



Software Reverse Engineering

COMP 272 | Spring 2022 | University of the Pacific | Jeff Shafer

Disassemblers and Debuggers

KNOW YOUR MALWARE 101

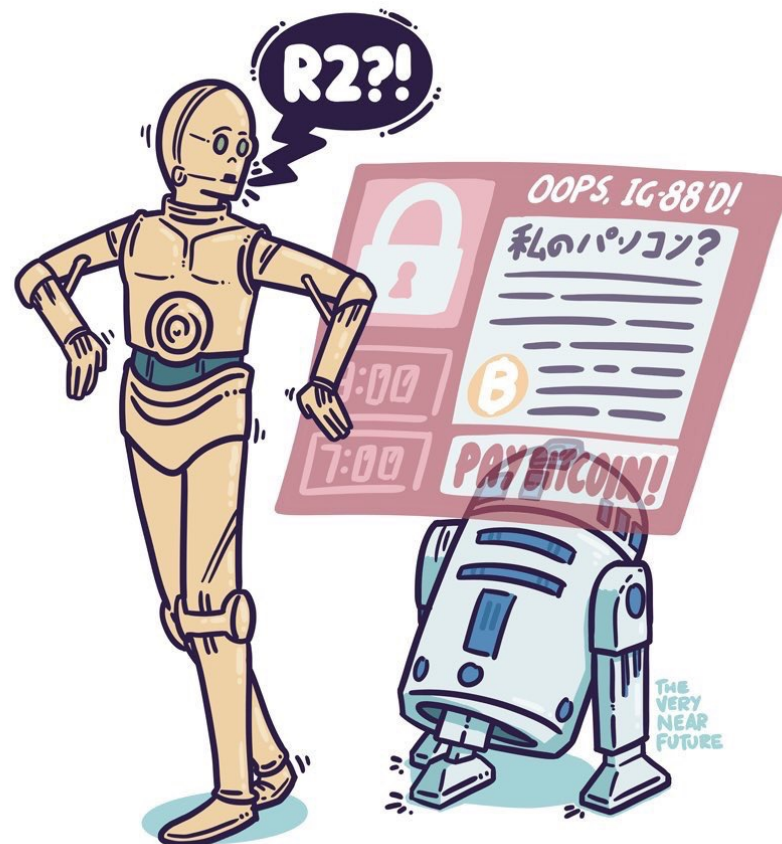


Malware



R2D2 – October 2011

- Backdoor/Trojan that communicates with remote server to obtain attacker commands
- Capabilities
 - Captures keystrokes in Skype, MSN messenger, Yahoo Messenger, ICQ
 - Capture keystrokes in web browsers (IE, Firefox, Opera)
 - Capture screen shots
 - Download and execute arbitrary files



- Revealed by “Chaos Computer Club” (European hacker club)
- Alleged to be “lawful intercept” program from German police forces (despite not being lawful in Germany)
 - Submitted to CCC anonymously
 - Allegedly installed as a laptop passed through Munich Airport customs control
- Poorly constructed malware – C2 infrastructure is not encrypted (!!) and lacks authentication
 - Other parties could take over an infected host

R2D2 Details

➤ Two files

➤ `mfc42u1.dll`

MD5 930712416770a8d5e6951f3e38548691

➤ `winsys32.sys`

MD5 d6791f5aa6239d143a22b2a15f627e72

R2D2 Details

- Named after function in DLL that triggers data transmission:
C3PO-r2d2-POE
- Communicates with 83.236.140.90 and 207.158.22.134 (German IP)

