HANDLING ADVANCED THREATS

SANS 2020 ONLINE SUMMIT

(EXTENDED VERSION)

by Alexandre Borges

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

- Cyber Security Researcher
- Speaker at DEF CON USA 2019
- Speaker at DEF CON USA 2018
- Speaker at DEF CON CHINA 2019
- Speaker at DC2711 (Johannesburg)
- Speaker at NO HAT 2019 (Italy)
- Speaker at HITB 2019 (Amsterdam)
- Speaker at CONFidence 2019 (Poland)
- Speaker at DevOpsDays BH 2019
- Speaker at BSIDES 2019/2018/2017/2016
- Speaker at H2HC 2016/2015
- Speaker at BHACK 2019/2018
- Researcher on Android/iOS Reversing, Rootkits and Digital Forensics.
- Referee on Digital Investigation: The International Journal of Digital Forensics & Incident Response

Agenda:

- ❖ Introduction
- ❖ Reversing
- ❖ Anti-Reversing
- ❖ De-obfuscation

❑Last conferences:

- BHACK 2019 (Belo Horizonte/Brazil)
- DC2711 (Johannesburg/South Africa)
- NO HAT Conference 2019 (Bergamo/Italy)
- DEF CON USA 2019 (Las Vegas / USA)
- CONFidence Conference 2019 (Krakow / Poland)
- DEF CON China 2019 (Beijing / China)
- HITB Security Conference 2019 (Amsterdam)
- BSIDES Sao Paulo 2019 (Sao Paulo / Brazil)
- DEF CON USA 2018 (Las Vegas / USA)

- Malwoverview Tool: https://github.com/alexandreborges/malwoverview

INTRODUCTION

INTRODUCTION

- During reverse engineering of a malware sample, we need to understand few aspects on the threat:

  - Is the malware packed?
  - Are DLLs and functions resolved dynamically?
  - Are strings encrypted?
  - Are there any anti-forensic techniques such as anti-vm, anti-debugging or anti-disassembly?
  - Is there any obfuscation technique being used?

- Unpacking malware is usually easy, but you might find sophisticated packers...

➢ INTRODUCTION

- Advanced threats are different from any daily malware because:

  - They don't use common packers.
  - Most of the time, they bring malicious device drives (rootkits).
  - Sometimes they try to compromise the platform (bootkits)
  - They can use 0-days to exploit the infrastructure and systems.
  - They bypass most of defenses and run under the radar.
  - C2 transmits beacons and data once per week with short duration.
  - They might implement tricks to prevent any memory acquisition.
  - Most certainly, there'll be anti-forensic techniques.
  - It's hard and take so much time to reverse it.

- If you have luck, so you'll have the opportunity to analyze them. ☺

INTRODUCTION

- There're many packers that we know about to unpack them or, at least, how to manage them (but it can be hard...):
  - Native code: ASPack, Armadillo, Petite, FSG, UPX, MPRESS, NSPack, PECompact, WinUnpack and so on...
  - .NET packers and obfuscators: .NET Reactor, Salamander .NET Obfuscator, Dotfuscator, Smart Assembly, CryptoObfuscator for .NET, Agile, ArmDot, babelfor.NET, Eazfuscator.NET, Spice.Net, Skater.NET, VM Protect 3.40+ and so on...
  - Android Packers/Obfuscators: DexGuard, DexProtect, DevGuard, Arxan, ApkGyard, and so on...

INTRODUCTION

- .NET packers use similar tricks of native code:
  - Control flow obfuscation and dead/junk code insertion.
  - Renaming: methods signatures, fields, methods implementation, namespaces, metadata and external references.
  - Re-encoding: changing printable to unprintable characters
  - Simple encryption of methods and strings.
  - Cross reference obfuscation.
- In native code, there’re well know memory APIs: VirtualAlloc/Ex( ), HeapCreate( ) / RtlCreateHeap( ), HeapReAlloc( ), GlobalAlloc( ), RtlAllocateHeap( )

INTRODUCTION

- Most packed binaries can be unpacked using debuggers, breakpoints and dumping unpacked content from memory.

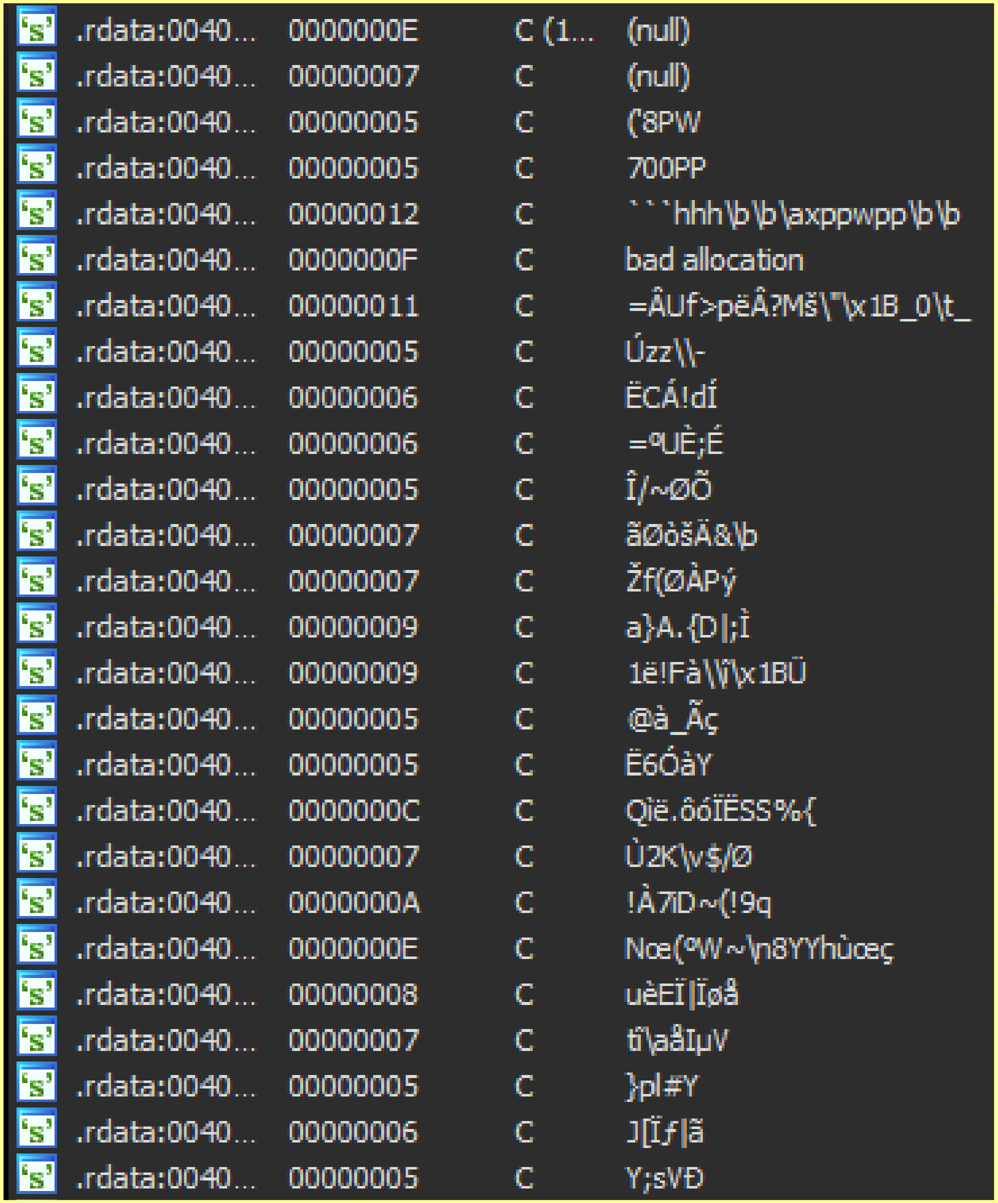
- Even when a binary uses customized packing techniques, it is still possible:

  - dumping the unpacked code from memory using Volatility.
  - fixing the ImageAddress field using few lines in Python its respective IAT using impscan plugin to analyze it in IDA Pro:

    - python vol.py -f memory.vmem procdump -p 2096 -D . --memory (to keep slack space)
    - python vol.py -f memory.vmem impscan --output=idc -p 2096

REVERSING

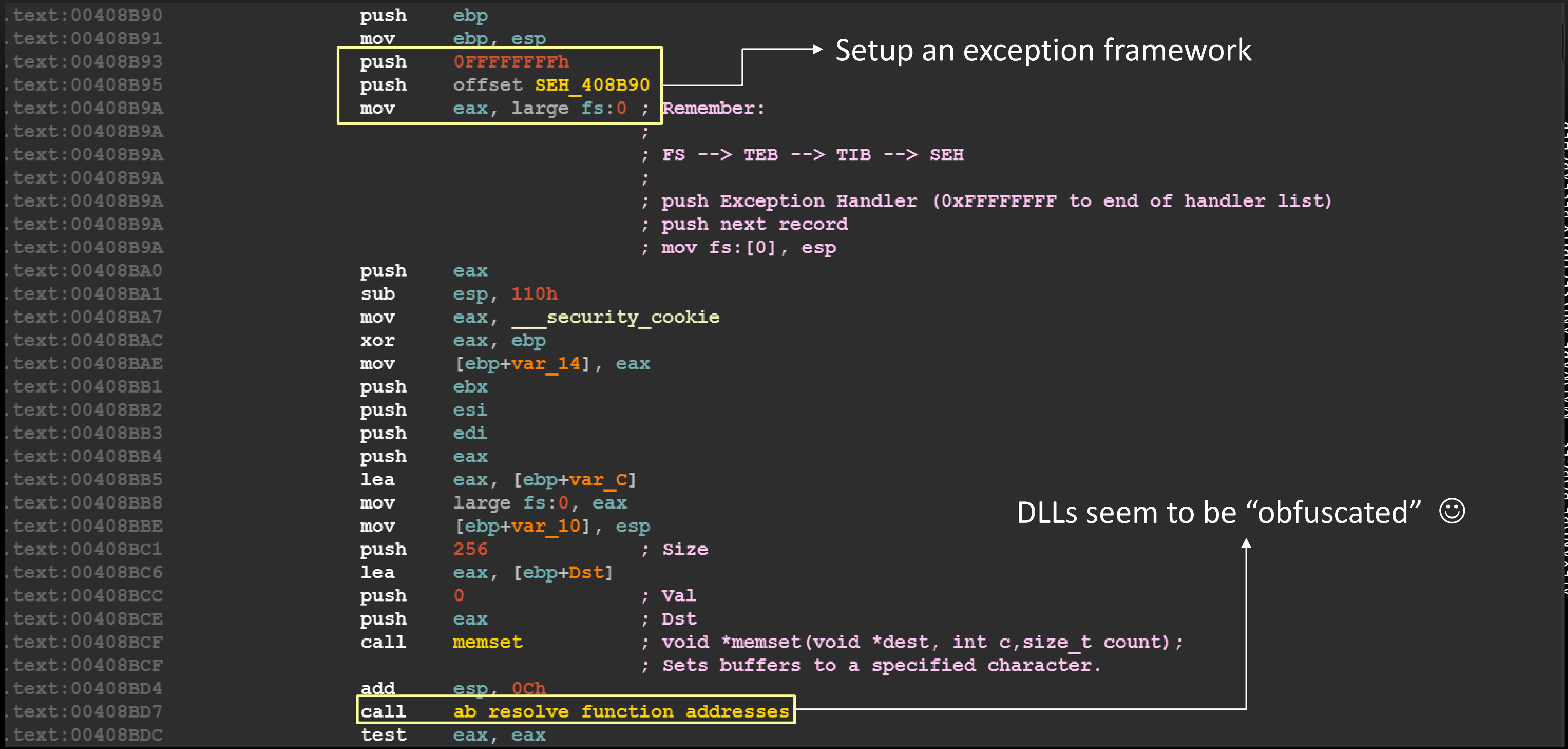
➢ INTRODUCTION



```
.rdata:0040...  0000000E  C (1...  (null)
.rdata:0040...  00000007  C        (null)
.rdata:0040...  00000005  C        ('8PW
.rdata:0040...  00000005  C        700PP
.rdata:0040...  00000012  C        ```hhh\b\b\axppwpp\b\b
.rdata:0040...  0000000F  C        bad allocation
.rdata:0040...  00000011  C        =ÂUf>pëÂ?Mš\"\x1B_0\t_
.rdata:0040...  00000005  C        Úzz\\-
.rdata:0040...  00000006  C        ËCÁ!dÍ
.rdata:0040...  00000006  C        =ºUÈ;É
.rdata:0040...  00000005  C        Î/~ØÕ
.rdata:0040...  00000007  C        ãØòšÄ&\b
.rdata:0040...  00000007  C        Žf(ØÀPý
.rdata:0040...  00000009  C        a}A.{D|;Ì
.rdata:0040...  00000009  C        1ë!Fà\\î\x1BÜ
.rdata:0040...  00000005  C        @à_Ãç
.rdata:0040...  00000005  C        Ë6ÓàY
.rdata:0040...  0000000C  C        Qìë.ôóÏËSS%{
.rdata:0040...  00000007  C        Ù2K\v$/Ø
.rdata:0040...  0000000A  C        !À7iD~(!9q
.rdata:0040...  0000000E  C        Nœ(ºW~\n8YYhùœç
.rdata:0040...  00000008  C        uèEÏ|Ïøå
.rdata:0040...  00000007  C        tî\aåIµV
.rdata:0040...  00000005  C        }pl#Y
.rdata:0040...  00000006  C        J[Ïƒ|ã
.rdata:0040...  00000005  C        Y;sVÐ
```

- ❖ As we've mentioned previously, strings are one of first references.
- ❖ However, they are all encrypted and writing YARA rules using them could not be so interesting ☺

➢ INTRODUCTION

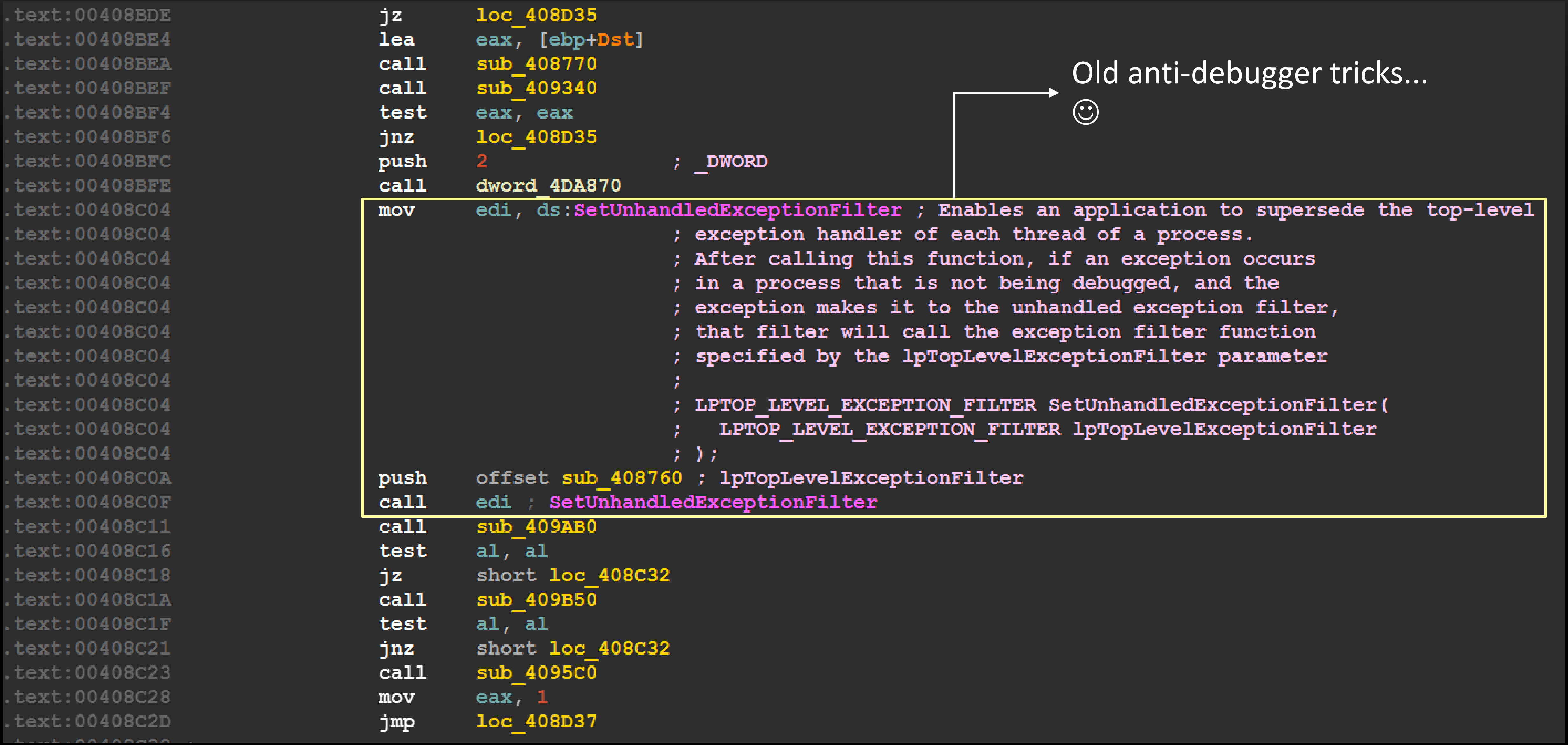


```
.text:00408B90                 push    ebp
.text:00408B91                 mov     ebp, esp
.text:00408B93                 push    0FFFFFFFFh
.text:00408B95                 push    offset SEH_408B90
.text:00408B9A                 mov     eax, large fs:0 ; Remember:
.text:00408B9A                                         ;
.text:00408B9A                                         ; FS --> TEB --> TIB --> SEH
.text:00408B9A                                         ;
.text:00408B9A                                         ; push Exception Handler (0xFFFFFFFF to end of handler list)
.text:00408B9A                                         ; push next record
.text:00408B9A                                         ; mov fs:[0], esp
.text:00408BA0                 push    eax
.text:00408BA1                 sub     esp, 110h
.text:00408BA7                 mov     eax, ___security_cookie
.text:00408BAC                 xor     eax, ebp
.text:00408BAE                 mov     [ebp+var_14], eax
.text:00408BB1                 push    ebx
.text:00408BB2                 push    esi
.text:00408BB3                 push    edi
.text:00408BB4                 push    eax
.text:00408BB5                 lea     eax, [ebp+var_C]
.text:00408BB8                 mov     large fs:0, eax
.text:00408BBE                 mov     [ebp+var_10], esp
.text:00408BC1                 push    256             ; Size
.text:00408BC6                 lea     eax, [ebp+Dst]
.text:00408BCC                 push    0               ; Val
.text:00408BCE                 push    eax             ; Dst
.text:00408BCF                 call    memset          ; void *memset(void *dest, int c,size_t count);
.text:00408BCF                                         ; Sets buffers to a specified character.
.text:00408BD4                 add     esp, 0Ch
.text:00408BD7                 call    ab_resolve_function_addresses
.text:00408BDC                 test    eax, eax
```

Setup an exception framework

DLLs seem to be “obfuscated” ☺

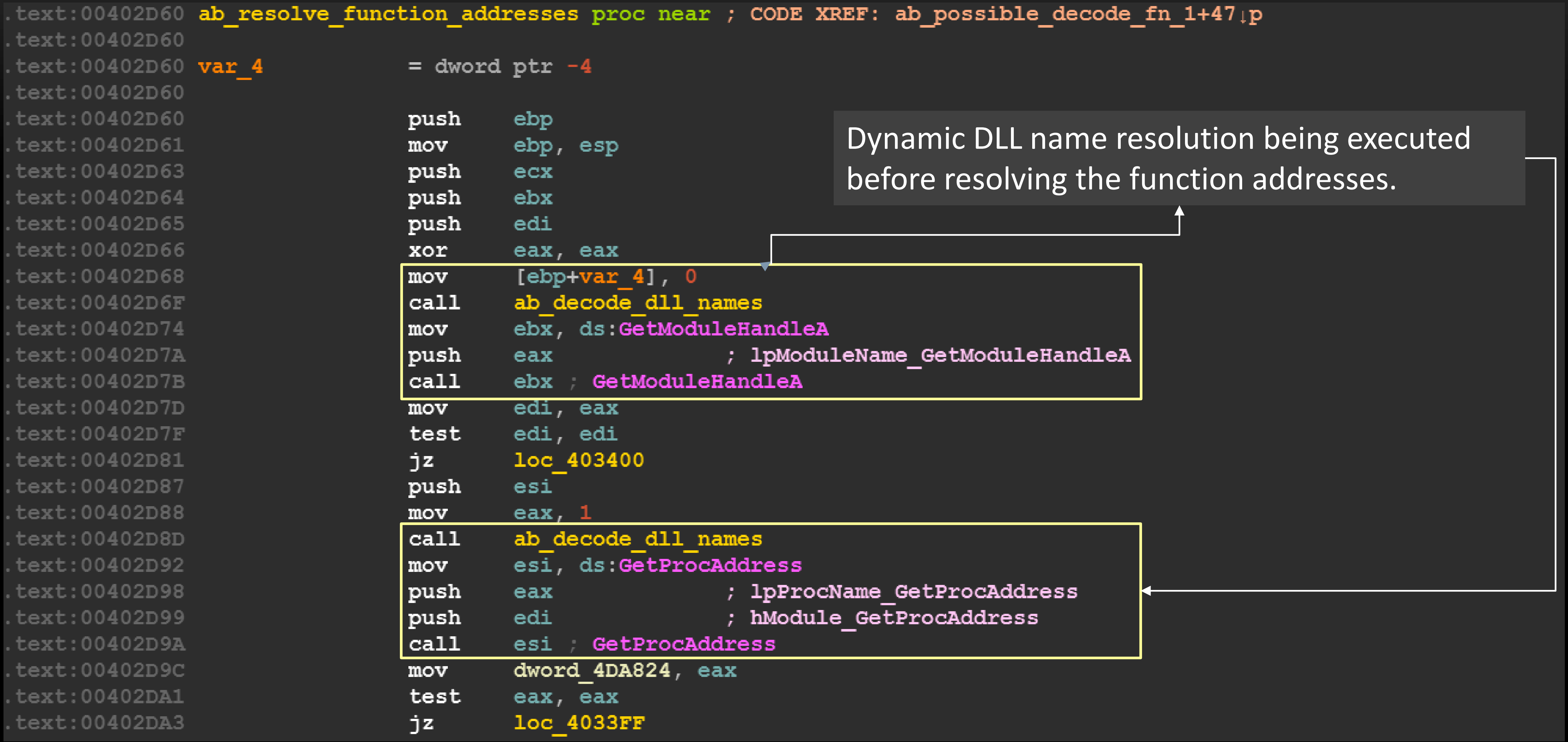
➢ INTRODUCTION



```
.text:00408BDE                 jz      loc_408D35
.text:00408BE4                 lea     eax, [ebp+Dst]
.text:00408BEA                 call    sub_408770
.text:00408BEF                 call    sub_409340
.text:00408BF4                 test    eax, eax
.text:00408BF6                 jnz     loc_408D35
.text:00408BFC                 push    2                ; _DWORD
.text:00408BFE                 call    dword_4DA870
.text:00408C04                 mov     edi, ds:SetUnhandledExceptionFilter ; Enables an application to supersede the top-level
.text:00408C04                                          ; exception handler of each thread of a process.
.text:00408C04                                          ; After calling this function, if an exception occurs
.text:00408C04                                          ; in a process that is not being debugged, and the
.text:00408C04                                          ; exception makes it to the unhandled exception filter,
.text:00408C04                                          ; that filter will call the exception filter function
.text:00408C04                                          ; specified by the lpTopLevelExceptionFilter parameter
.text:00408C04                                          ;
.text:00408C04                                          ; LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(
.text:00408C04                                          ;   LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter
.text:00408C04                                          ; );
.text:00408C0A                 push    offset sub_408760 ; lpTopLevelExceptionFilter
.text:00408C0F                 call    edi ; SetUnhandledExceptionFilter
.text:00408C11                 call    sub_409AB0
.text:00408C16                 test    al, al
.text:00408C18                 jz      short loc_408C32
.text:00408C1A                 call    sub_409B50
.text:00408C1F                 test    al, al
.text:00408C21                 jnz     short loc_408C32
.text:00408C23                 call    sub_4095C0
.text:00408C28                 mov     eax, 1
.text:00408C2D                 jmp     loc_408D37
```

Old anti-debugger tricks... ☺

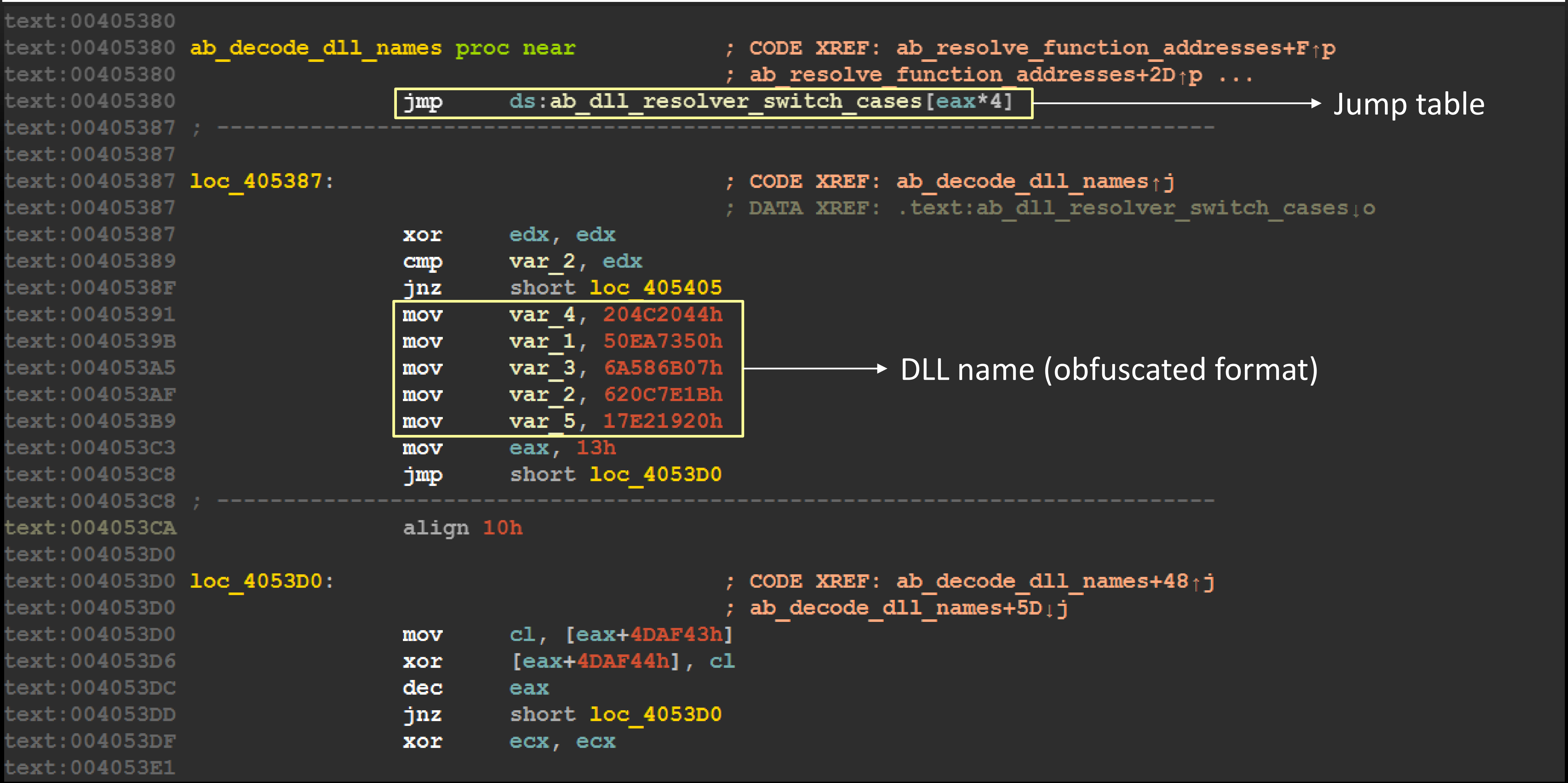
➢ INTRODUCTION



```
.text:00402D60 ab_resolve_function_addresses proc near ; CODE XREF: ab_possible_decode_fn_1+47↓p
.text:00402D60
.text:00402D60 var_4           = dword ptr -4
.text:00402D60
.text:00402D60                 push    ebp
.text:00402D61                 mov     ebp, esp
.text:00402D63                 push    ecx
.text:00402D64                 push    ebx
.text:00402D65                 push    edi
.text:00402D66                 xor     eax, eax
.text:00402D68                 mov     [ebp+var_4], 0
.text:00402D6F                 call    ab_decode_dll_names
.text:00402D74                 mov     ebx, ds:GetModuleHandleA
.text:00402D7A                 push    eax             ; lpModuleName_GetModuleHandleA
.text:00402D7B                 call    ebx ; GetModuleHandleA
.text:00402D7D                 mov     edi, eax
.text:00402D7F                 test    edi, edi
.text:00402D81                 jz      loc_403400
.text:00402D87                 push    esi
.text:00402D88                 mov     eax, 1
.text:00402D8D                 call    ab_decode_dll_names
.text:00402D92                 mov     esi, ds:GetProcAddress
.text:00402D98                 push    eax             ; lpProcName_GetProcAddress
.text:00402D99                 push    edi             ; hModule_GetProcAddress
.text:00402D9A                 call    esi ; GetProcAddress
.text:00402D9C                 mov     dword_4DA824, eax
.text:00402DA1                 test    eax, eax
.text:00402DA3                 jz      loc_4033FF
```

Dynamic DLL name resolution being executed before resolving the function addresses.

