Static Malware Analysis

CS 483: Digital Forensics

Lesson 8

Lesson Learning Objectives

- Hands-on reverse engineering using IDA Pro
- Pros and Cons of static analysis

Underpinning this lesson are program analysis techniques:

1. **Control Flow (data flow, dependence, etc.)**
2. **Conditional Constructs (if, switch, etc.)**
3. **Loop structures (nested, etc.)**
4. **Data structures (stacks, etc.)**

Tools are useful, but understanding the techniques are essential

Why Static Analysis?

- Can find precise location of vulnerabilities or capabilities
- Low-cost & low-threat
- Malware cannot evade if static
- Augments other sophisticated techniques (can help trouble shoot)

Why not Static Analysis?

- Time consuming
- Obfuscation and packing
- No run-time information
- Tedious
- Frustrating
- Makes me angry

Binary Dissasembly

Linear Sweep

1. Go through .text sections of the executable code and dissemble everything sequentially
2. Start at the first byte then decode each subsequent byte until an illegal instruction is reached
3. Susceptible to mistakes or intentionally placed bombs (who would do such a thing!?)

Objdump, WinDbg

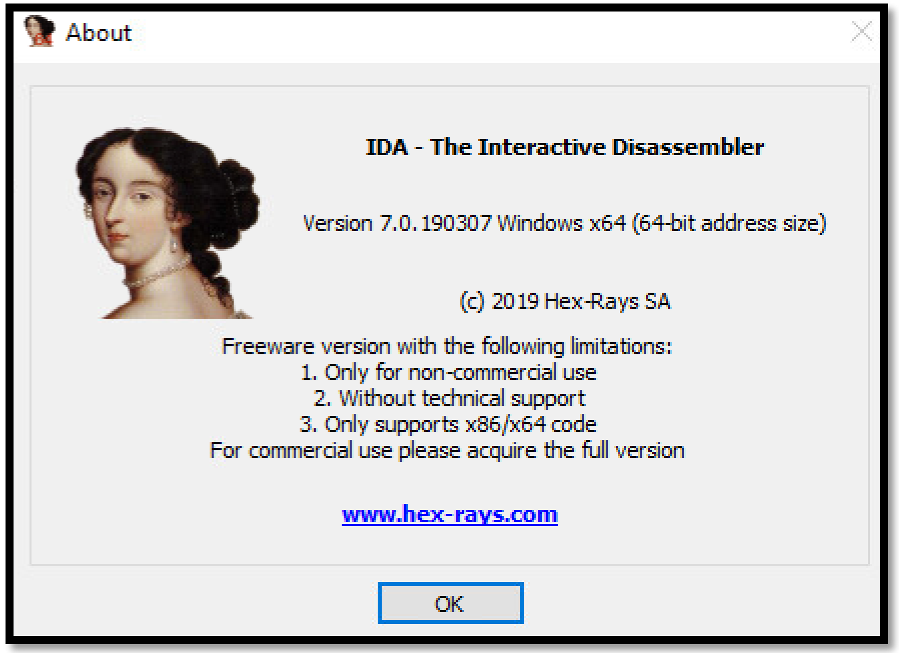
Recursive Descent

1. Each jump and branch instructions are followed
2. It's a linear sweep until a branch is encountered, then we follow it and linear sweep again
3. Susceptible to indirect jumps

Olly, IDA, Ghidra

Interactive DisAssembler (a tool)

- When IDA Pro starts it will ask you to start a new disassembly or open a previous one
- The defaults are almost always correct…unless you are dealing with nasty malware



IDA - The Interactive Disassembler

Version 7.0.190307 Windows x64 (64-bit address size)

(c) 2019 Hex-Rays SA

Freeware version with the following limitations:
1. Only for non-commercial use
2. Without technical support
3. Only supports x86/x64 code
For commercial use please acquire the full version

www.hex-rays.com

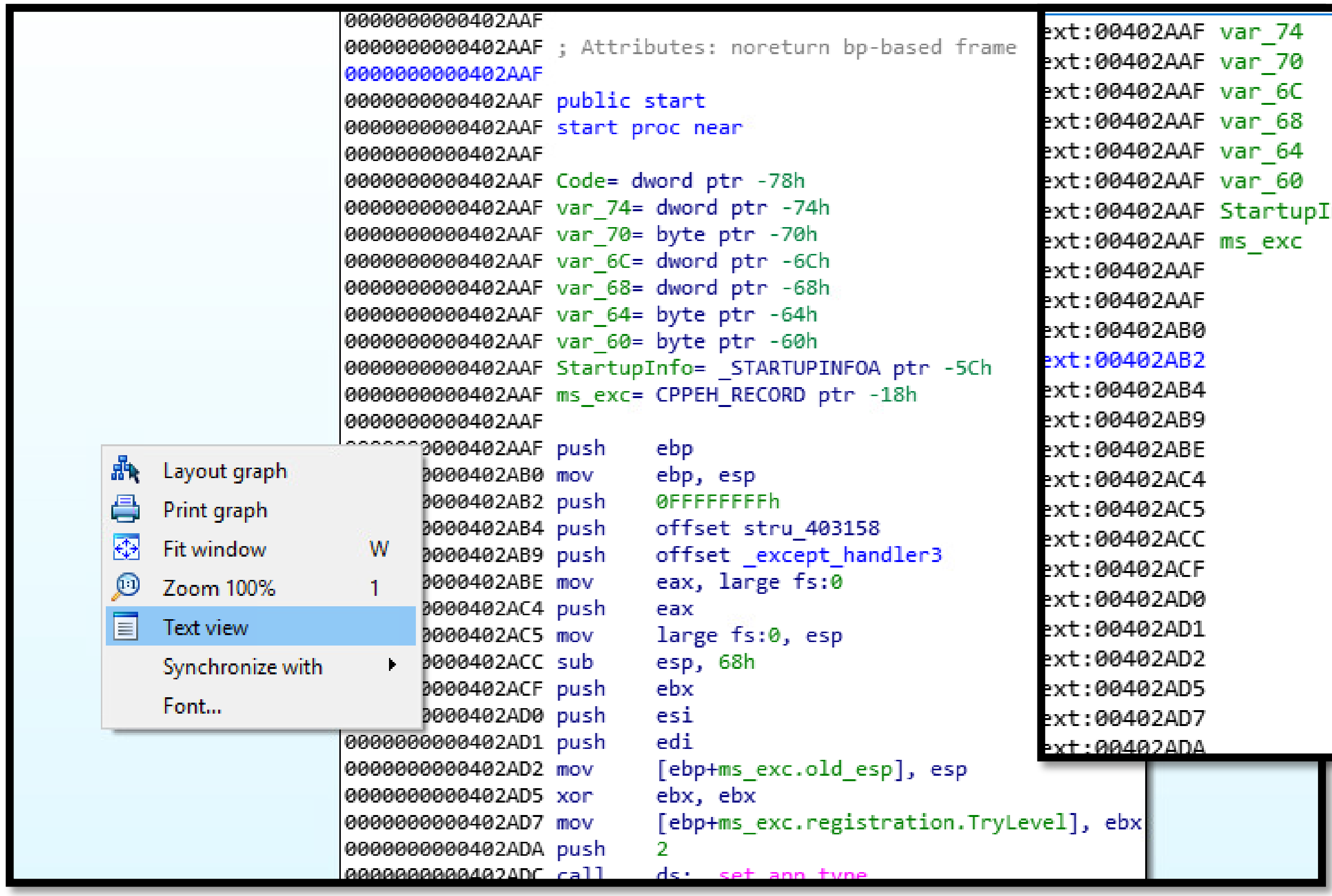
Greencat (APT 1)

https://github.com/fullerj/PMA/raw/main/webc2-greencat-2.7z

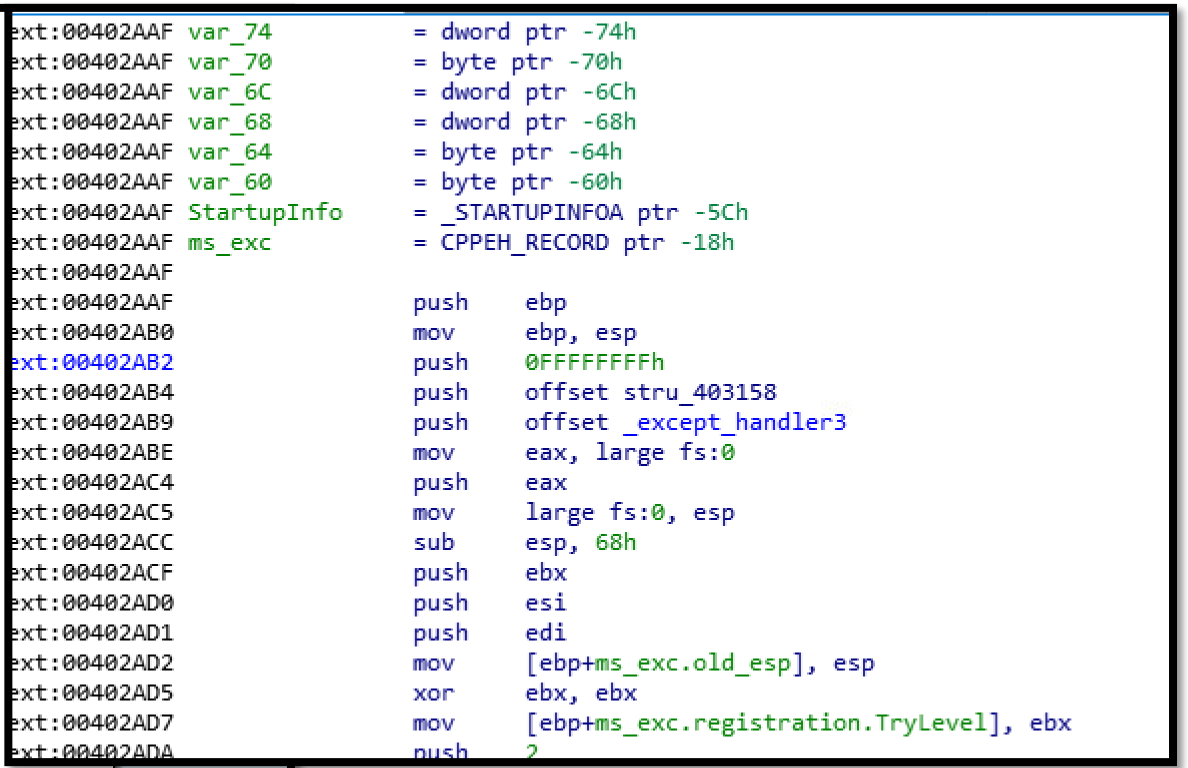
It's "infected", so be careful.

IDA will open in CFG view

- Right-click and select Text View for flat disassembled code (or Space bar)

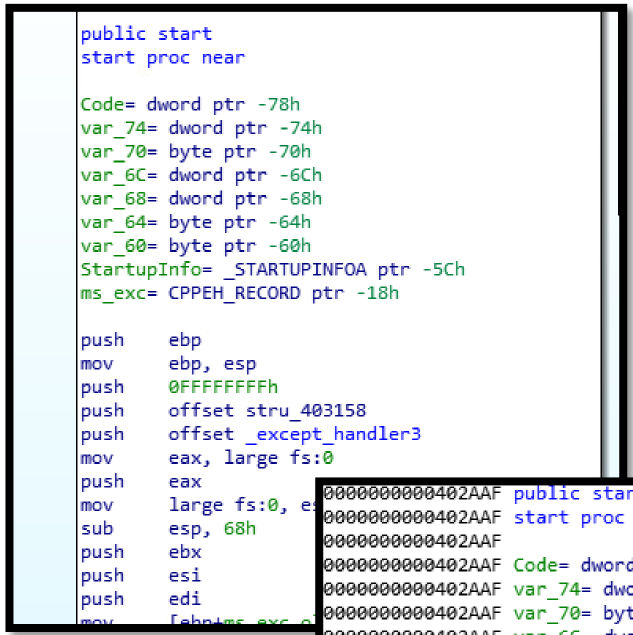


```
0000000000402AAF
0000000000402AAF ; Attributes: noreturn bp-based frame
0000000000402AAF
0000000000402AAF public start
0000000000402AAF start proc near
0000000000402AAF
0000000000402AAF Code= dword ptr -78h
0000000000402AAF var_74= dword ptr -74h
0000000000402AAF var_70= byte ptr -70h
0000000000402AAF var_6C= dword ptr -6Ch
0000000000402AAF var_68= dword ptr -68h
0000000000402AAF var_64= byte ptr -64h
0000000000402AAF var_60= byte ptr -60h
0000000000402AAF StartupInfo= _STARTUPINFOA ptr -5Ch
0000000000402AAF ms_exc= CPPEH_RECORD ptr -18h
0000000000402AAF
0000000000402AAF push    ebp
0000402AB0 mov     ebp, esp
0000402AB2 push    0FFFFFFFFh
0000402AB4 push    offset stru_403158
0000402AB9 push    offset _except_handler3
0000402ABE mov     eax, large fs:0
0000402AC4 push    eax
0000402AC5 mov     large fs:0, esp
0000402ACC sub     esp, 68h
0000402ACF push    ebx
0000402AD0 push    esi
0000000000402AD1 push    edi
0000000000402AD2 mov     [ebp+ms_exc.old_esp], esp
0000000000402AD5 xor     ebx, ebx
0000000000402AD7 mov     [ebp+ms_exc.registration.TryLevel], ebx
0000000000402ADA push    2
0000000000402ADC call    ds:__set_app_type
```

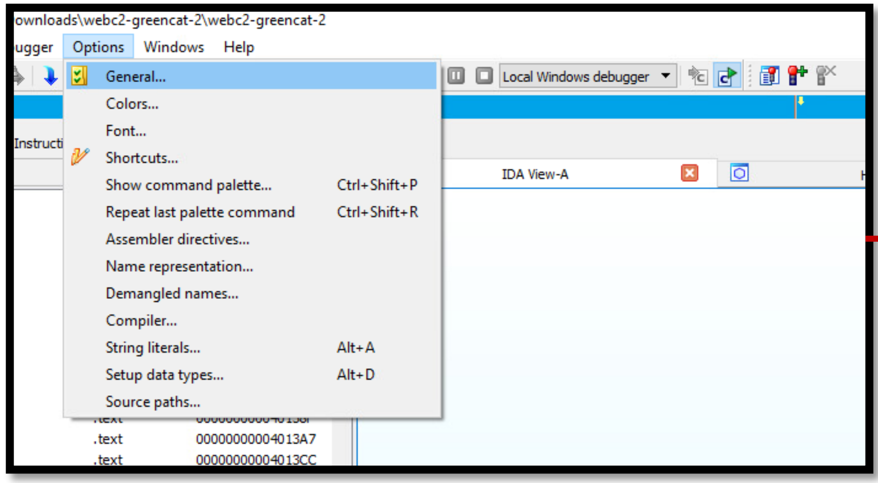


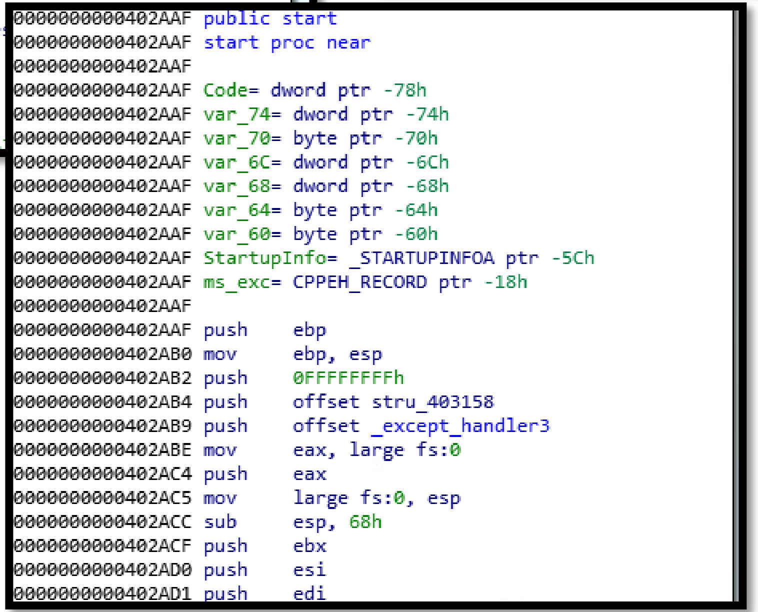
```
ext:00402AAF var_74         = dword ptr -74h
ext:00402AAF var_70         = byte ptr -70h
ext:00402AAF var_6C         = dword ptr -6Ch
ext:00402AAF var_68         = dword ptr -68h
ext:00402AAF var_64         = byte ptr -64h
ext:00402AAF var_60         = byte ptr -60h
ext:00402AAF StartupInfo    = _STARTUPINFOA ptr -5Ch
ext:00402AAF ms_exc         = CPPEH_RECORD ptr -18h
ext:00402AAF
ext:00402AAF                push    ebp
ext:00402AB0                mov     ebp, esp
ext:00402AB2                push    0FFFFFFFFh
ext:00402AB4                push    offset stru_403158
ext:00402AB9                push    offset _except_handler3
ext:00402ABE                mov     eax, large fs:0
ext:00402AC4                push    eax
ext:00402AC5                mov     large fs:0, esp
ext:00402ACC                sub     esp, 68h
ext:00402ACF                push    ebx
ext:00402AD0                push    esi
ext:00402AD1                push    edi
ext:00402AD2                mov     [ebp+ms_exc.old_esp], esp
ext:00402AD5                xor     ebx, ebx
ext:00402AD7                mov     [ebp+ms_exc.registration.TryLevel], ebx
ext:00402ADA                push    2
```