R2D2 Details

```
remnux@remnux:~/Desktop$ strings mfc42ul.dll | grep .exe
skype.exe
seamonkey.exe
navigator.exe
opera.exe
iexplore.exe
firefox.exe
%s~tmp%08x~.exe
SkypePM.exe
\sipgatexlite.exe
\x-lite.exe
\yahoомessenger.exe
\msnmsgr.exe
\explorer.exe
\SkypePM.exe
\Skype.exe
```

```

.text:1000F590
.text:1000F590 ; :::::::::::::::::::: S U B R O U T I N E ::::::::::::::::::::
.text:1000F590
.text:1000F590
.text:1000F590 ; BOOL __stdcall DllMain(HINSTANCE hinstDLL,DWORD fdwReason,LPVOID lpvReserved)
.text:1000F590 _DllMain@12      proc near          ; CODE XREF: DllEntryPoint+4B↓p
.text:1000F590
.text:1000F590 var_C           = dword ptr -0Ch
.text:1000F590 var_4          = dword ptr -4
.text:1000F590 hinstDLL       = dword ptr 4
.text:1000F590 fdwReason      = dword ptr 8
.text:1000F590 lpvReserved    = dword ptr 0Ch
.text:1000F590
* .text:1000F590      mov     eax, large fs:0
* .text:1000F596      push   0FFFFFFFh
* .text:1000F598      push   offset loc_1003C07B
* .text:1000F59D      push   eax
* .text:1000F59E      mov     eax, [esp+0Ch+fdwReason]
* .text:1000F5A2      mov     large fs:0, esp
* .text:1000F5A9      cmp     eax, 1
* .text:1000F5AC      push   esi
* .text:1000F5AD      jnz    loc_1000F68A
* .text:1000F5B3      push   offset aSkype_exe ; "\\Skype.exe"
* .text:1000F5B8      xor     esi, esi
* .text:1000F5BA      call   sub_10003990
* .text:1000F5BF      add     esp, 4
* .text:1000F5C2      test   al, al
* .text:1000F5C4      jz     short loc_1000F5CB
* .text:1000F5C6      mov     esi, 1
* .text:1000F5CB      loc_1000F5CB:          ; CODE XREF: DllMain(x,x,x)+34↑j
* .text:1000F5CB      push   offset aSkypepm_exe ; "\\SkypePM.exe"
* .text:1000F5D0      call   sub_10003990
* .text:1000F5D5      add     esp, 4
* .text:1000F5D8      test   al, al
* .text:1000F5DA      jz     short loc_1000F5E1
* .text:1000F5DC      mov     esi, 2
* .text:1000F5E1      loc_1000F5E1:          ; CODE XREF: DllMain(x,x,x)+4A↑j
* .text:1000F5E1      push   offset aExplorer_exe ; "\\explorer.exe"
* .text:1000F5E6      call   sub_10003990
* .text:1000F5EB      add     esp, 4
* .text:1000F5EE      test   al, al
* .text:1000F5F0      jz     short loc_1000F5F7
* .text:1000F5F2      mov     esi, 0Fh

```


R2D2 Resources

- <https://nakedsecurity.sophos.com/2011/10/10/german-government-r2d2-trojan-faq/>
- <https://blog.trendmicro.com/trendlabs-security-intelligence/backdoor-snoops-on-skype-msn-and-yahoo-messenger/>
- <http://www.stoned-vienna.com/analysis-of-german-bundestrojaner.html>
- <http://ccc.de/en/updates/2011/staatstrojaner>
 - Download binary

Behavioral Analysis

- You can observe a lot from behavioral analysis
 - **We found many potential IOCs in Exam 1**
- **What are some things that we don't yet know about the malware?**
 - What is the **malicious** purpose of this malware?
The programmers did a lot of work to accomplish what?
 - Networking related (dealing with remote systems)
 - Why did the malware disable the firewall? (Will we be receiving something?)
 - What is supposed to happen after it retrieves `/Api/G.jsp` from the remote server?
 - What is supposed to happen if a response is received from one of the `sandai.net` servers?
 - Is the Server-Key HTTP header exfiltrated data, or not?



Time to
Dig Deep!