➢ INTRODUCTION



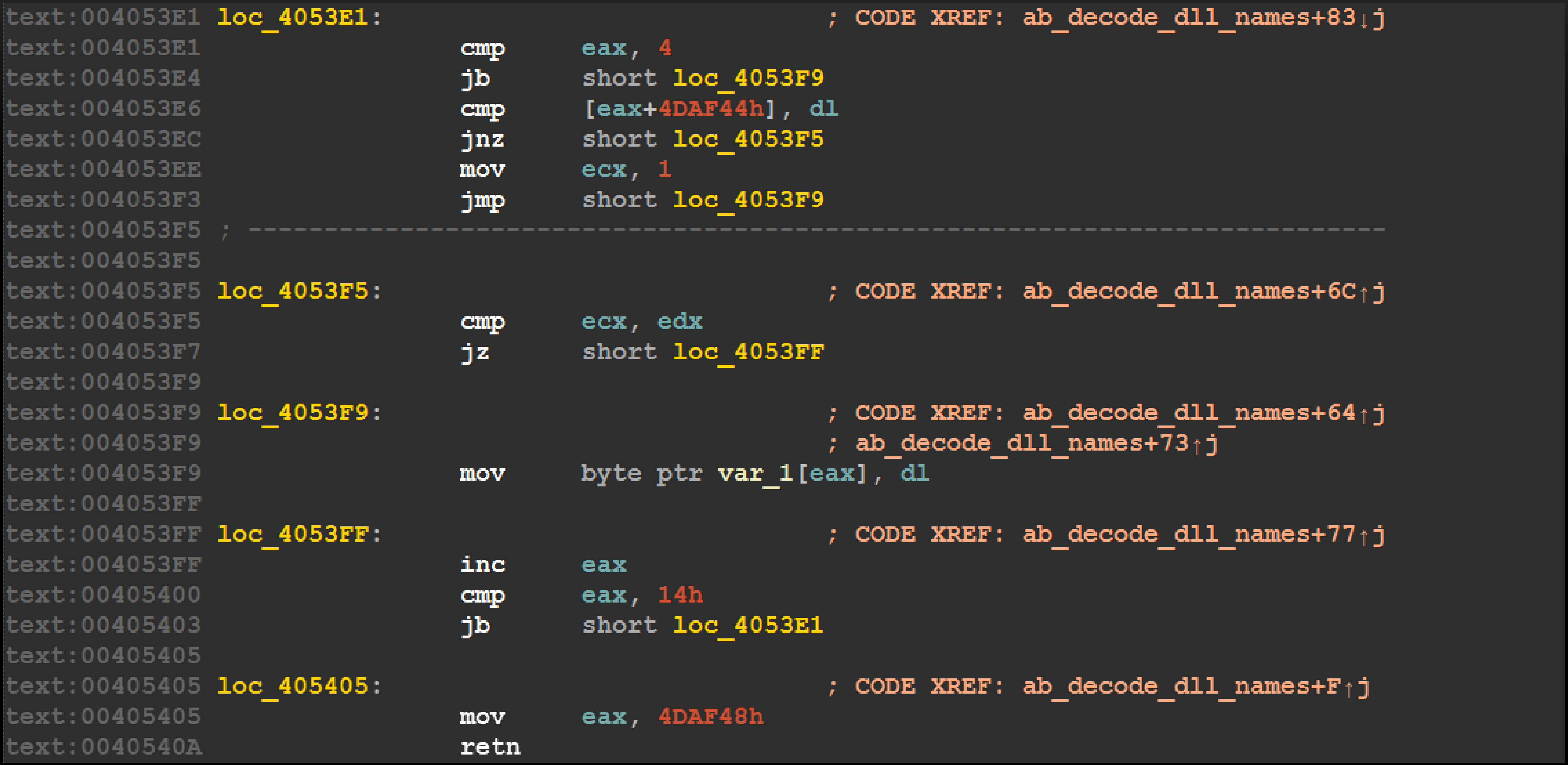
```
text:00405380
text:00405380 ab_decode_dll_names proc near           ; CODE XREF: ab_resolve_function_addresses+F↑p
text:00405380                                         ; ab_resolve_function_addresses+2D↑p ...
text:00405380                 jmp     ds:ab_dll_resolver_switch_cases[eax*4]
text:00405387 ; ---------------------------------------------------------------------------
text:00405387
text:00405387 loc_405387:                             ; CODE XREF: ab_decode_dll_names↑j
text:00405387                                         ; DATA XREF: .text:ab_dll_resolver_switch_cases↓o
text:00405387                 xor     edx, edx
text:00405389                 cmp     var_2, edx
text:0040538F                 jnz     short loc_405405
text:00405391                 mov     var_4, 204C2044h
text:0040539B                 mov     var_1, 50EA7350h
text:004053A5                 mov     var_3, 6A586B07h
text:004053AF                 mov     var_2, 620C7E1Bh
text:004053B9                 mov     var_5, 17E21920h
text:004053C3                 mov     eax, 13h
text:004053C8                 jmp     short loc_4053D0
text:004053C8 ; ---------------------------------------------------------------------------
text:004053CA                 align 10h
text:004053D0
text:004053D0 loc_4053D0:                             ; CODE XREF: ab_decode_dll_names+48↑j
text:004053D0                                         ; ab_decode_dll_names+5D↓j
text:004053D0                 mov     cl, [eax+4DAF43h]
text:004053D6                 xor     [eax+4DAF44h], cl
text:004053DC                 dec     eax
text:004053DD                 jnz     short loc_4053D0
text:004053DF                 xor     ecx, ecx
text:004053E1
```

Jump table

DLL name (obfuscated format)

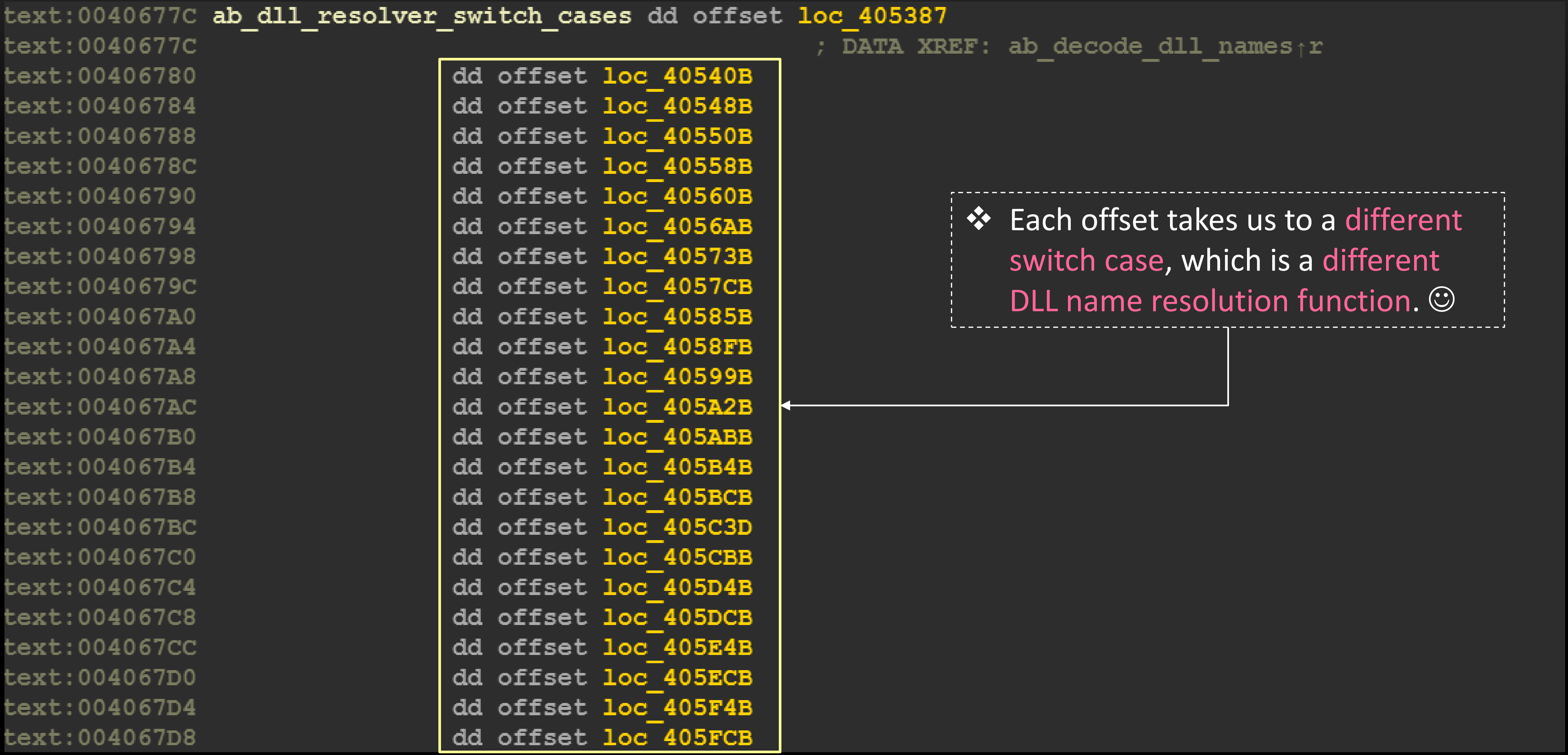
ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

➢ INTRODUCTION



```
text:004053E1 loc_4053E1:                             ; CODE XREF: ab_decode_dll_names+83↓j
text:004053E1                 cmp     eax, 4
text:004053E4                 jb      short loc_4053F9
text:004053E6                 cmp     [eax+4DAF44h], dl
text:004053EC                 jnz     short loc_4053F5
text:004053EE                 mov     ecx, 1
text:004053F3                 jmp     short loc_4053F9
text:004053F5 ; ---------------------------------------------------------------------------
text:004053F5
text:004053F5 loc_4053F5:                             ; CODE XREF: ab_decode_dll_names+6C↑j
text:004053F5                 cmp     ecx, edx
text:004053F7                 jz      short loc_4053FF
text:004053F9
text:004053F9 loc_4053F9:                             ; CODE XREF: ab_decode_dll_names+64↑j
text:004053F9                                         ; ab_decode_dll_names+73↑j
text:004053F9                 mov     byte ptr var_1[eax], dl
text:004053FF
text:004053FF loc_4053FF:                             ; CODE XREF: ab_decode_dll_names+77↑j
text:004053FF                 inc     eax
text:00405400                 cmp     eax, 14h
text:00405403                 jb      short loc_4053E1
text:00405405
text:00405405 loc_405405:                             ; CODE XREF: ab_decode_dll_names+F↑j
text:00405405                 mov     eax, 4DAF48h
text:0040540A                 retn
```

➢ INTRODUCTION

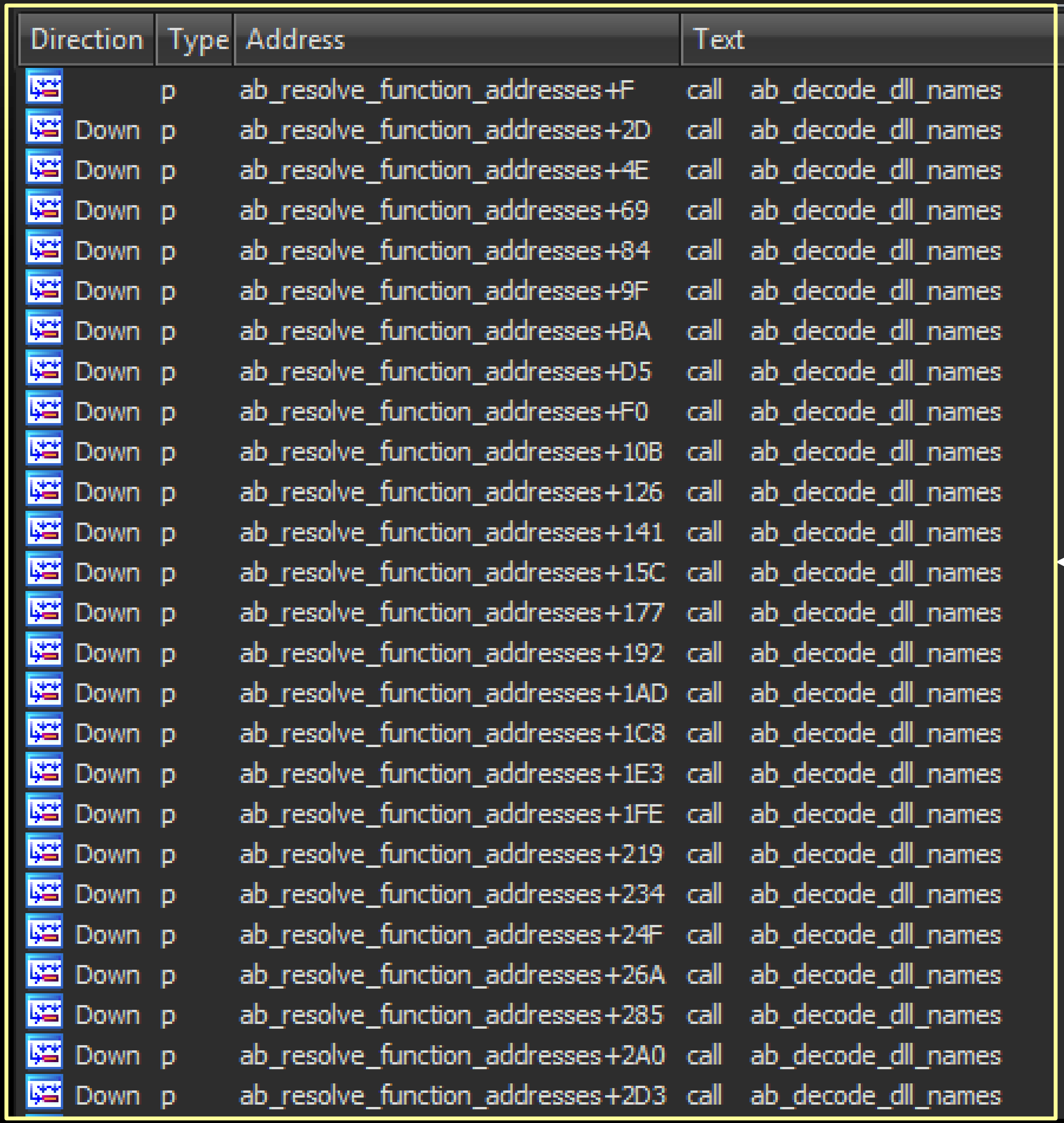


```
text:0040677C ab_dll_resolver_switch_cases dd offset loc_405387
text:0040677C                                                   ; DATA XREF: ab_decode_dll_names↑r
text:00406780                     dd offset loc_40540B
text:00406784                     dd offset loc_40548B
text:00406788                     dd offset loc_40550B
text:0040678C                     dd offset loc_40558B
text:00406790                     dd offset loc_40560B
text:00406794                     dd offset loc_4056AB
text:00406798                     dd offset loc_40573B
text:0040679C                     dd offset loc_4057CB
text:004067A0                     dd offset loc_40585B
text:004067A4                     dd offset loc_4058FB
text:004067A8                     dd offset loc_40599B
text:004067AC                     dd offset loc_405A2B
text:004067B0                     dd offset loc_405ABB
text:004067B4                     dd offset loc_405B4B
text:004067B8                     dd offset loc_405BCB
text:004067BC                     dd offset loc_405C3D
text:004067C0                     dd offset loc_405CBB
text:004067C4                     dd offset loc_405D4B
text:004067C8                     dd offset loc_405DCB
text:004067CC                     dd offset loc_405E4B
text:004067D0                     dd offset loc_405ECB
text:004067D4                     dd offset loc_405F4B
text:004067D8                     dd offset loc_405FCB
```

❖ Each offset takes us to a different switch case, which is a different DLL name resolution function. ☺

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

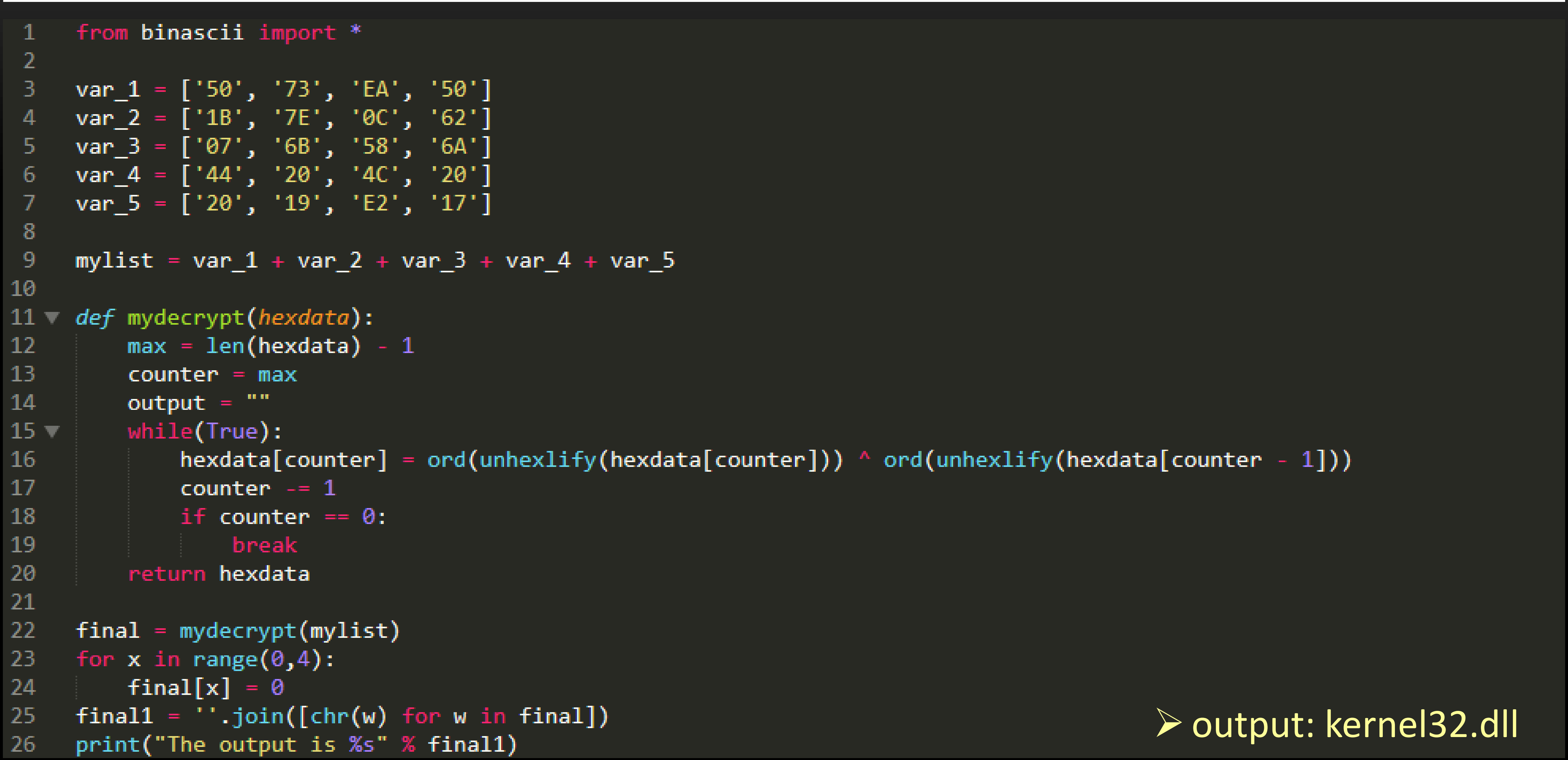
INTRODUCTION



Direction	Type	Address	Text
	p	ab_resolve_function_addresses+F	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+2D	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+4E	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+69	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+84	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+9F	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+BA	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+D5	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+F0	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+10B	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+126	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+141	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+15C	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+177	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+192	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+1AD	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+1C8	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+1E3	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+1FE	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+219	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+234	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+24F	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+26A	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+285	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+2A0	call ab_decode_dll_names
Down	p	ab_resolve_function_addresses+2D3	call ab_decode_dll_names

❖ As expected, there’re many calls to the same function for “decrypting” the DLL names ☺

➢ INTRODUCTION



```python
from binascii import *

var_1 = ['50', '73', 'EA', '50']
var_2 = ['1B', '7E', '0C', '62']
var_3 = ['07', '6B', '58', '6A']
var_4 = ['44', '20', '4C', '20']
var_5 = ['20', '19', 'E2', '17']

mylist = var_1 + var_2 + var_3 + var_4 + var_5

def mydecrypt(hexdata):
    max = len(hexdata) - 1
    counter = max
    output = ""
    while(True):
        hexdata[counter] = ord(unhexlify(hexdata[counter])) ^ ord(unhexlify(hexdata[counter - 1]))
        counter -= 1
        if counter == 0:
            break
    return hexdata

final = mydecrypt(mylist)
for x in range(0,4):
    final[x] = 0
final1 = ''.join([chr(w) for w in final])
print("The output is %s" % final1)
```

➢ output: kernel32.dll

➢ INTRODUCTION

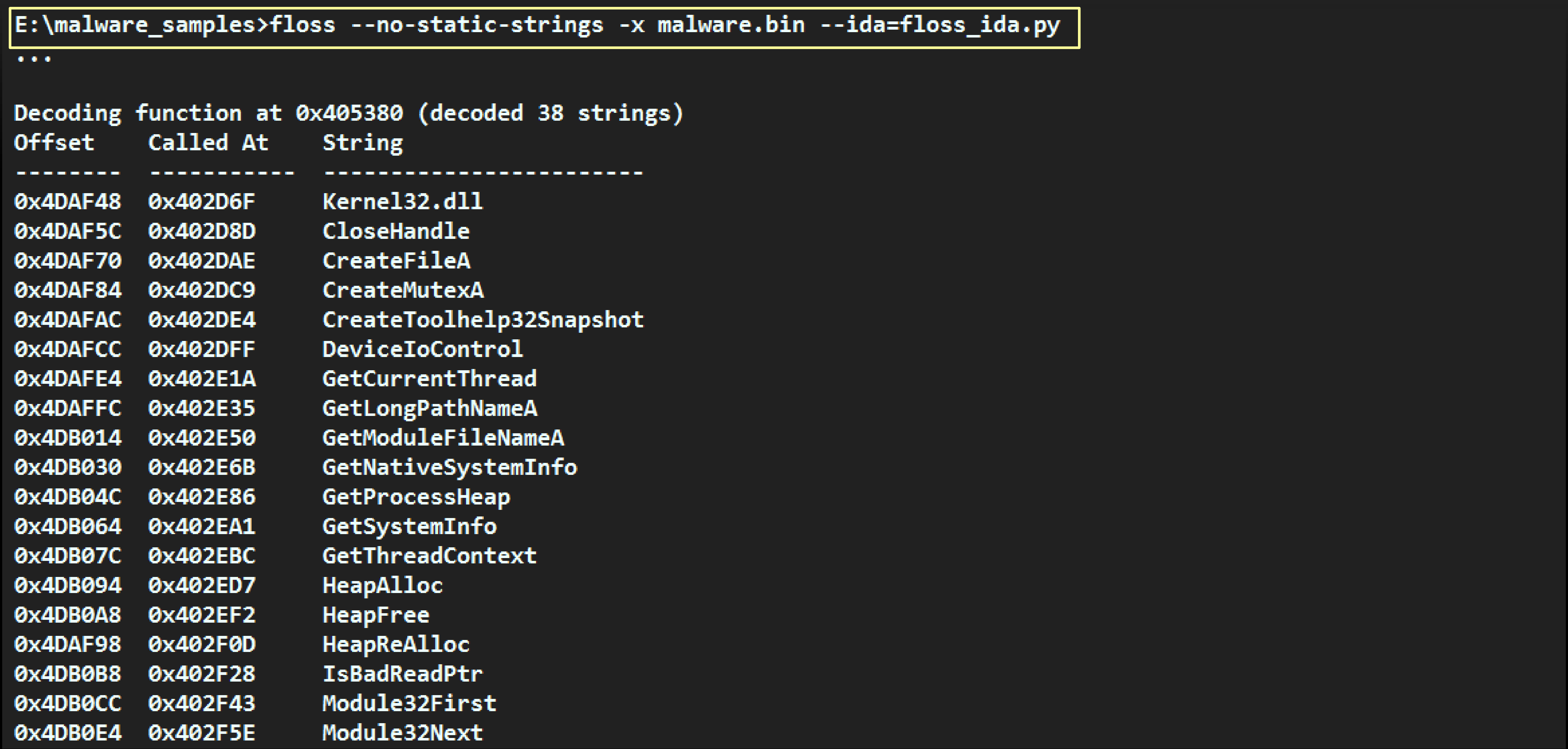
- Of course, we could try to improve and automatize the de-obfuscation of all functions names by using:

  - IDA Python: using IDA Python you can de-obfuscated function names and save them into the idb.
  - IDC: it’s a bit more complicated, but very powerful.