Pro tip #1: Add line prefixes

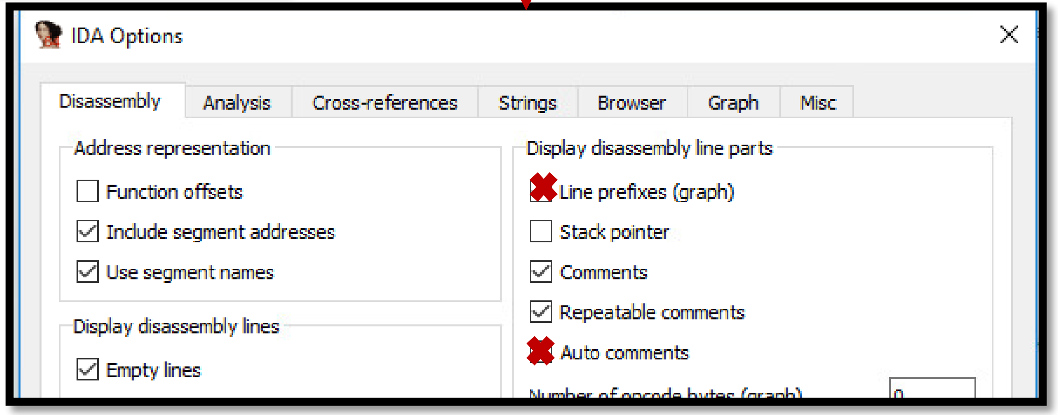


```
public start
start proc near

Code= dword ptr -78h
var_74= dword ptr -74h
var_70= byte ptr -70h
var_6C= dword ptr -6Ch
var_68= dword ptr -68h
var_64= byte ptr -64h
var_60= byte ptr -60h
StartupInfo= _STARTUPINFOA ptr -5Ch
ms_exc= CPPEH_RECORD ptr -18h

push    ebp
mov     ebp, esp
push    0FFFFFFFFh
push    offset stru_403158
push    offset _except_handler3
mov     eax, large fs:0
push    eax
mov     large fs:0, e
sub     esp, 68h
push    ebx
push    esi
push    edi
```





```
0000000000402AAF public start
0000000000402AAF start proc near
0000000000402AAF
0000000000402AAF Code= dword ptr -78h
0000000000402AAF var_74= dword ptr -74h
0000000000402AAF var_70= byte ptr -70h
0000000000402AAF var_6C= dword ptr -6Ch
0000000000402AAF var_68= dword ptr -68h
0000000000402AAF var_64= byte ptr -64h
0000000000402AAF var_60= byte ptr -60h
0000000000402AAF StartupInfo= _STARTUPINFOA ptr -5Ch
0000000000402AAF ms_exc= CPPEH_RECORD ptr -18h
0000000000402AAF
0000000000402AAF push    ebp
0000000000402AB0 mov     ebp, esp
0000000000402AB2 push    0FFFFFFFFh
0000000000402AB4 push    offset stru_403158
0000000000402AB9 push    offset _except_handler3
0000000000402ABE mov     eax, large fs:0
0000000000402AC4 push    eax
0000000000402AC5 mov     large fs:0, esp
0000000000402ACC sub     esp, 68h
0000000000402ACF push    ebx
0000000000402AD0 push    esi
0000000000402AD1 push    edi
```



IDA Options

Disassembly | Analysis | Cross-references | Strings | Browser | Graph | Misc

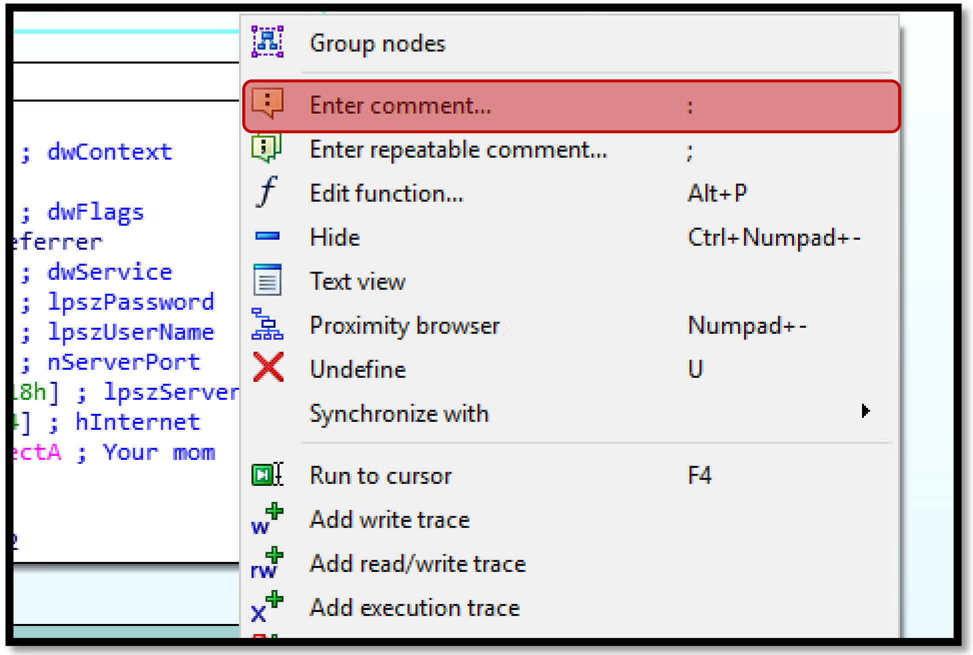
Address representation
- [ ] Function offsets
- [x] Include segment addresses
- [x] Use segment names

Display disassembly lines
- [x] Empty lines

Display disassembly line parts
- ✖ Line prefixes (graph)
- [ ] Stack pointer
- [x] Comments
- [x] Repeatable comments
- ✖ Auto comments

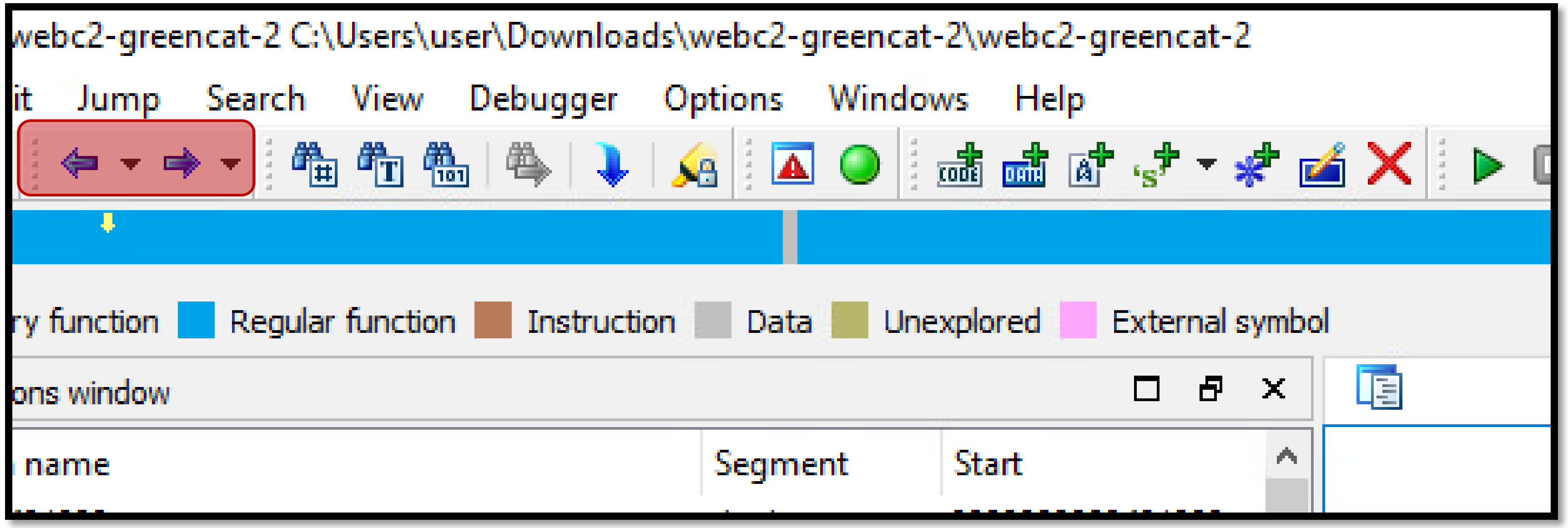
Number of opcode bytes (graph) 0

Pro tip #2: Right-click and select comments or press ":"



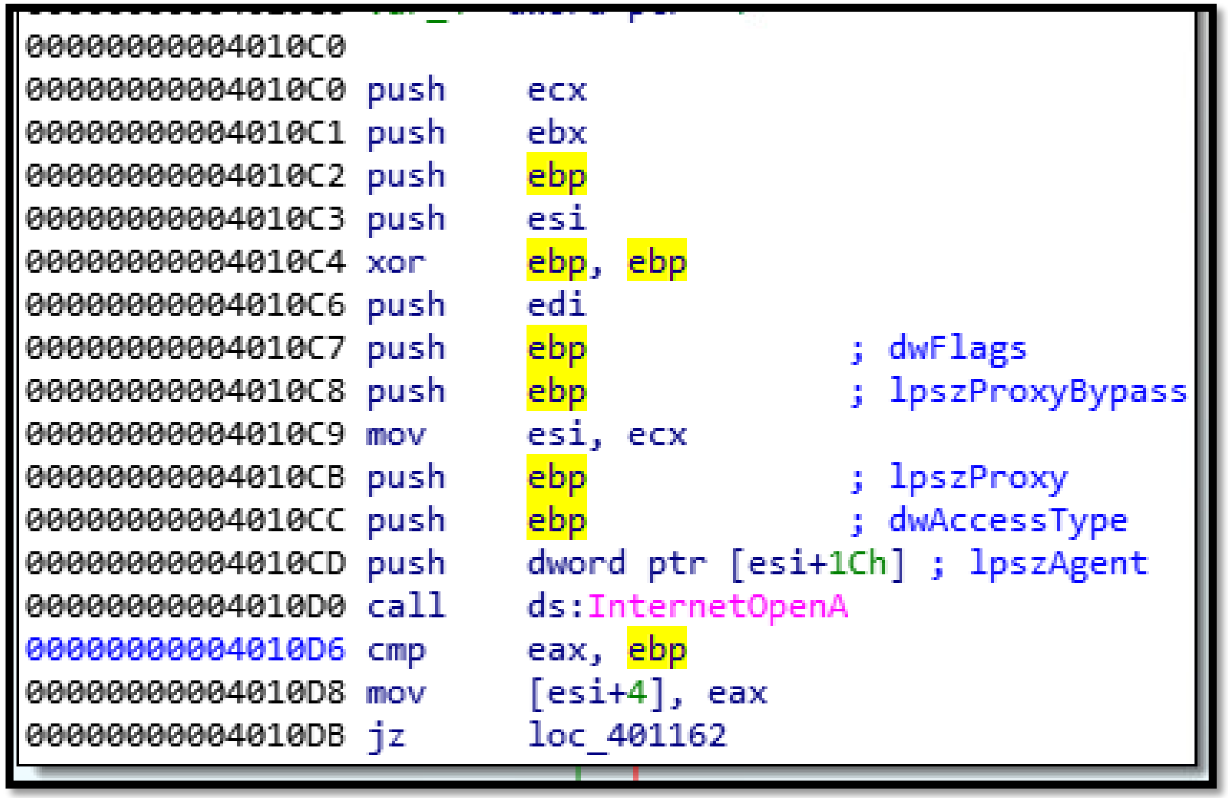
```
; dwContext

; dwFlags
eferrer
; dwService
; lpszPassword
; lpszUserName
; nServerPort
18h] ; lpszServer
4] ; hInternet
ectA ; Your mom
```

Pro tip #3: Wait, where was I?



Pro tip #4: Rename Symbolic Constants

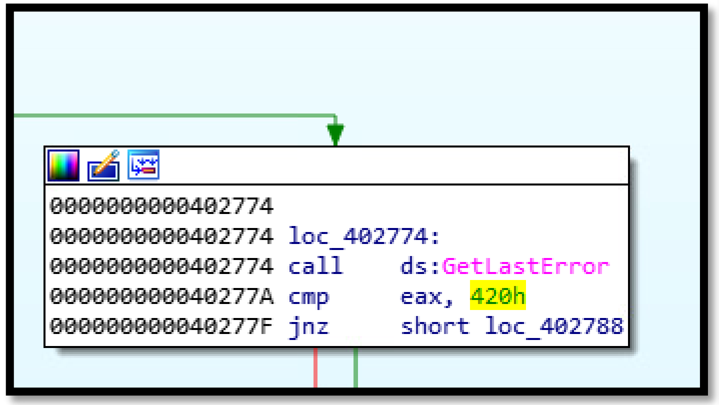
- IDA's FLIRT signatures know the arguments for common APIs



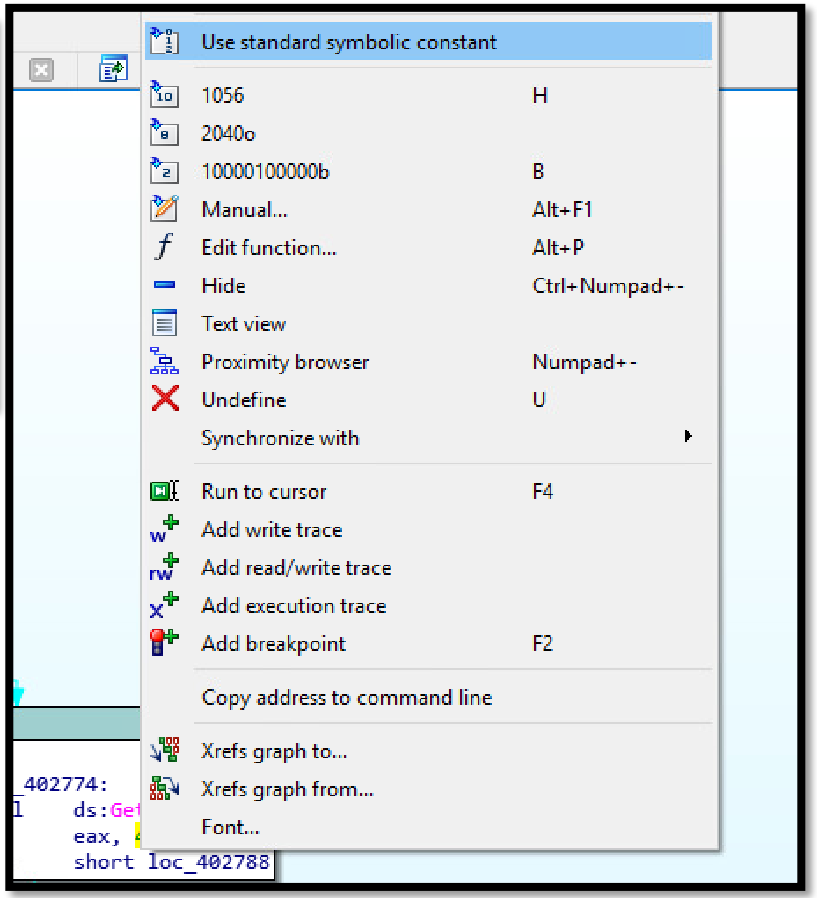
```
00000000004010C0
00000000004010C0 push    ecx
00000000004010C1 push    ebx
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
00000000004010C6 push    edi
00000000004010C7 push    ebp                 ; dwFlags
00000000004010C8 push    ebp                 ; lpszProxyBypass
00000000004010C9 mov     esi, ecx
00000000004010CB push    ebp                 ; lpszProxy
00000000004010CC push    ebp                 ; dwAccessType
00000000004010CD push    dword ptr [esi+1Ch] ; lpszAgent
00000000004010D0 call    ds:InternetOpenA
00000000004010D6 cmp     eax, ebp
00000000004010D8 mov     [esi+4], eax
00000000004010DB jz      loc_401162
```