Disassemblers, Decompilers, Debuggers

Disassemblers, Decompilers, Debuggers

Executable

- Binary machine code
- Designed to be read by CPU



Disassembler



Assembly Code

- Assembly code
- Designed to be read by humans
(*some of us, anyway*)

```

push    edi
mov     eax, init_val
mov     [ebp+size], eax
mov     al, byte_40A12C
mov     byte ptr [ebp+ptr],
mov     esi, offset default
lea    edi, [ebp+array]
mov     ecx, 5
rep movsd
push    5
lea    eax, [ebp+size]
push   eax
call   adjust_array

```

Disassembler

- Disassembly *should* be straightforward process
 1. Read in a sequence of bytes
 2. Look it up in a table (provided by CPU designer) to see what assembly instruction corresponds
 - There should only be *one* possible match, as CPUs are deterministic
 3. Emit assembly instruction as output

Disassembler

- Big challenge for disassembler construction:
Separating *Code* from *Data*
 - Here are some bytes. Are they *data* bytes or *instructions* bytes?
 - Code and data may not be fully separated into `.text` and `.data` sections
 - A human can *guess* or use heuristics but can't say for certain

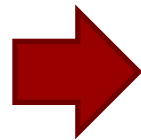
- One method to provide reasonable results on normal executables
 1. Start at the entry point of the executable
 2. Assume that is a valid instruction
 3. Walk to all code reachable from that location (*recursively*)
 4. Everything else is “data” (you hope)

Disassemblers, Decompilers, Debuggers

Disassembler

Decompiler

Executable



Assembly
Code



High Level
Language

- Binary machine code
- Designed to be read by CPU

- Assembly code
- Designed to be read by humans
(*some of us, anyway*)

- C code
- Designed to be read by humans

```
001001010010100110101101100101100100
0101010010100101010010100010100101000
01010100010101010101000100101010100101
0101010100010101001010100101010100101
00101010101001010101000101101000010101
0101010001010001010101010101010010100
010101011100100101001010100101001010010
10100100101011101100101101010100010101
01001010101010001001010001010100010101
01010010101001010001010010101001010101
00101010010100101010101001010010100101
10101010010101110110101001010111010101
11010101010110101010101001010010101010
0110010101010010101010101010010101001
1100101010010101010101010101010101001
01010011010001010101010101010010101010
010101010101001010101010101001010101010
00101010010101001010101010010100101010
110010111001010101110101010101001111
```


```
push    edi
mov     eax, init_val
mov     [ebp+size], eax
mov     al, byte_40A12C
mov     byte ptr [ebp+ptr],
mov     esi, offset default
lea    edi, [ebp+array]
mov     ecx, 5
rep movsd
push    5
lea    eax, [ebp+size]
push    eax
call   adjust_array
add    esp, 8
```

```
// Include needed header files
#include <OC_const.h> // OC
#include <Common.h> // Common
#include <stdio.h> // Stdio
#include <Data.h> // Data
#include <OC_nag8.h> // OC

int GetData()
{
    // Declare local variable hold
    int iErr;

    // *** Declare and initialize
    // *** passed into the NAG fun
    int nPts; // Input numb
    int nTdx; // Input numb
    matrix mX; // Input matr
    vector vY; // Input vect
    vector vWT; // Input vect
```