- If your time is short, so you could try emulation tools such as Floss (https://github.com/fireeye/flare-floss) to decode possible obfuscated strings and create an IDA script to decorate the reversed code:

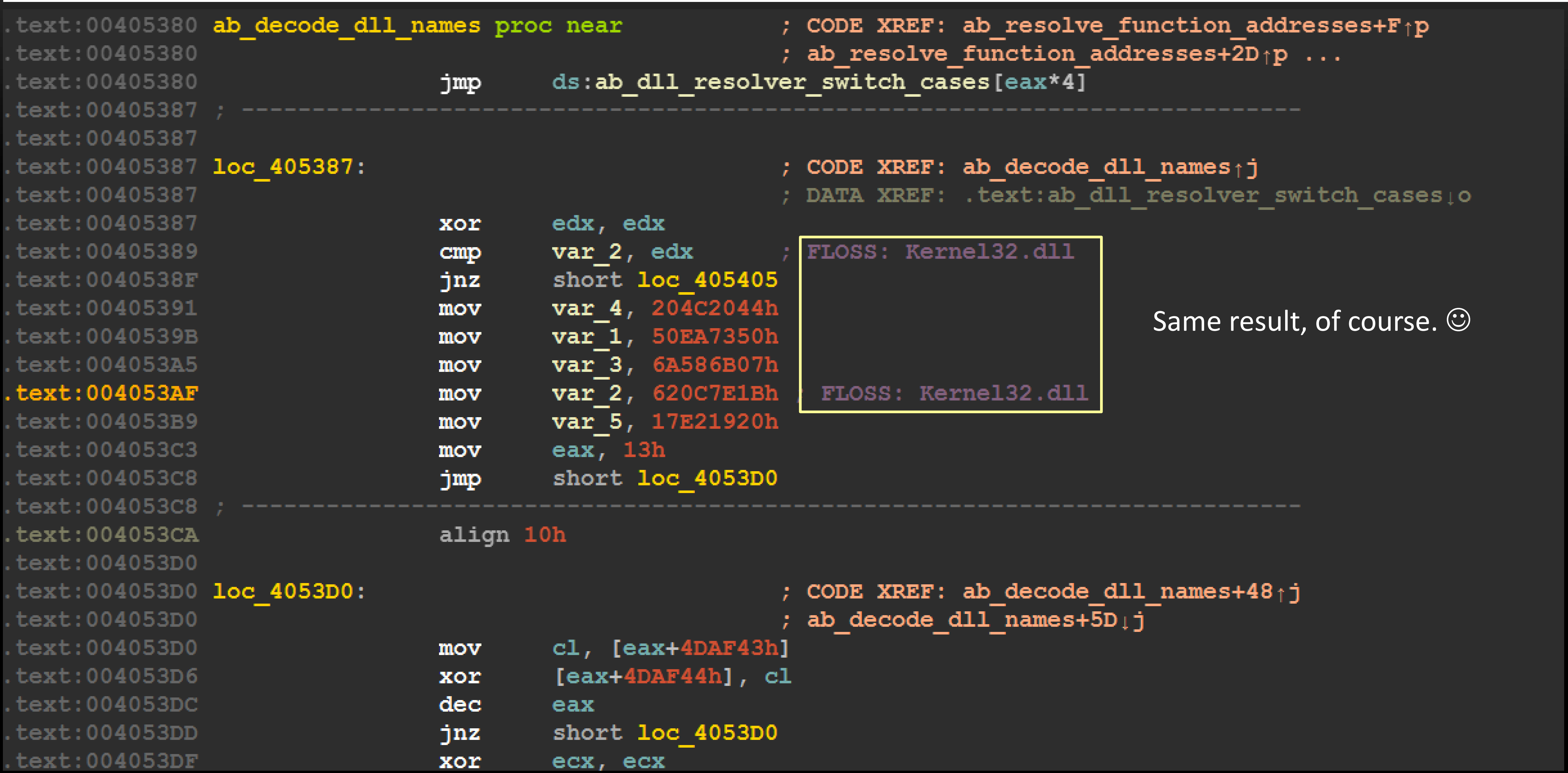
  - floss --no-static-strings -x malware.bin --ida=floss_ida.py

➢ INTRODUCTION



```
E:\malware_samples>floss --no-static-strings -x malware.bin --ida=floss_ida.py
...

Decoding function at 0x405380 (decoded 38 strings)
Offset    Called At    String
--------  -----------  ------------------------
0x4DAF48  0x402D6F     Kernel32.dll
0x4DAF5C  0x402D8D     CloseHandle
0x4DAF70  0x402DAE     CreateFileA
0x4DAF84  0x402DC9     CreateMutexA
0x4DAFAC  0x402DE4     CreateToolhelp32Snapshot
0x4DAFCC  0x402DFF     DeviceIoControl
0x4DAFE4  0x402E1A     GetCurrentThread
0x4DAFFC  0x402E35     GetLongPathNameA
0x4DB014  0x402E50     GetModuleFileNameA
0x4DB030  0x402E6B     GetNativeSystemInfo
0x4DB04C  0x402E86     GetProcessHeap
0x4DB064  0x402EA1     GetSystemInfo
0x4DB07C  0x402EBC     GetThreadContext
0x4DB094  0x402ED7     HeapAlloc
0x4DB0A8  0x402EF2     HeapFree
0x4DAF98  0x402F0D     HeapReAlloc
0x4DB0B8  0x402F28     IsBadReadPtr
0x4DB0CC  0x402F43     Module32First
0x4DB0E4  0x402F5E     Module32Next
```

➢ INTRODUCTION



```
.text:00405380 ab_decode_dll_names proc near           ; CODE XREF: ab_resolve_function_addresses+F↑p
.text:00405380                                         ; ab_resolve_function_addresses+2D↑p ...
.text:00405380                 jmp     ds:ab_dll_resolver_switch_cases[eax*4]
.text:00405387 ; ---------------------------------------------------------------------------
.text:00405387
.text:00405387 loc_405387:                             ; CODE XREF: ab_decode_dll_names↑j
.text:00405387                                         ; DATA XREF: .text:ab_dll_resolver_switch_cases↓o
.text:00405387                 xor     edx, edx
.text:00405389                 cmp     var_2, edx      ; FLOSS: Kernel32.dll
.text:0040538F                 jnz     short loc_405405
.text:00405391                 mov     var_4, 204C2044h
.text:0040539B                 mov     var_1, 50EA7350h
.text:004053A5                 mov     var_3, 6A586B07h
.text:004053AF                 mov     var_2, 620C7E1Bh ; FLOSS: Kernel32.dll
.text:004053B9                 mov     var_5, 17E21920h
.text:004053C3                 mov     eax, 13h
.text:004053C8                 jmp     short loc_4053D0
.text:004053C8 ; ---------------------------------------------------------------------------
.text:004053CA                 align 10h
.text:004053D0
.text:004053D0 loc_4053D0:                             ; CODE XREF: ab_decode_dll_names+48↑j
.text:004053D0                                         ; ab_decode_dll_names+5D↓j
.text:004053D0                 mov     cl, [eax+4DAF43h]
.text:004053D6                 xor     [eax+4DAF44h], cl
.text:004053DC                 dec     eax
.text:004053DD                 jnz     short loc_4053D0
.text:004053DF                 xor     ecx, ecx
```

Same result, of course. ☺

INTRODUCTION

- IDA Python is ALWAYS very handy and we can use IDA Pro SDK to write plugins for:

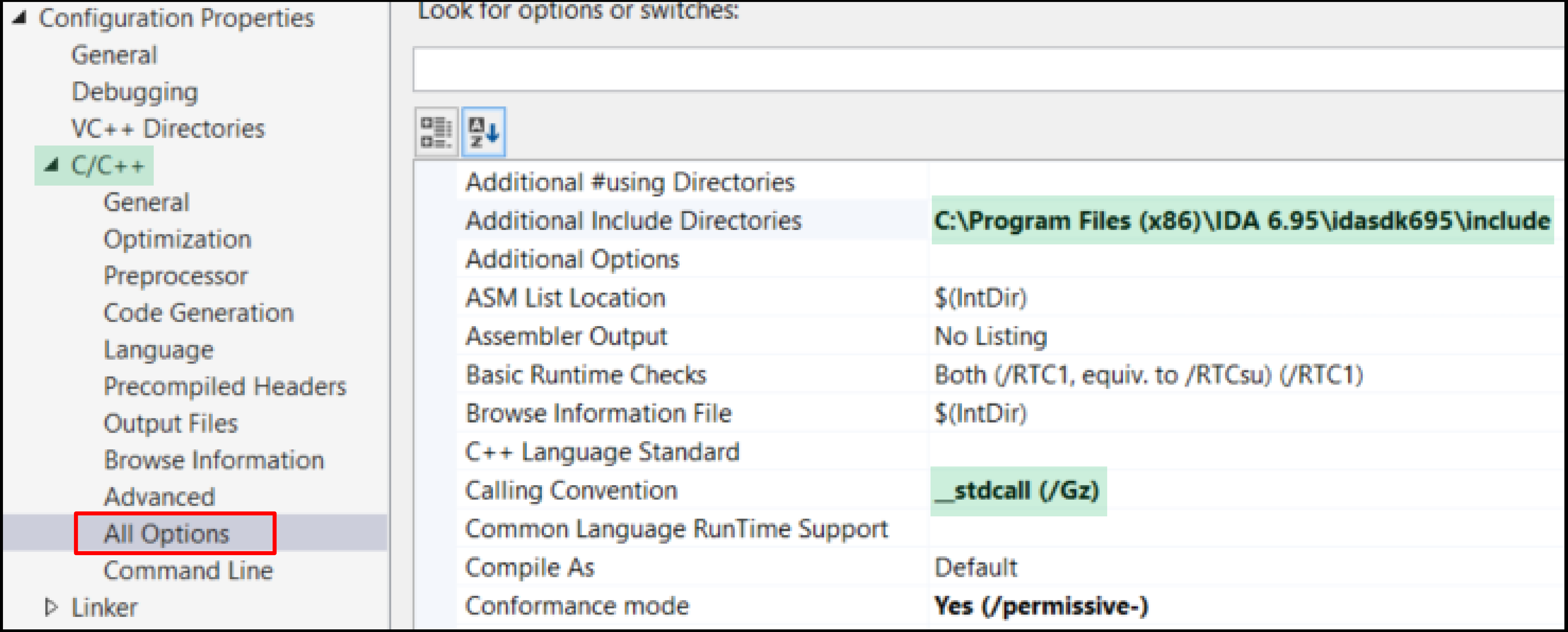
  - extending the IDA Pro functionalities
  - analyzing piece of obsfuscated code and data flow
  - automatizing unpacking of strange malicious files
  - decoding and loading encrypted / modified MBRs

- It is quick to create a simple IDA Pro plugin.

  - Download the IDA SDK from https://www.hex-rays.com/products/ida/support/download.shtml (likely, you will need a professional account).
  - Copy it to a folder (idasdk695/) within the IDA Pro installation directory.

➢ INTRODUCTION

- Create a project in Visual Studio 2017/2019 (File → New → Create Project → Visual C++ → Windows Desktop → Dynamic-Link Library (DLL)).
- Change few project properties as shown in this slide and next ones.



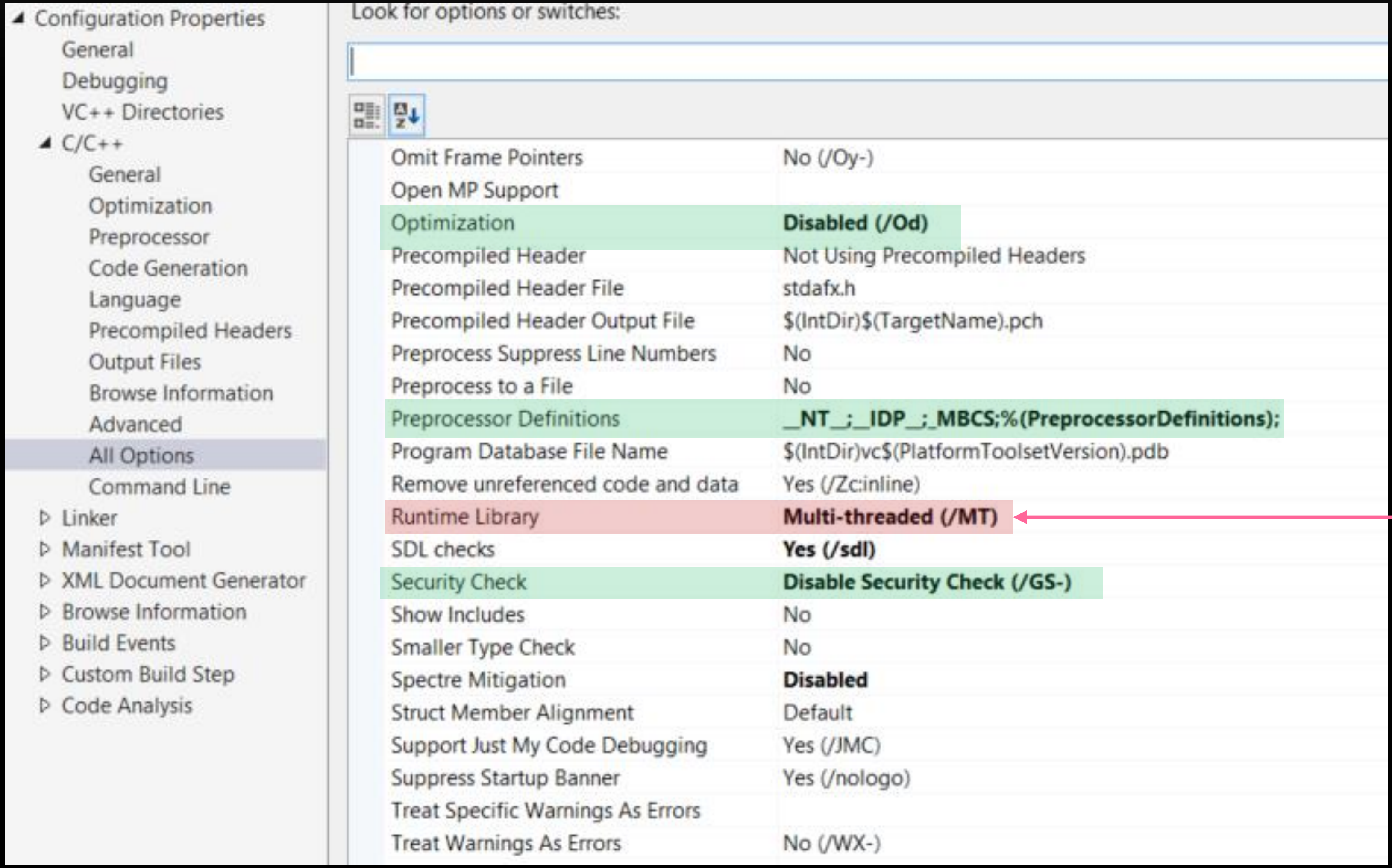
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation
    - Language
    - Precompiled Headers
    - Output Files
    - Browse Information
    - Advanced
    - All Options
    - Command Line
  - Linker

Look for options or switches:

Additional #using Directories	
Additional Include Directories	C:\Program Files (x86)\IDA 6.95\idasdk695\include
Additional Options	
ASM List Location	$(IntDir)
Assembler Output	No Listing
Basic Runtime Checks	Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Browse Information File	$(IntDir)
C++ Language Standard	
Calling Convention	__stdcall (/Gz)
Common Language RunTime Support	
Compile As	Default
Conformance mode	**Yes (/permissive-)**

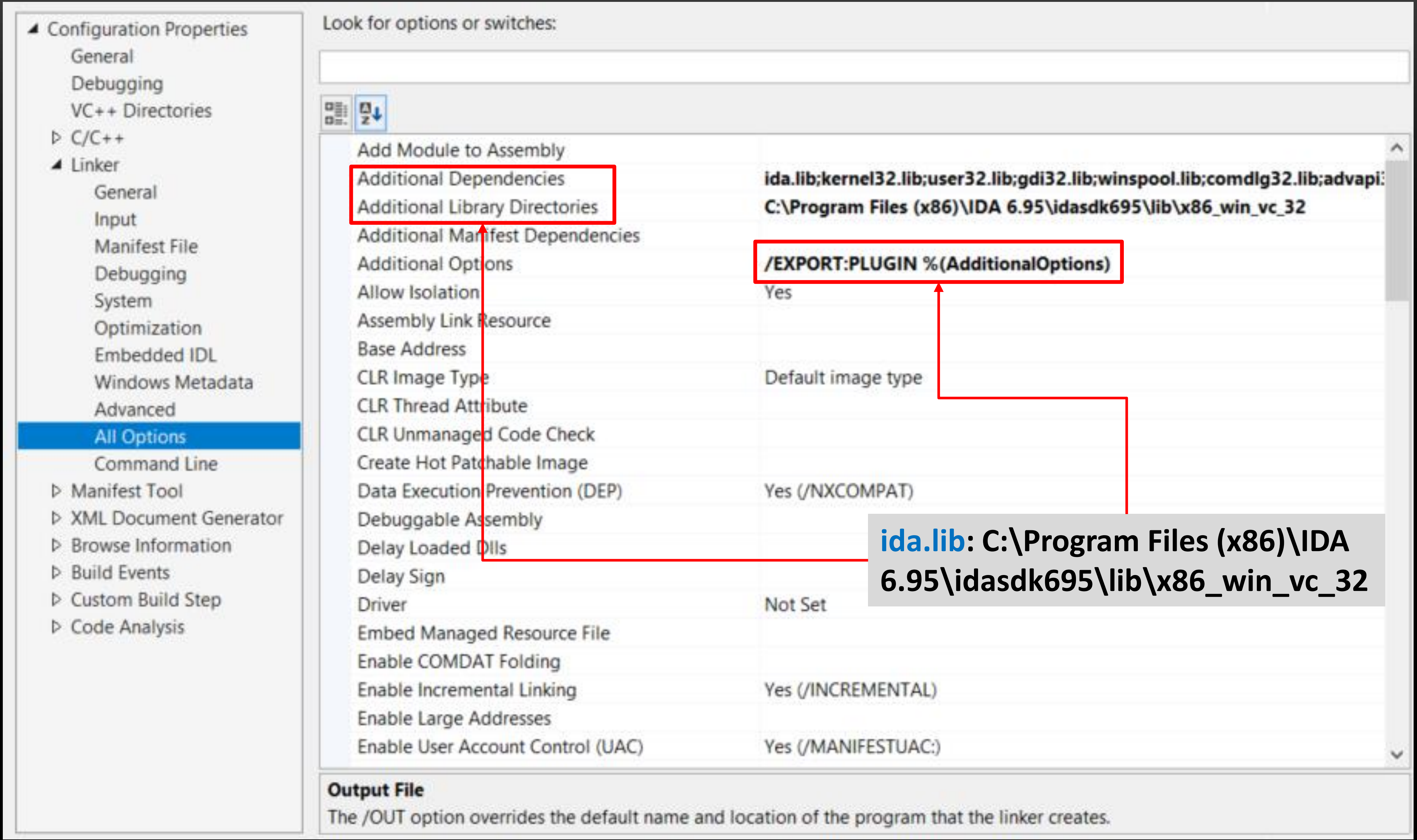
INTRODUCTION

- Include the "__NT__;__IDP__" in Processor Definitions and change Runtime Library to "Multi-threaded" (MT).



Omit Frame Pointers	No (/Oy-)
Open MP Support	
Optimization	**Disabled (/Od)**
Precompiled Header	Not Using Precompiled Headers
Precompiled Header File	stdafx.h
Precompiled Header Output File	$(IntDir)$(TargetName).pch
Preprocess Suppress Line Numbers	No
Preprocess to a File	No
Preprocessor Definitions	**__NT__;__IDP__;_MBCS;%(PreprocessorDefinitions);**
Program Database File Name	$(IntDir)vc$(PlatformToolsetVersion).pdb
Remove unreferenced code and data	Yes (/Zc:inline)
Runtime Library	**Multi-threaded (/MT)**
SDL checks	**Yes (/sdl)**
Security Check	**Disable Security Check (/GS-)**
Show Includes	No
Smaller Type Check	No
Spectre Mitigation	**Disabled**
Struct Member Alignment	Default
Support Just My Code Debugging	Yes (/JMC)
Suppress Startup Banner	Yes (/nologo)
Treat Specific Warnings As Errors	
Treat Warnings As Errors	No (/WX-)

Warning!
It is NOT /MTd

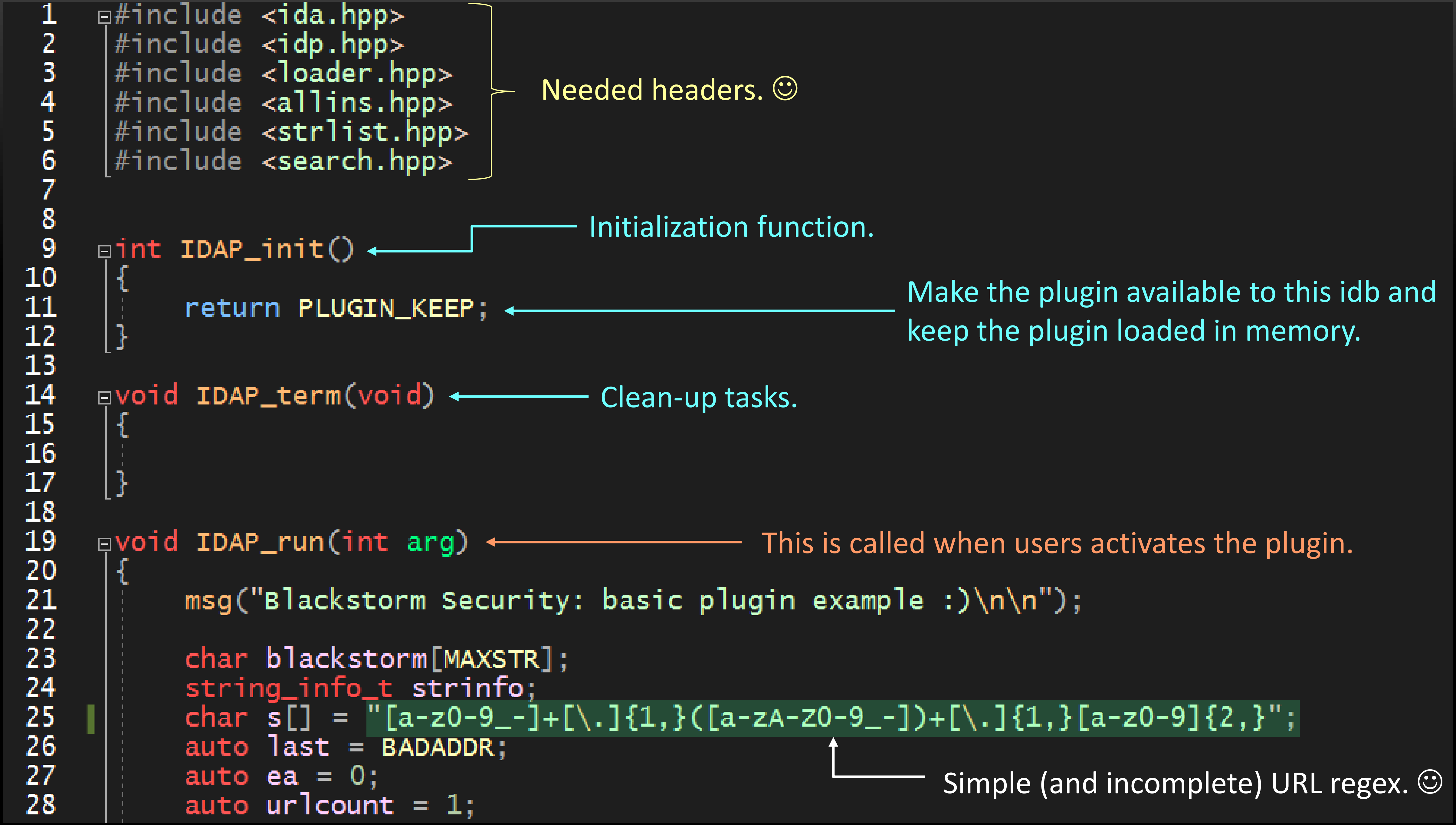


Add Module to Assembly	
Additional Dependencies	ida.lib;kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi:
Additional Library Directories	C:\Program Files (x86)\IDA 6.95\idasdk695\lib\x86_win_vc_32
Additional Manifest Dependencies	
Additional Options	/EXPORT:PLUGIN %(AdditionalOptions)
Allow Isolation	Yes
Assembly Link Resource	
Base Address	
CLR Image Type	Default image type
CLR Thread Attribute	
CLR Unmanaged Code Check	
Create Hot Patchable Image	
Data Execution Prevention (DEP)	Yes (/NXCOMPAT)
Debuggable Assembly	
Delay Loaded Dlls	
Delay Sign	
Driver	Not Set
Embed Managed Resource File	
Enable COMDAT Folding	
Enable Incremental Linking	Yes (/INCREMENTAL)
Enable Large Addresses	
Enable User Account Control (UAC)	Yes (/MANIFESTUAC:)

Output File

The /OUT option overrides the default name and location of the program that the linker creates.

ida.lib: C:\Program Files (x86)\IDA 6.95\idasdk695\lib\x86_win_vc_32



```
#include <ida.hpp>
#include <idp.hpp>
#include <loader.hpp>
#include <allins.hpp>
#include <strlist.hpp>
#include <search.hpp>


int IDAP_init()
{
    return PLUGIN_KEEP;
}

void IDAP_term(void)
{
    
    
}

void IDAP_run(int arg)
{
    msg("Blackstorm Security: basic plugin example :)\n\n");

    char blackstorm[MAXSTR];
    string_info_t strinfo;
    char s[] = "[a-z0-9_-]+[\.]{1,}([a-zA-Z0-9_-])+[\.]{1,}[a-z0-9]{2,}";
    auto last = BADADDR;
    auto ea = 0;
    auto urlcount = 1;
```

Needed headers. ☺

Initialization function.

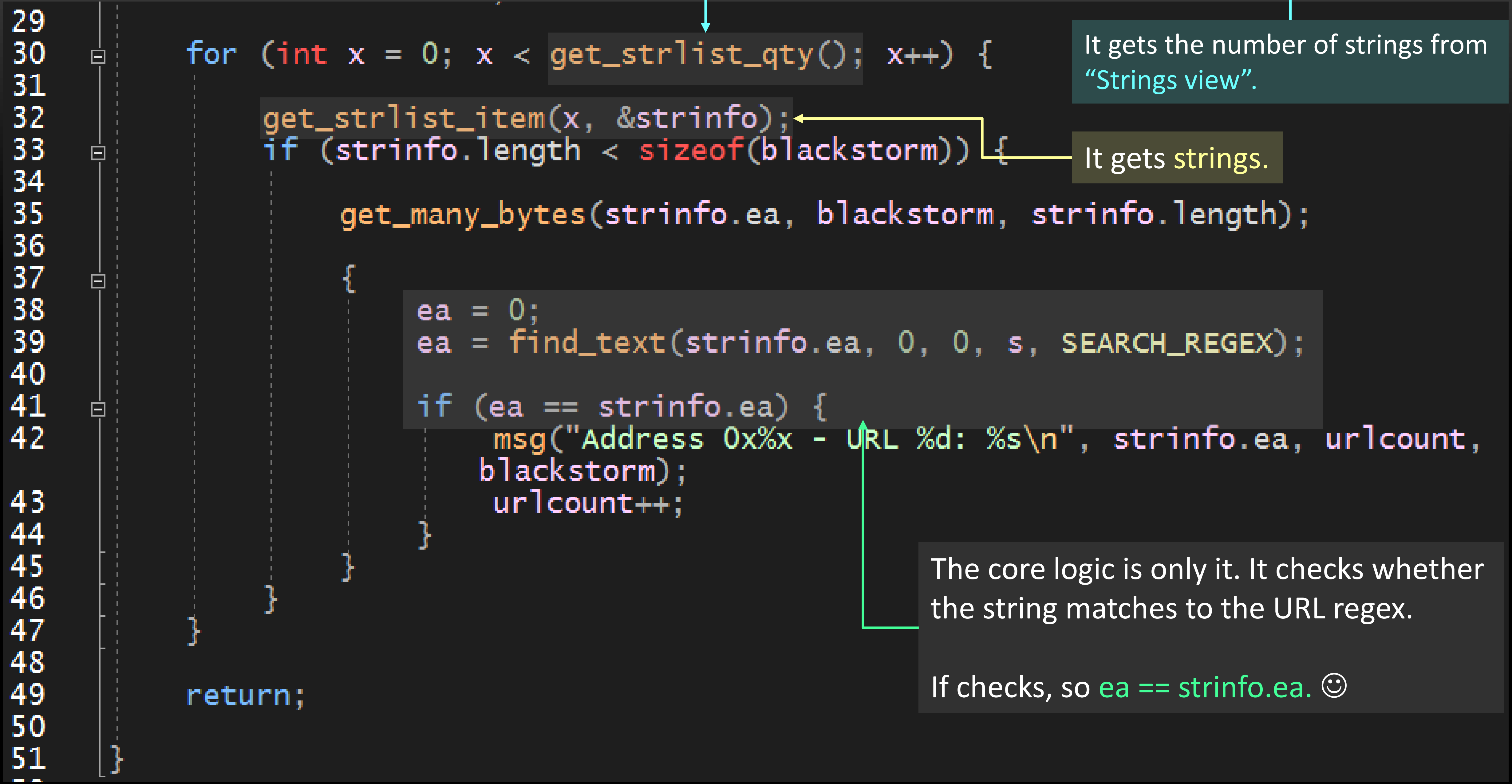
Make the plugin available to this idb and keep the plugin loaded in memory.

Clean-up tasks.

This is called when users activates the plugin.

Simple (and incomplete) URL regex. ☺

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER



```
    for (int x = 0; x < get_strlist_qty(); x++) {

        get_strlist_item(x, &strinfo);
        if (strinfo.length < sizeof(blackstorm)) {

            get_many_bytes(strinfo.ea, blackstorm, strinfo.length);

            {
                ea = 0;
                ea = find_text(strinfo.ea, 0, 0, s, SEARCH_REGEX);

                if (ea == strinfo.ea) {
                    msg("Address 0x%x - URL %d: %s\n", strinfo.ea, urlcount,
                   blackstorm);
                    urlcount++;
                }
            }
        }
    }

    return;

}
```

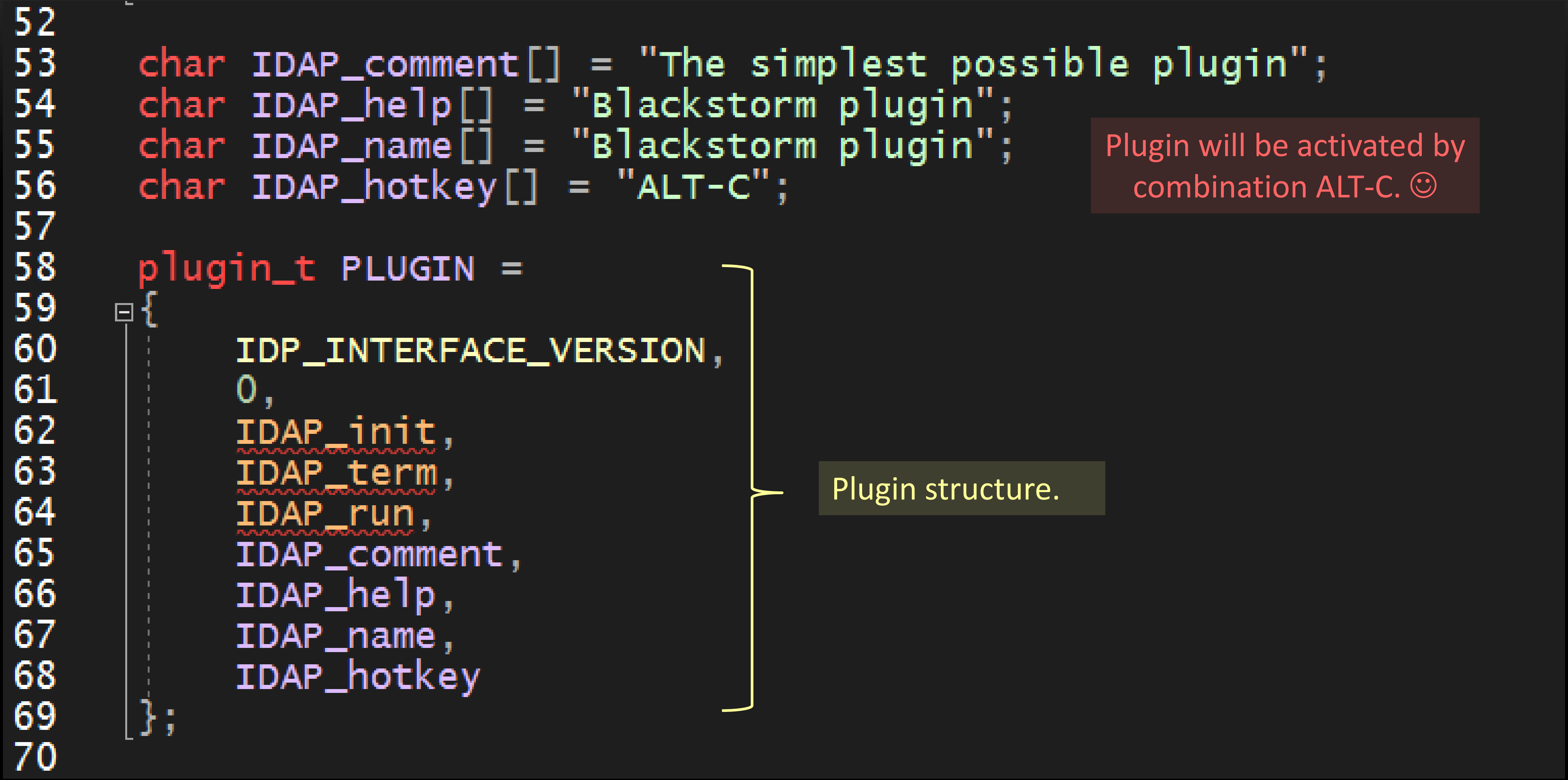
It gets the number of strings from "Strings view".