Pro tip #4: Rename Symbolic Constants

- IDA's FLIRT signatures know the arguments for common APIs
- IDA also knows symbolic names for defined constants! Just tell IDA what to look for



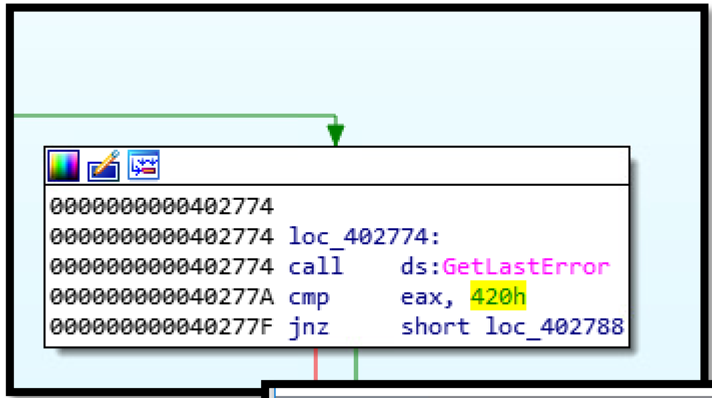
```
0000000000402774
0000000000402774 loc_402774:
0000000000402774 call    ds:GetLastError
000000000040277A cmp     eax, 420h
000000000040277F jnz     short loc_402788
```



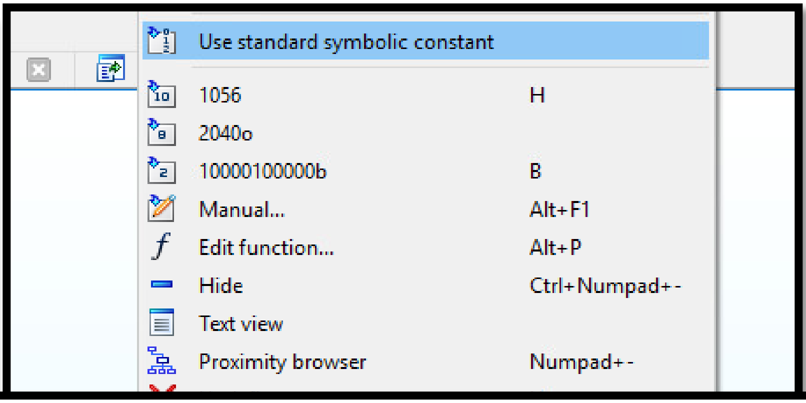
https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes--1000-1299-

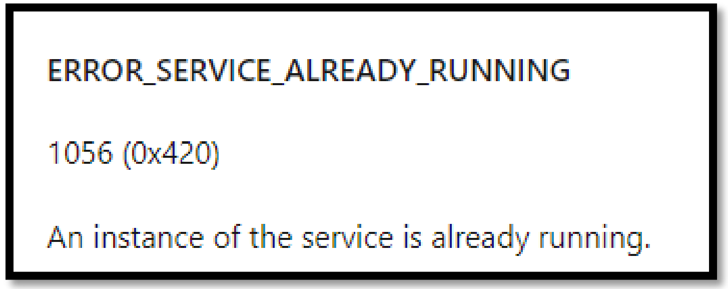
Pro tip #4: Rename Symbolic Constants

- IDA's FLIRT signatures know the arguments for common APIs
- IDA also knows symbolic names for defined constants! Just tell IDA what to look for



```
0000000000402774
0000000000402774 loc_402774:
0000000000402774 call    ds:GetLastError
000000000040277A cmp     eax, 420h
000000000040277F jnz     short loc_402788
```

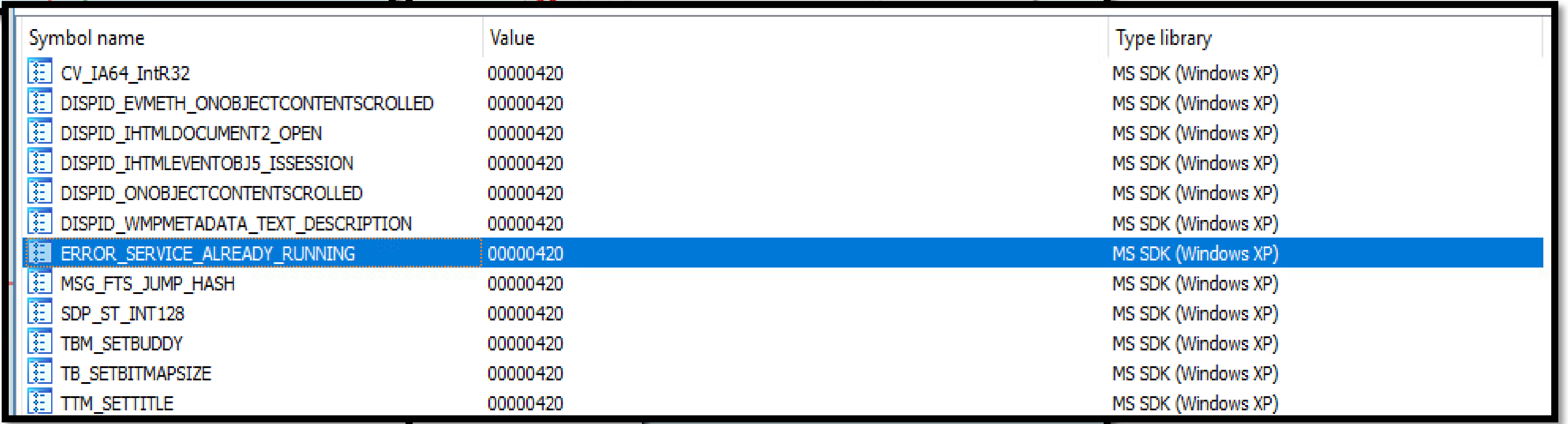




ERROR_SERVICE_ALREADY_RUNNING

1056 (0x420)

An instance of the service is already running.

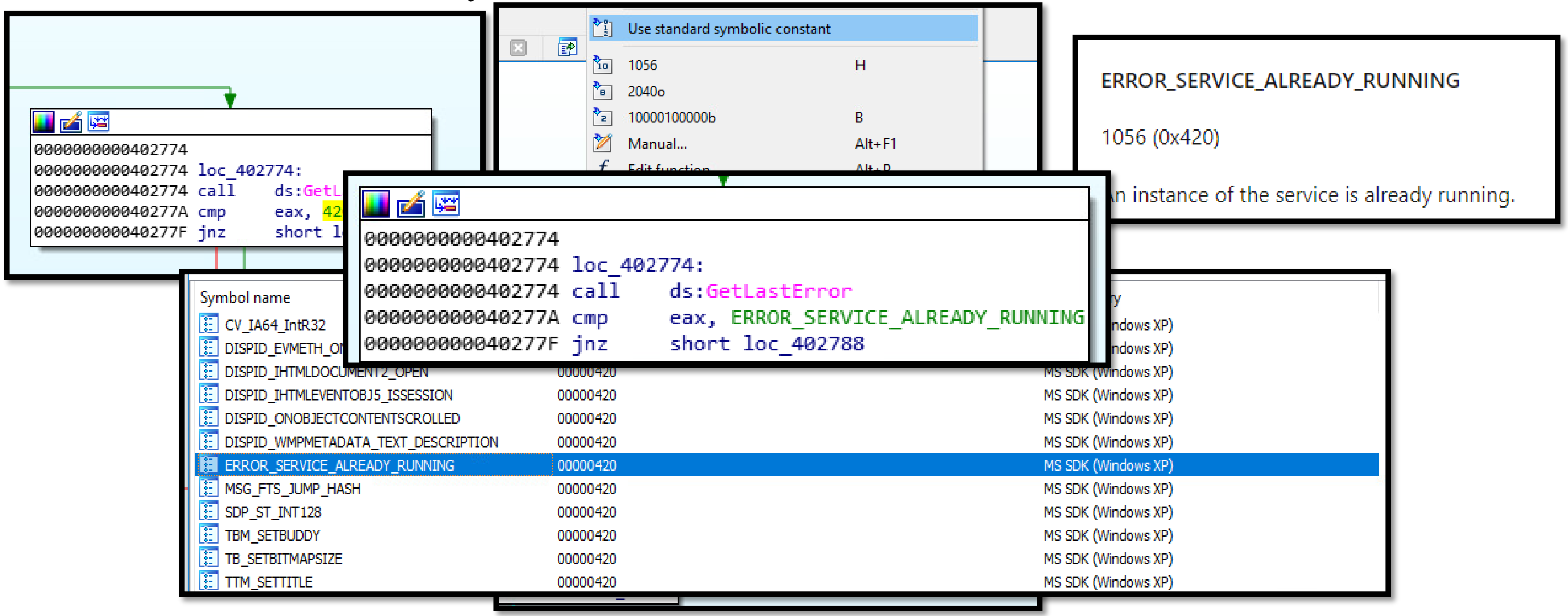


Symbol name	Value	Type library
CV_IA64_IntR32	00000420	MS SDK (Windows XP)
DISPID_EVMETH_ONOBJECTCONTENTSCROLLED	00000420	MS SDK (Windows XP)
DISPID_IHTMLDOCUMENT2_OPEN	00000420	MS SDK (Windows XP)
DISPID_IHTMLEVENTOBJ5_ISSESSION	00000420	MS SDK (Windows XP)
DISPID_ONOBJECTCONTENTSCROLLED	00000420	MS SDK (Windows XP)
DISPID_WMPMETADATA_TEXT_DESCRIPTION	00000420	MS SDK (Windows XP)
ERROR_SERVICE_ALREADY_RUNNING	00000420	MS SDK (Windows XP)
MSG_FTS_JUMP_HASH	00000420	MS SDK (Windows XP)
SDP_ST_INT128	00000420	MS SDK (Windows XP)
TBM_SETBUDDY	00000420	MS SDK (Windows XP)
TB_SETBITMAPSIZE	00000420	MS SDK (Windows XP)
TTM_SETTITLE	00000420	MS SDK (Windows XP)

https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes--1000-1299-

Pro tip #4: Rename Symbolic Constants

- IDA's FLIRT signatures know the arguments for common APIs
- IDA also knows symbolic names for defined constants! Just tell IDA what to look for



```
0000000000402774
0000000000402774 loc_402774:
0000000000402774 call    ds:GetL
000000000040277A cmp     eax, 42
000000000040277F jnz     short l
```

ERROR_SERVICE_ALREADY_RUNNING

1056 (0x420)

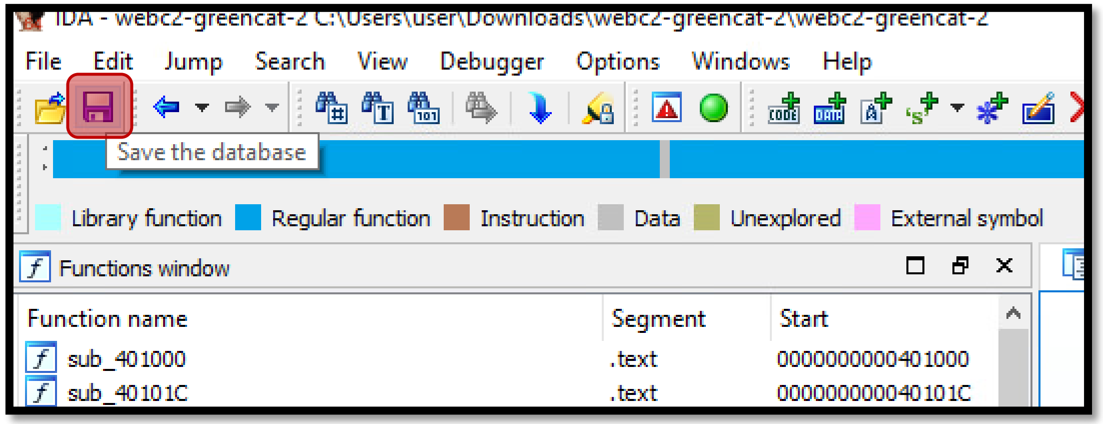
n instance of the service is already running.

```
0000000000402774
0000000000402774 loc_402774:
0000000000402774 call    ds:GetLastError
000000000040277A cmp     eax, ERROR_SERVICE_ALREADY_RUNNING
000000000040277F jnz     short loc_402788
```

Symbol name		y
CV_IA64_IntR32		ndows XP)
DISPID_EVMETH_O		ndows XP)
DISPID_IHTMLDOCUMENT2_OPEN	00000420	MS SDK (Windows XP)
DISPID_IHTMLEVENTOBJ5_ISSESSION	00000420	MS SDK (Windows XP)
DISPID_ONOBJECTCONTENTSCROLLED	00000420	MS SDK (Windows XP)
DISPID_WMPMETADATA_TEXT_DESCRIPTION	00000420	MS SDK (Windows XP)
ERROR_SERVICE_ALREADY_RUNNING	00000420	MS SDK (Windows XP)
MSG_FTS_JUMP_HASH	00000420	MS SDK (Windows XP)
SDP_ST_INT128	00000420	MS SDK (Windows XP)
TBM_SETBUDDY	00000420	MS SDK (Windows XP)
TB_SETBITMAPSIZE	00000420	MS SDK (Windows XP)
TTM_SETTITLE	00000420	MS SDK (Windows XP)

Pro tip #5: Save often!

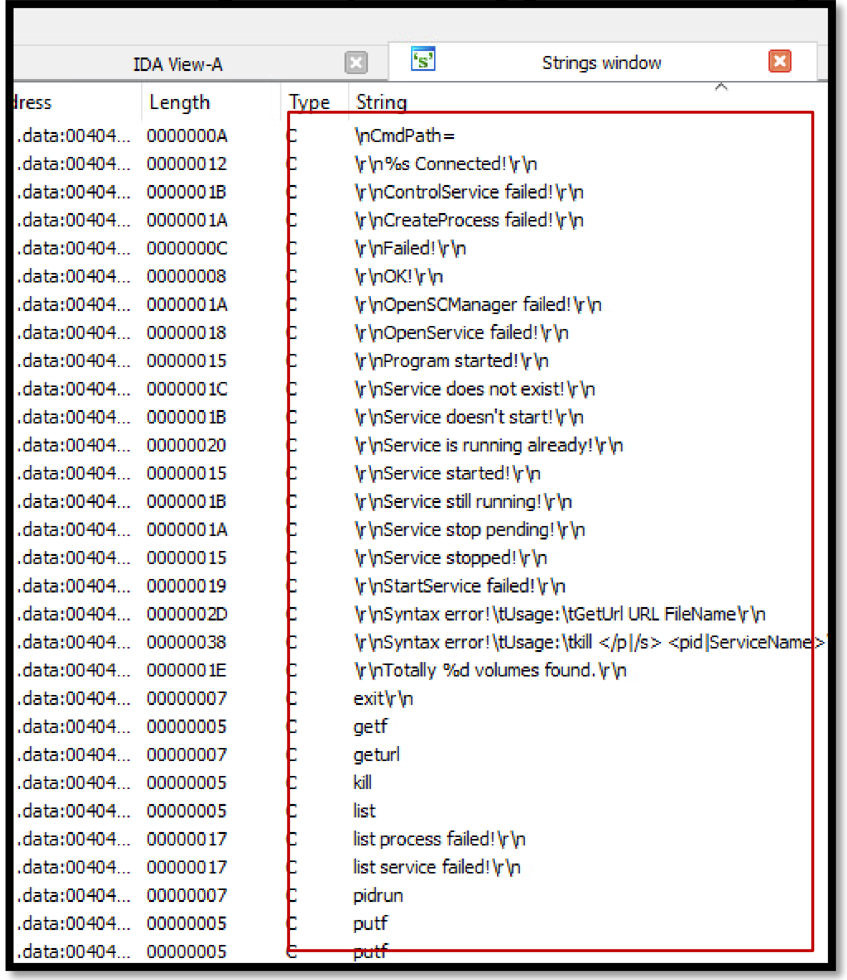
- Losing hours of reverse engineering can be hazardous to your health!


IDA - webc2-greencat-2 C:\Users\user\Downloads\webc2-greencat-2\webc2-greencat-2
File Edit Jump Search View Debugger Options Windows Help
Save the database
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window

Function name	Segment	Start
sub_401000	.text	0000000000401000
sub_40101C	.text	000000000040101C

Pro tip #6: Where to begin?