Decompiler

- A decompiler is a very complicated program
 - There are multiple possible high-level programs that could map into the same binary program
 - Compiler aggressively inlines and reorders machine code for speed
 - Important information is *lost* in the original compilation process
 - Symbol table (variable names, function names, ...)
 - Different decompilers will produce different results
- **Don't think** “a decompiler gives me back the original source code as the malware author wrote it” 
- **Think** “a decompiler gives me pseudo-code resembling the original program”
 - The more **expensive** the decompiler, the better the pseudo-code!

```
; ===== S U B R O U T I N E =====
```

```
; int __cdecl sub_4061C0(char *Str, char *Dest)
sub_4061C0      proc near                ; CODE XREF: sub_4062F0+15p
                                           ; sub_4063D4+21p ...
```

```
Str            = dword ptr  4
Dest           = dword ptr  8
```

```
    push      esi
    push      offset aSmtp_      ; "smtp."
    push      [esp+8+Dest]      ; Dest
    call      _strcpy
    mov       esi, [esp+0Ch+Str]
    push      esi                ; Str
    call      _strlen
    add       esp, 0Ch
    xor       ecx, ecx
    test      eax, eax
    jle      short loc_4061ED
```

```
loc_4061E2:                ; CODE XREF: sub_4061C0+2Bj
    cmp       byte ptr [ecx+esi], 40h
    jz        short loc_4061ED
    inc       ecx
    cmp       ecx, eax
    jl        short loc_4061E2
```

IDA Disassembler
Output

```

signed int __cdecl sub_4061C0(char *Str, char *Dest)
{
    int len; // eax@1
    int i; // ecx@1
    char *str2; // esi@1
    signed int result; // eax@5

    strcpy(Dest, "smtp.");
    str2 = Str;
    len = strlen(Str);
    for ( i = 0; i < len; ++i )
    {
        if ( str2[i] == 64 )
            break;
    }
    if ( i < len - 1 )
    {
        strcat(Dest, &str2[i + 1]);
        result = 1;
    }
    else

```

IDA Decompiler
Output

Debugger

- Tool that allows you to *view* and *change* the *state* of a running program
- Useful in bypassing obfuscation techniques used in the static binary
 - E.g. The data you seek may be scrambled or encrypted on the disk but it has to be descrambled/decrypted *sometime!* Find the right place to set a breakpoint (tricky) and you can steal it right out of memory
- You've presumably used the Visual Studio debugger to step through your C++ code
- We'll use a debugger packaged with a disassembler (x64dbg) to step through malware assembly code



Analysis Tools

Analysis Tools

- Tools we will use in course
 - **IDA Pro** (demo version)
 - **x32dbg / x64dbg**

- Other (current) hot tools
 - **Ghidra** – Released March 2019 by NSA

- Other tools
 - Retdec
 - Hopper
 - Radare2, Cutter
 - WinDbg
 - OllyDbg (*an old classic*)

Analysis Tool – IDA Pro

- Commercial disassembler
 - Platform: Windows, Linux, Mac
 - Disassembles binaries for more processor architectures than you've heard of!

- Features
 - Scriptable (Python)
 - Code labeling (w/propagation)
 - Visualization (flow view)
 - Decompiler (w/additional purchase)
 - Debugger (for dynamic analysis)

Analysis Tool – IDA Pro

- Example cost (Feb 2022)
 - IDA Pro Named License - \$1879 USD
 - x64 Decompiler Fixed License - \$2629
 - x32 Decompiler Fixed License - \$2629
- *Discussion on IDA, piracy, and career risks*

Analysis Tool – IDA Pro

- **IDA Freeware 5.0** for Windows
 - 32-bit PE executables only, other features disabled
 - No commercial use
 - Version 5.x is old – 7.x is current
 - *No longer available as-of 2018 but we have it in Windows VM (including installer)*

- **IDA Freeware 7.0** for Windows, Mac, Linux
 - 64-bit PE or ELF executables only
 - No commercial use
 - No Python scripting, no debugging, no X,Y,Z – But free!
 - Silently released Feb 2 2018 except for Twitter message from Igor Skochinsky (developer) – **Long awaited!**

Welcome to Reddit, the front page of the internet.

BECOME A REDDITOR

and subscribe to one of thousands of communities.



IDA freeware version is 7.0?! (hex-rays.com)

15 points submitted 25 days ago by galapag0

45 comments share save hide report

all 45 comments

sorted by: **best** ▼

Want to add to the discussion?

Post a comment!

