanding of the company infrastructure, allowing them to make the spread of the malware doable on the company's systems.

Removal

The removal of the malware is quite simple.

- Ending the task in task manager will stop all connectivity with the bot master.
- Removing the process from the registry will remove the ability for the malware to function on startup (figure 24)
- removing the executable from the system to make sure the user cannot accidentally run it again

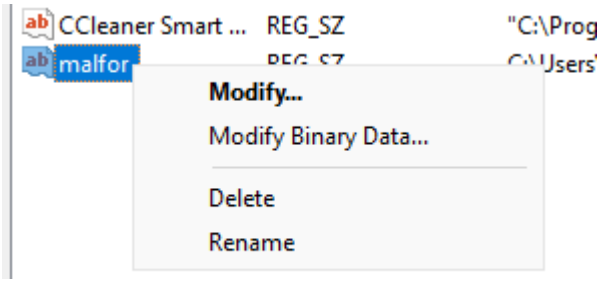


Figure 34: registry

Conclusion

This report is in response to Longtext's issue with a rogue employee stealing plans for Gazprom. This report demonstrates how the attacker accomplished this attack. The volatility of the cyber landscape presents a strong reason for the company to ensure they are well protected.

As the ever-changing cyber security world evolves, it is beneficial that the company prepares proper incident and potential business continuity plans to prevent all future and possible attacks. The potential for a security assessment could also help prevent future attacks.

References

Analysis of a Botnet Takeover | IEEE Journals & Magazine | IEEE Xplore. (n.d.). Ieeexplore.ieee.org. Retrieved November 27, 2023, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5560627

Equation Group (Threat Actor). (n.d.). Malpedia.caad.fkie.fraunhofer.de. https://malpedia.caad.fkie.fraunhofer.de/actor/equation_group

Port 6667 (tcp/udp). (2017). SpeedGuide. https://www.speedguide.net/port.php?port=6667
Software Protection, Software Licensing, Software Virtualization. (n.d.-a).

Enigmaprotector.com. Retrieved December 5, 2023, from https://enigmaprotector.com/en/aboutvb.html#:~:text=Enigma%20Virtual%20Box%20enables%20application

Software Protection, Software Licensing, Software Virtualization. (n.d.-b). Enigmaprotector.com. Retrieved December 5, 2023, from https://enigmaprotector.com/en/aboutvb.html#:~:text=Enigma%20Virtual%20Box%20is%20used