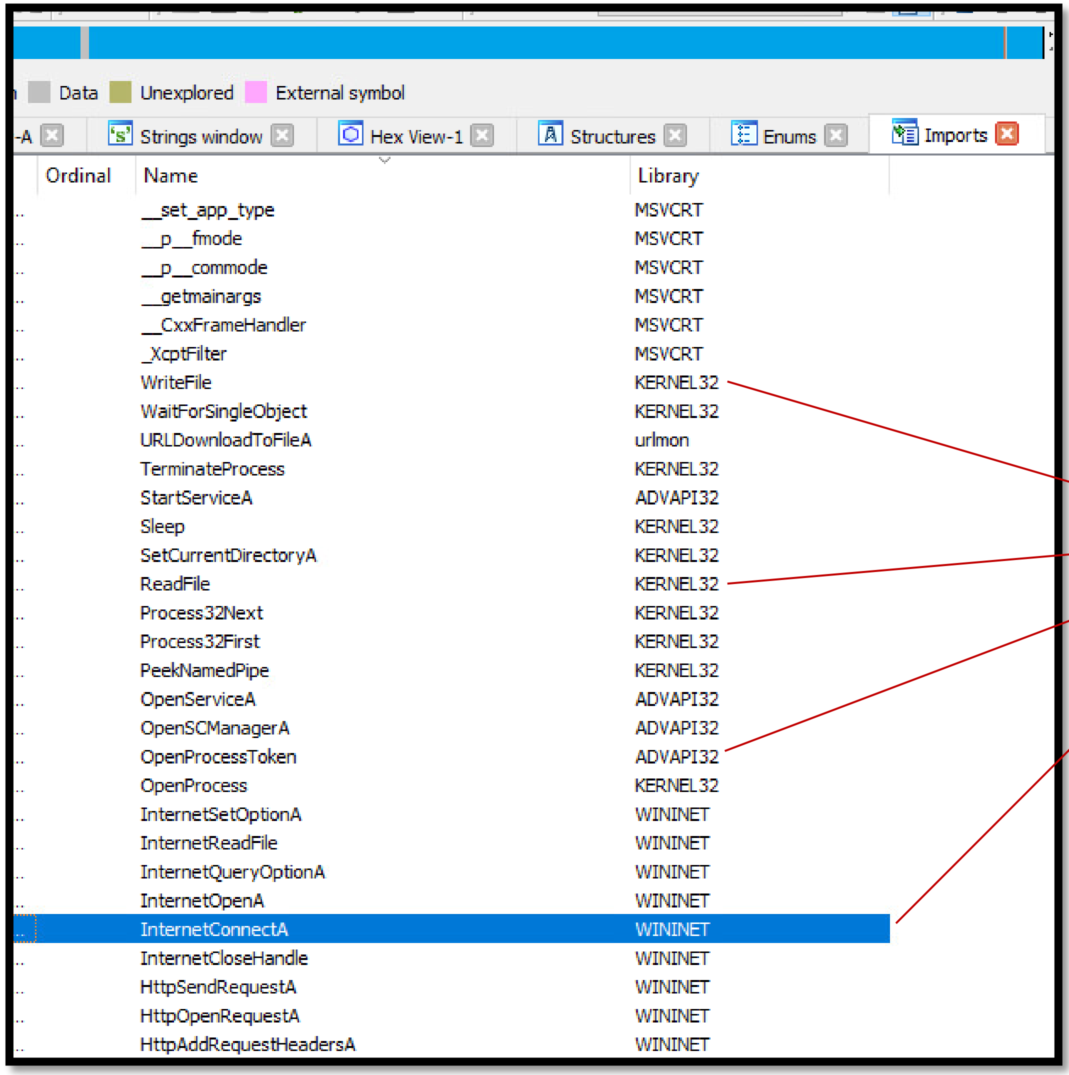
- Strings -- Quick approach to high-level overview (Shift-F12)



dress	Length	Type	String
.data:00404...	0000000A	C	\nCmdPath=
.data:00404...	00000012	C	\r\n%s Connected!\r\n
.data:00404...	0000001B	C	\r\nControlService failed!\r\n
.data:00404...	0000001A	C	\r\nCreateProcess failed!\r\n
.data:00404...	0000000C	C	\r\nFailed!\r\n
.data:00404...	00000008	C	\r\nOK!\r\n
.data:00404...	0000001A	C	\r\nOpenSCManager failed!\r\n
.data:00404...	00000018	C	\r\nOpenService failed!\r\n
.data:00404...	00000015	C	\r\nProgram started!\r\n
.data:00404...	0000001C	C	\r\nService does not exist!\r\n
.data:00404...	0000001B	C	\r\nService doesn't start!\r\n
.data:00404...	00000020	C	\r\nService is running already!\r\n
.data:00404...	00000015	C	\r\nService started!\r\n
.data:00404...	0000001B	C	\r\nService still running!\r\n
.data:00404...	0000001A	C	\r\nService stop pending!\r\n
.data:00404...	00000015	C	\r\nService stopped!\r\n
.data:00404...	00000019	C	\r\nStartService failed!\r\n
.data:00404...	0000002D	C	\r\nSyntax error!\tUsage:\tGetUrl URL FileName\r\n
.data:00404...	00000038	C	\r\nSyntax error!\tUsage:\tkill </p|/s> <pid|ServiceName>
.data:00404...	0000001E	C	\r\nTotally %d volumes found.\r\n
.data:00404...	00000007	C	exit\r\n
.data:00404...	00000005	C	getf
.data:00404...	00000007	C	geturl
.data:00404...	00000005	C	kill
.data:00404...	00000005	C	list
.data:00404...	00000017	C	list process failed!\r\n
.data:00404...	00000017	C	list service failed!\r\n
.data:00404...	00000007	C	pidrun
.data:00404...	00000005	C	putf
.data:00404...	00000005	C	putf

What are those??!!?!?!??!??!!

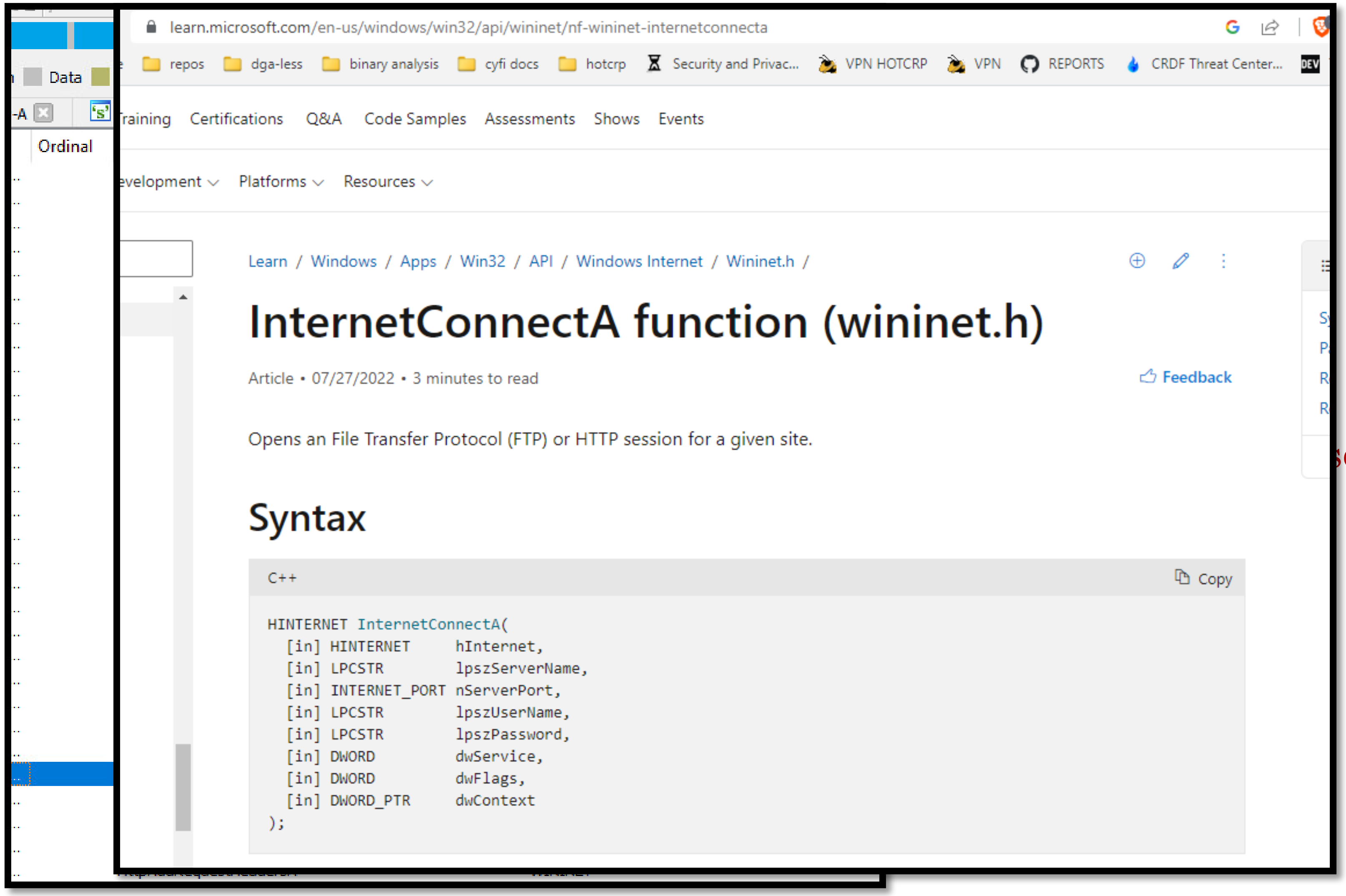
Pro tip #6: Where to begin? IAT



Ordinal	Name	Library
	__set_app_type	MSVCRT
	__p__fmode	MSVCRT
	__p__commode	MSVCRT
	__getmainargs	MSVCRT
	__CxxFrameHandler	MSVCRT
	_XcptFilter	MSVCRT
	WriteFile	KERNEL32
	WaitForSingleObject	KERNEL32
	URLDownloadToFileA	urlmon
	TerminateProcess	KERNEL32
	StartServiceA	ADVAPI32
	Sleep	KERNEL32
	SetCurrentDirectoryA	KERNEL32
	ReadFile	KERNEL32
	Process32Next	KERNEL32
	Process32First	KERNEL32
	PeekNamedPipe	KERNEL32
	OpenServiceA	ADVAPI32
	OpenSCManagerA	ADVAPI32
	OpenProcessToken	ADVAPI32
	OpenProcess	KERNEL32
	InternetSetOptionA	WININET
	InternetReadFile	WININET
	InternetQueryOptionA	WININET
	InternetOpenA	WININET
	InternetConnectA	WININET
	InternetCloseHandle	WININET
	HttpSendRequestA	WININET
	HttpOpenRequestA	WININET
	HttpAddRequestHeadersA	WININET

What do these do?

Pro tip #6: Where to begin? IAT



InternetConnectA function (wininet.h)

Article • 07/27/2022 • 3 minutes to read

Opens an File Transfer Protocol (FTP) or HTTP session for a given site.

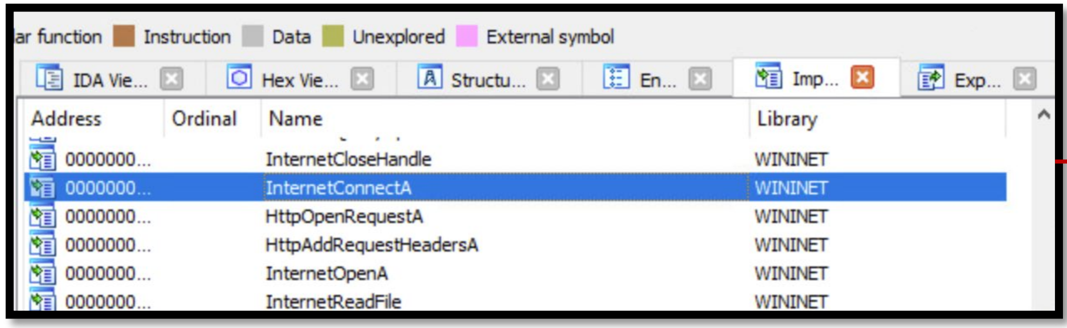
Syntax

C++

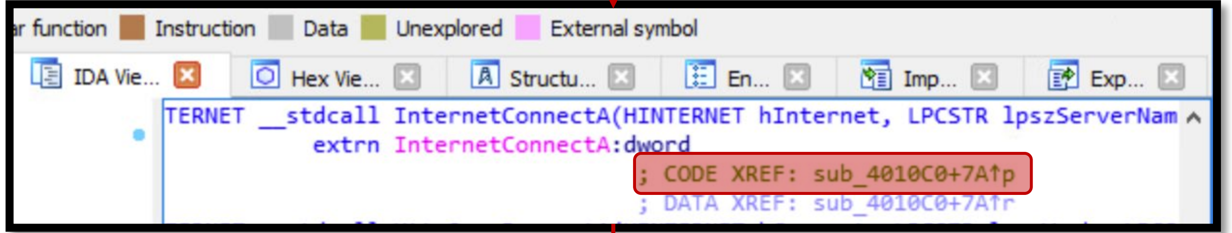
```
HINTERNET InternetConnectA(
  [in] HINTERNET     hInternet,
  [in] LPCSTR        lpszServerName,
  [in] INTERNET_PORT nServerPort,
  [in] LPCSTR        lpszUserName,
  [in] LPCSTR        lpszPassword,
  [in] DWORD         dwService,
  [in] DWORD         dwFlags,
  [in] DWORD_PTR     dwContext
);
```

se do?

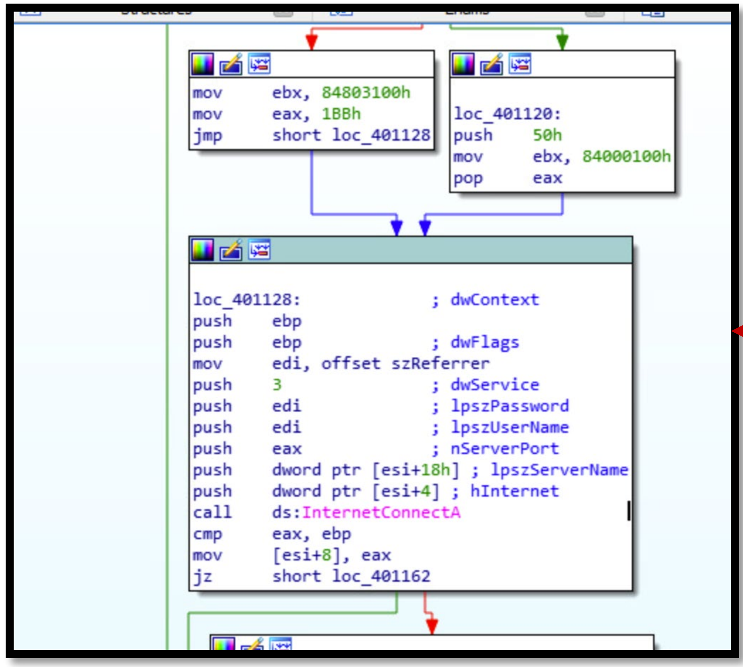
Pro tip #6: IAT Motivated Search



Address	Ordinal	Name	Library
0000000...		InternetCloseHandle	WININET
0000000...		InternetConnectA	WININET
0000000...		HttpOpenRequestA	WININET
0000000...		HttpAddRequestHeadersA	WININET
0000000...		InternetOpenA	WININET
0000000...		InternetReadFile	WININET



```
TERNET __stdcall InternetConnectA(HINTERNET hInternet, LPCSTR lpszServerNam
          extrn InternetConnectA:dword
                                        ; CODE XREF: sub_4010C0+7A↑p
                                        ; DATA XREF: sub_4010C0+7A↑r
```



```
mov     ebx, 84803100h
mov     eax, 1BBh
jmp     short loc_401128
```