CREATE AN ACCOUNT

[-] **odd100** 20 points 25 days ago

OMG OMG OMG OMG

permlink embed save report reply

[-] **revolct** 13 points 25 days ago

is this real?

permlink embed save report reply

load more comments (16 replies)

[-] **C0rn3j** 11 points 25 days ago

Seems real - TOS from the installer

<https://haste.c0rn3j.com/muticizaja.txt>

About

Freeware version with the following limitations:

1. Only for non-commercial use
2. Without technical support
3. Only supports x64 code

For commercial use please acquire the full version

permlink embed save report reply

Analysis Tool – x32dbg / x64dbg

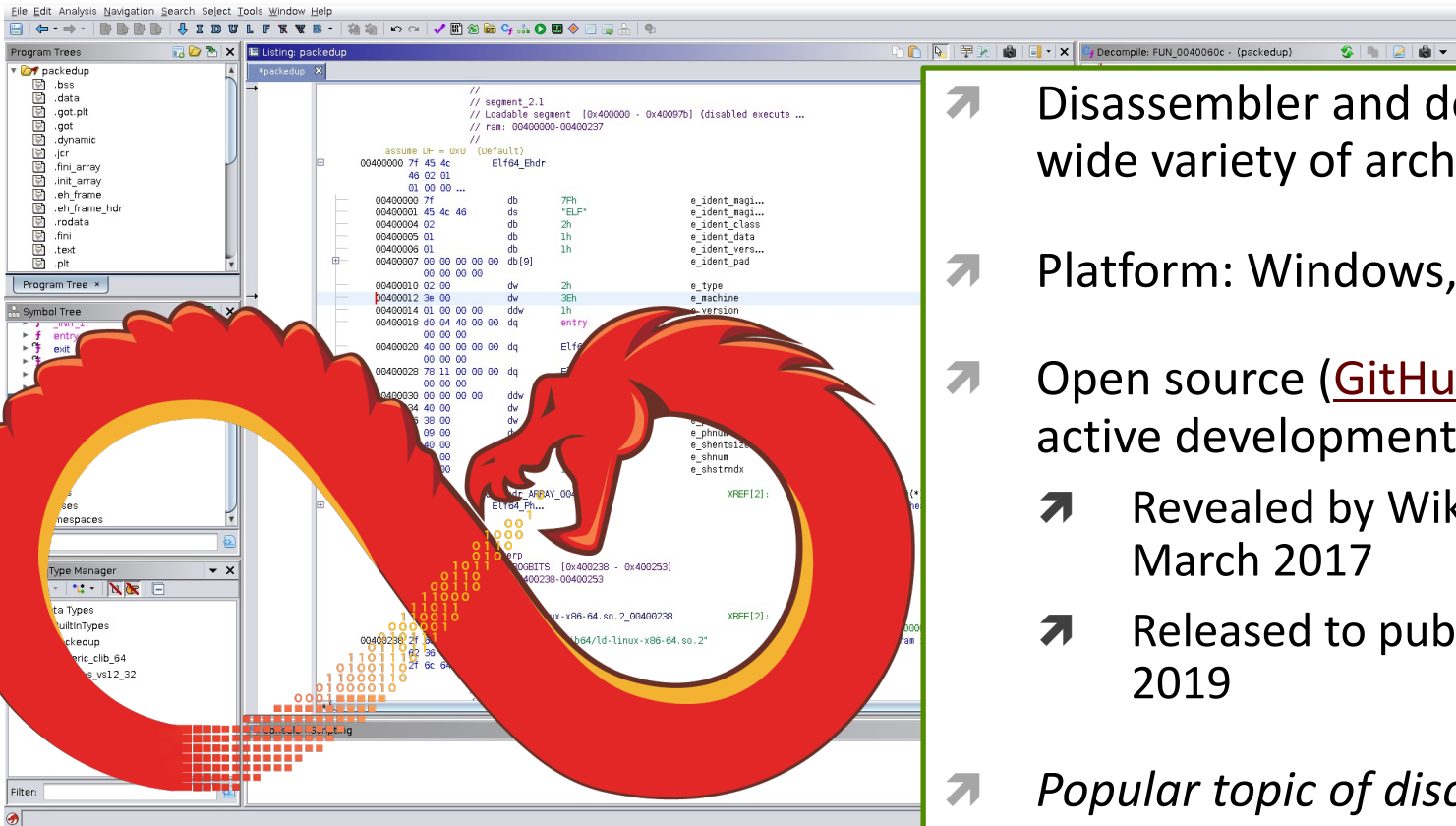
The screenshot shows the x64dbg disassembler interface. The CPU register window on the left shows EAX, EIP, and ECX. The main window displays assembly code with hex addresses, hex bytes, and mnemonics. The code includes instructions like `call crackme.40139C`, `leave`, `retn 10`, `lea edx, dword ptr ds:[...]`, `push edx`, `call crackme.4013FC`, `mov ebp, eax`, `mov ecx, 5`, `xor esi, esi`, `xor eax, eax`, `mov cl, byte ptr ds:[...]`, `mov bl, cl`, `xor bl, byte ptr ds:[...]`, `inc eax`, `cmp eax, 5`, `mov byte ptr ds:[edx], ...`, `mov byte ptr ds:[eax], ...`, `jnz crackme.401196`, `xor eax, eax`, `inc esi`, `cmp esi, ebp`, `jb crackme.40117A`, `xor edi, edi`, `xor ecx, ecx`, `test ebp, ebp`, `jbe crackme.4011C9`, `mov bl, byte ptr ds:[...]`, `mov esi, ebp`, `sub esi, ecx`, `dec esi`, `mov al, byte ptr ds:[...]`, and `xor bl, al`.