It gets strings.

The core logic is only it. It checks whether the string matches to the URL regex.

If checks, so ea == strinfo.ea. ☺

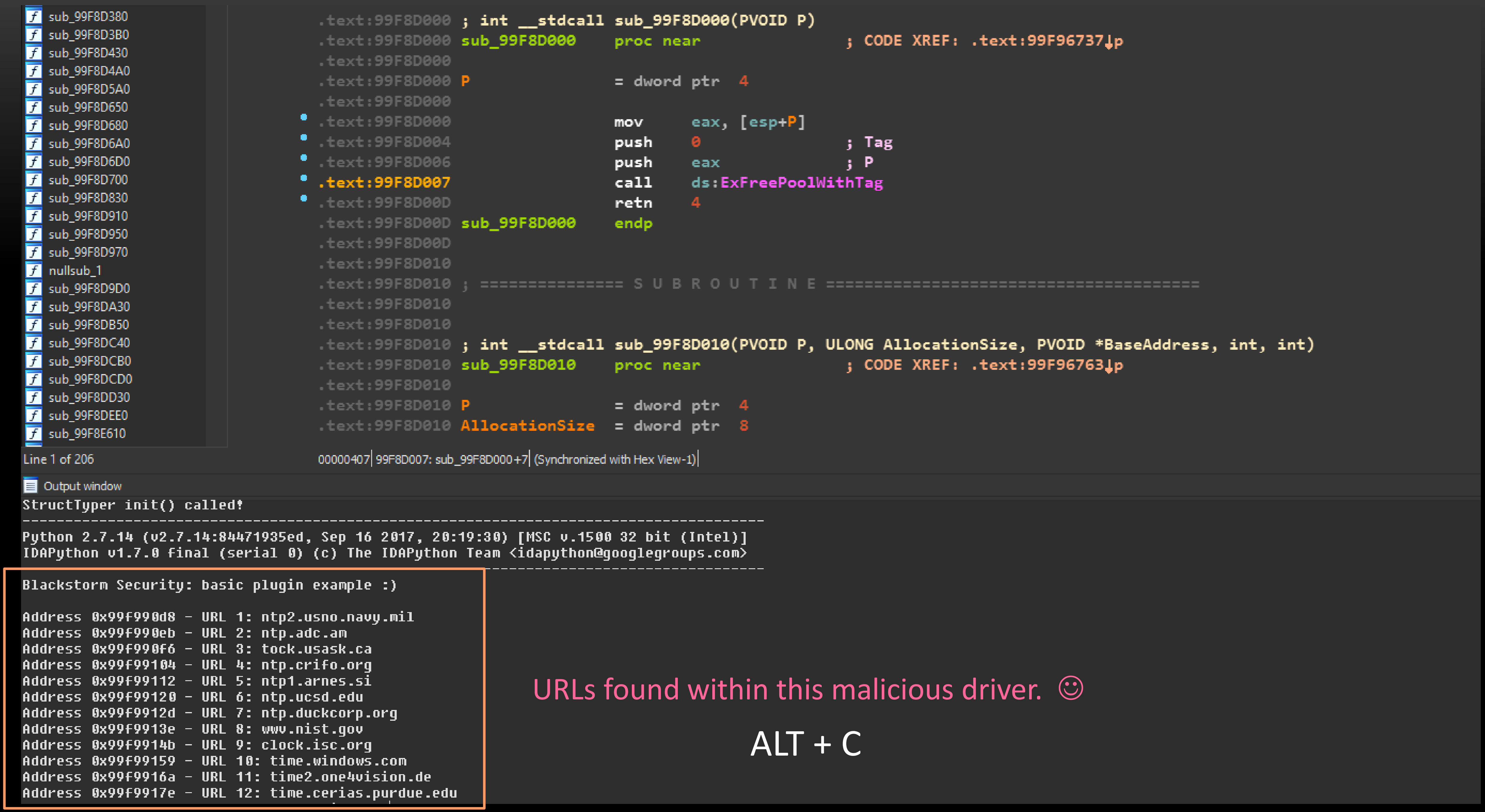
➢ INTRODUCTION



```
    char IDAP_comment[] = "The simplest possible plugin";
    char IDAP_help[] = "Blackstorm plugin";
    char IDAP_name[] = "Blackstorm plugin";
    char IDAP_hotkey[] = "ALT-C";

    plugin_t PLUGIN =
    {
        IDP_INTERFACE_VERSION,
        0,
        IDAP_init,
        IDAP_term,
        IDAP_run,
        IDAP_comment,
        IDAP_help,
        IDAP_name,
        IDAP_hotkey
    };
```

Plugin will be activated by combination ALT-C. ☺

Plugin structure.



```
sub_99F8D380
sub_99F8D3B0
sub_99F8D430
sub_99F8D4A0
sub_99F8D5A0
sub_99F8D650
sub_99F8D680
sub_99F8D6A0
sub_99F8D6D0
sub_99F8D700
sub_99F8D830
sub_99F8D910
sub_99F8D950
sub_99F8D970
nullsub_1
sub_99F8D9D0
sub_99F8DA30
sub_99F8DB50
sub_99F8DC40
sub_99F8DCB0
sub_99F8DCD0
sub_99F8DD30
sub_99F8DEE0
sub_99F8E610
Line 1 of 206
```

```
.text:99F8D000 ; int __stdcall sub_99F8D000(PVOID P)
.text:99F8D000 sub_99F8D000    proc near               ; CODE XREF: .text:99F96737↓p
.text:99F8D000
.text:99F8D000 P               = dword ptr  4
.text:99F8D000
.text:99F8D000                 mov     eax, [esp+P]
.text:99F8D004                 push    0               ; Tag
.text:99F8D006                 push    eax             ; P
.text:99F8D007                 call    ds:ExFreePoolWithTag
.text:99F8D00D                 retn    4
.text:99F8D00D sub_99F8D000    endp
.text:99F8D00D
.text:99F8D010
.text:99F8D010 ; =============== S U B R O U T I N E =======================================
.text:99F8D010
.text:99F8D010
.text:99F8D010 ; int __stdcall sub_99F8D010(PVOID P, ULONG AllocationSize, PVOID *BaseAddress, int, int)
.text:99F8D010 sub_99F8D010    proc near               ; CODE XREF: .text:99F96763↓p
.text:99F8D010
.text:99F8D010 P               = dword ptr  4
.text:99F8D010 AllocationSize  = dword ptr  8
```

```
00000407 99F8D007: sub_99F8D000+7 (Synchronized with Hex View-1)
```

Output window

```
StructTyper init() called!
-----------------------------------------------------------------------------------------
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (Intel)]
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
-----------------------------------------------------------------------------------------
Blackstorm Security: basic plugin example :)

Address 0x99f990d8 - URL 1: ntp2.usno.navy.mil
Address 0x99f990eb - URL 2: ntp.adc.am
Address 0x99f990f6 - URL 3: tock.usask.ca
Address 0x99f99104 - URL 4: ntp.crifo.org
Address 0x99f99112 - URL 5: ntp1.arnes.si
Address 0x99f99120 - URL 6: ntp.ucsd.edu
Address 0x99f9912d - URL 7: ntp.duckcorp.org
Address 0x99f9913e - URL 8: www.nist.gov
Address 0x99f9914b - URL 9: clock.isc.org
Address 0x99f99159 - URL 10: time.windows.com
Address 0x99f9916a - URL 11: time2.one4vision.de
Address 0x99f9917e - URL 12: time.cerias.purdue.edu
```

URLs found within this malicious driver. ☺

ALT + C

ANTI-REVERSING

ANTI-REVERSING

- Advanced threats don't use standard tricks and there're many lots of facts about them:

  - They use similar techniques from modern packers such as Themida, Arxan, Agile .NET, Tigress, Obfuscator-LLVM and so on.

  - Most of them are focused on 64-bit code.

  - Obviously, almost all functions are removed from IAT. Remember that Themida packer usually keeps only TlsSetValue( ).

  - String encryption is also a common technique.

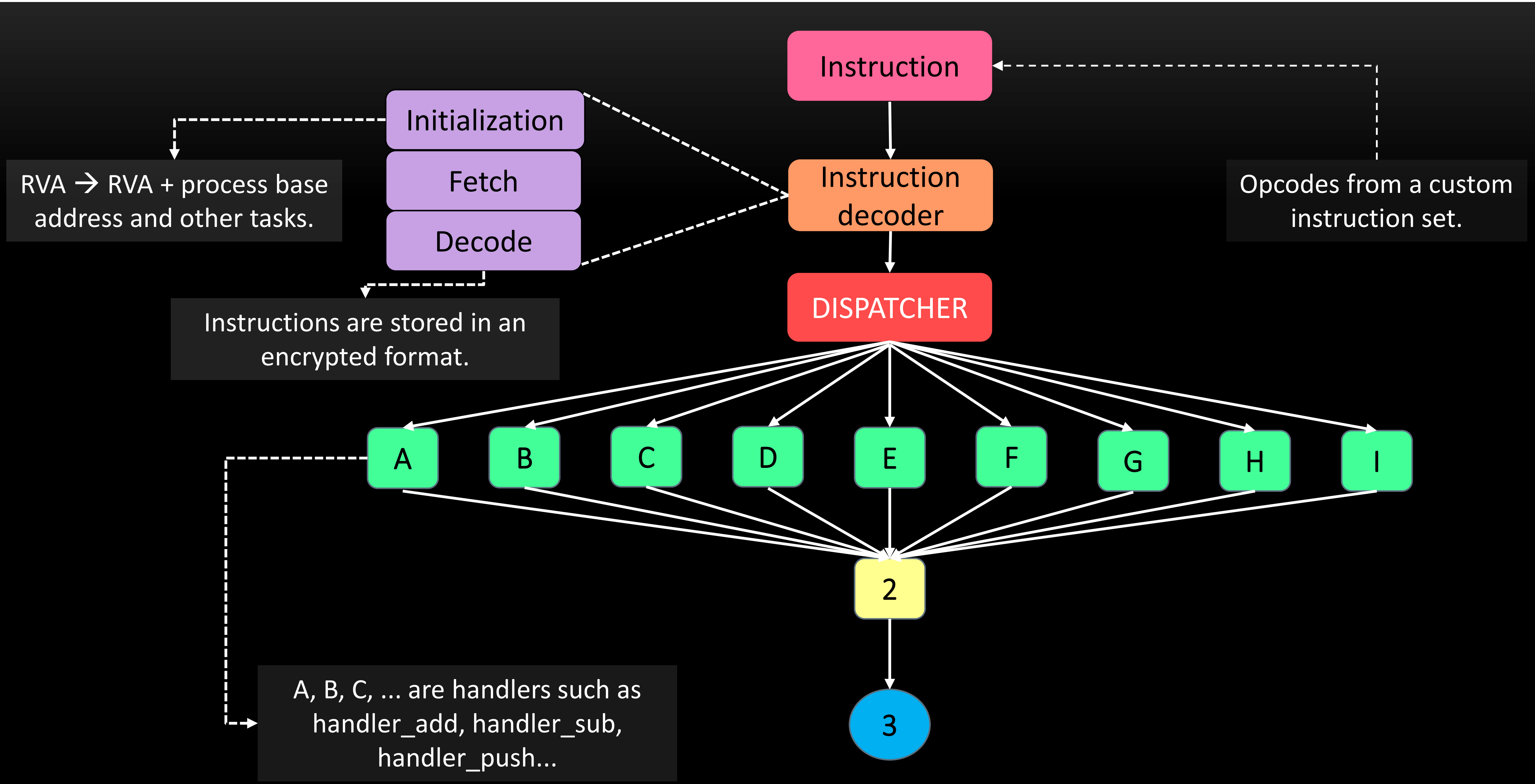
➢ ANTI-REVERSING

- Advanced threats are concerned to integrity:
  - They protect and check the memory integrity.
  - Thus, it is not possible to dump a clean executable from the memory (using Volatility, for example) because original instructions are not completely decoded in the memory.
  - There could be checksum functions verifing the integrity of the code itself. Therefore, any attempt of changing the code may break it up.
  - Additionally, advanced threats may use watermark to control the "ownership". This is the same technique used to program copyright. ☺

ANTI-REVERSING

- Many additional techniques are used:
  - There are also fake push instructions.
  - There are many dead and useless pieces of code.
  - There is some code reordering using unconditional jumps.
  - All obfuscators use code flattening.
  - Packers have few anti-debugger and anti-vm tricks. Weird anti-vm methods based on temperature, for example.

ANTI-REVERSING

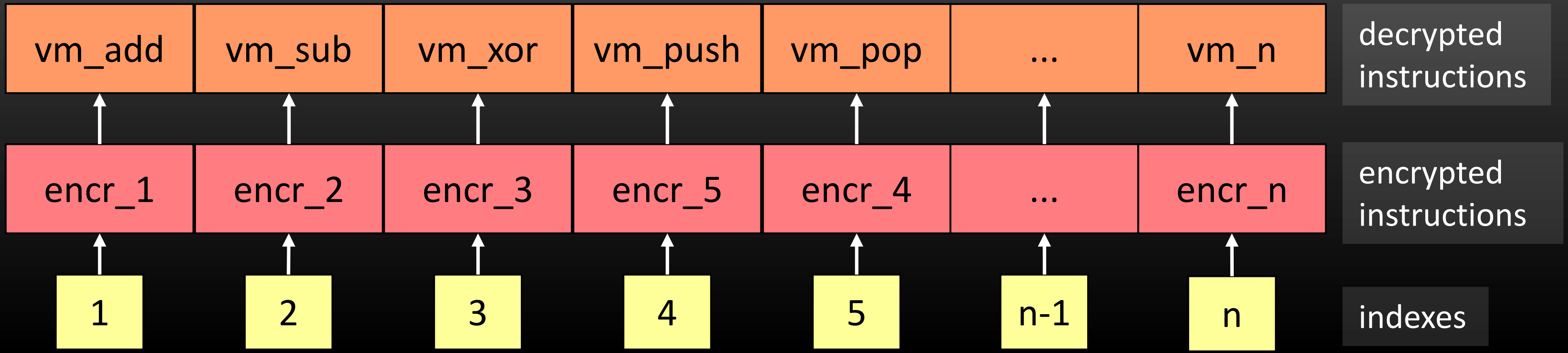


RVA → RVA + process base address and other tasks.

Opcodes from a custom instruction set.

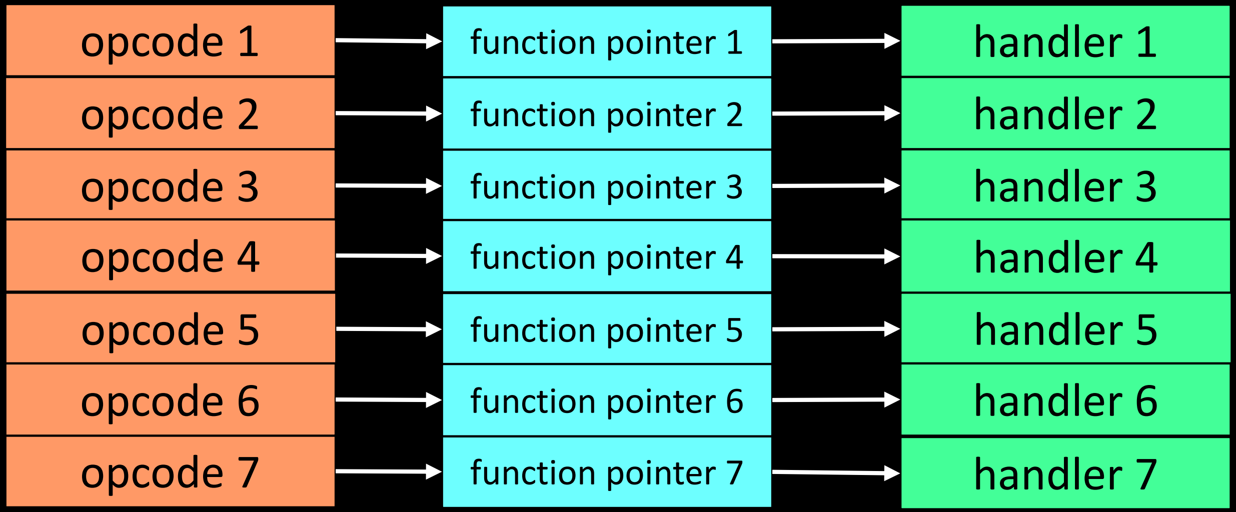
Instructions are stored in an encrypted format.

A, B, C, ... are handlers such as handler_add, handler_sub, handler_push...



vm_add	vm_sub	vm_xor	vm_push	vm_pop	...	vm_n	decrypted instructions
encr_1	encr_2	encr_3	encr_5	encr_4	...	encr_n	encrypted instructions
1	2	3	4	5	n-1	n	indexes

recovering and decrypting functions



opcode 1	function pointer 1	handler 1
opcode 2	function pointer 2	handler 2
opcode 3	function pointer 3	handler 3
opcode 4	function pointer 4	handler 4
opcode 5	function pointer 5	handler 5
opcode 6	function pointer 6	handler 6
opcode 7	function pointer 7	handler 7

function pointer table (likely encrypted)

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

➢ ANTI-REVERSING

Remember: obfuscating is transforming a code from A to B by using different techniques (including **virtualization**).

What're the **transition points** from **native code to virtualized code** and vice-versa?

Prologues and epilogues from each function **could not be virtualized**. Take care. ☺

Have you tried to open an advanced packer in **IDA Pro**? First sight: **only red and grey blocks (non-functions and data).** ☹

ANTI-REVERSING

- In few cases the **VM handlers** come **from data blocks.**

- Original **code section could be "splitted" and "scattered"** around the program. In this case, **data and instructions are mixed in the binary**, without having just one instruction block.

- **Instructions which reference imported functions** could have been either **zeroed or replaced by NOP.** ☹

- Most certainly, they will be **restored (re-inserted) dynamically** by the packer later.

➢ ANTI-REVERSING

- If **references** are not zeroed, so they are usually **translated to short jumps using RVA**, for the **same import address** ("IAT obfuscation") ☺

- **API names** could be **hashed** (as used in shellcodes). ☺

- Custom packers usually **don't virtualize all native (x86/x64) instructions.**

- There's a **mix** between **virtualized, native instructions** and data after the packing procedure.

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

ANTI-REVERSING

- **Native APIs** could be **redirected to stub code (proxy)**, which **forwards the call to (copied) native DLLs** (from the respective APIs).

- The **“hidden” function code** could be **copied** (memcpy( )) to **memory allocated** by VirtualAlloc( ) ☺ Of course, there must be a **fixup in the code to get these instructions**.

ANTI-REVERSING

- By the way, **how many virtualized instructions** exist in the binary?

- It is recommended to try to **find handlers to native x86 instructions** (non-virtualized instruction)

- Try to **classify virtualized instructions in groups** according to operands and their purpose such as **memory access, conditional/unconditional jumps, arithmetic, general**...

- Try to understand the **size of virtualized instructions** which we might fit into a **structure that represents encryption key, data, RVA (location), opcode (type)** and so on.

ANTI-REVERSING

- Pay attention to **instruction's stem** to put **similar classes of instructions** together as, for example, **jump instructions, direct calls, indirect calls and** so on.

- Find the **transition instructions from native mode to virtualized mode** and vice versa.

- Find **similarity between virtualized instructions and x86 instructions.**

- **x86 instructions** are also kept **encrypted and compressed together with the virtualized instructions**.

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

ANTI-REVERSING

- Constant unfolding: technique used by obfuscators to replace a contant by a bunch of code that produces the same resulting constant's value.

- Pattern-based obfuscation: exchange of one instruction by a set of equivalent instructions.

- Abusing inline functions.

- Anti-VM techniques: prevents the malware sample to run inside a VM.

- Dead (garbage) code: this technique is implemented by inserting codes whose results will be overwritten in next lines of code or, worse, they won't be used anymore.

➢ ANTI-REVERSING

- Code duplication: different paths coming into the same destination (used by virtualization obfuscators).

- Control indirection 1: call instruction → stack pointer update → return skipping some junk code after the call instruction (RET x).

- Control indirection 2: malware trigger an exception → registered exception is called → new branch of instructions.

- Anti-debugging: used as irritating techniques to slow the process analysis.

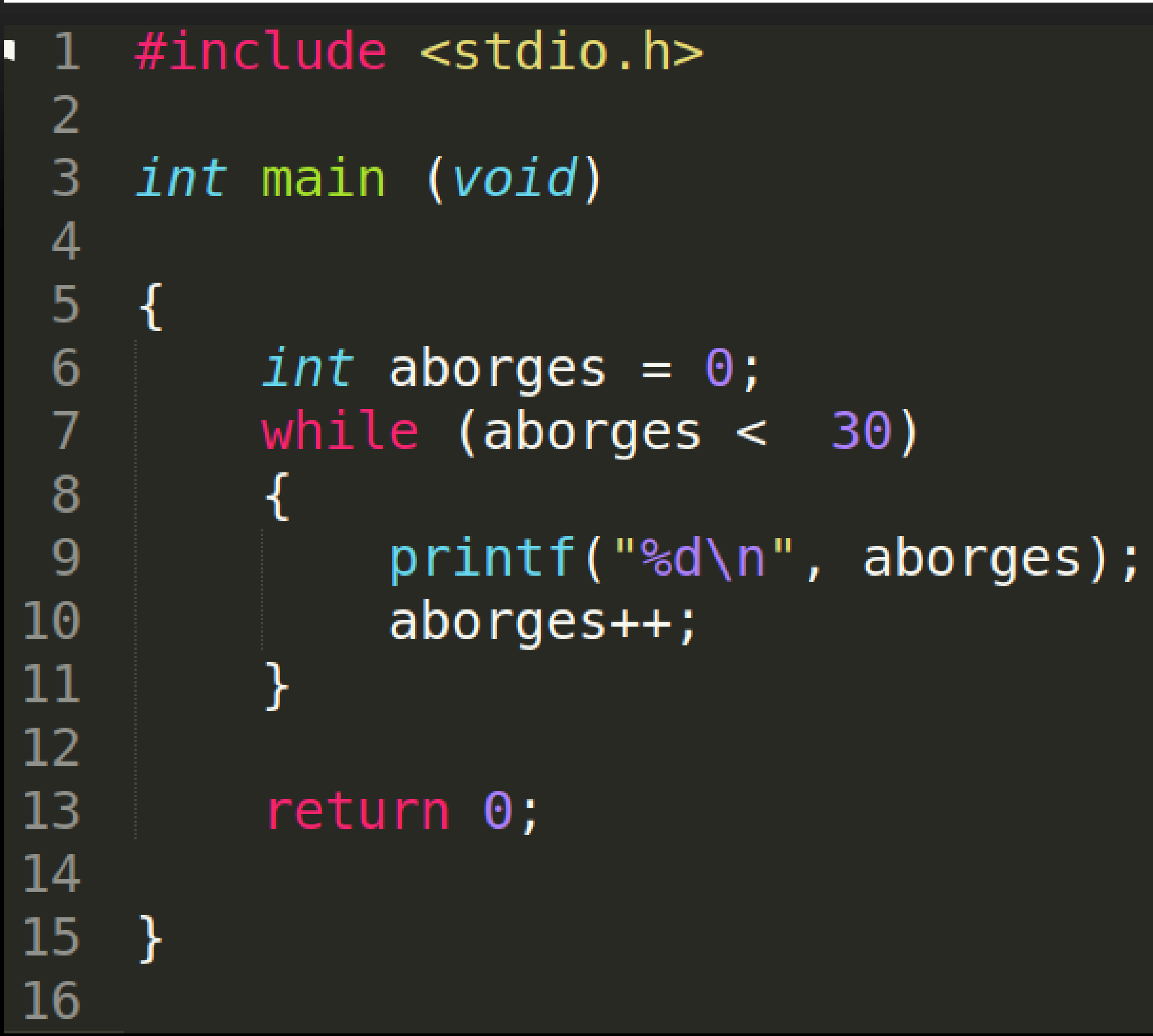
ANTI-REVERSING

- Opaque predicate: Although apparently there is an evaluation (conditional jump: jz/jnz), the result is always evaluated to true (or false), which means an unconditional jump. Thus, there is a dead branch.

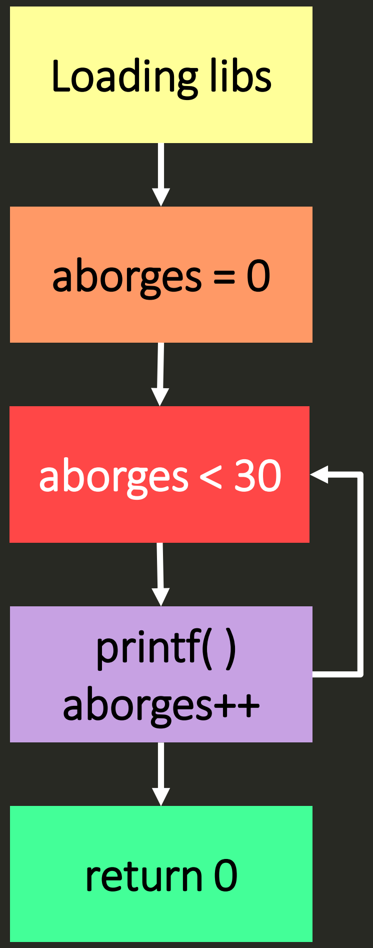
- Polymorphism: it is produced by self-modification code (like shellcodes) and by encrypting resources (similar most malware samples).