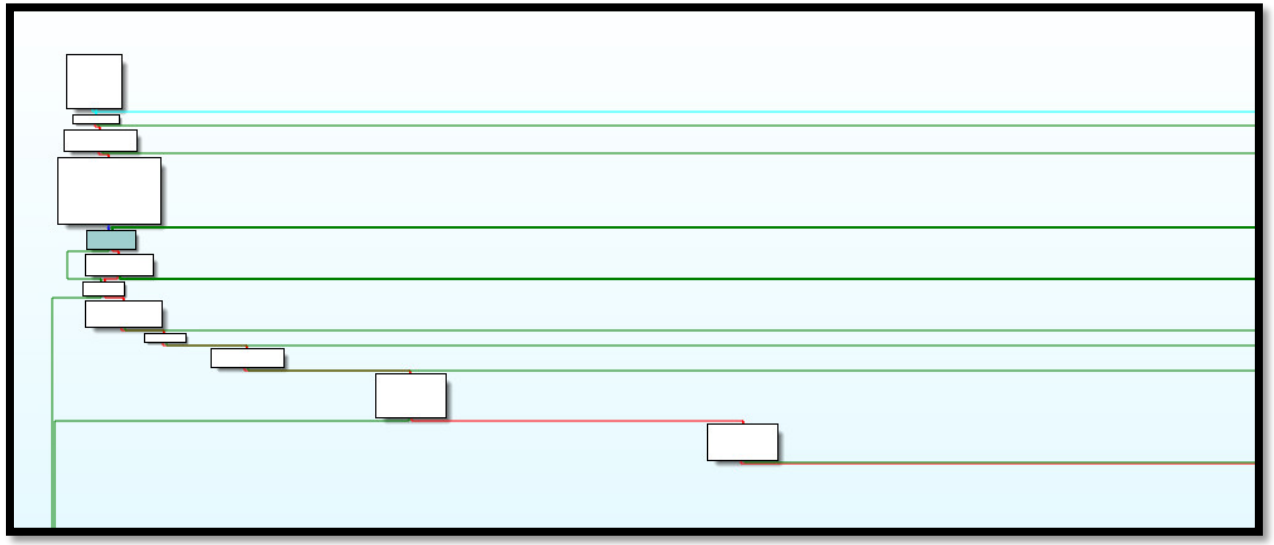
```
loc_401120:
push    50h
mov     ebx, 84000100h
pop     eax
```

```
loc_401128:             ; dwContext
push    ebp
push    ebp             ; dwFlags
mov     edi, offset szReferrer
push    3               ; dwService
push    edi             ; lpszPassword
push    edi             ; lpszUserName
push    eax             ; nServerPort
push    dword ptr [esi+18h] ; lpszServerName
push    dword ptr [esi+4] ; hInternet
call    ds:InternetConnectA
cmp     eax, ebp
mov     [esi+8], eax
jz      short loc_401162
```

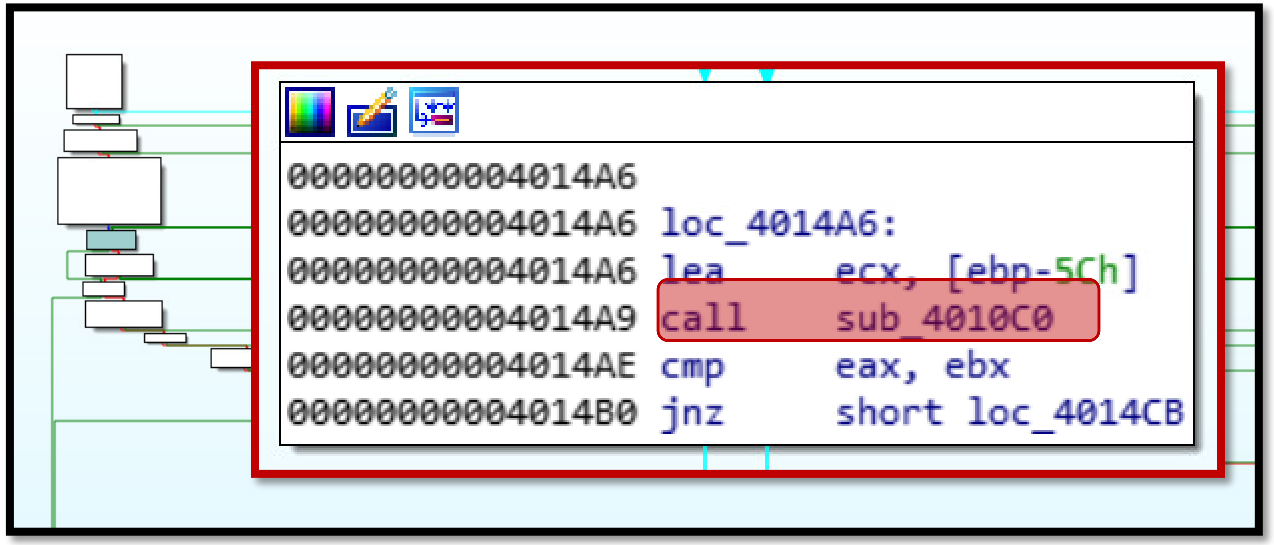
Recursive Descent RE?

- How should we go about REing malware?
- We can learn from recursive descent disassembly
- Follow a path to the end
  - Through switch/case statements
  - Through loops
  - Across multiple functions
- Create labels
  - Name function
  - Name variables
  - Comment where you’re fairly sure

Let's try `0x401406`

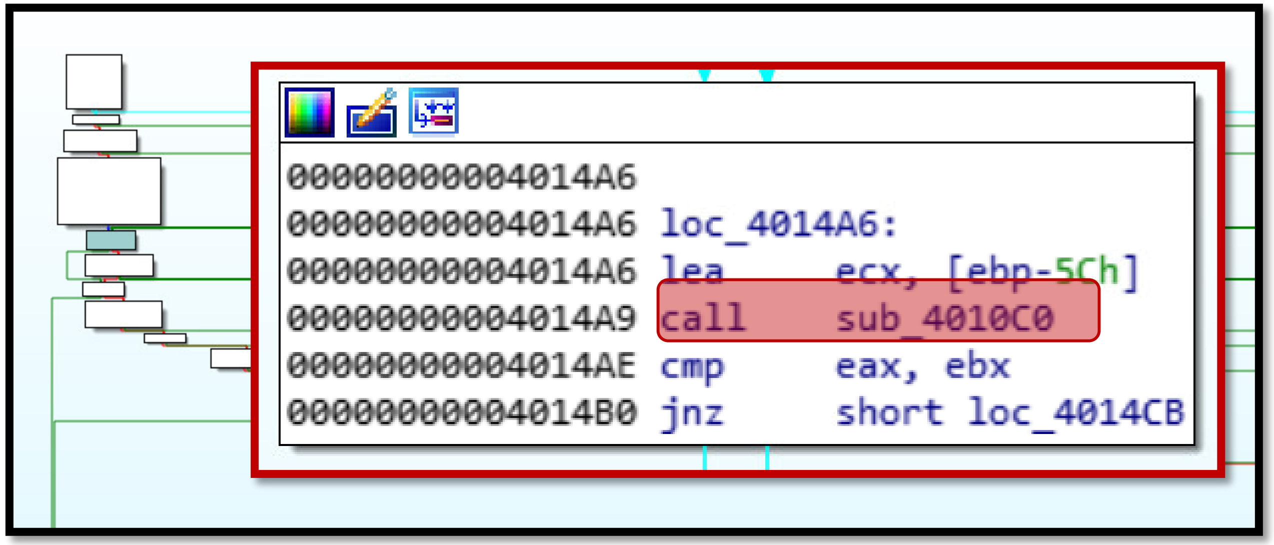


Let's try `0x401406`

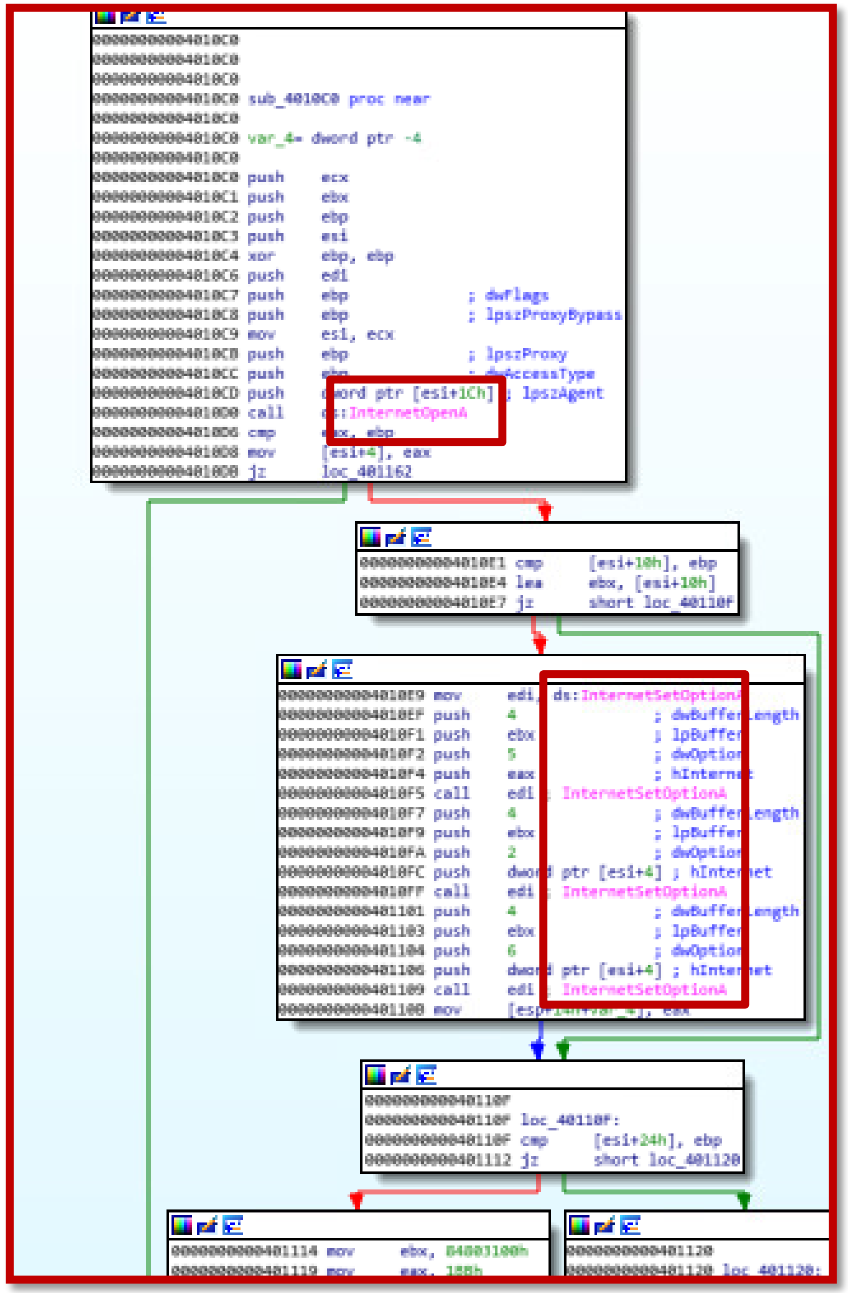


```
00000000004014A6
00000000004014A6 loc_4014A6:
00000000004014A6 lea     ecx, [ebp-5Ch]
00000000004014A9 call    sub_4010C0
00000000004014AE cmp     eax, ebx
00000000004014B0 jnz     short loc_4014CB
```

Let's try 0x401406



```
00000000004014A6
00000000004014A6 loc_4014A6:
00000000004014A6 lea     ecx, [ebp-5Ch]
00000000004014A9 call    sub_4010C0
00000000004014AE cmp     eax, ebx
00000000004014B0 jnz     short loc_4014CB
```



```
00000000004010C0
00000000004010C0
00000000004010C0
00000000004010C0 sub_4010C0 proc near
00000000004010C0
00000000004010C0 var_4= dword ptr -4
00000000004010C0
00000000004010C0 push    ecx
00000000004010C1 push    ebx
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
00000000004010C6 push    edi
00000000004010C7 push    ebp                     ; dwFlags
00000000004010C8 push    ebp                     ; lpszProxyBypass
00000000004010C9 mov     esi, ecx
00000000004010CB push    ebp                     ; lpszProxy
00000000004010CC push    ebp                     ; dwAccessType
00000000004010CD push    dword ptr [esi+1Ch] ; lpszAgent
00000000004010D0 call    ds:InternetOpenA
00000000004010D6 cmp     eax, ebp
00000000004010D8 mov     [esi+4], eax
00000000004010DB jz      loc_401162
```

```
00000000004010E1 cmp     [esi+10h], ebp
00000000004010E4 lea     ebx, [esi+10h]
00000000004010E7 jz      short loc_40110F
```

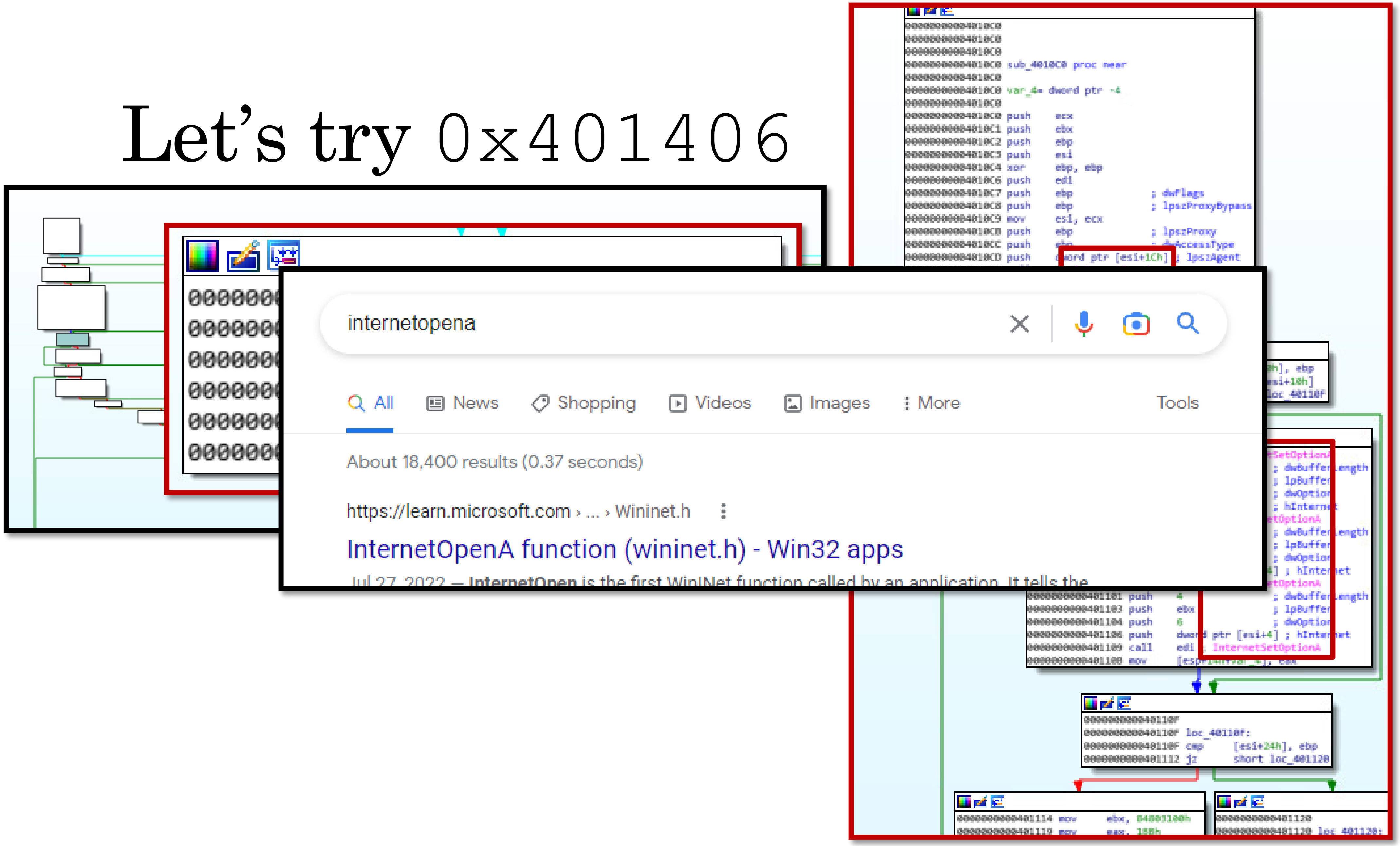
```
00000000004010E9 mov     edi, ds:InternetSetOptionA
00000000004010EF push    4                  ; dwBufferLength
00000000004010F1 push    ebx                ; lpBuffer
00000000004010F2 push    5                  ; dwOption
00000000004010F4 push    eax                ; hInternet
00000000004010F5 call    edi ; InternetSetOptionA
00000000004010F7 push    4                  ; dwBufferLength
00000000004010F9 push    ebx                ; lpBuffer
00000000004010FA push    2                  ; dwOption
00000000004010FC push    dword ptr [esi+4] ; hInternet
00000000004010FF call    edi ; InternetSetOptionA
0000000000401101 push    4                  ; dwBufferLength
0000000000401103 push    ebx                ; lpBuffer
0000000000401104 push    6                  ; dwOption
0000000000401106 push    dword ptr [esi+4] ; hInternet
0000000000401109 call    edi ; InternetSetOptionA
000000000040110B mov     [esp+14h+var_4], eax
```

```
000000000040110F
000000000040110F loc_40110F:
000000000040110F cmp     [esi+24h], ebp
0000000000401112 jz      short loc_401120
```

```
0000000000401114 mov     ebx, 84803100h
0000000000401119 mov     eax, 1BBh
```

```
0000000000401120
0000000000401120 loc_401120:
```