➔ Free disassembler and **debugger** for 32 and 64 bit executables

➔ Platform: Windows-only

➔ Open source ([GitHub](https://github.com)) and under active development

Analysis Tool - Ghidra



- Disassembler and decompiler for wide variety of architectures
- Platform: Windows, Mac, Linux
- Open source ([GitHub](https://github.com)) and under active development by NSA
 - Revealed by WikiLeaks in March 2017
 - Released to public in March 2019
- *Popular topic of discussion: Do I need IDA Pro if I have this?*

Ghidra

Analysis Tool - RetDec

- Retargetable Decompiler
- Free! 😊 Very memory intensive 😞 Slow 😞
- Cross platform: Windows, Linux, and (sorta) Mac
- Inputs
 - PE and ELF file formats (among others)
 - Intel x86 (but not x64), ARM, MIPS, PowerPC
- Outputs
 - “C”
 - “Python-like” language
 - Call graphs, control flow graphs, statistics...

Demo of RetDec using Lab 3 Executable

(It's tiny and 32 bits. It's just a
dropper, remember?)

<https://retdec.com/decompilation/>

```

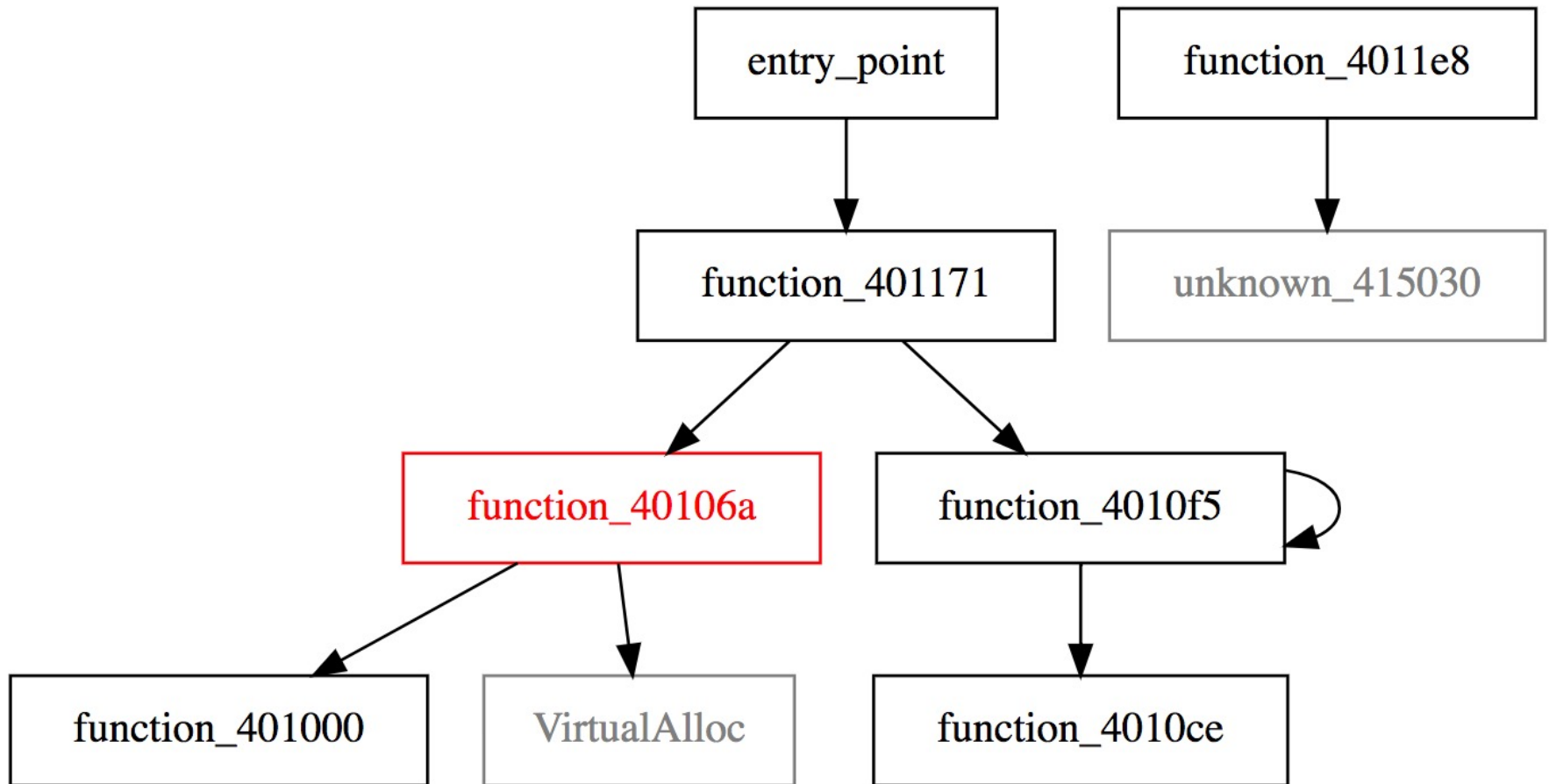
1 ;;
2 ;; This file was generated by the Retargetable Decompiler
3 ;; Website: https://retdec.com
4 ;; Copyright (c) 2018 Retargetable Decompiler <info@retdec.com>
5 ;;
6 ;; Decompiler release: v2.2.1 (2016-09-07)
7 ;; Decompilation date: 2018-02-28 08:25:39
8 ;; Architecture: x86
9 ;;
10
11 ;;
12 ;; Code Segment
13 ;;
14
15 ; section: .text
16 ; function: function_401000 at 0x401000 -- 0x401069
17 0x401000: 55          push ebp
18 0x401001: 89 e5       mov ebp, esp
19 0x401003: 57          push edi
20 0x401004: 56          push esi
21 0x401005: 53          push ebx