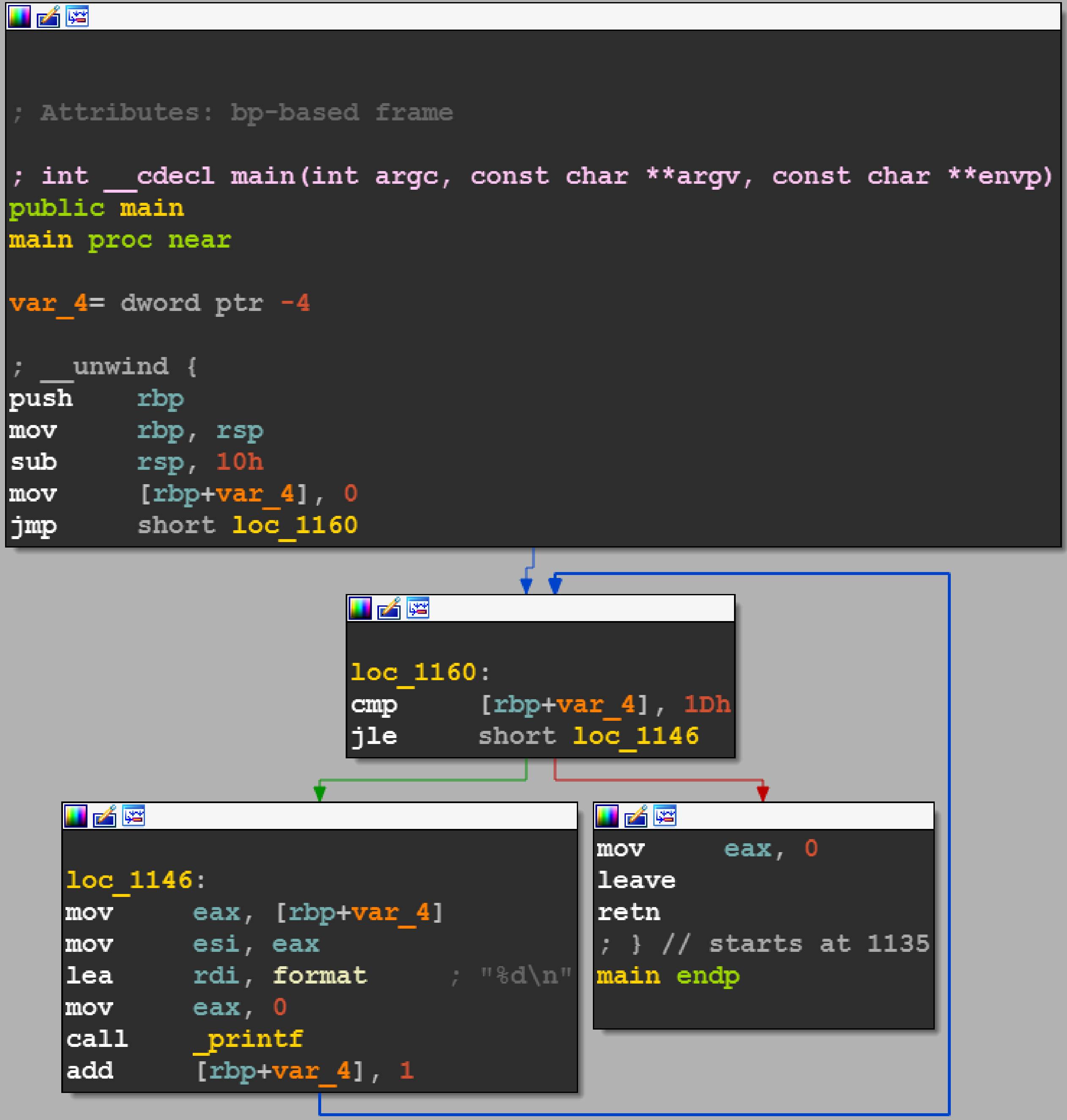
ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

ANTI-REVERSING



```c
#include <stdio.h>

int main (void)

{
    int aborges = 0;
    while (aborges <  30)
    {
        printf("%d\n", aborges);
        aborges++;
    }

    return 0;

}

```





```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], 0
jmp     short loc_1160
```

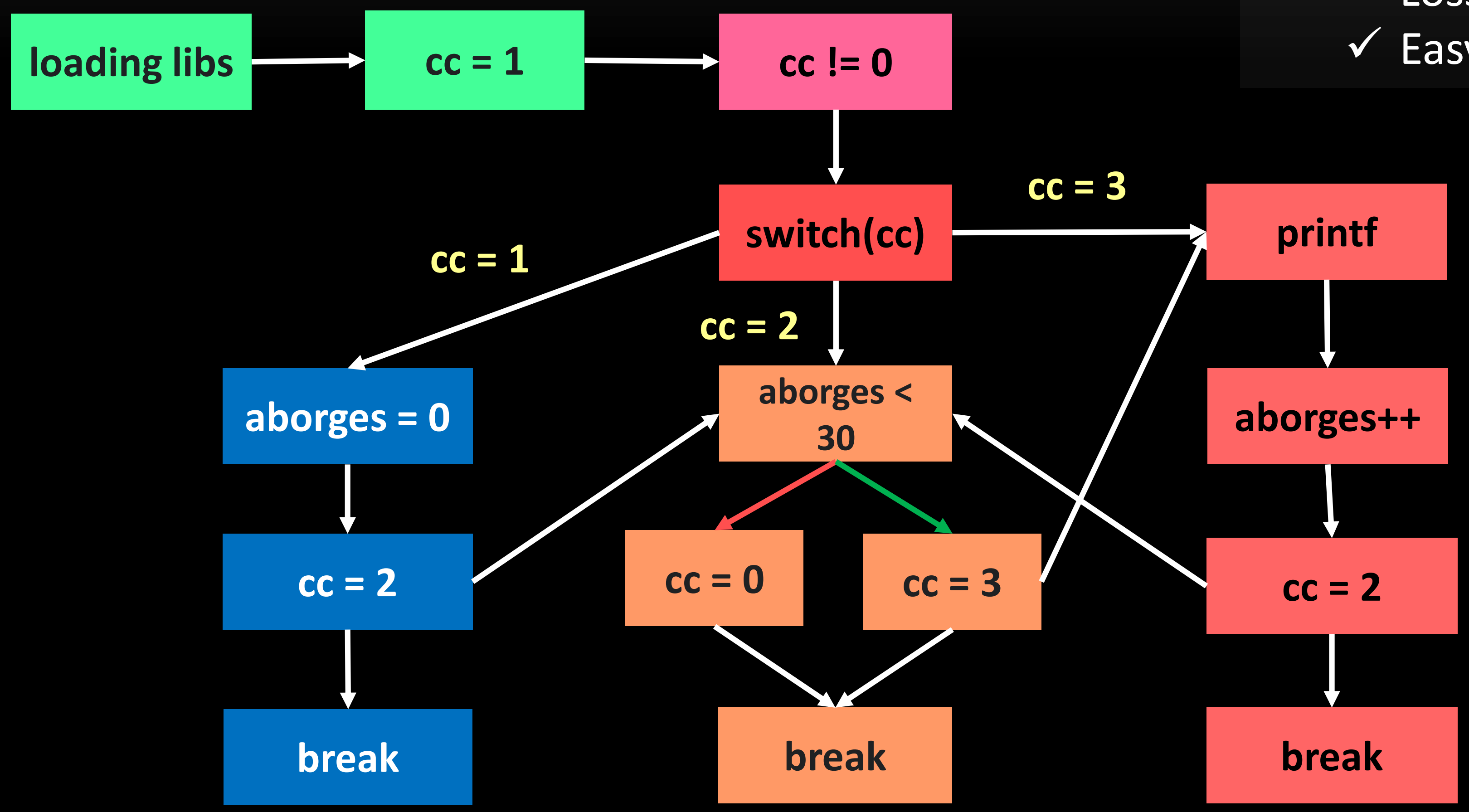
```
loc_1160:
cmp     [rbp+var_4], 1Dh
jle     short loc_1146
```

```
loc_1146:
mov     eax, [rbp+var_4]
mov     esi, eax
lea     rdi, format     ; "%d\n"
mov     eax, 0
call    _printf
add     [rbp+var_4], 1
```

```
mov     eax, 0
leave
retn
; } // starts at 1135
main endp
```

Original Program

ANTI-REVERSING



- Disavantages:
  - ✓ Loss of performance
  - ✓ Easy to identify the Code flattening

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

ANTI-REVERSING

- The obfuscator-llvm is an excellent project to be used for code obsfuscation. To install it, it is recommended to add a swap file first (because the linkage stage):

  - fallocate -l 8GB /swapfile ; chmod 600 /swapfile
  - mkswap /swapfile ; swapon /swapfile ; swapon --show
  - apt-get install llvm-4.0
  - apt-get install gcc-multilib (install gcc lib support to 32 bit)
  - git clone -b llvm-4.0 https://github.com/obfuscator-llvm/obfuscator.git
  - mkdir build ; cd build/
  - cmake -DCMAKE_BUILD_TYPE=Release -DLLVM_INCLUDE_TESTS=OFF ../obfuscator/
  - make -j7

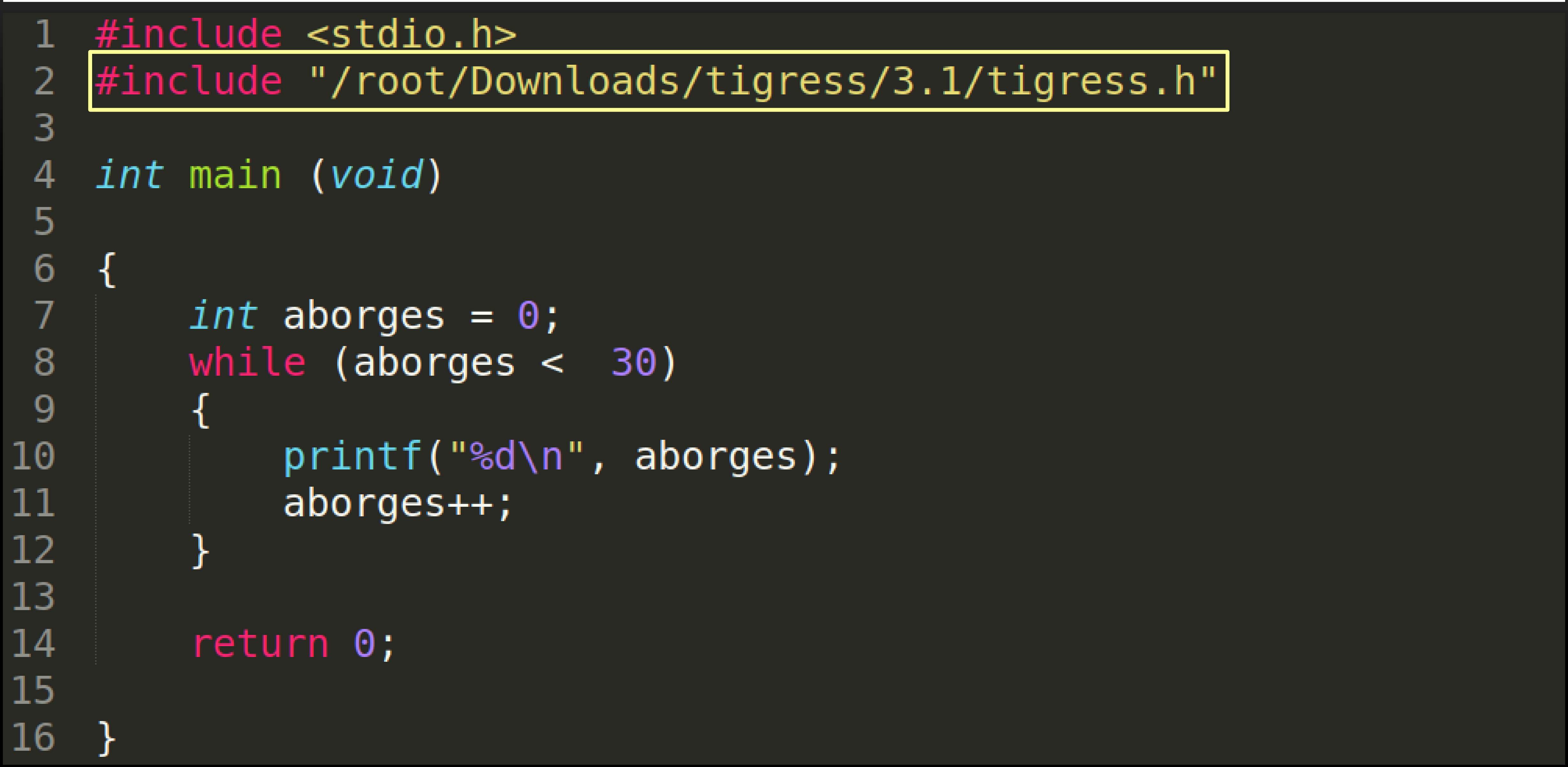
ANTI-REVERSING

- Possible usages:
  - ./build/bin/clang alexborges.c -o alexborges -mllvm -fla
  - ./build/bin/clang alexborges.c -m32 -o alexborges -mllvm -fla
  - ./build/bin/clang alexborges.c -o alexborges -mllvm -fla -mllvm -sub
- Where:
  - fla: Control Flow Flattening
  - sub: Instruction Substitution
  - bcf: Opaque Predicate

ANTI-REVERSING

- A better option would be using Tigress.
- Download Tigress binary from https://tigress.wtf/download.html
- Install Tigress is pretty easy:
  - unzip tigress-3.1-bin.zip
  - Export the TIGRESS_HOME environment variable:
    - export TIGRESS_HOME=/root/Downloads/tigress/3.1
  - Add the Tigress installation directory to the PATH variable:
    - export PATH=$PATH:/root/Downloads/tigress/3.1

ANTI-REVERSING



```c
#include <stdio.h>
#include "/root/Downloads/tigress/3.1/tigress.h"

int main (void)

{
    int aborges = 0;
    while (aborges <  30)
    {
        printf("%d\n", aborges);
        aborges++;
    }

    return 0;

}
```

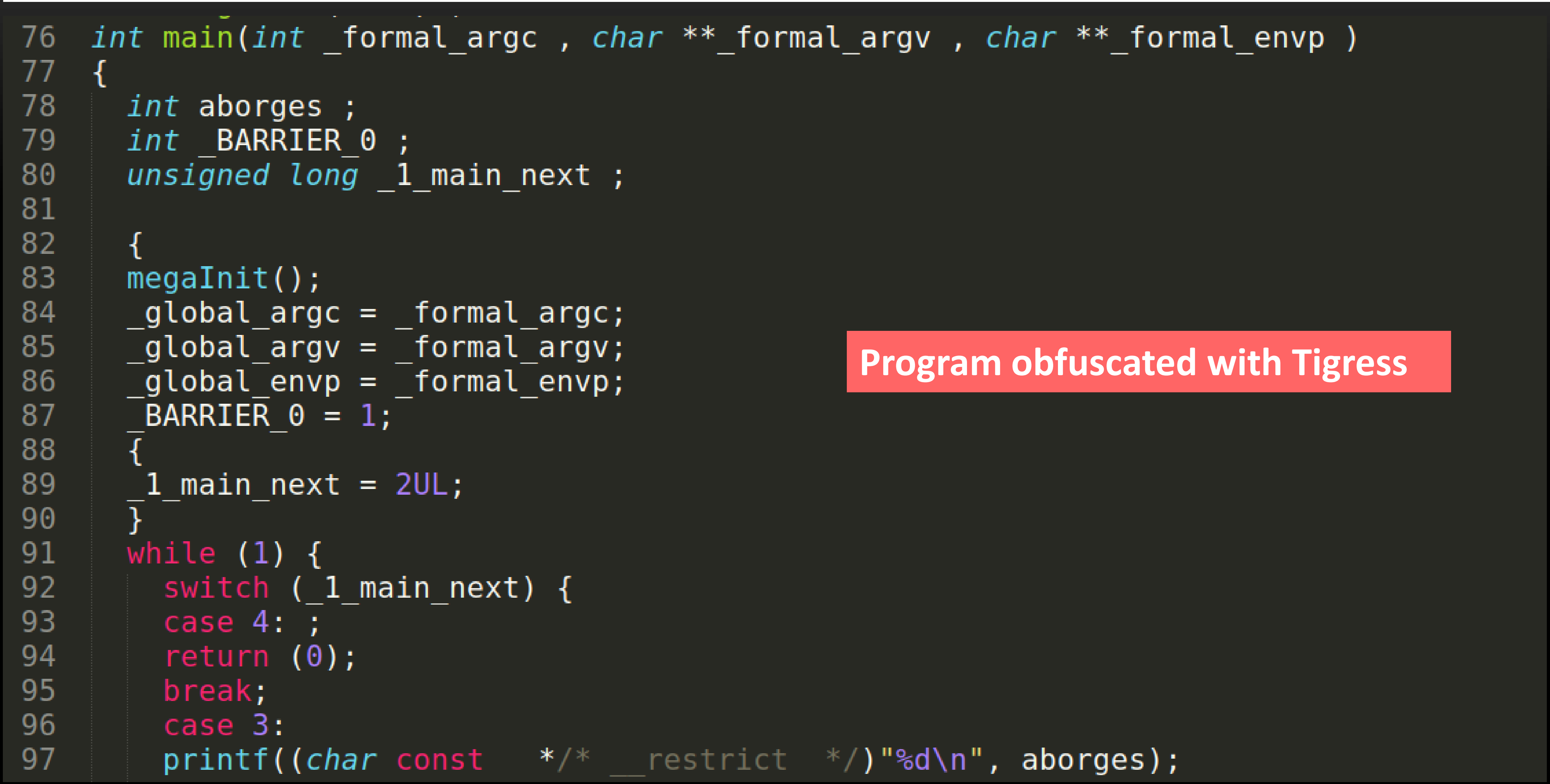
ANTI-REVERSING

- To transform a C source using Tigress:
  - tigress --Environment=x86_64:Linux:Gcc:4.6 --Transform=Flatten --Functions=main --out=aleborges_obfuscated.c aleborges_trigess.c
- There're many notes about the command above:
  - We should pick up one or more functions to be transformed. Of course, I've chosen the main( ) only for educational purposes.
  - The argument for Environment must be according to your environment (x86_64:Linux:Gcc:4.6, x86_64:Darwin:Clang:5.1, armv7:Linux:Gcc:4.6, armv8:Linux:Gcc:4.6)

ANTI-REVERSING

- Additional notes:
  - We could use multiple transformations (specifying the --Transform option and --Function option multiple times) such as:
    - Opaque Predicate (InitOpaque)
    - Virtualization Obfuscation (Virtualize)
    - Split and Merge
    - Encode Literals
    - Encode Branches
    - AntiTaintAnalysis
    - ...and much more...

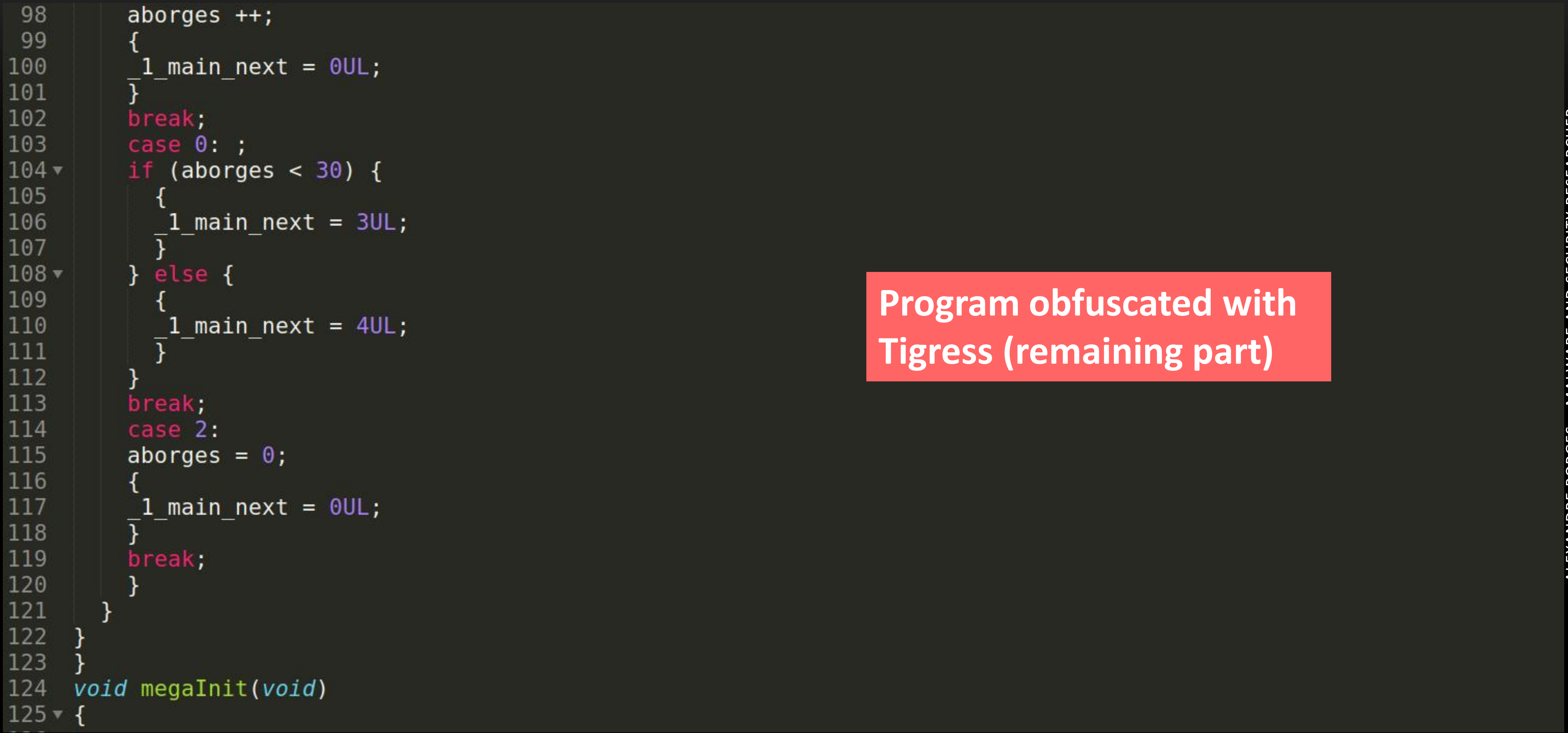
ANTI-REVERSING



```
int main(int _formal_argc , char **_formal_argv , char **_formal_envp )
{
  int aborges ;
  int _BARRIER_0 ;
  unsigned long _1_main_next ;

  {
  megaInit();
  _global_argc = _formal_argc;
  _global_argv = _formal_argv;
  _global_envp = _formal_envp;
  _BARRIER_0 = 1;
  {
  _1_main_next = 2UL;
  }
  while (1) {
    switch (_1_main_next) {
    case 4: ;
    return (0);
    break;
    case 3:
    printf((char const   */* __restrict  */)"%d\n", aborges);
```

Program obfuscated with Tigress

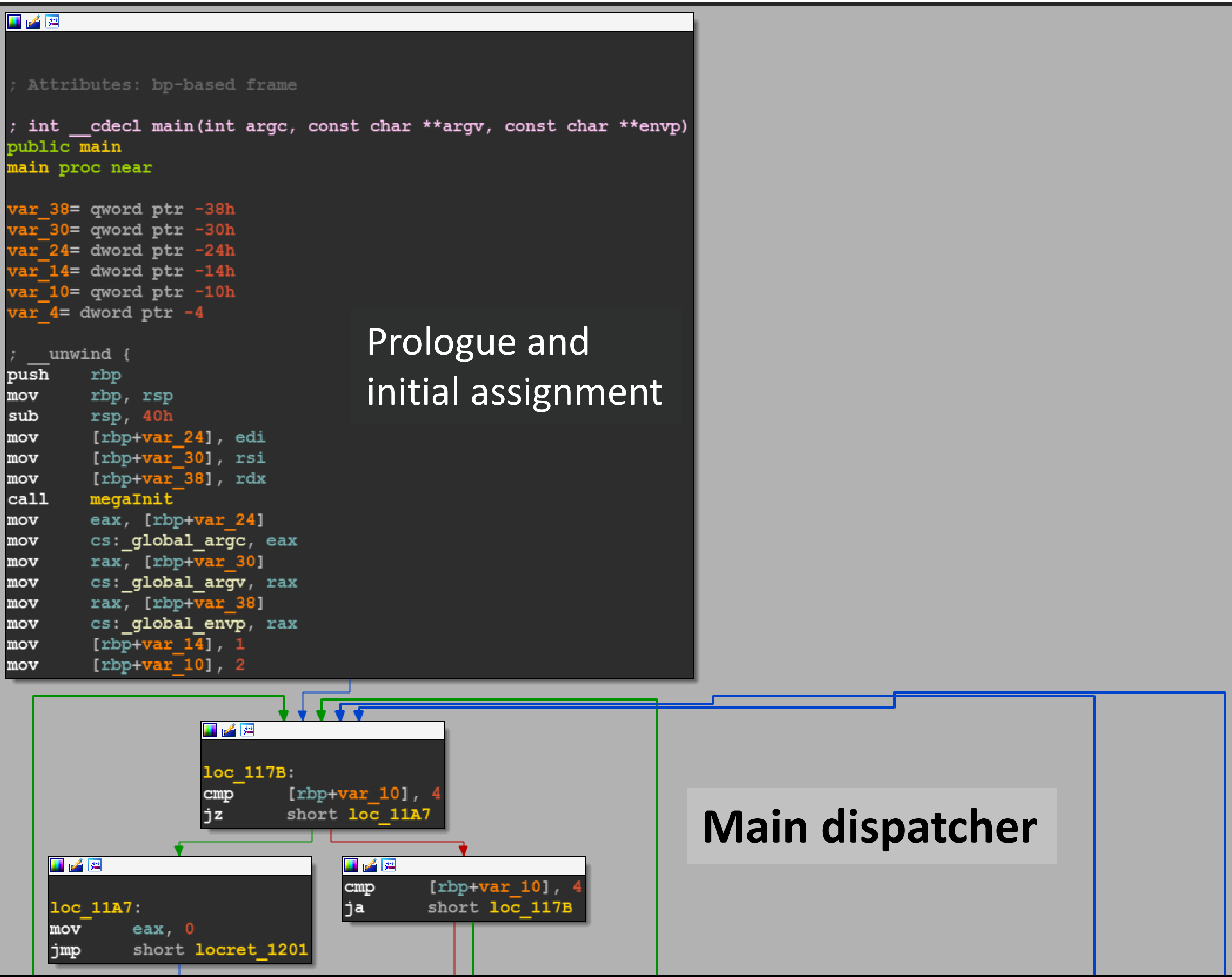
➢ ANTI-REVERSING



```
    aborges ++;
    {
    _1_main_next = 0UL;
    }
    break;
    case 0: ;
    if (aborges < 30) {
      {
      _1_main_next = 3UL;
      }
    } else {
      {
      _1_main_next = 4UL;
      }
    }
    break;
    case 2:
    aborges = 0;
    {
    _1_main_next = 0UL;
    }
    break;
    }
  }
}
}
void megaInit(void)
{
```

Program obfuscated with Tigress (remaining part)

➢ ANTI-REVERSING



```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_38= qword ptr -38h
var_30= qword ptr -30h
var_24= dword ptr -24h
var_14= dword ptr -14h
var_10= qword ptr -10h
var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 40h
mov     [rbp+var_24], edi
mov     [rbp+var_30], rsi
mov     [rbp+var_38], rdx
call    megaInit
mov     eax, [rbp+var_24]
mov     cs:_global_argc, eax
mov     rax, [rbp+var_30]
mov     cs:_global_argv, rax
mov     rax, [rbp+var_38]
mov     cs:_global_envp, rax
mov     [rbp+var_14], 1
mov     [rbp+var_10], 2
```

Prologue and initial assignment

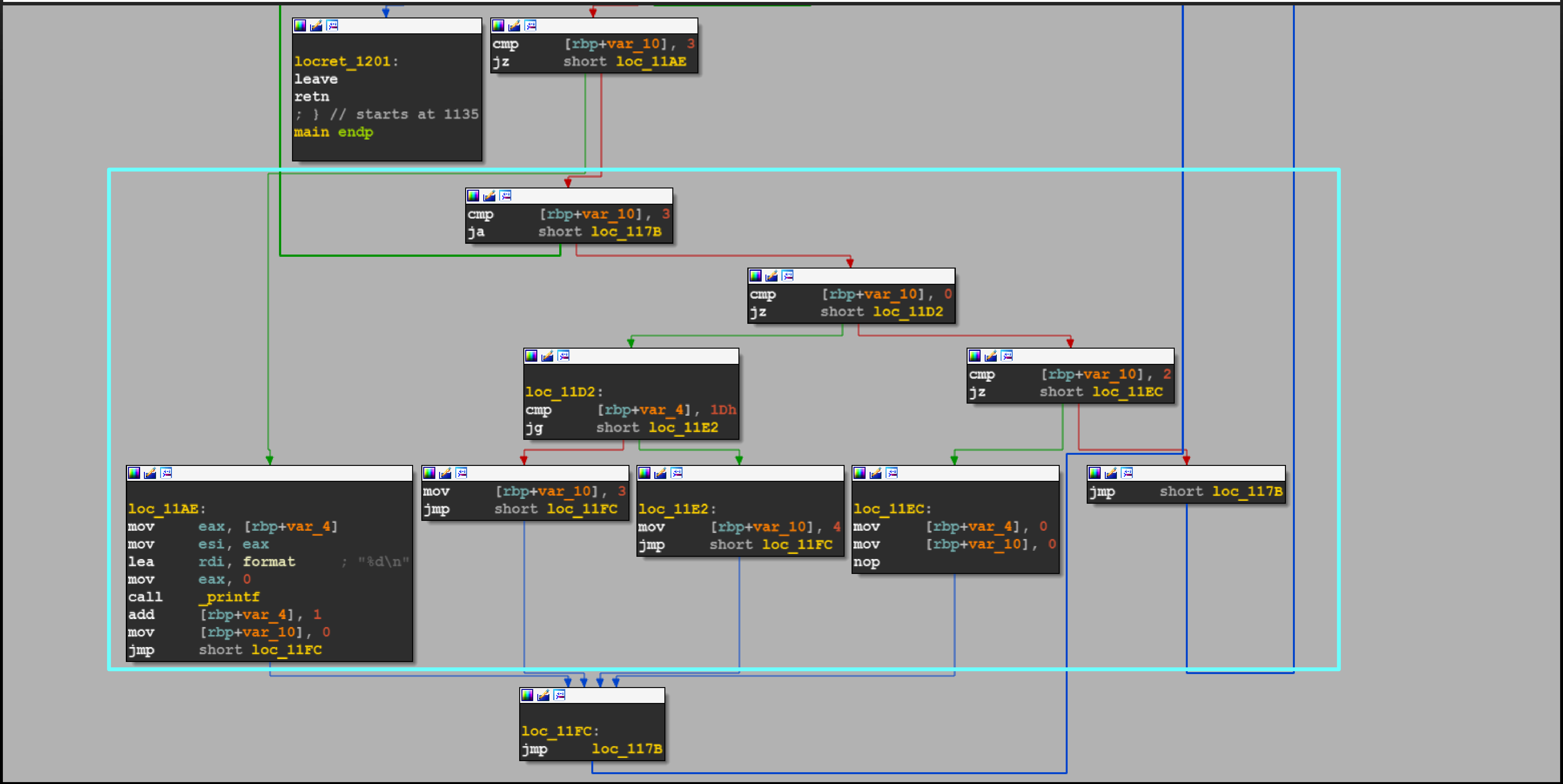
```
loc_117B:
cmp     [rbp+var_10], 4
jz      short loc_11A7
```

```
loc_11A7:
mov     eax, 0
jmp     short locret_1201
```

```
cmp     [rbp+var_10], 4
ja      short loc_117B
```

Main dispatcher

➢ ANTI-REVERSING



```
locret_1201:
leave
retn
; } // starts at 1135
main endp
```

```
cmp     [rbp+var_10], 3
jz      short loc_11AE
```

```
cmp     [rbp+var_10], 3
ja      short loc_117B
```

```
cmp     [rbp+var_10], 0
jz      short loc_11D2
```

```
loc_11D2:
cmp     [rbp+var_4], 1Dh
jg      short loc_11E2
```

```
cmp     [rbp+var_10], 2
jz      short loc_11EC
```

```
loc_11AE:
mov     eax, [rbp+var_4]
mov     esi, eax
lea     rdi, format     ; "%d\n"
mov     eax, 0
call    _printf
add     [rbp+var_4], 1
mov     [rbp+var_10], 0
jmp     short loc_11FC
```

```
mov     [rbp+var_10], 3
jmp     short loc_11FC
```

```
loc_11E2:
mov     [rbp+var_10], 4
jmp     short loc_11FC
```

```
loc_11EC:
mov     [rbp+var_4], 0
mov     [rbp+var_10], 0
nop
```

```
jmp     short loc_117B
```

```
loc_11FC:
jmp     loc_117B
```