Let’s try `0x401406`



```
00000000004010C0
00000000004010C0
00000000004010C0
00000000004010C0 sub_4010C0 proc near
00000000004010C0
00000000004010C0 var_4= dword ptr -4
00000000004010C0
00000000004010C0 push    ecx
00000000004010C1 push    ebx
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
00000000004010C6 push    edi
00000000004010C7 push    ebp                     ; dwFlags
00000000004010C8 push    ebp                     ; lpszProxyBypass
00000000004010C9 mov     esi, ecx
00000000004010CB push    ebp                     ; lpszProxy
00000000004010CC push    ebp                     ; dwAccessType
00000000004010CD push    dword ptr [esi+1Ch]     ; lpszAgent
```

internetopena

All News Shopping Videos Images More Tools

About 18,400 results (0.37 seconds)

https://learn.microsoft.com › ... › Wininet.h

InternetOpenA function (wininet.h) - Win32 apps

Jul 27, 2022 — InternetOpen is the first WinINet function called by an application. It tells the

```
0h], ebp
esi+10h]
loc_40110F
```

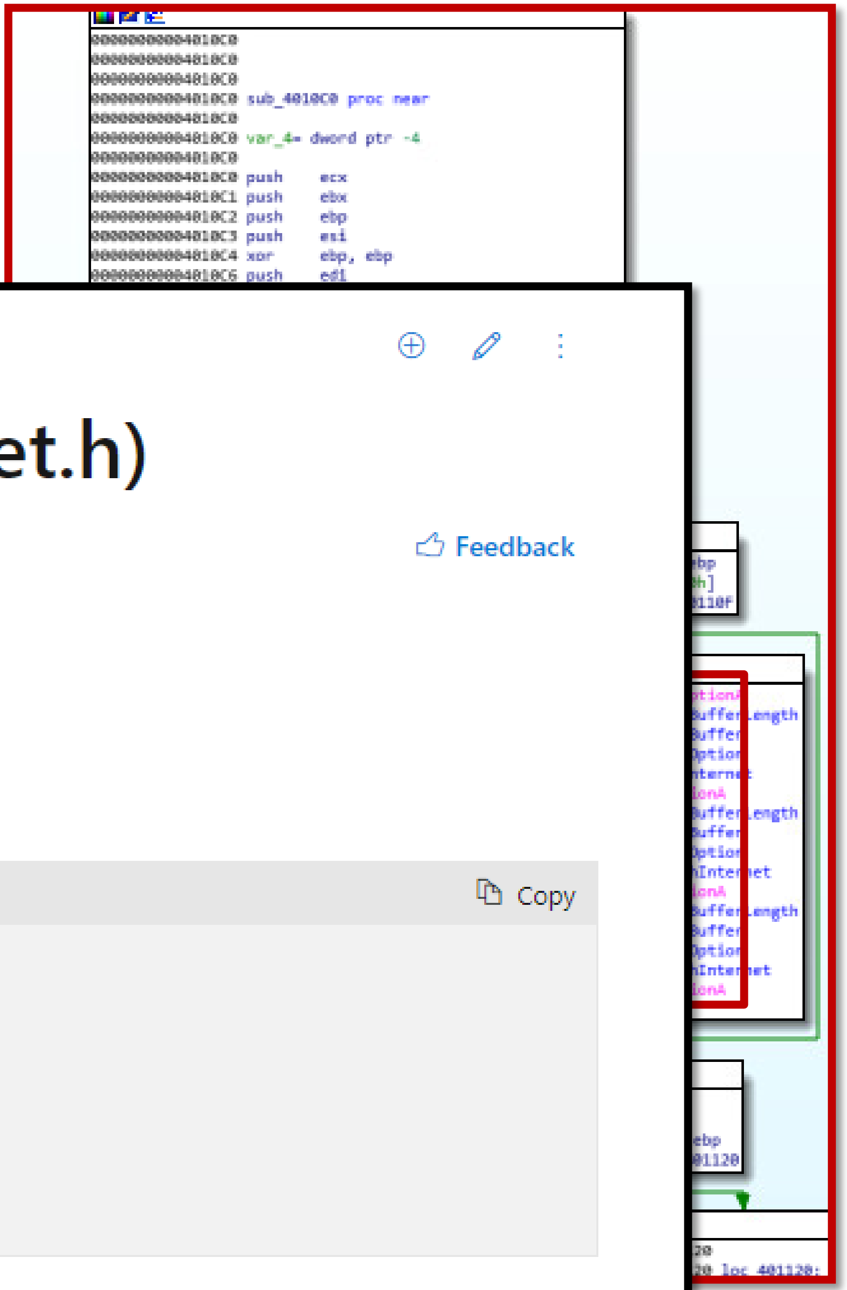
```
tSetOptionA
       ; dwBufferLength
       ; lpBuffer
       ; dwOption
       ; hInternet
etOptionA
       ; dwBufferLength
       ; lpBuffer
       ; dwOption
4] ; hInternet
etOptionA
0000000000401101 push    4                       ; dwBufferLength
0000000000401103 push    ebx                     ; lpBuffer
0000000000401104 push    6                       ; dwOption
0000000000401106 push    dword ptr [esi+4] ; hInternet
0000000000401109 call    edi ; InternetSetOptionA
000000000040110B mov     [esp+14h+var_4], eax
```

```
000000000040110F
000000000040110F loc_40110F:
000000000040110F cmp     [esi+24h], ebp
0000000000401112 jz      short loc_401120
```

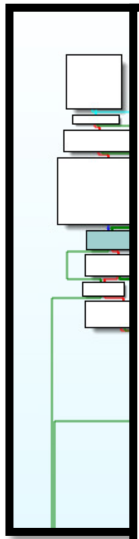
```
0000000000401114 mov     ebx, 84803100h
0000000000401119 mov     eax, 180h
```

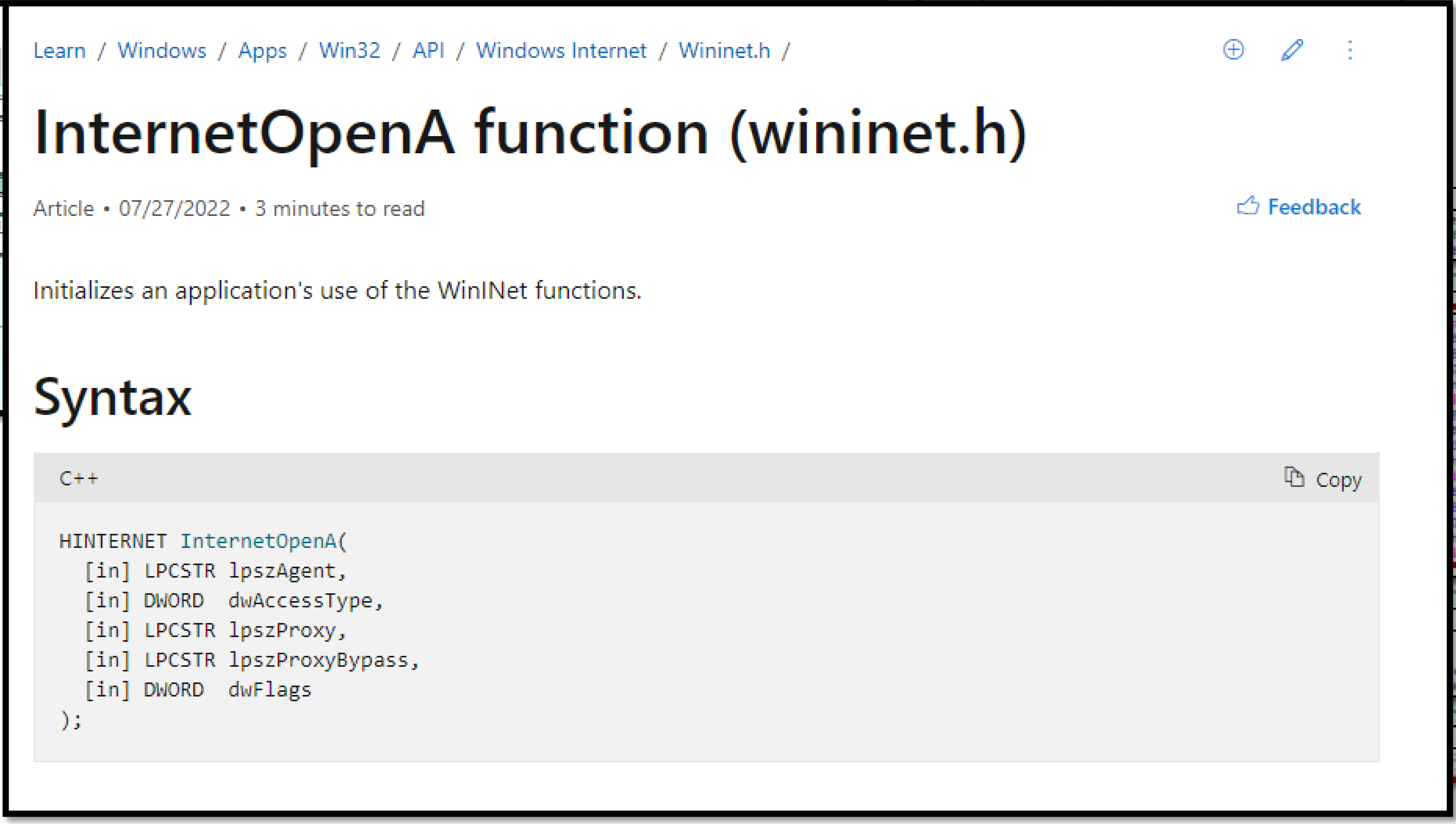
```
0000000000401120
0000000000401120 loc_401120:
```

Let's try `0x401406`



```
00000000004010C0
00000000004010C0
00000000004010C0
00000000004010C0 sub_4010C0 proc near
00000000004010C0
00000000004010C0 var_4= dword ptr -4
00000000004010C0
00000000004010C0 push    ecx
00000000004010C1 push    ebx
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
00000000004010C6 push    edi
```





Learn / Windows / Apps / Win32 / API / Windows Internet / Wininet.h /

InternetOpenA function (wininet.h)

Article • 07/27/2022 • 3 minutes to read

Feedback

Initializes an application's use of the WinINet functions.

Syntax

C++ Copy

```cpp
HINTERNET InternetOpenA(
  [in] LPCSTR lpszAgent,
  [in] DWORD  dwAccessType,
  [in] LPCSTR lpszProxy,
  [in] LPCSTR lpszProxyBypass,
  [in] DWORD  dwFlags
);
```

cdecl

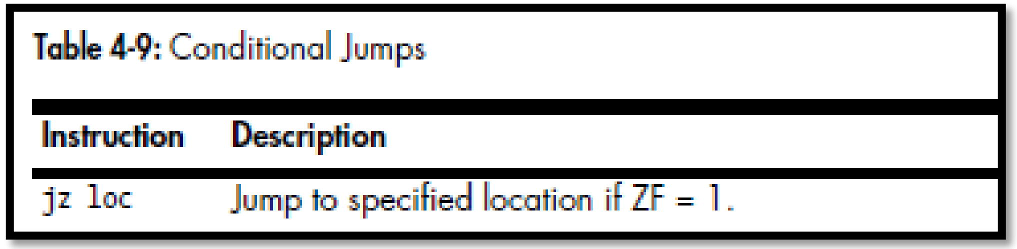
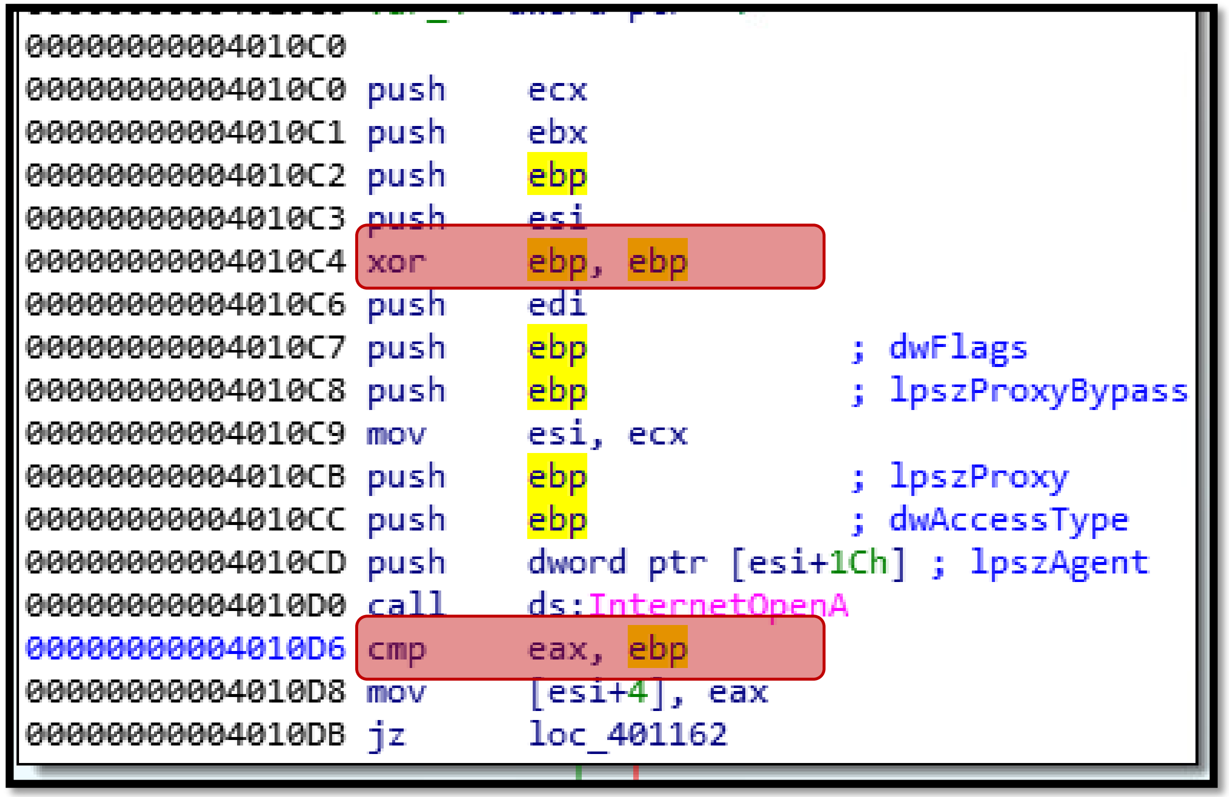


Table 4-9: Conditional Jumps

Instruction	Description
jz loc	Jump to specified location if ZF = 1.



```
00000000004010C0
00000000004010C0 push    ecx
00000000004010C1 push    ebx
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
00000000004010C6 push    edi
00000000004010C7 push    ebp               ; dwFlags
00000000004010C8 push    ebp               ; lpszProxyBypass
00000000004010C9 mov     esi, ecx
00000000004010CB push    ebp               ; lpszProxy
00000000004010CC push    ebp               ; dwAccessType
00000000004010CD push    dword ptr [esi+1Ch] ; lpszAgent
00000000004010D0 call    ds:InternetOpenA
00000000004010D6 cmp     eax, ebp
00000000004010D8 mov     [esi+4], eax
00000000004010DB jz      loc_401162
```