22 0x401006: 52          push edx
23 0x401007: 31 d2       xor edx, edx
24 0x401009: 31 c9       xor ecx, ecx
25 0x40100b: 31 f6       xor esi, esi
26 0x40100d: eb 50       jmp 0x40105f <function_401000+0x5f>
27 0x40100f: 8b 7d 08    mov edi, dword [ ebp + 0x8 ]
28 0x401012: 8a 1c 0f    mov bl, byte [ edi + ecx * 0x0 ]
29 0x401015: f6 c3 df    test bl, 0xdf
30 0x401018: 75 03       jnz 0x40101d <function_401000+0x1d>
31 0x40101a: 41         inc ecx
32 0x40101b: eb 42       jmp 0x40105f <function_401000+0x5f>
33 0x40101d: 89 f0       mov eax, esi
34 0x40101f: 83 e0 01    and eax, 0x1
35 0x401022: 83 f8 01    cmp eax, 0x1
36 0x401025: 19 c0       sbb eax, eax
37 0x401027: 83 e0 d5    and eax, 0xfffff5d
38 0x40102a: 83 c0 6c    add eax, 0x6c
39 0x40102d: 88 45 f3    mov byte [ ebp + 0xfffffff3 ], al
40 0x401030: 29 c3       sub ebx, eax
41 0x401032: c1 e3 04    shl ebx, 0x4
42 0x401035: 8b 7d 08    mov edi, dword [ ebp + 0x8 ]
43 0x401038: 8a 44 0f 01 mov al, byte [ edi + ecx * 0x0 + 0x1 ]
44 0x40103c: 2a 45 f3    sub al, byte [ ebp + 0xfffffff3 ]
45 0x40103f: 83 e0 0f    and eax, 0xf
46 0x401042: 09 c3       or ebx, eax
47 0x401044: 83 c1 02    add ecx, 0x2
48 0x401047: 8b 7d 14    mov edi, dword [ ebp + 0x14 ]
49 0x40104a: 32 1c 17    xor bl, byte [ edi + edx * 0x0 ]
50 0x40104d: 8b 7d 10    mov edi, dword [ ebp + 0x10 ]
51 0x401050: 88 1c 37    mov byte [ edi + esi * 0x0 ], bl
52 0x401053: 46         inc esi
53 0x401054: 8d 42 01    lea eax, dword [ edx + 0x1 ]
54 0x401057: bb 06 00 00 00 mov ebx, 0x6
55 0x40105c: 99         cdq
56 0x40105d: f7 fb     idiv ebx
57 0x40105f: 7b 44 0e    cmp ebx, dword [ ebp + 0xe ]