➢ ANTI-REVERSING

Simple opaque predicate and anti-disassembly technique

```
.text:00401000 loc_401000:                       ; CODE XREF: _main+Fp

.text:00401000              push    ebp
.text:00401001              mov     ebp, esp
.text:00401003              xor     eax, eax
.text:00401005              jz      short near ptr loc_40100D+1
.text:00401007              jnz     near ptr loc_40100D+4

.text:0040100D
.text:0040100D loc_40100D:                   ; CODE XREF: .text:00401005j
.text:0040100D                              ; .text:00401007j

.text:0040100D              jmp    near ptr 0D0A8837h
```

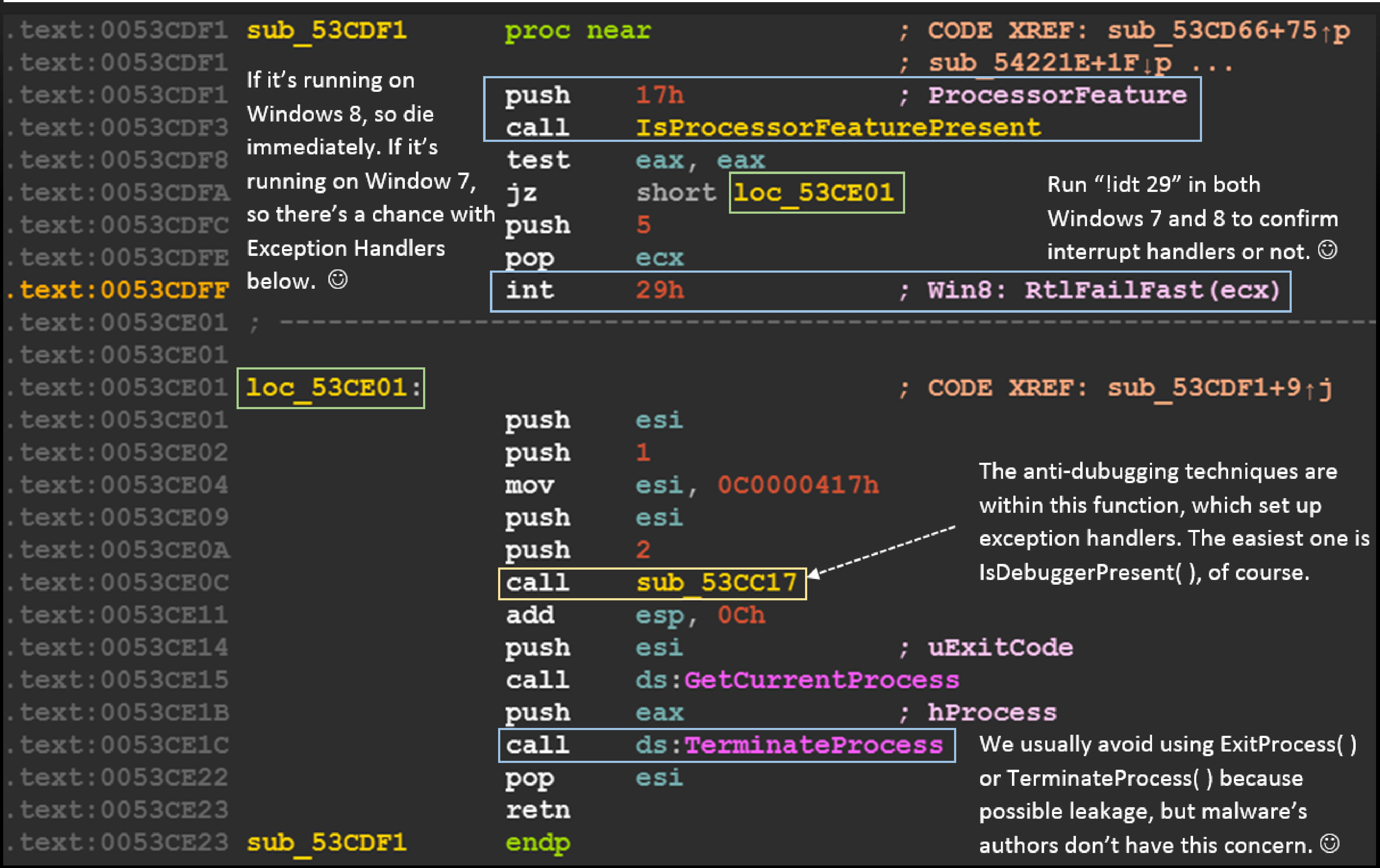
ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

ANTI-REVERSING

```
00401040        call + $5
00401045        pop ecx
00401046        inc ecx
00401047        inc ecx
00401048        add ecx, 4
00401049        add ecx, 4
0040104A        push ecx
0040104B        ret
0040104C        sub ecx, 6
0040104D        dec ecx
0040104E        dec ecx
0040104F        jmp 0x401320
```

- Call stack manipulation:
  - Do you know what’s happening here? ☺

ANTI-REVERSING



```
.text:0053CDF1 sub_53CDF1      proc near               ; CODE XREF: sub_53CD66+75↑p
.text:0053CDF1                                         ; sub_54221E+1F↓p ...
.text:0053CDF1                 push    17h             ; ProcessorFeature
.text:0053CDF3                 call    IsProcessorFeaturePresent
.text:0053CDF8                 test    eax, eax
.text:0053CDFA                 jz      short loc_53CE01
.text:0053CDFC                 push    5
.text:0053CDFE                 pop     ecx
.text:0053CDFF                 int     29h             ; Win8: RtlFailFast(ecx)
.text:0053CE01 ; ---------------------------------------------------------------
.text:0053CE01
.text:0053CE01 loc_53CE01:                             ; CODE XREF: sub_53CDF1+9↑j
.text:0053CE01                 push    esi
.text:0053CE02                 push    1
.text:0053CE04                 mov     esi, 0C0000417h
.text:0053CE09                 push    esi
.text:0053CE0A                 push    2
.text:0053CE0C                 call    sub_53CC17
.text:0053CE11                 add     esp, 0Ch
.text:0053CE14                 push    esi             ; uExitCode
.text:0053CE15                 call    ds:GetCurrentProcess
.text:0053CE1B                 push    eax             ; hProcess
.text:0053CE1C                 call    ds:TerminateProcess
.text:0053CE22                 pop     esi
.text:0053CE23                 retn
.text:0053CE23 sub_53CDF1      endp
```

If it's running on Windows 8, so die immediately. If it's running on Window 7, so there's a chance with Exception Handlers below. ☺

Run "!idt 29" in both Windows 7 and 8 to confirm interrupt handlers or not. ☺

The anti-dubugging techniques are within this function, which set up exception handlers. The easiest one is IsDebuggerPresent( ), of course.

We usually avoid using ExitProcess( ) or TerminateProcess( ) because possible leakage, but malware's authors don't have this concern. ☺

- Few lines of code could have several anti-debugging tricks. Maybe some non-recommended functions too ☺

➢ ANTI-REVERSING

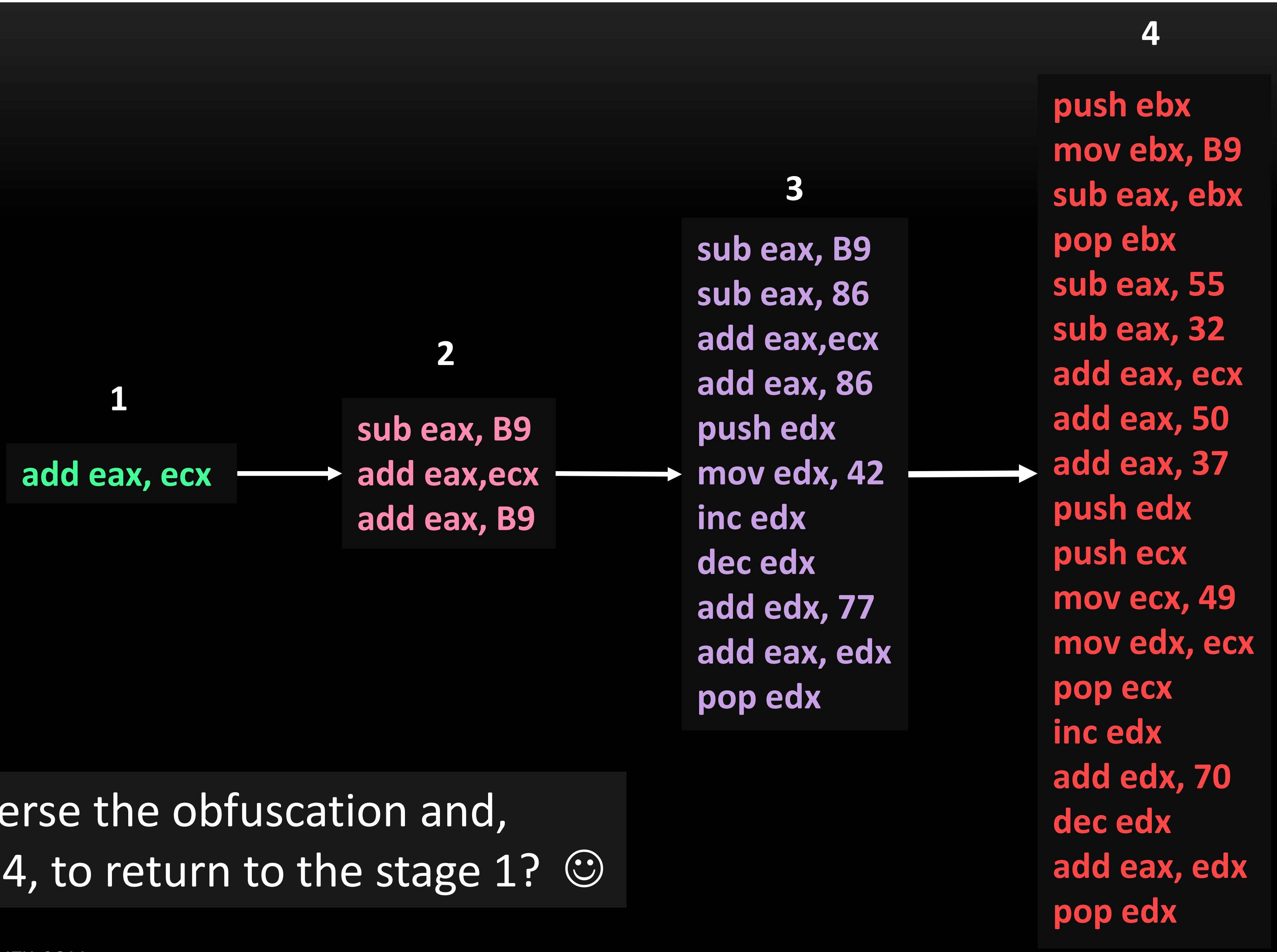
- Are there only these anti-forensic techniques? No.... ☺
  - Instruction substitution
  - Garbage insertion
  - Self-modifying code (like shellcode)
  - Method renaming
  - Cryptography
  - Variable reordering
  - Instruction reordering
  - Randomized Stack frame
  - Inherance manipulation
  - And more...much more ☺

DEOBFUSCATION

METASM

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

DEOBFUSCATION



1

```
add eax, ecx
```

2

```
sub eax, B9
add eax,ecx
add eax, B9
```

3

```
sub eax, B9
sub eax, 86
add eax,ecx
add eax, 86
push edx
mov edx, 42
inc edx
dec edx
add edx, 77
add eax, edx
pop edx
```

4

```
push ebx
mov ebx, B9
sub eax, ebx
pop ebx
sub eax, 55
sub eax, 32
add eax, ecx
add eax, 50
add eax, 37
push edx
push ecx
mov ecx, 49
mov edx, ecx
pop ecx
inc edx
add edx, 70
dec edx
add eax, edx
pop edx
```

- How to reverse the obfuscation and, from stage 4, to return to the stage 1? ☺

DEOBFUSCATION

- METASM works as disassembler, assembler, debugger, compiler and linker.

- Key features:

  - Written in Ruby
  - C compiler and decompiler
  - Automatic backtracking
  - Live process manipulation
  - Supports the following architecture:

    - Intel IA32 (16/32/64 bits)
    - PPC
    - MIPS

➢ DEOBFUSCATION

- root@kali:~/github# git clone https://github.com/jjyg/metasm.git

- Include the following line into .bashrc file to indicate the Metasm directory installation:

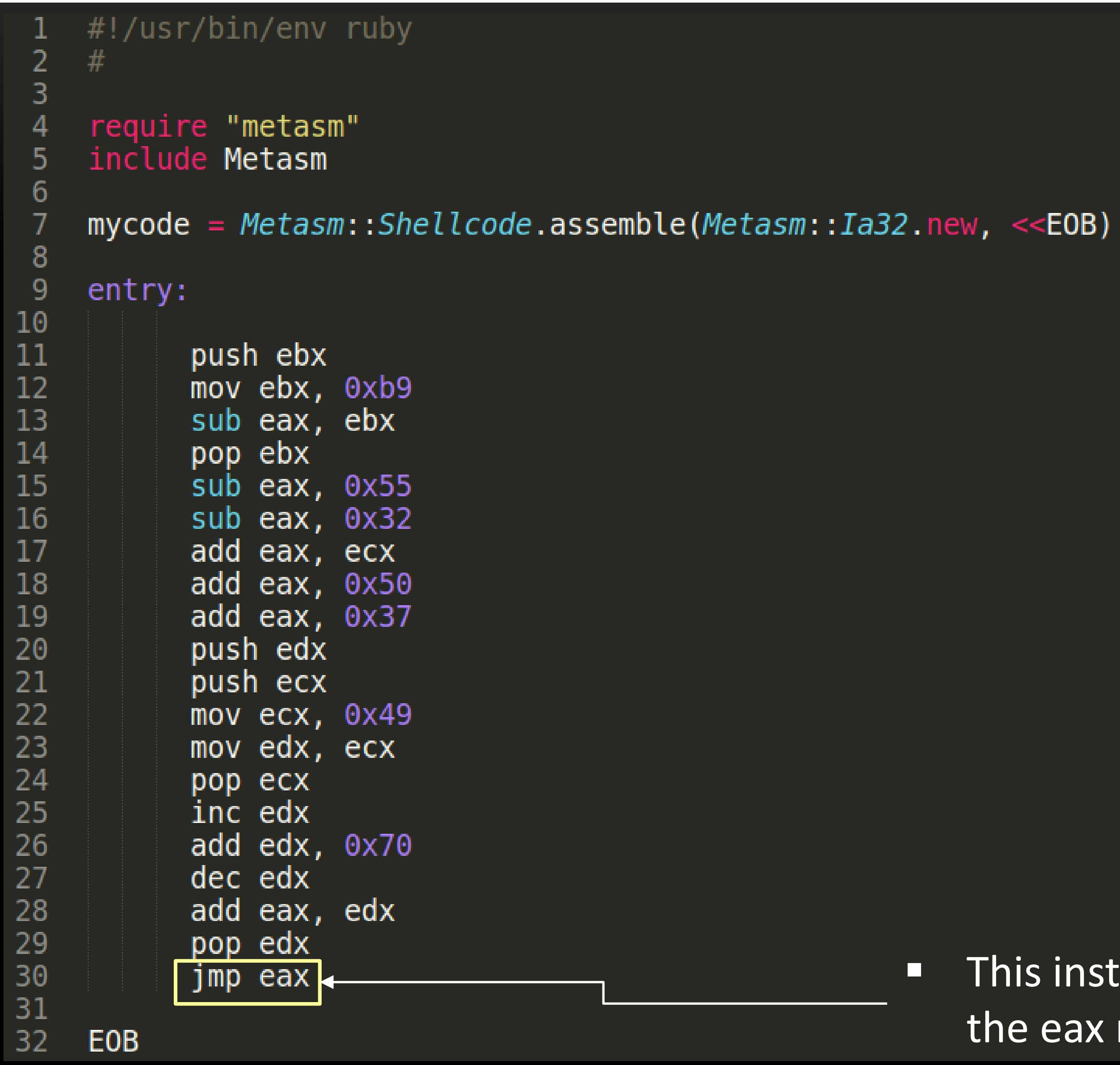
  - export RUBYLIB=$RUBYLIB:~/github/metasm

- Test metasm:

  - ruby -r metasm -e 'p Metasm::VERSION‘

- You should see a number in the output. It’s done ☺

DEOBFUSCATION

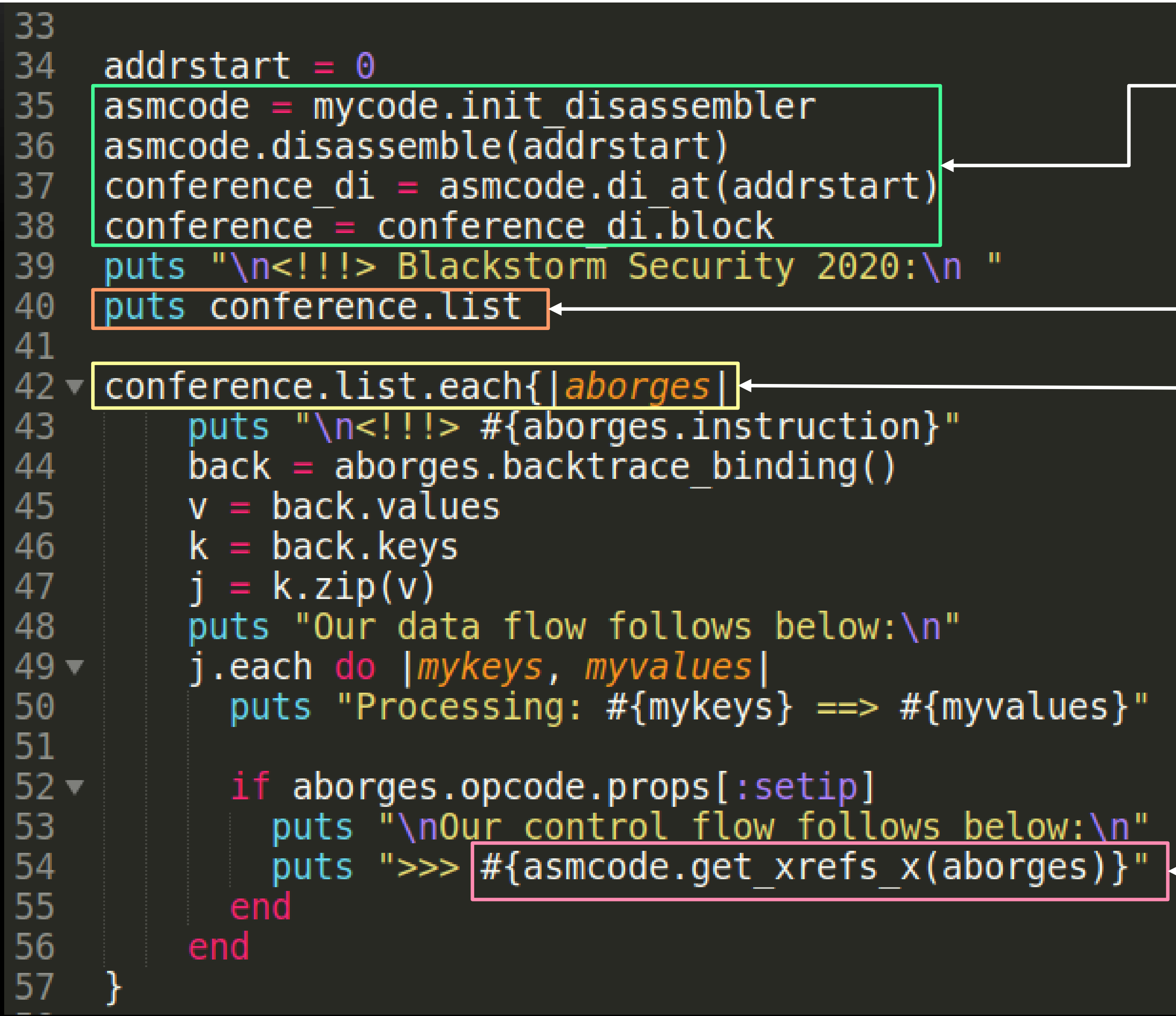


```
#!/usr/bin/env ruby
#

require "metasm"
include Metasm

mycode = Metasm::Shellcode.assemble(Metasm::Ia32.new, <<EOB)

entry:

        push ebx
        mov ebx, 0xb9
        sub eax, ebx
        pop ebx
        sub eax, 0x55
        sub eax, 0x32
        add eax, ecx
        add eax, 0x50
        add eax, 0x37
        push edx
        push ecx
        mov ecx, 0x49
        mov edx, ecx
        pop ecx
        inc edx
        add edx, 0x70
        dec edx
        add eax, edx
        pop edx
        jmp eax

EOB
```

- Based on metasm.rb file and Bruce Dang’s code.

- This instruction was inserted to make the eax register evaluation easier. ☺

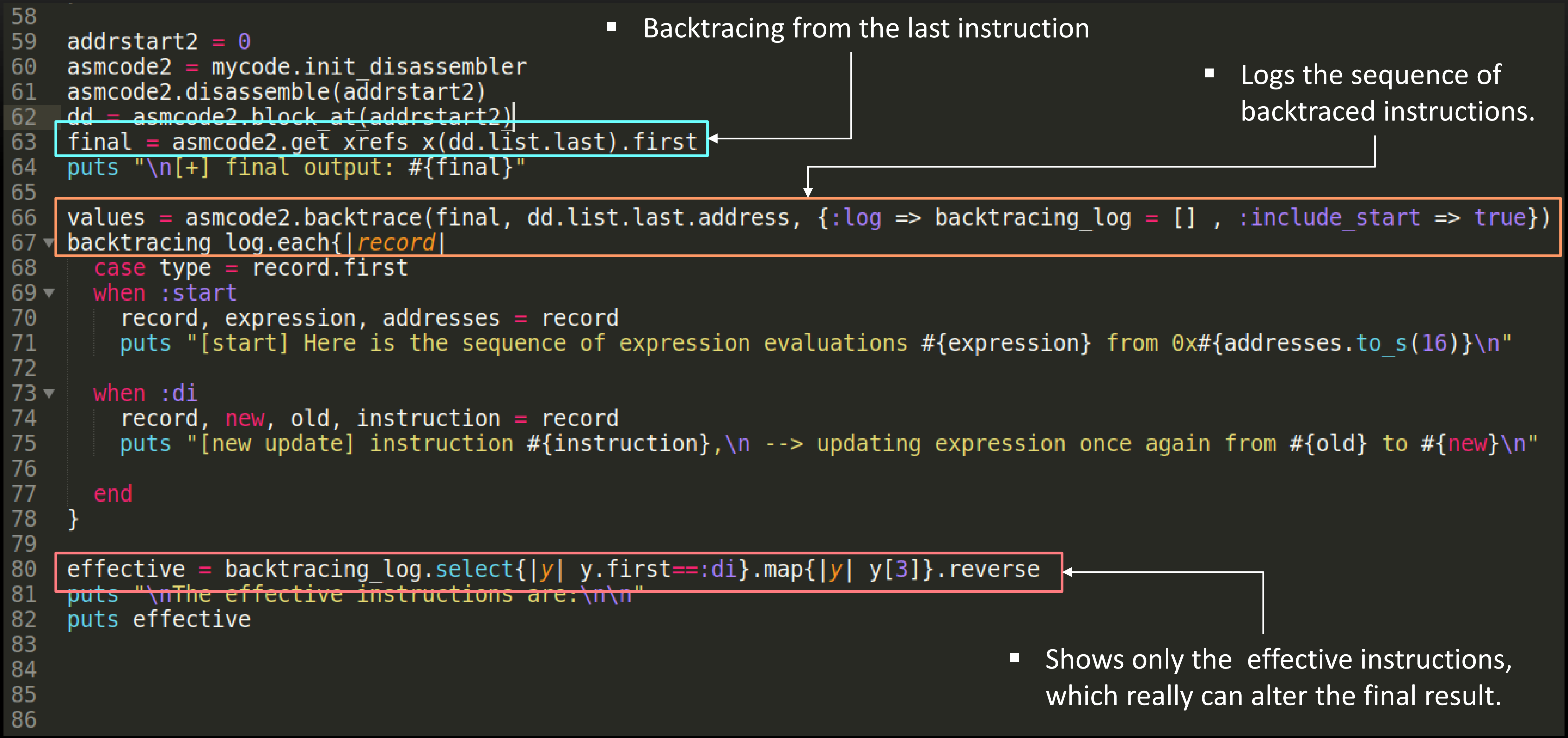
DEOBFUSCATION



```
addrstart = 0
asmcode = mycode.init_disassembler
asmcode.disassemble(addrstart)
conference_di = asmcode.di_at(addrstart)
conference = conference_di.block
puts "\n<!!!> Blackstorm Security 2020:\n "
puts conference.list

conference.list.each{|aborges|
    puts "\n<!!!> #{aborges.instruction}"
    back = aborges.backtrace_binding()
    v = back.values
    k = back.keys
    j = k.zip(v)
    puts "Our data flow follows below:\n"
    j.each do |mykeys, myvalues|
      puts "Processing: #{mykeys} ==> #{myvalues}"

      if aborges.opcode.props[:setip]
        puts "\nOur control flow follows below:\n"
        puts ">>> #{asmcode.get_xrefs_x(aborges)}"
      end
    end
}
```

- initialize and disassemble code since beginning (start).
- list the assemble code
- initialize the backtracking engine
- determines which is the final instruction to walk back from there

➢ DEOBFUSCATION



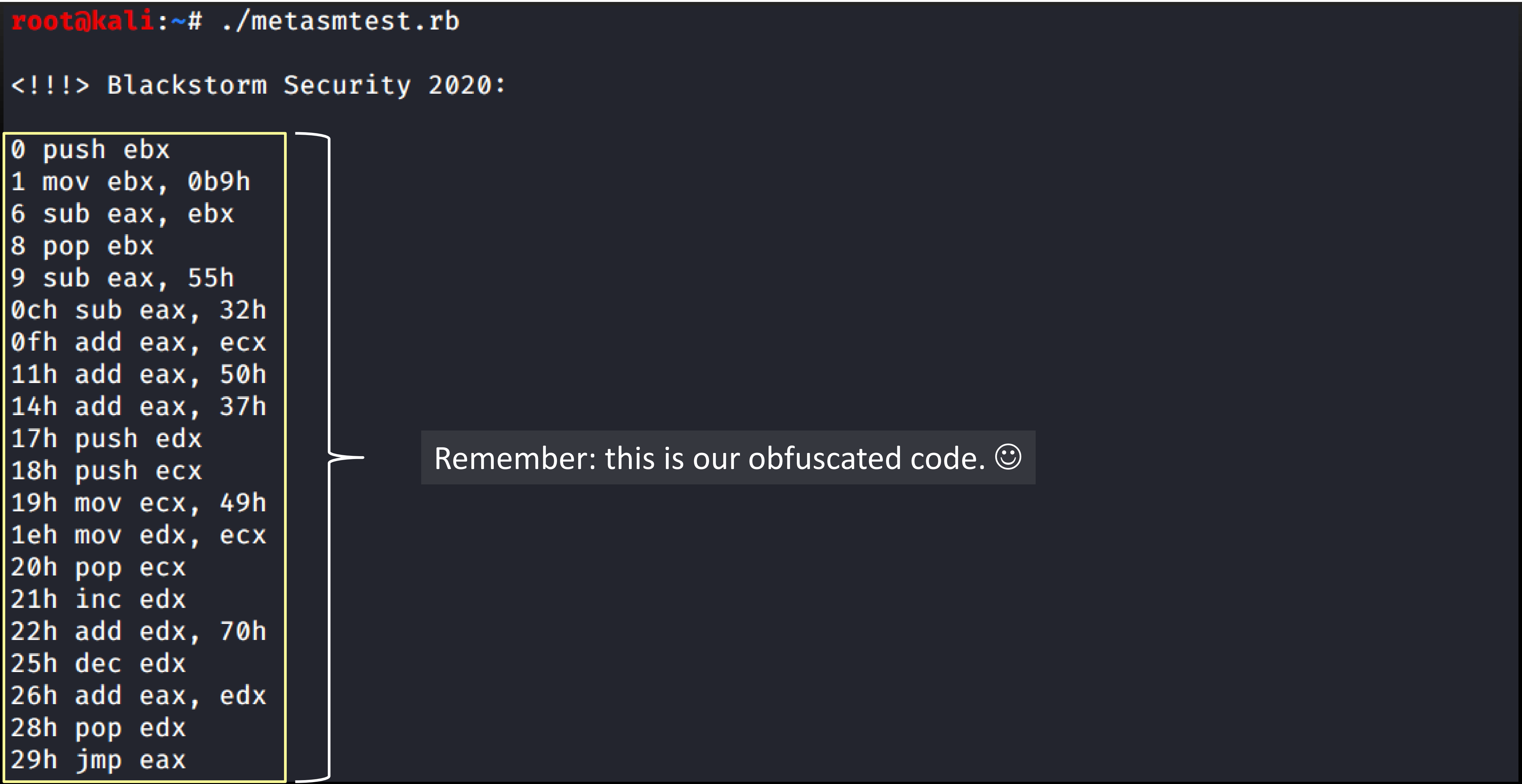
```ruby

addrstart2 = 0
asmcode2 = mycode.init_disassembler
asmcode2.disassemble(addrstart2)
dd = asmcode2.block_at(addrstart2)
final = asmcode2.get_xrefs_x(dd.list.last).first
puts "\n[+] final output: #{final}"

values = asmcode2.backtrace(final, dd.list.last.address, {:log => backtracing_log = [] , :include_start => true})
backtracing_log.each{|record|
  case type = record.first
  when :start
    record, expression, addresses = record
    puts "[start] Here is the sequence of expression evaluations #{expression} from 0x#{addresses.to_s(16)}\n"

  when :di
    record, new, old, instruction = record
    puts "[new update] instruction #{instruction},\n --> updating expression once again from #{old} to #{new}\n"

  end
}

effective = backtracing_log.select{|y| y.first==:di}.map{|y| y[3]}.reverse
puts "\nThe effective instructions are:\n\n"
puts effective
```

- Backtracing from the last instruction

- Logs the sequence of backtraced instructions.