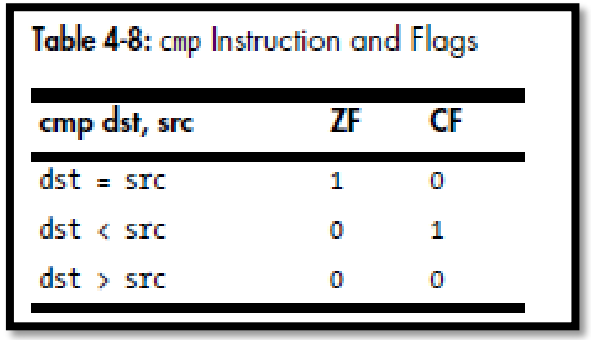
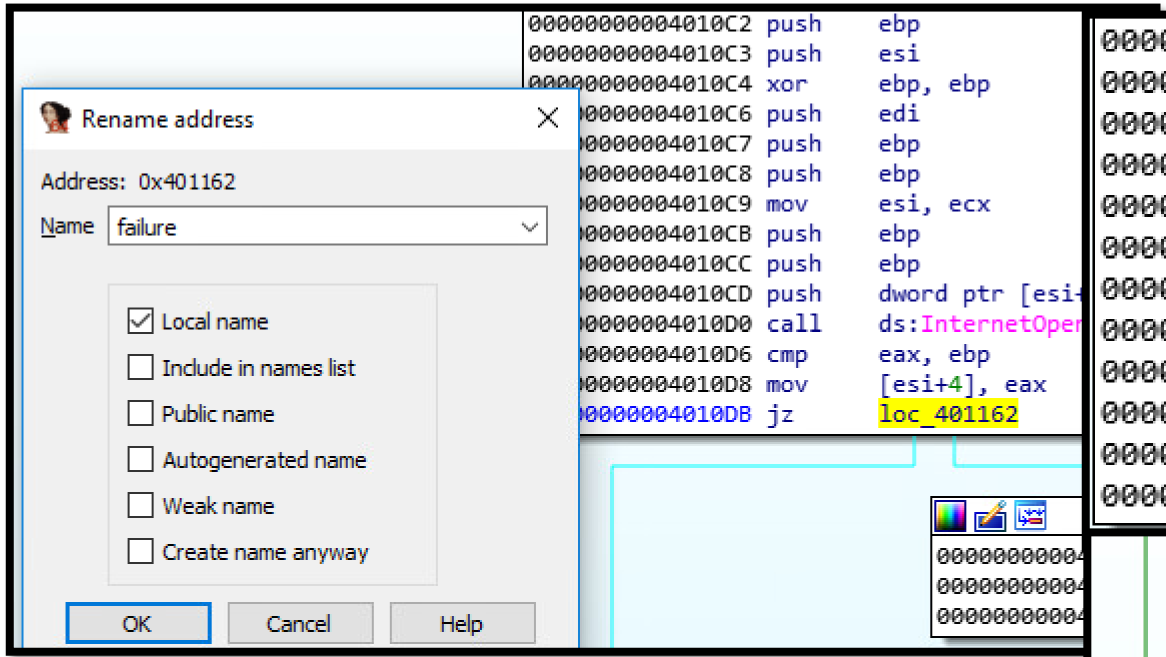


Table 4-8: cmp Instruction and Flags

cmp dst, src	ZF	CF
dst = src	1	0
dst < src	0	1
dst > src	0	0

Helpful naming…



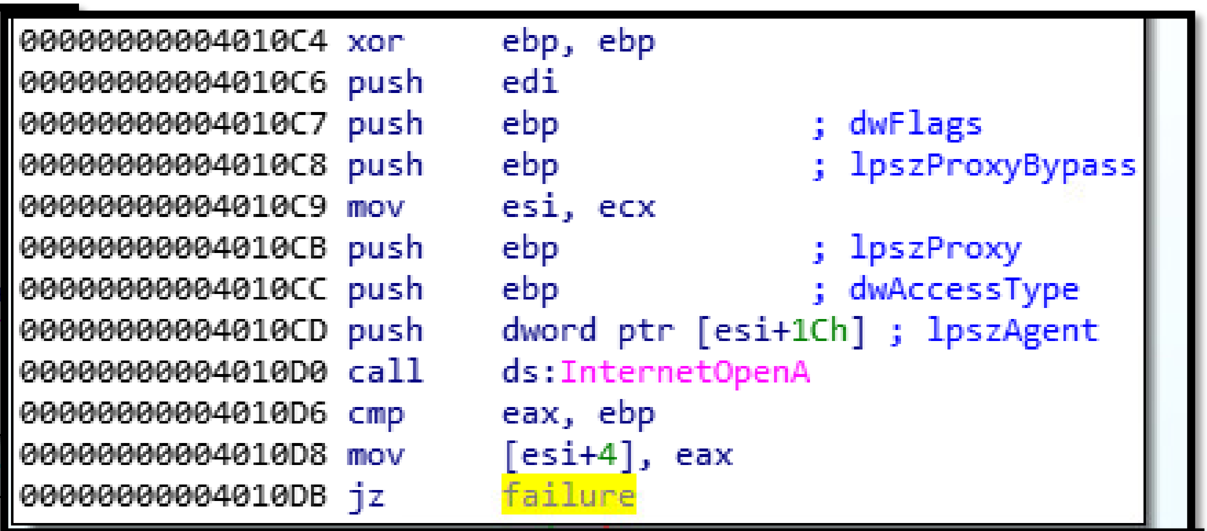
```
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
)0000000004010C6 push    edi
)0000000004010C7 push    ebp
)0000000004010C8 push    ebp
)0000000004010C9 mov     esi, ecx
)0000000004010CB push    ebp
)0000000004010CC push    ebp
)0000000004010CD push    dword ptr [esi+
)0000000004010D0 call    ds:InternetOpen
)0000000004010D6 cmp     eax, ebp
)0000000004010D8 mov     [esi+4], eax
)0000000004010DB jz      loc_401162
```

Rename address

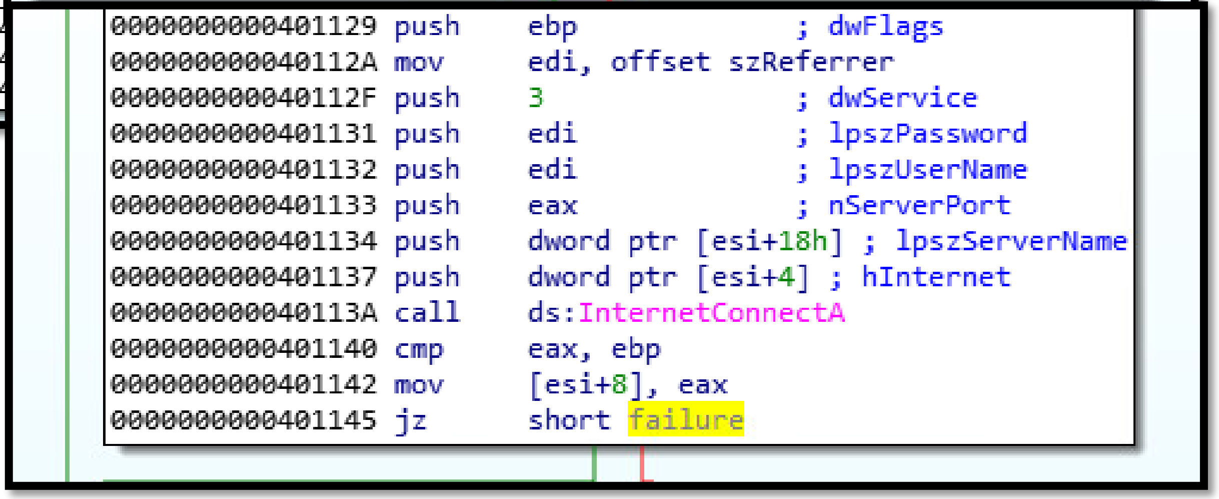
Address: 0x401162

Name: failure

[✓] Local name
[ ] Include in names list
[ ] Public name
[ ] Autogenerated name
[ ] Weak name
[ ] Create name anyway



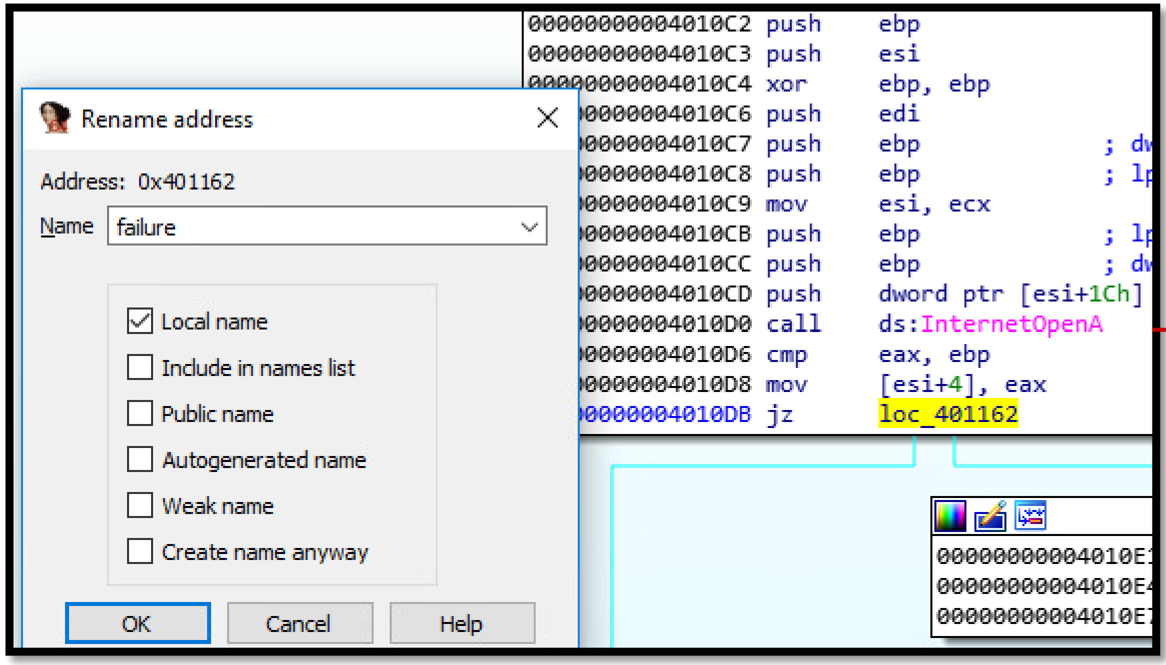
```
00000000004010C4 xor     ebp, ebp
00000000004010C6 push    edi
00000000004010C7 push    ebp                 ; dwFlags
00000000004010C8 push    ebp                 ; lpszProxyBypass
00000000004010C9 mov     esi, ecx
00000000004010CB push    ebp                 ; lpszProxy
00000000004010CC push    ebp                 ; dwAccessType
00000000004010CD push    dword ptr [esi+1Ch] ; lpszAgent
00000000004010D0 call    ds:InternetOpenA
00000000004010D6 cmp     eax, ebp
00000000004010D8 mov     [esi+4], eax
00000000004010DB jz      failure
```



```
0000000000401129 push    ebp                 ; dwFlags
000000000040112A mov     edi, offset szReferrer
000000000040112F push    3                   ; dwService
0000000000401131 push    edi                 ; lpszPassword
0000000000401132 push    edi                 ; lpszUserName
0000000000401133 push    eax                 ; nServerPort
0000000000401134 push    dword ptr [esi+18h] ; lpszServerName
0000000000401137 push    dword ptr [esi+4]  ; hInternet
000000000040113A call    ds:InternetConnectA
0000000000401140 cmp     eax, ebp
0000000000401142 mov     [esi+8], eax
0000000000401145 jz      short failure
```

“Easy on the eyes”

- We can group basic blocks to make visual inspection more appealing



Rename address

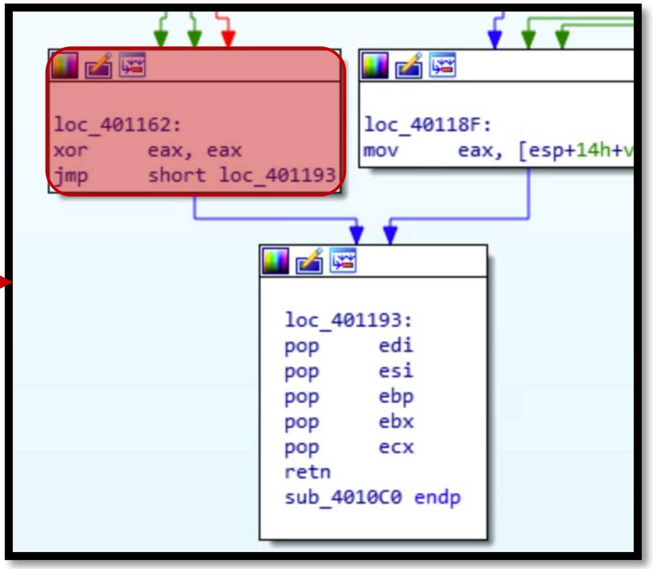
Address: 0x401162

Name: failure

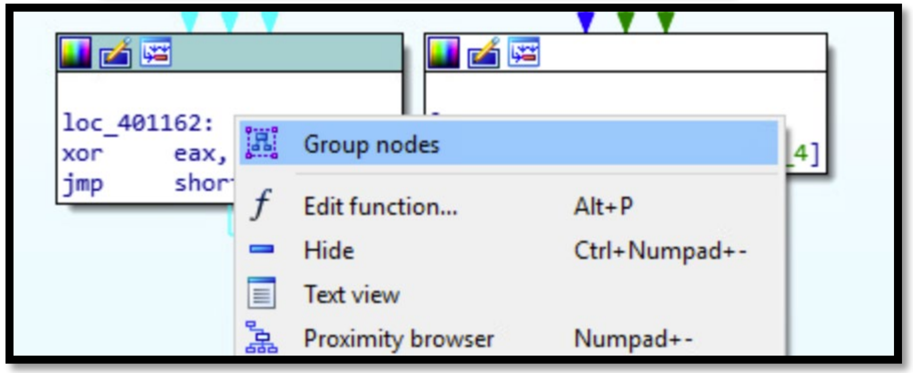
[✓] Local name
[ ] Include in names list
[ ] Public name
[ ] Autogenerated name
[ ] Weak name
[ ] Create name anyway

OK | Cancel | Help

```
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
0000000004010C6 push     edi
0000000004010C7 push     ebp              ; dw
0000000004010C8 push     ebp              ; lp
0000000004010C9 mov      esi, ecx
0000000004010CB push     ebp              ; lp
0000000004010CC push     ebp              ; dw
0000000004010CD push     dword ptr [esi+1Ch]
0000000004010D0 call     ds:InternetOpenA
0000000004010D6 cmp      eax, ebp
0000000004010D8 mov      [esi+4], eax
0000000004010DB jz       loc_401162

00000000004010E
00000000004010E
00000000004010E
```



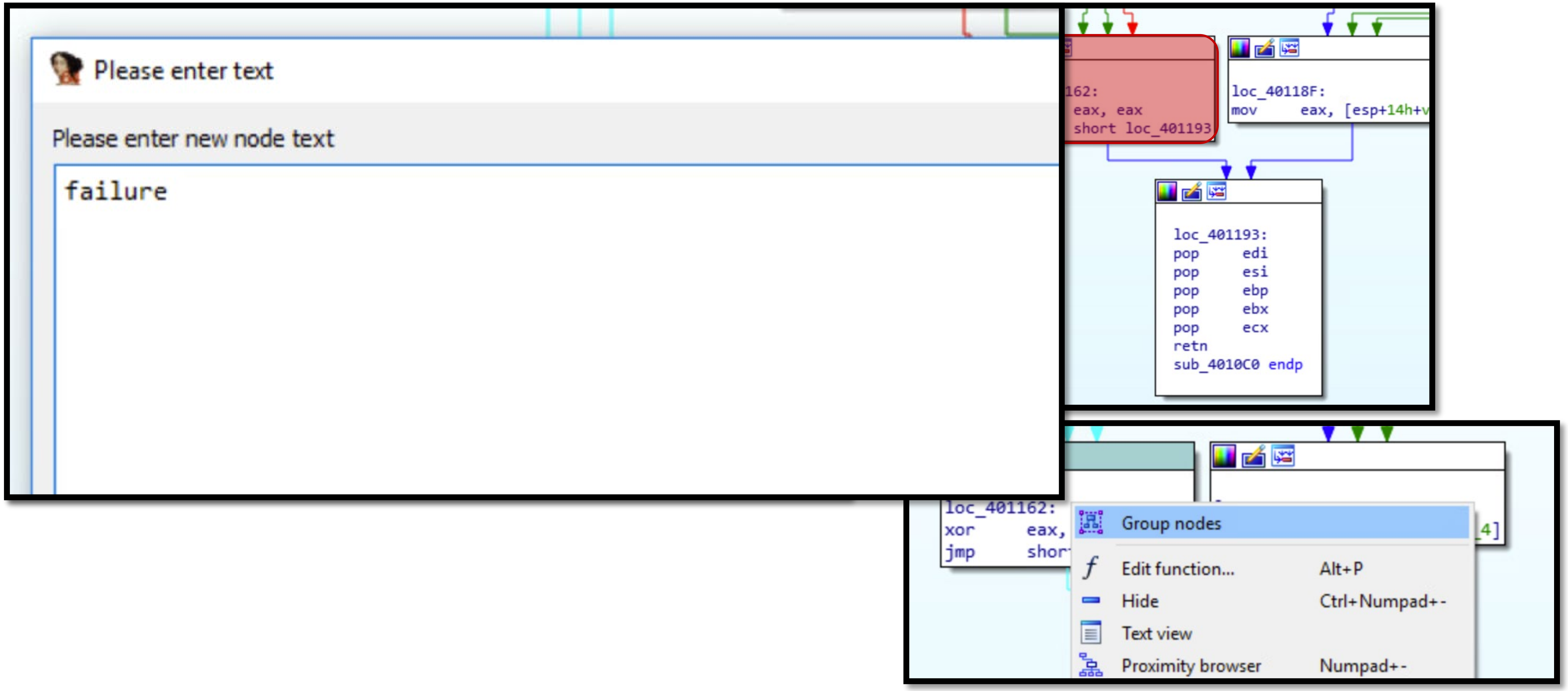
```
loc_401162:
xor     eax, eax
jmp     short loc_401193

loc_40118F:
mov     eax, [esp+14h+v

loc_401193:
pop     edi
pop     esi
pop     ebp
pop     ebx
pop     ecx
retn
sub_4010C0 endp
```



```
loc_401162:
xor     eax,
jmp     shor
```