- Shows only the effective instructions, which really can alter the final result.

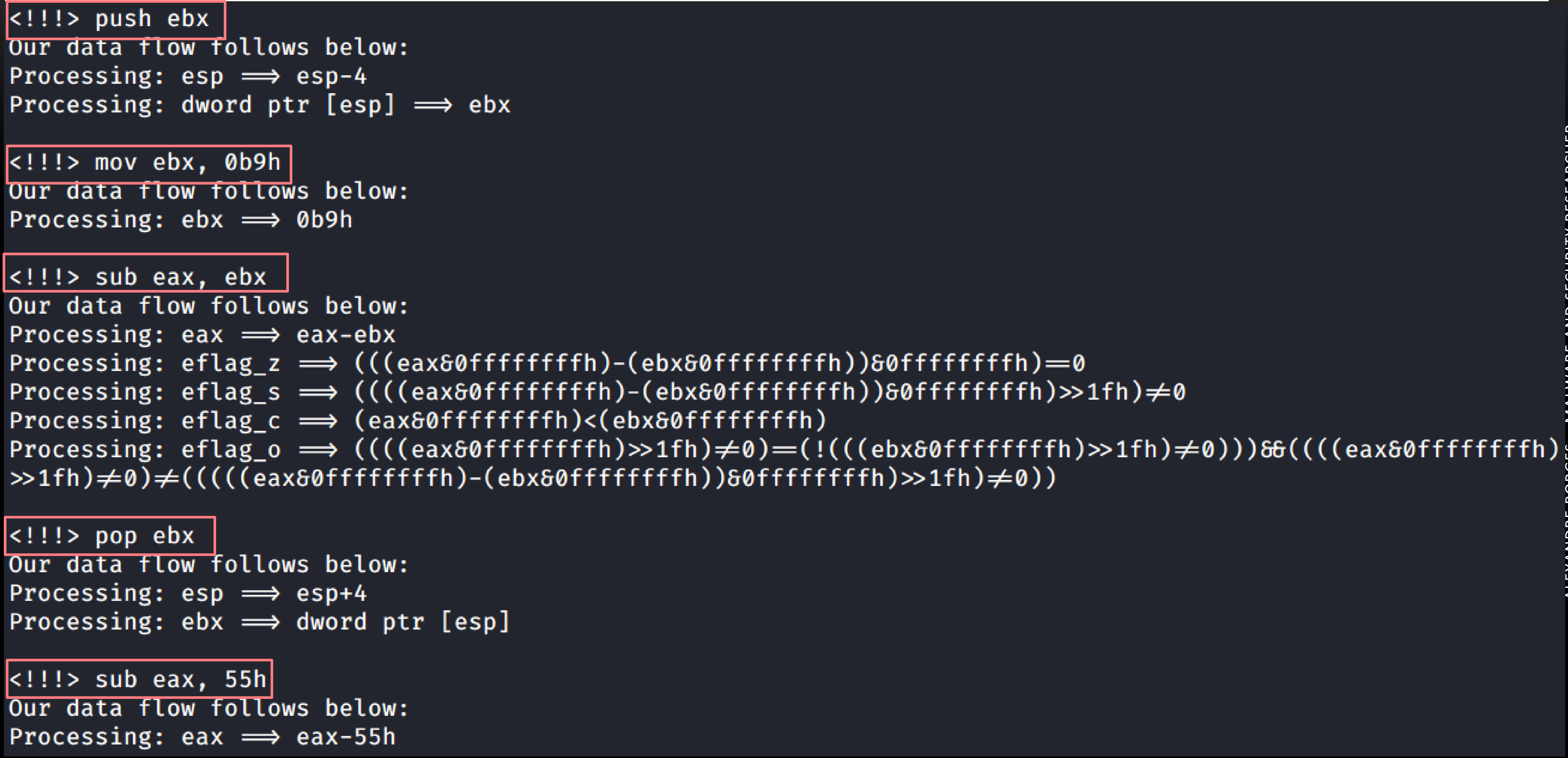
DEOBFUSCATION



```
root@kali:~# ./metasmtest.rb

<!!!> Blackstorm Security 2020:

0 push ebx
1 mov ebx, 0b9h
6 sub eax, ebx
8 pop ebx
9 sub eax, 55h
0ch sub eax, 32h
0fh add eax, ecx
11h add eax, 50h
14h add eax, 37h
17h push edx
18h push ecx
19h mov ecx, 49h
1eh mov edx, ecx
20h pop ecx
21h inc edx
22h add edx, 70h
25h dec edx
26h add eax, edx
28h pop edx
29h jmp eax
```

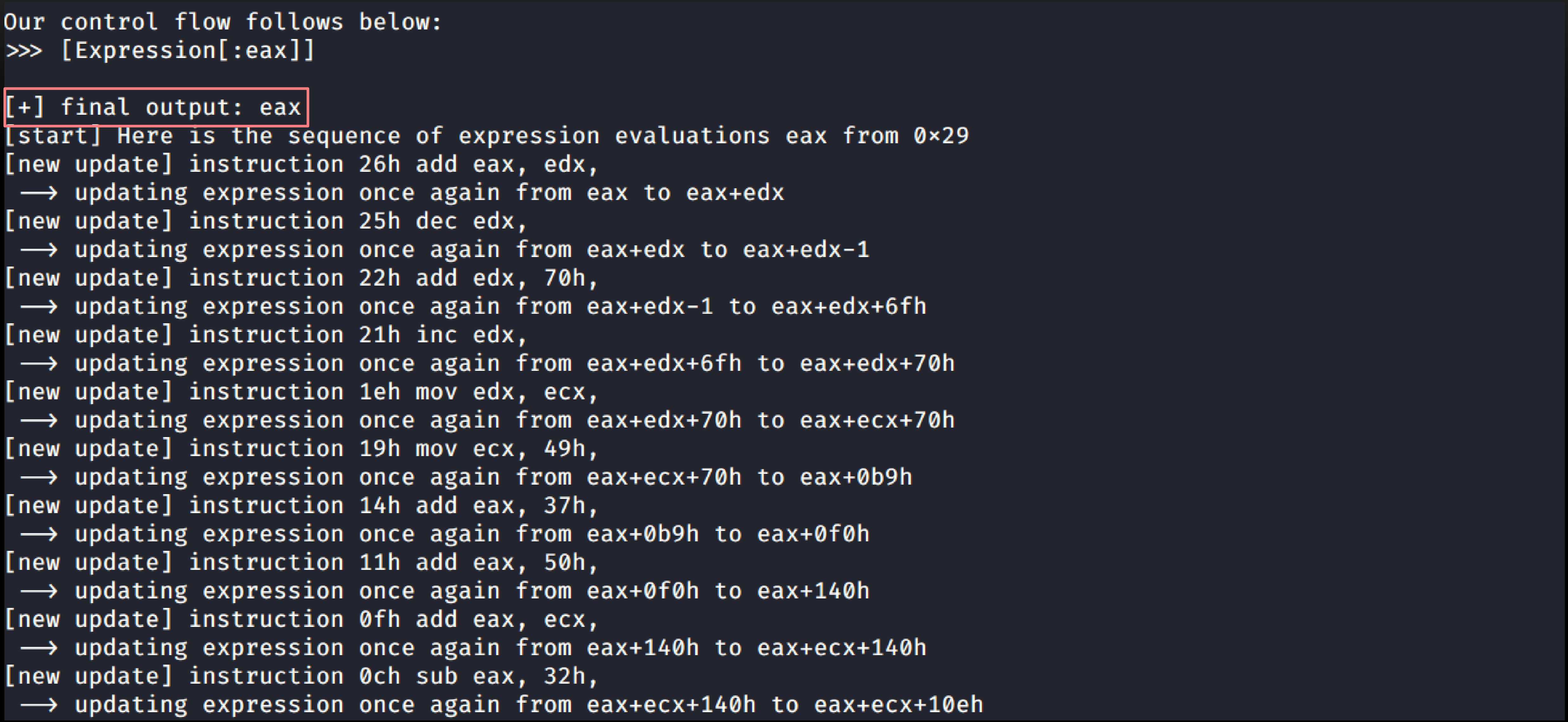
Remember: this is our obfuscated code. ☺

➢ DEOBFUSCATION



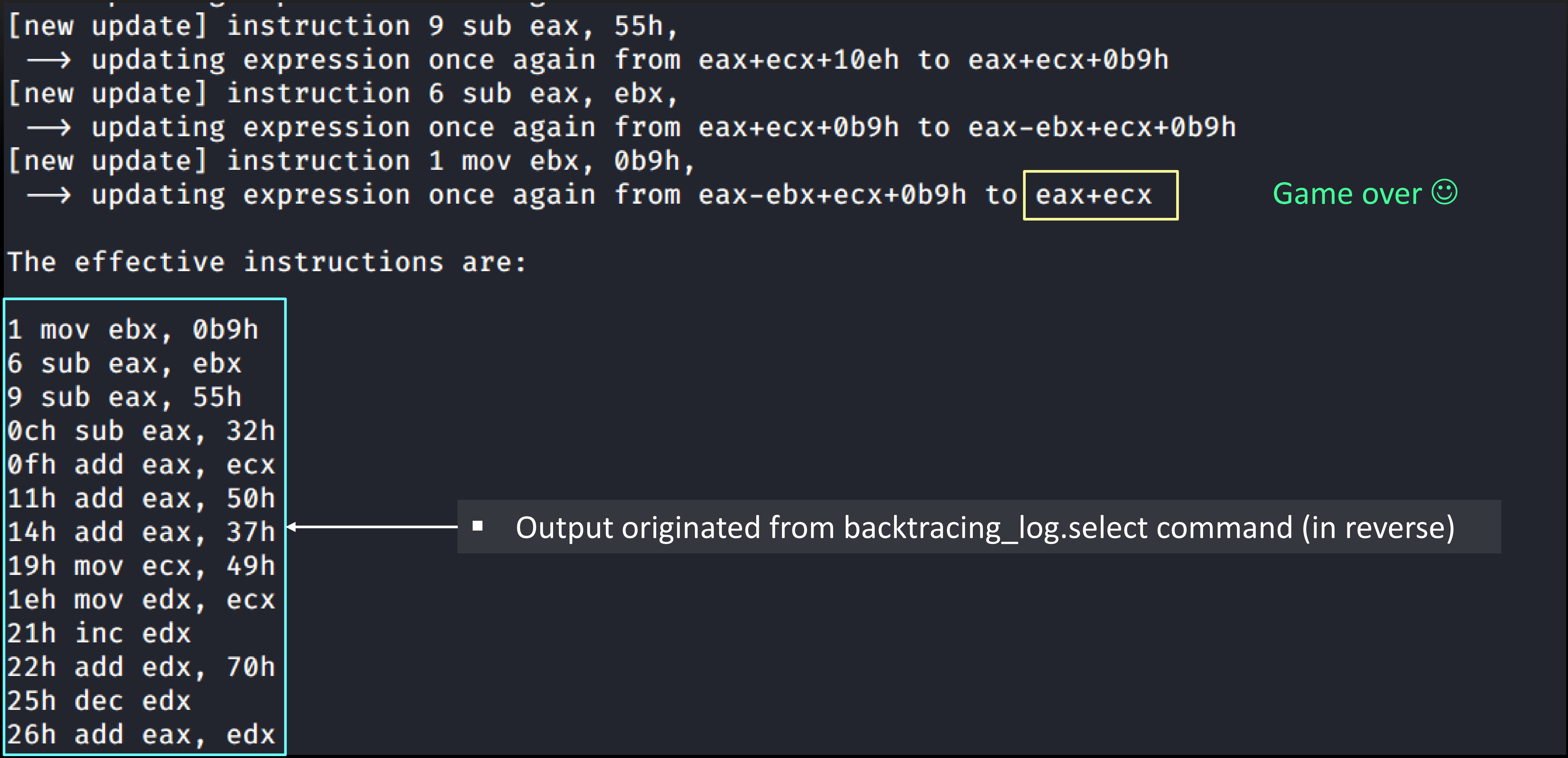
```
<!!!> push ebx
Our data flow follows below:
Processing: esp ⟹ esp-4
Processing: dword ptr [esp] ⟹ ebx

<!!!> mov ebx, 0b9h
Our data flow follows below:
Processing: ebx ⟹ 0b9h

<!!!> sub eax, ebx
Our data flow follows below:
Processing: eax ⟹ eax-ebx
Processing: eflag_z ⟹ (((eax&0ffffffffh)-(ebx&0ffffffffh))&0ffffffffh)==0
Processing: eflag_s ⟹ ((((eax&0ffffffffh)-(ebx&0ffffffffh))&0ffffffffh)>>1fh)≠0
Processing: eflag_c ⟹ (eax&0ffffffffh)<(ebx&0ffffffffh)
Processing: eflag_o ⟹ ((((eax&0ffffffffh)>>1fh)≠0)==(!(((ebx&0ffffffffh)>>1fh)≠0)))&&((((eax&0ffffffffh)
>>1fh)≠0)≠(((((eax&0ffffffffh)-(ebx&0ffffffffh))&0ffffffffh)>>1fh)≠0))

<!!!> pop ebx
Our data flow follows below:
Processing: esp ⟹ esp+4
Processing: ebx ⟹ dword ptr [esp]

<!!!> sub eax, 55h
Our data flow follows below:
Processing: eax ⟹ eax-55h
```

➢ DEOBFUSCATION



```
Our control flow follows below:
>>> [Expression[:eax]]

[+] final output: eax
[start] Here is the sequence of expression evaluations eax from 0×29
[new update] instruction 26h add eax, edx,
 ⟶ updating expression once again from eax to eax+edx
[new update] instruction 25h dec edx,
 ⟶ updating expression once again from eax+edx to eax+edx-1
[new update] instruction 22h add edx, 70h,
 ⟶ updating expression once again from eax+edx-1 to eax+edx+6fh
[new update] instruction 21h inc edx,
 ⟶ updating expression once again from eax+edx+6fh to eax+edx+70h
[new update] instruction 1eh mov edx, ecx,
 ⟶ updating expression once again from eax+edx+70h to eax+ecx+70h
[new update] instruction 19h mov ecx, 49h,
 ⟶ updating expression once again from eax+ecx+70h to eax+0b9h
[new update] instruction 14h add eax, 37h,
 ⟶ updating expression once again from eax+0b9h to eax+0f0h
[new update] instruction 11h add eax, 50h,
 ⟶ updating expression once again from eax+0f0h to eax+140h
[new update] instruction 0fh add eax, ecx,
 ⟶ updating expression once again from eax+140h to eax+ecx+140h
[new update] instruction 0ch sub eax, 32h,
 ⟶ updating expression once again from eax+ecx+140h to eax+ecx+10eh
```

DEOBFUSCATION



```
[new update] instruction 9 sub eax, 55h,
 ⟶ updating expression once again from eax+ecx+10eh to eax+ecx+0b9h
[new update] instruction 6 sub eax, ebx,
 ⟶ updating expression once again from eax+ecx+0b9h to eax-ebx+ecx+0b9h
[new update] instruction 1 mov ebx, 0b9h,
 ⟶ updating expression once again from eax-ebx+ecx+0b9h to eax+ecx

The effective instructions are:

1 mov ebx, 0b9h
6 sub eax, ebx
9 sub eax, 55h
0ch sub eax, 32h
0fh add eax, ecx
11h add eax, 50h
14h add eax, 37h
19h mov ecx, 49h
1eh mov edx, ecx
21h inc edx
22h add edx, 70h
25h dec edx
26h add eax, edx
```

Game over ☺

- Output originated from backtracing_log.select command (in reverse)

MIASM

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

DEOBFUSCATION

- MIASM is one of most impressive framework for reverse engineering, which is able to analyze, generate and modify several different types of programs.

- MIASM supports assembling and disassembling programs from different platforms such as ARM, x86, MIPS and so on, and it also is able to emulate by using JIT.

- Therefore, MIASM is excellent to de-obfuscation.

- Installing MIASM (python 2.7.x):

  - git clone https://github.com/serpilliere/elfesteem.git elfesteem
  - cd elfesteem/

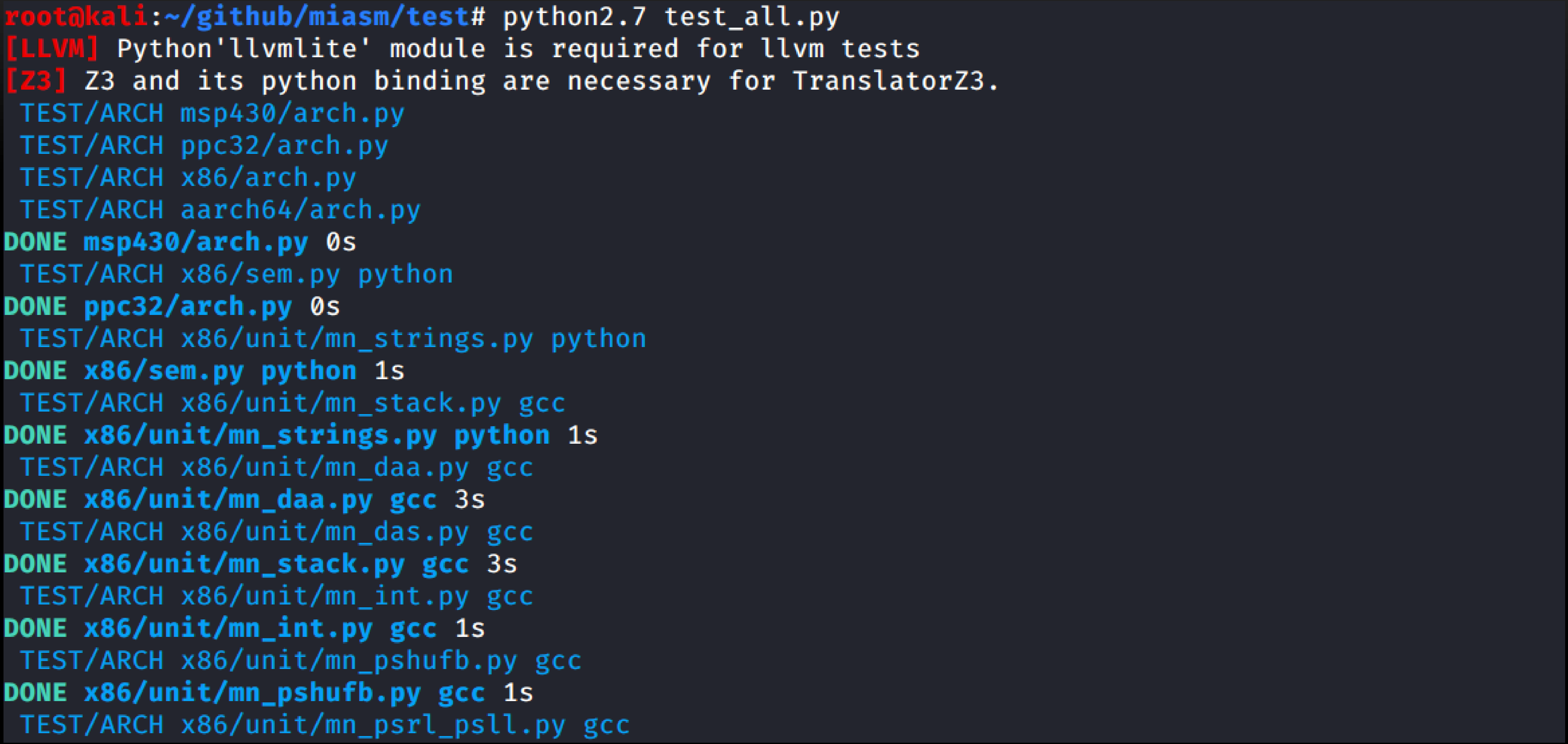
➢ DEOBFUSCATION

- python setup.py build
- python setup.py install
- apt-get install clang texinfo texi2html
- apt-get remove libtcc-dev
- apt-get install llvm
- cd ..
- git clone http://repo.or.cz/tinycc.git
- cd tinycc/
- git checkout release_0_9_26
- ./configure --disable-static
- make
- make install

DEOBFUSCATION

- pip install llvmlite
- pip install future
- apt-get install z3
- apt-get install python-pycparser
- pip install pyparsing
- cd ..
- git clone https://github.com/cea-sec/miasm.git
- cd miasm
- python setup.py build
- python setup.py install
- cd test/
- python test_all.py

➢ DEOBFUSCATION



```
root@kali:~/github/miasm/test# python2.7 test_all.py
[LLVM] Python'llvmlite' module is required for llvm tests
[Z3] Z3 and its python binding are necessary for TranslatorZ3.
 TEST/ARCH msp430/arch.py
 TEST/ARCH ppc32/arch.py
 TEST/ARCH x86/arch.py
 TEST/ARCH aarch64/arch.py
DONE msp430/arch.py 0s
 TEST/ARCH x86/sem.py python
DONE ppc32/arch.py 0s
 TEST/ARCH x86/unit/mn_strings.py python
DONE x86/sem.py python 1s
 TEST/ARCH x86/unit/mn_stack.py gcc
DONE x86/unit/mn_strings.py python 1s
 TEST/ARCH x86/unit/mn_daa.py gcc
DONE x86/unit/mn_daa.py gcc 3s
 TEST/ARCH x86/unit/mn_das.py gcc
DONE x86/unit/mn_stack.py gcc 3s
 TEST/ARCH x86/unit/mn_int.py gcc
DONE x86/unit/mn_int.py gcc 1s
 TEST/ARCH x86/unit/mn_pshufb.py gcc
DONE x86/unit/mn_pshufb.py gcc 1s
 TEST/ARCH x86/unit/mn_psrl_psll.py gcc
```

DEOBFUSCATION

- Before proceding with MIASM, we need to create a binary containing our code, so we need an assembler and Keystone is great.

- Keystone Engine acts an assembler and:

  - Supports x86, Mips, Arm and many other architectures.
  - It is implemented in C/C++ and has bindings to Python, Ruby, Powershell and C# (among other languages).

- Installing Keystone:

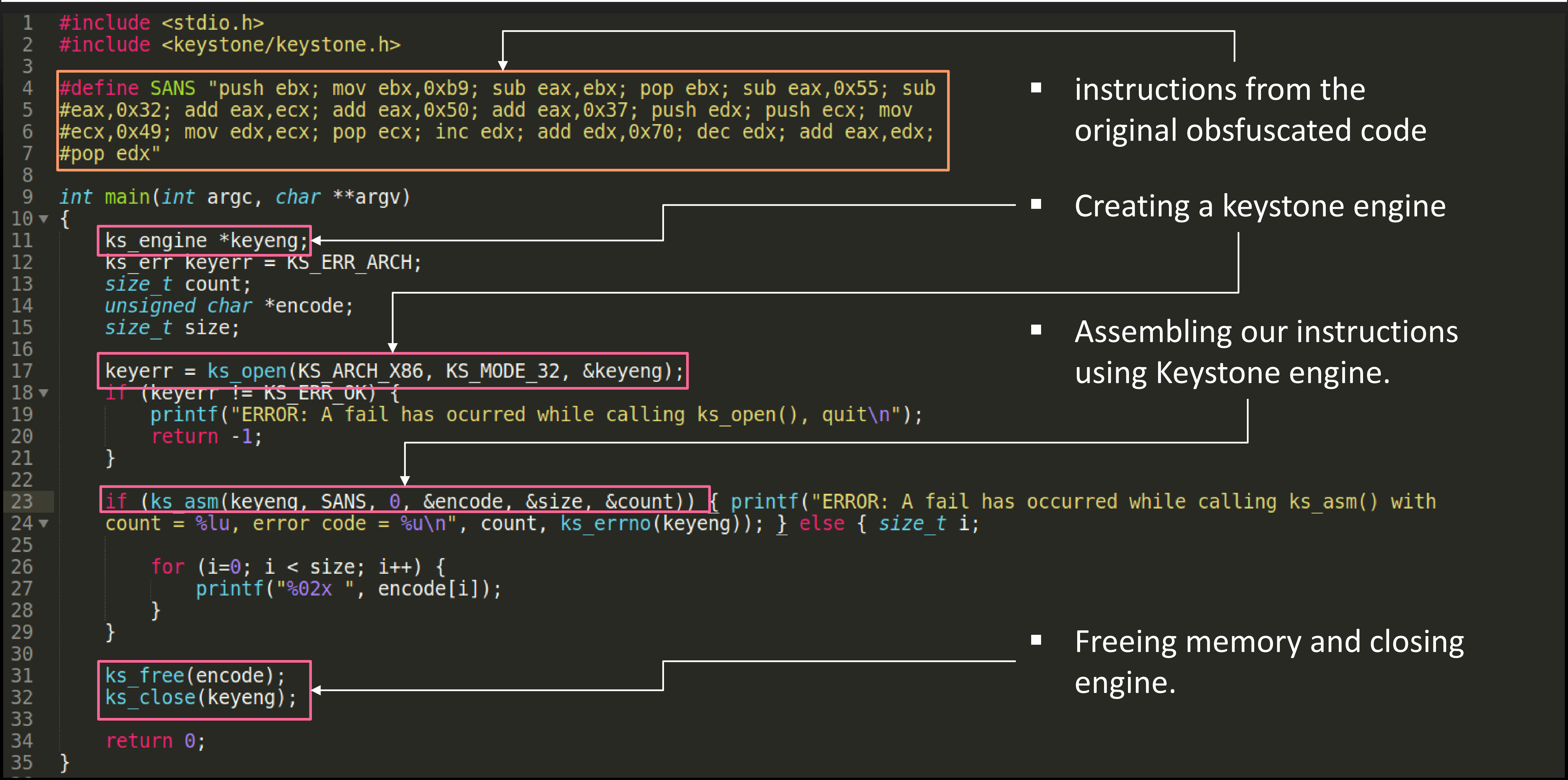
  - root@kali:~/Desktop# wget https://github.com/keystone-engine/keystone/archive/0.9.1.tar.gz

DEOBFUSCATION

```
root@kali:~/programs# cp /root/Desktop/keystone-0.9.1.tar.gz .
root@kali:~/programs# tar -zxvf keystone-0.9.1.tar.gz
 root@kali:~/programs/keystone-0.9.1# apt-get install cmake
root@kali:~/programs/keystone-0.9.1# mkdir build ; cd build
root@kali:~/programs/keystone-0.9.1/build# apt-get install time
root@kali:~/programs/keystone-0.9.1/build# ../make-share.sh
root@kali:~/programs/keystone-0.9.1/build# make install
root@kali:~/programs/keystone-0.9.1/build# ldconfig
root@kali:~/programs/keystone-0.9.1/build# tail -2 /root/.bashrc
```

```
export RUBYLIB=$RUBYLIB:~/programs/metasm
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

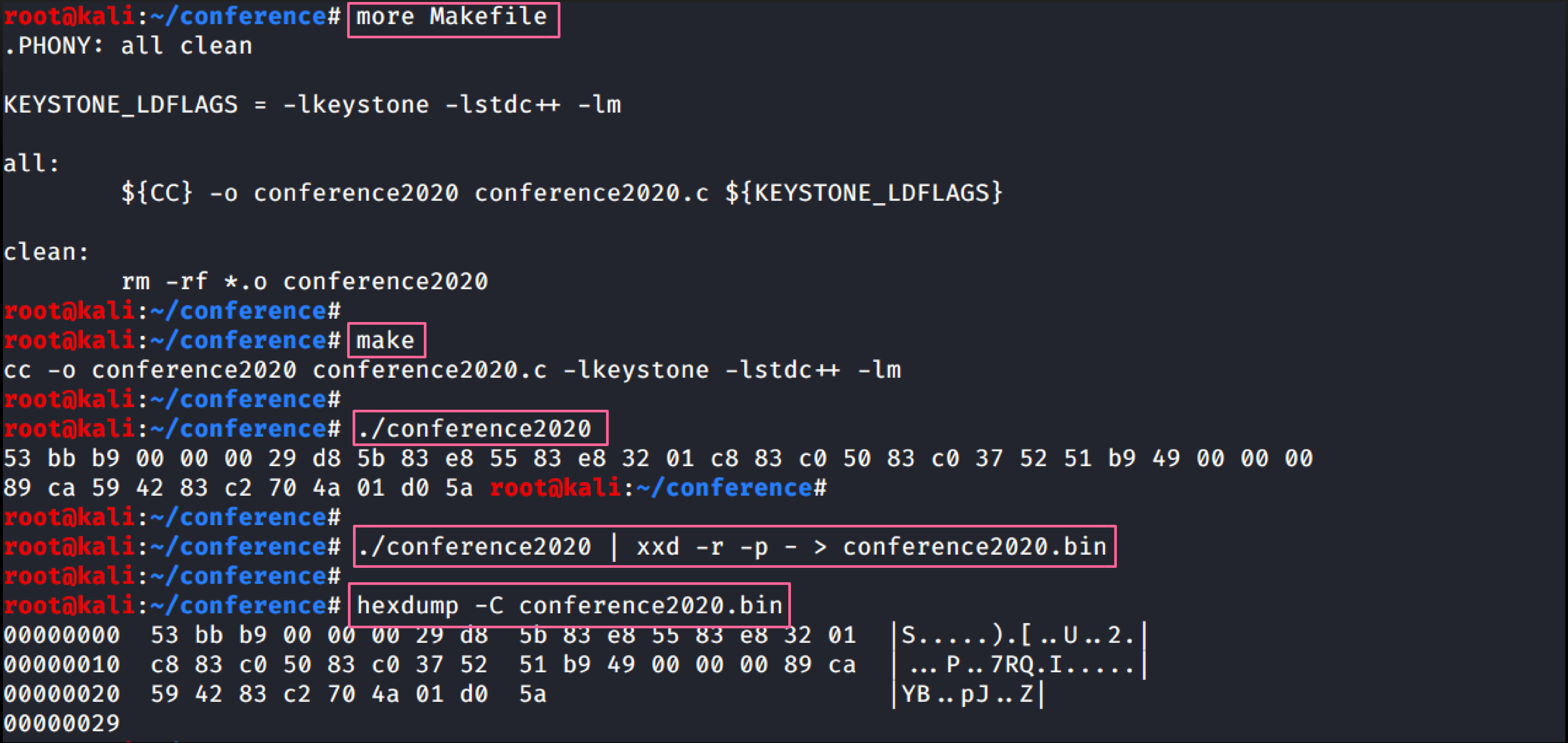
➢ DEOBFUSCATION