“Easy on the eyes”

- We can group basic blocks to make visual inspection more appealing



Please enter text

Please enter new node text

```
failure
```

```
162:
 eax, eax
 short loc_401193
```

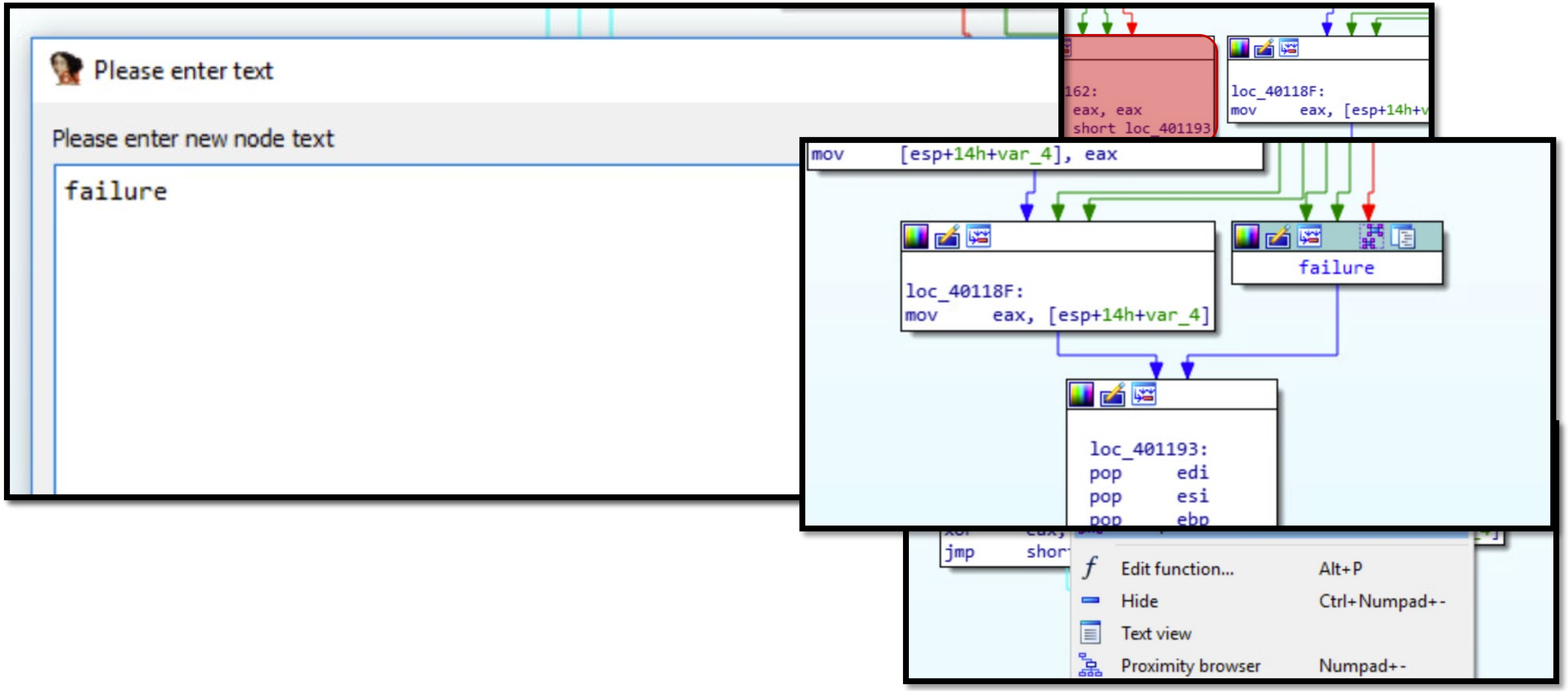
```
loc_40118F:
mov     eax, [esp+14h+v
```

```
loc_401193:
pop     edi
pop     esi
pop     ebp
pop     ebx
pop     ecx
retn
sub_4010C0 endp
```

```
loc_401162:
xor     eax,
jmp     shor
```

“Easy on the eyes”

- We can group basic blocks to make visual inspection more appealing



```
162:
 eax, eax
 short loc_401193
```

```
loc_40118F:
mov      eax, [esp+14h+v
```

```
mov      [esp+14h+var_4], eax
```

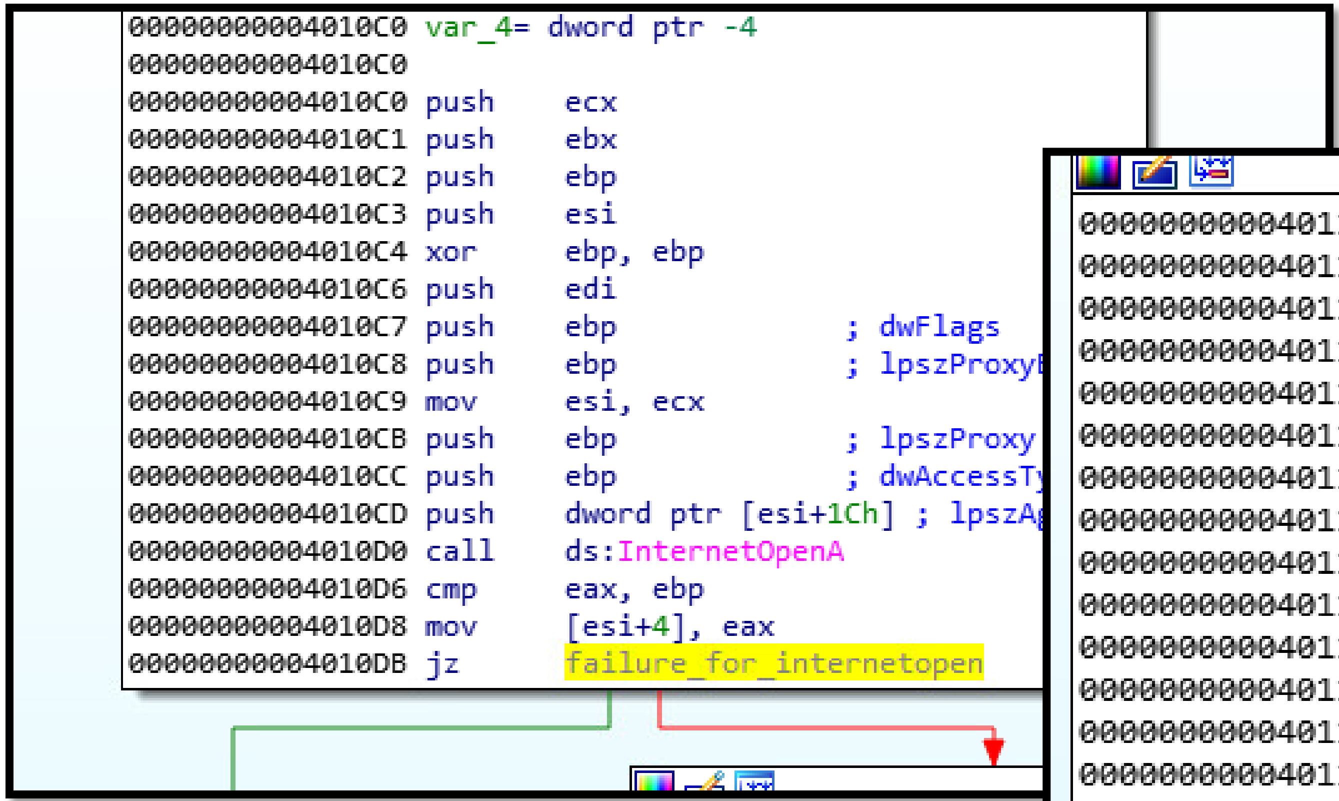
```
loc_40118F:
mov      eax, [esp+14h+var_4]
```

```
failure
```

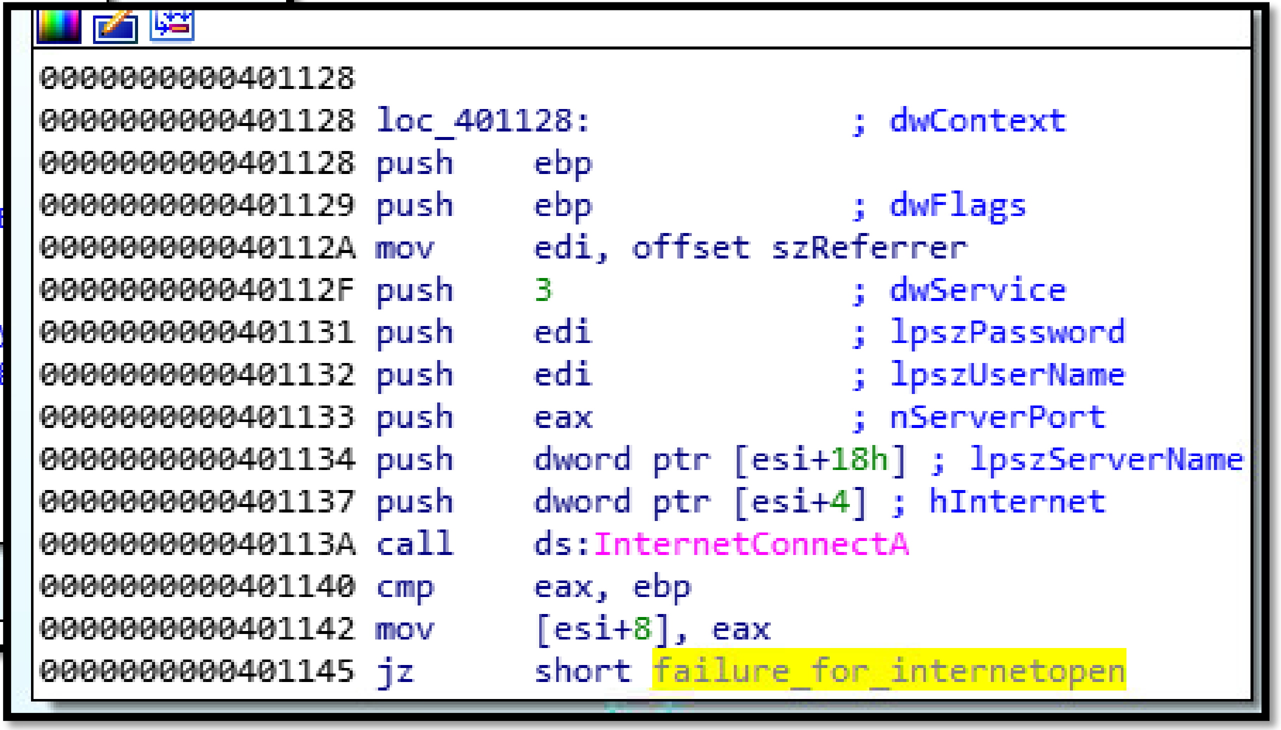
```
loc_401193:
pop      edi
pop      esi
pop      ebp
```

```
jmp      shor
```

Unhelpful naming…



```
00000000004010C0 var_4= dword ptr -4
00000000004010C0
00000000004010C0 push    ecx
00000000004010C1 push    ebx
00000000004010C2 push    ebp
00000000004010C3 push    esi
00000000004010C4 xor     ebp, ebp
00000000004010C6 push    edi
00000000004010C7 push    ebp                ; dwFlags
00000000004010C8 push    ebp                ; lpszProxyB
00000000004010C9 mov     esi, ecx
00000000004010CB push    ebp                ; lpszProxy
00000000004010CC push    ebp                ; dwAccessTy
00000000004010CD push    dword ptr [esi+1Ch] ; lpszAg
00000000004010D0 call    ds:InternetOpenA
00000000004010D6 cmp     eax, ebp
00000000004010D8 mov     [esi+4], eax
00000000004010DB jz      failure_for_internetopen
```



```
0000000000401128
0000000000401128 loc_401128:                 ; dwContext
0000000000401128 push    ebp
0000000000401129 push    ebp                 ; dwFlags
000000000040112A mov     edi, offset szReferrer
000000000040112F push    3                   ; dwService
0000000000401131 push    edi                 ; lpszPassword
0000000000401132 push    edi                 ; lpszUserName
0000000000401133 push    eax                 ; nServerPort
0000000000401134 push    dword ptr [esi+18h] ; lpszServerName
0000000000401137 push    dword ptr [esi+4] ; hInternet
000000000040113A call    ds:InternetConnectA
0000000000401140 cmp     eax, ebp
0000000000401142 mov     [esi+8], eax
0000000000401145 jz      short failure_for_internetopen
```

Exercise #1: Press ‘g’, enter `4010c0`

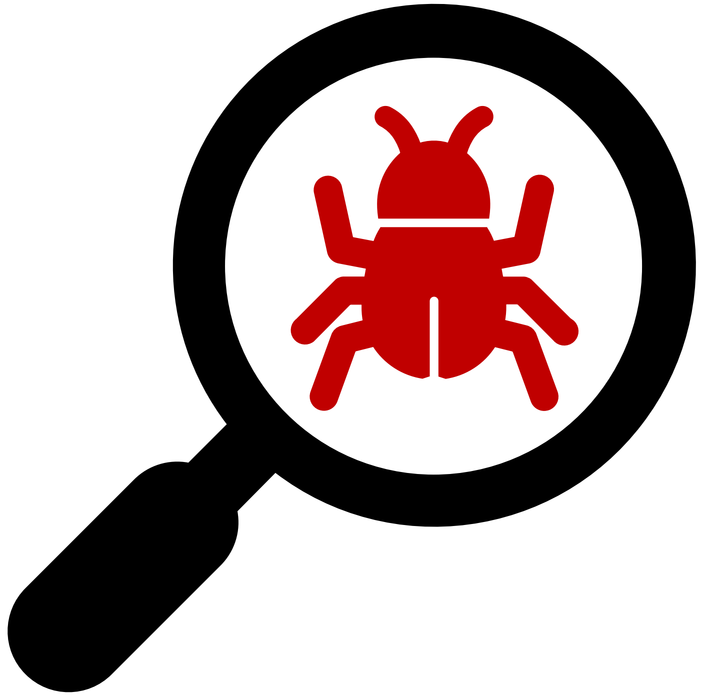
- As a team, tell me what this function is responsible for. I need a **high-level** overview. (I don’t need “the prologue is prepared by…..”) 🥱 Who read the book?
- When complete and your team agrees, rename the `sub_4010C0` function to a descriptive term

Hints:

1. Search for API documentation https://learn.microsoft.com/en-us/windows/win32/api/
2. Look at returns (cdecl) and how they are used
3. Symbolic constants (API arguments)
4. Ask questions!

Exercise #2: Go to `4013CC`

- Super easy!
- Rename it



Exercise #3: Go to `402645`

- As a team, tell me what this function is responsible for. I need a **high-level** overview.
- When complete and your team agrees
  1. rename the `sub_402645` function to a descriptive term
  2. Provide the sequence of APIs that lead to your derived capability

     `InternetReadFile -> WriteFile -> ShellExecute` = Execute Dropped File

Hints:

1. Search for API documentation https://learn.microsoft.com/en-us/windows/win32/api/
2. Look at hard-coded strings for help
3. There are 3 main paths (1 of them is the failure path)

Summary

- Static analysis is “fun”
- Not very useful on sophisticated (obfuscated and packed malware)
- Useful to analyze benign software (when source code isn’t available) to find bugs
- Useful to troubleshoot more sophisticated binary analysis tools (i.e., my tool breaks down at instruction X…why?)
- What IOCs can you extract from Greencat?
  - A sequence of APIs can be used to classify maliciousness
  - Sequence of APIs can also be used to identify capabilities (…like CAPA)

For Next Lesson

- Read Chapter 7 from *Practical Malware Analysis*