```c
#include <stdio.h>
#include <keystone/keystone.h>

#define SANS "push ebx; mov ebx,0xb9; sub eax,ebx; pop ebx; sub eax,0x55; sub
#eax,0x32; add eax,ecx; add eax,0x50; add eax,0x37; push edx; push ecx; mov
#ecx,0x49; mov edx,ecx; pop ecx; inc edx; add edx,0x70; dec edx; add eax,edx;
#pop edx"

int main(int argc, char **argv)
{
    ks_engine *keyeng;
    ks_err keyerr = KS_ERR_ARCH;
    size_t count;
    unsigned char *encode;
    size_t size;

    keyerr = ks_open(KS_ARCH_X86, KS_MODE_32, &keyeng);
    if (keyerr != KS_ERR_OK) {
        printf("ERROR: A fail has ocurred while calling ks_open(), quit\n");
        return -1;
    }

    if (ks_asm(keyeng, SANS, 0, &encode, &size, &count)) { printf("ERROR: A fail has occurred while calling ks_asm() with
    count = %lu, error code = %u\n", count, ks_errno(keyeng)); } else { size_t i;

        for (i=0; i < size; i++) {
            printf("%02x ", encode[i]);
        }
    }

    ks_free(encode);
    ks_close(keyeng);

    return 0;
}
```

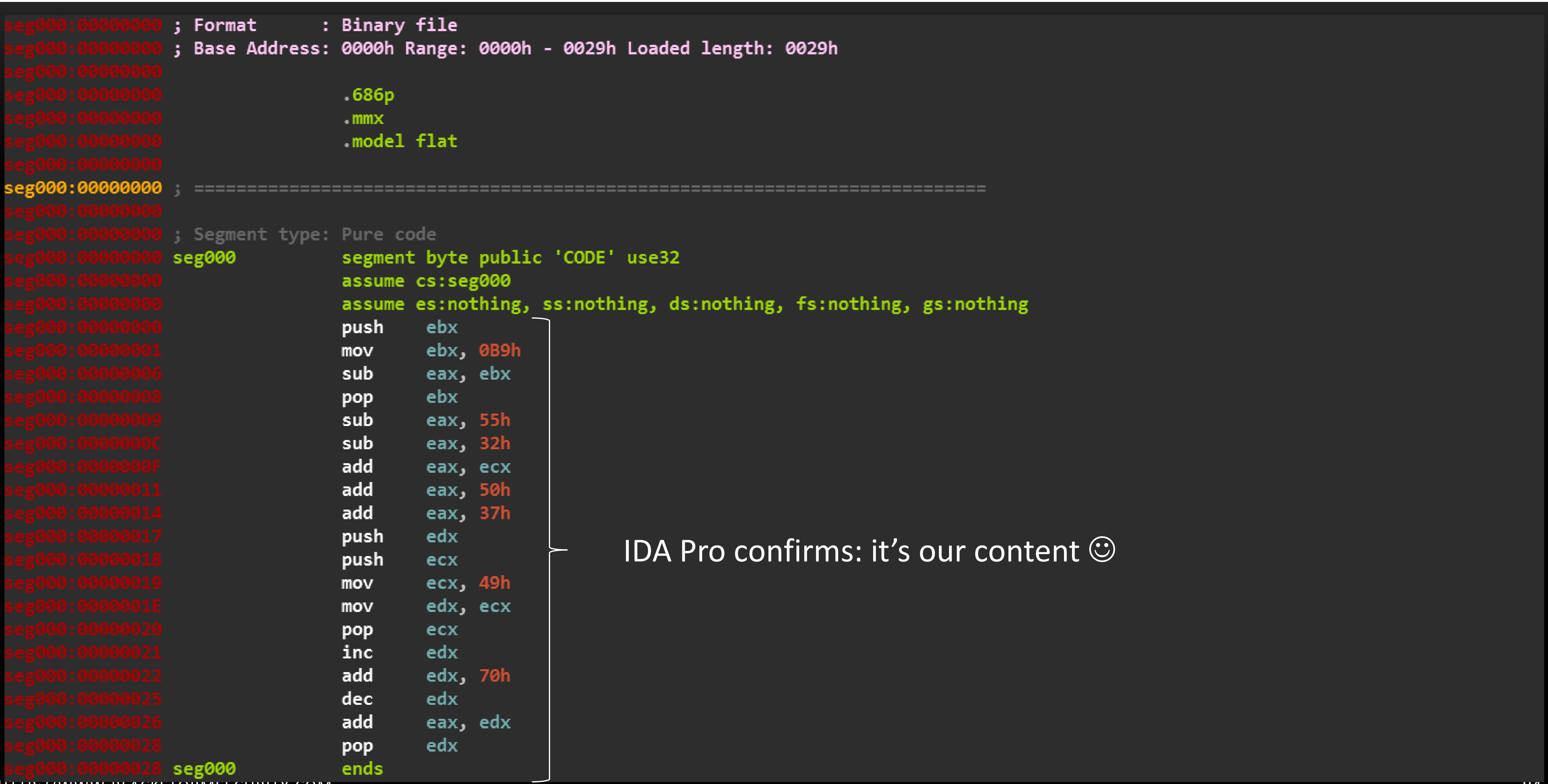
- instructions from the original obsfuscated code
- Creating a keystone engine
- Assembling our instructions using Keystone engine.
- Freeing memory and closing engine.

➢ DEOBFUSCATION



```
root@kali:~/conference# more Makefile
.PHONY: all clean

KEYSTONE_LDFLAGS = -lkeystone -lstdc++ -lm

all:
        ${CC} -o conference2020 conference2020.c ${KEYSTONE_LDFLAGS}

clean:
        rm -rf *.o conference2020
root@kali:~/conference#
root@kali:~/conference# make
cc -o conference2020 conference2020.c -lkeystone -lstdc++ -lm
root@kali:~/conference#
root@kali:~/conference# ./conference2020
53 bb b9 00 00 00 29 d8 5b 83 e8 55 83 e8 32 01 c8 83 c0 50 83 c0 37 52 51 b9 49 00 00 00
89 ca 59 42 83 c2 70 4a 01 d0 5a root@kali:~/conference#
root@kali:~/conference#
root@kali:~/conference# ./conference2020 | xxd -r -p - > conference2020.bin
root@kali:~/conference#
root@kali:~/conference# hexdump -C conference2020.bin
00000000  53 bb b9 00 00 00 29 d8  5b 83 e8 55 83 e8 32 01  |S.....).[..U..2.|
00000010  c8 83 c0 50 83 c0 37 52  51 b9 49 00 00 00 89 ca  |...P..7RQ.I.....|
00000020  59 42 83 c2 70 4a 01 d0  5a                       |YB..pJ..Z|
00000029
```

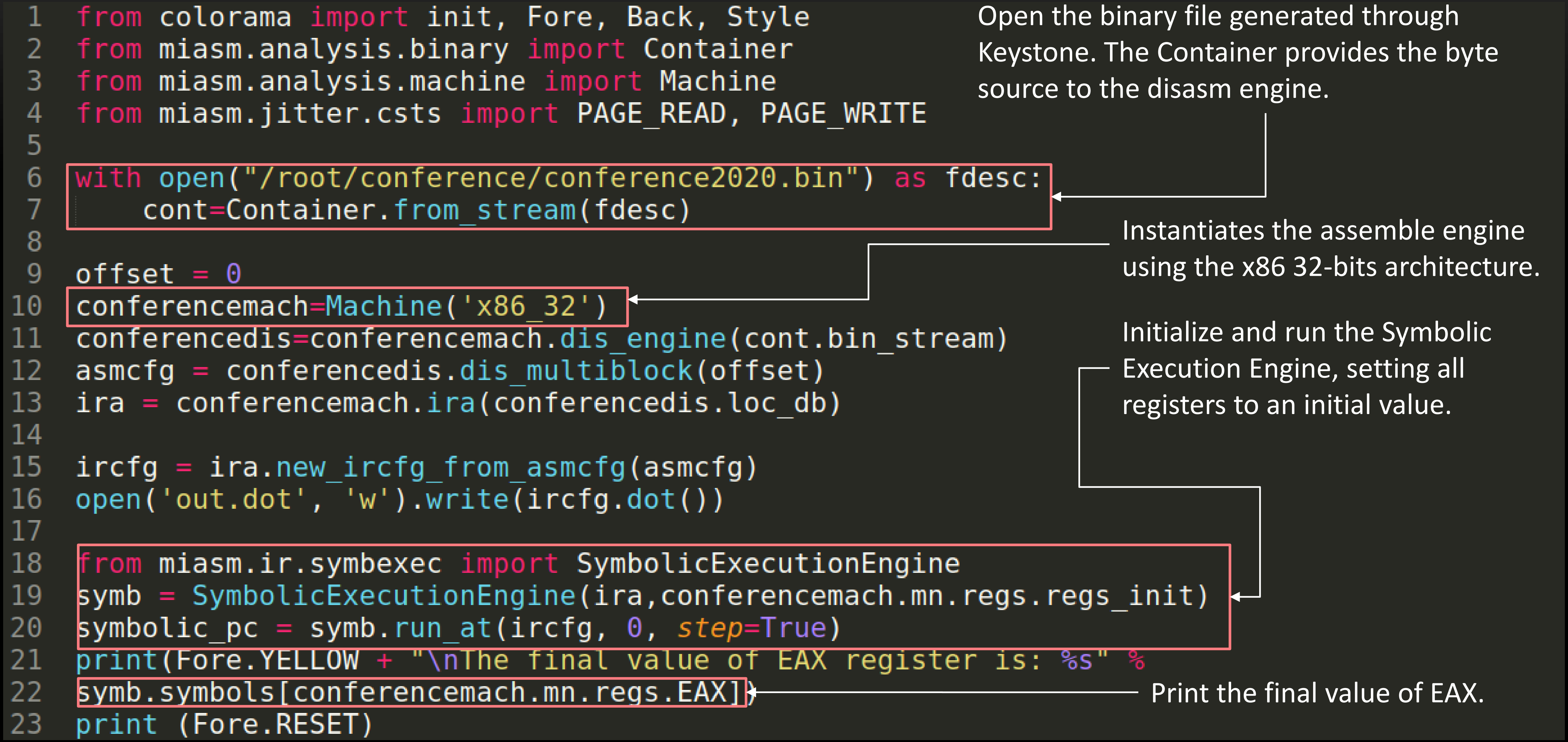
➢ DEOBFUSCATION



```
seg000:00000000 ; Format      : Binary file
seg000:00000000 ; Base Address: 0000h Range: 0000h - 0029h Loaded length: 0029h
seg000:00000000
seg000:00000000                 .686p
seg000:00000000                 .mmx
seg000:00000000                 .model flat
seg000:00000000
seg000:00000000 ; ===========================================================================
seg000:00000000
seg000:00000000 ; Segment type: Pure code
seg000:00000000 seg000          segment byte public 'CODE' use32
seg000:00000000                 assume cs:seg000
seg000:00000000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000                 push    ebx
seg000:00000001                 mov     ebx, 0B9h
seg000:00000006                 sub     eax, ebx
seg000:00000008                 pop     ebx
seg000:00000009                 sub     eax, 55h
seg000:0000000C                 sub     eax, 32h
seg000:0000000F                 add     eax, ecx
seg000:00000011                 add     eax, 50h
seg000:00000014                 add     eax, 37h
seg000:00000017                 push    edx
seg000:00000018                 push    ecx
seg000:00000019                 mov     ecx, 49h
seg000:0000001E                 mov     edx, ecx
seg000:00000020                 pop     ecx
seg000:00000021                 inc     edx
seg000:00000022                 add     edx, 70h
seg000:00000025                 dec     edx
seg000:00000026                 add     eax, edx
seg000:00000028                 pop     edx
seg000:00000028 seg000          ends
```

IDA Pro confirms: it’s our content ☺

DEOBFUSCATION



```python
from colorama import init, Fore, Back, Style
from miasm.analysis.binary import Container
from miasm.analysis.machine import Machine
from miasm.jitter.csts import PAGE_READ, PAGE_WRITE

with open("/root/conference/conference2020.bin") as fdesc:
    cont=Container.from_stream(fdesc)

offset = 0
conferencemach=Machine('x86_32')
conferencedis=conferencemach.dis_engine(cont.bin_stream)
asmcfg = conferencedis.dis_multiblock(offset)
ira = conferencemach.ira(conferencedis.loc_db)

ircfg = ira.new_ircfg_from_asmcfg(asmcfg)
open('out.dot', 'w').write(ircfg.dot())

from miasm.ir.symbexec import SymbolicExecutionEngine
symb = SymbolicExecutionEngine(ira,conferencemach.mn.regs.regs_init)
symbolic_pc = symb.run_at(ircfg, 0, step=True)
print(Fore.YELLOW + "\nThe final value of EAX register is: %s" %
symb.symbols[conferencemach.mn.regs.EAX])
print (Fore.RESET)
```

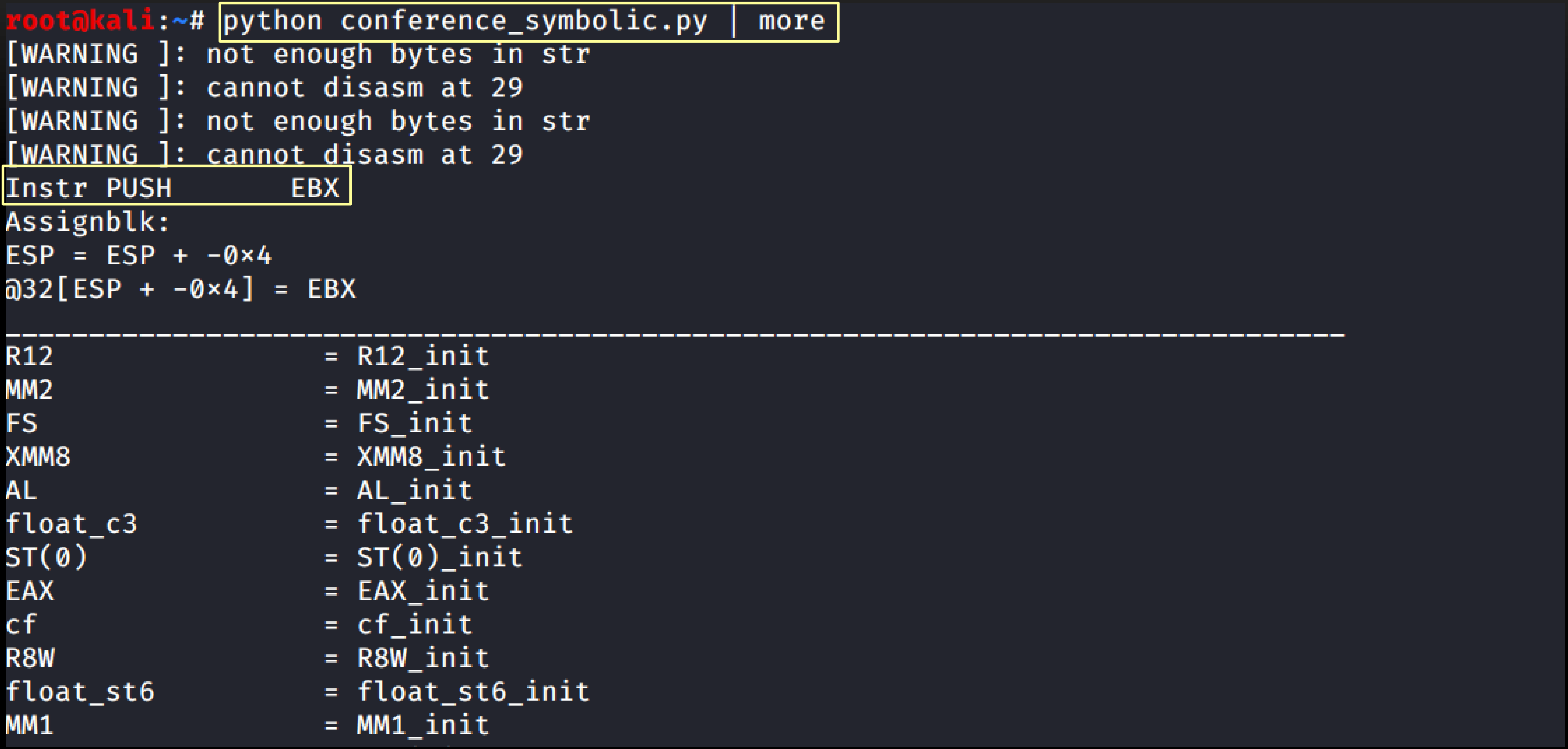
Open the binary file generated through Keystone. The Container provides the byte source to the disasm engine.

Instantiates the assemble engine using the x86 32-bits architecture.

Initialize and run the Symbolic Execution Engine, setting all registers to an initial value.

Print the final value of EAX.

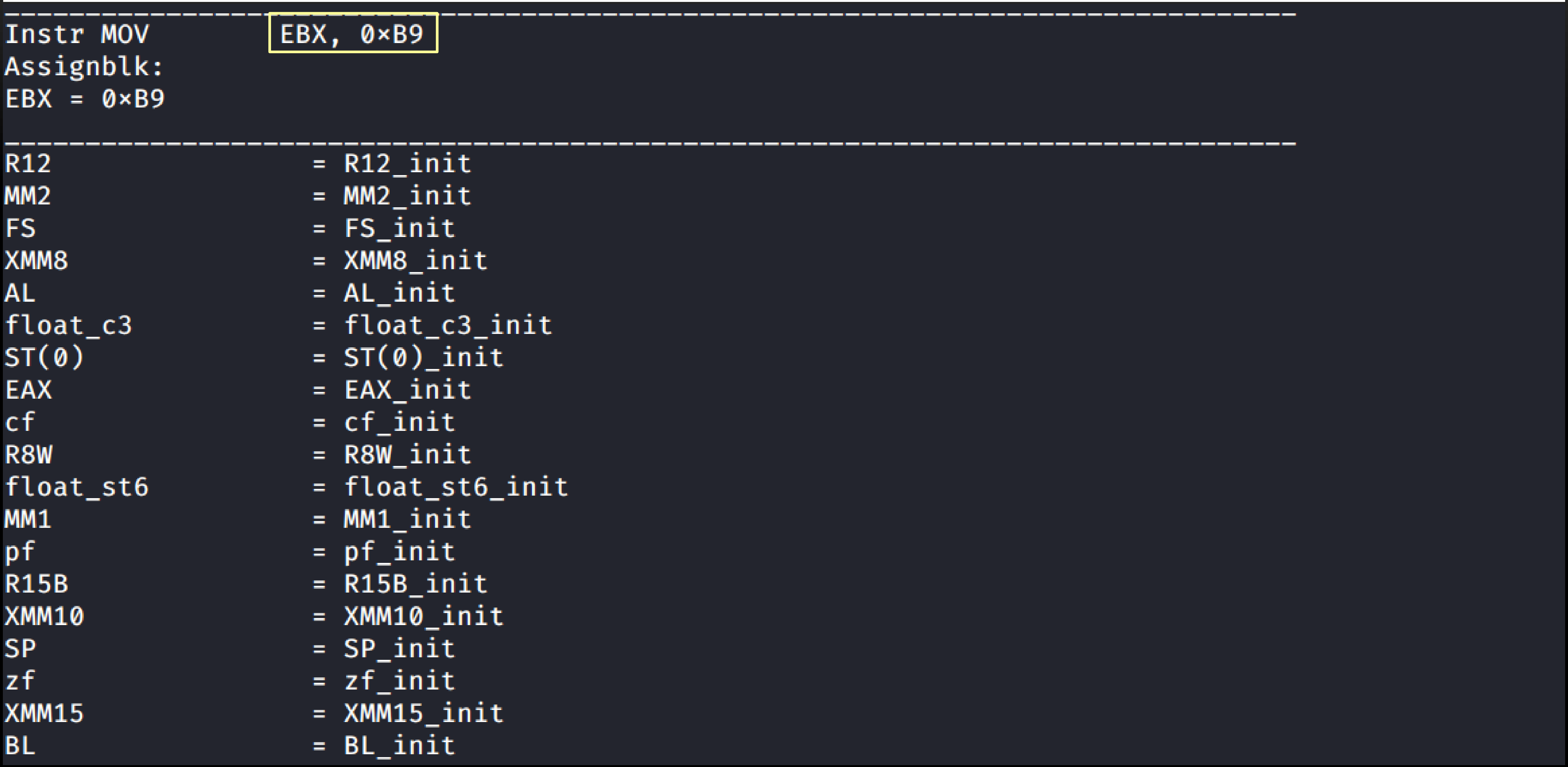
➢ DEOBFUSCATION



```
root@kali:~# python conference_symbolic.py | more
[WARNING ]: not enough bytes in str
[WARNING ]: cannot disasm at 29
[WARNING ]: not enough bytes in str
[WARNING ]: cannot disasm at 29
Instr PUSH      EBX
Assignblk:
ESP = ESP + -0x4
@32[ESP + -0x4] = EBX

--------------------------------------------------------------------------------
R12                 = R12_init
MM2                 = MM2_init
FS                  = FS_init
XMM8                = XMM8_init
AL                  = AL_init
float_c3            = float_c3_init
ST(0)               = ST(0)_init
EAX                 = EAX_init
cf                  = cf_init
R8W                 = R8W_init
float_st6           = float_st6_init
MM1                 = MM1_init
```

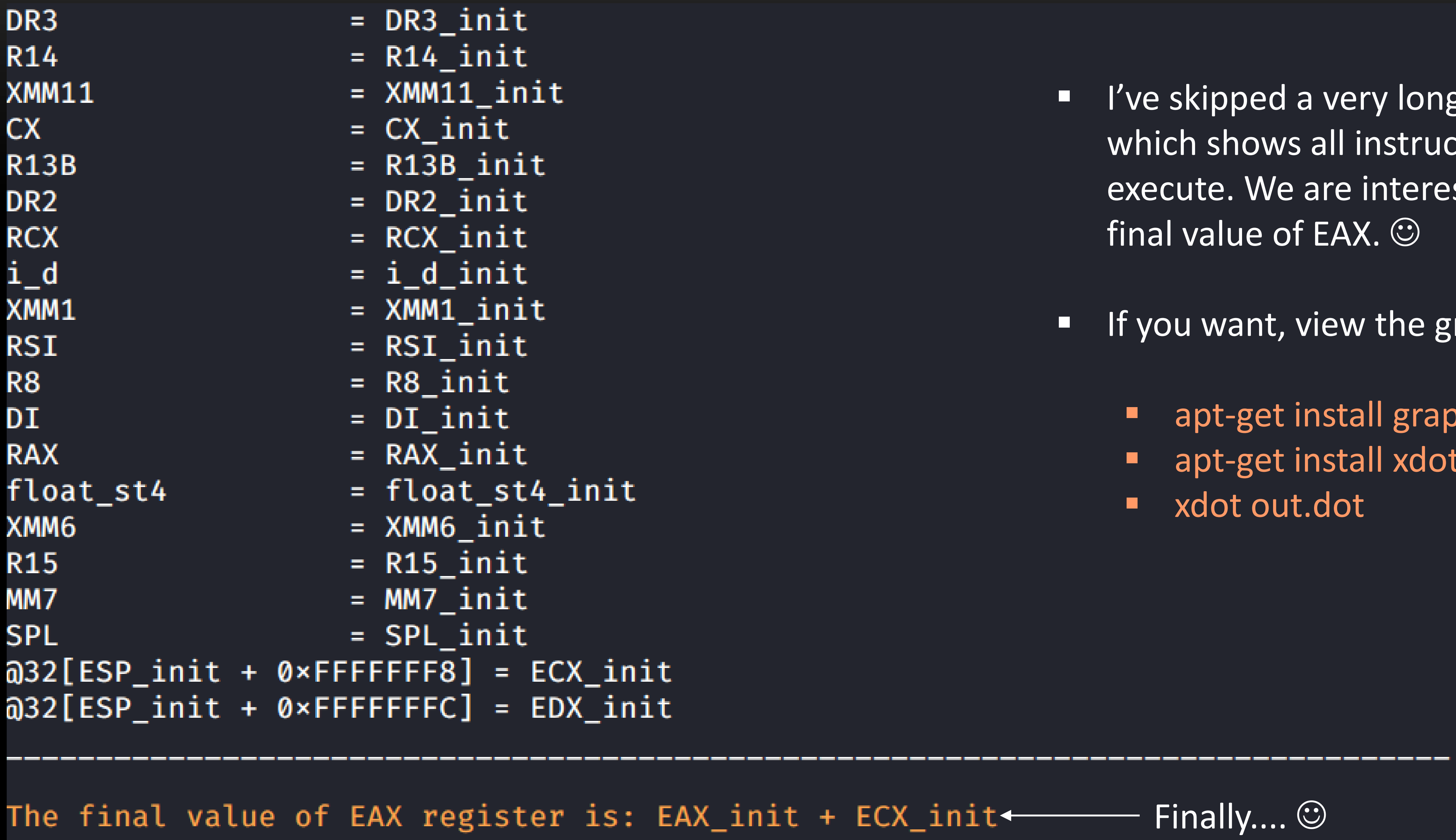
ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

DEOBFUSCATION



```
--------------------------------------------------------------------------------
Instr MOV        EBX, 0×B9
Assignblk:
EBX = 0×B9

--------------------------------------------------------------------------------
R12                = R12_init
MM2                = MM2_init
FS                 = FS_init
XMM8               = XMM8_init
AL                 = AL_init
float_c3           = float_c3_init
ST(0)              = ST(0)_init
EAX                = EAX_init
cf                 = cf_init
R8W                = R8W_init
float_st6          = float_st6_init
MM1                = MM1_init
pf                 = pf_init
R15B               = R15B_init
XMM10              = XMM10_init
SP                 = SP_init
zf                 = zf_init
XMM15              = XMM15_init
BL                 = BL_init
```

➢ DEOBFUSCATION



```
DR3                  = DR3_init
R14                  = R14_init
XMM11                = XMM11_init
CX                   = CX_init
R13B                 = R13B_init
DR2                  = DR2_init
RCX                  = RCX_init
i_d                  = i_d_init
XMM1                 = XMM1_init
RSI                  = RSI_init
R8                   = R8_init
DI                   = DI_init
RAX                  = RAX_init
float_st4            = float_st4_init
XMM6                 = XMM6_init
R15                  = R15_init
MM7                  = MM7_init
SPL                  = SPL_init
@32[ESP_init + 0xFFFFFFF8] = ECX_init
@32[ESP_init + 0xFFFFFFFC] = EDX_init

--------------------------------------------------------------------------------

The final value of EAX register is: EAX_init + ECX_init
```

← Finally.... ☺

- I've skipped a very long output, which shows all instruction being execute. We are interested in the final value of EAX. ☺

- If you want, view the graph:

  - apt-get install graphviz
  - apt-get install xdot
  - xdot out.dot

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

➢ Closing thoughts and Acknowledgments...

- Honestly, I didn’t even scratch the surface of this topic....

- There’re tons of obfuscation techniques and de-obfuscation/reversing techniques to explain... it would take one or two entire weeks...and probaly you wouldn’t to know about it... ☺

- I’d like to thank SANS for the event and, in special, my friend Stephen Sims for the invite. ☺

- And, of course, I’d like to thank you (the audience) for attending my talk.

THANK YOU FOR ATTENDING MY TALK ☺

- Security Researcher
- Speaker at DEF CON USA 2019
- Speaker at DEF CON USA 2018
- Speaker at DEF CON CHINA 2019
- Speaker at DC2711 (Johannesburg)
- Speaker at NO HAT 2019 (Italy)
- Speaker at HITB 2019 (Amsterdam)
- Speaker at CONFidence 2019 (Poland)
- Speaker at DevOpsDays BH 2019
- Speaker at BSIDES 2019/2018/2017/2016
- Speaker at H2HC 2016/2015
- Speaker at BHACK 2018
- Researcher on Android/iOS Reversing, Rootkits and Digital Forensics.
- Referee on Digital Investigation: The International Journal of Digital Forensics & Incident Response

- Twitter:

  @ale_sp_brazil
  @blackstormsecbr

- Website: http://www.blackstormsecurity.com

- LinkedIn: http://www.linkedin.com/in/aleborges

- E-mail: alexandreborges@blackstormsecurity.com