```

```

1 //
2 // This file was generated by the Retargetable Decompiler
3 // Website: https://retdec.com
4 // Copyright (c) 2018 Retargetable Decompiler <info@retdec.com>
5 //
6 //
7 #include <stdbool.h>
8 #include <stdint.h>
9 #include <stdlib.h>
10 #include <windows.h>
11
12 // ----- Function Prototypes -----
13
14 int32_t entry_point(void);
15 int32_t function_401000(int32_t a1, int32_t a2, int32_t a3, int32_t a4);
16 char * function_40106a(char * a1, int32_t dwSize, int32_t a3);
17 int32_t function_4010ce(int32_t a1, int32_t a2);
18 int32_t function_4010f5(int32_t a1, char * a2, int32_t a3);
19 int32_t function_401171(void);
20 void function_4011e8(void);
21 int32_t unknown_415030(void);
22
23 // ----- Global Variables -----
24
25 int32_t g1 = 0; // eax
26 int32_t g2 = 0; // ebp
27 int32_t g3 = 0; // ebx
28 int32_t g4 = 0; // edi
29 int32_t g5 = 0; // edx
30 int32_t g6 = 0; // esi
31 char * g7 = "\xc0\x5c\x80\xcc\x6d\xf2"; // 0x402000
32 char * g8 = "\x82\x06\xc3\x88\x28\xb6"; // 0x402008
33
34 // ----- Functions -----
35
36 // Address range: 0x401000 - 0x401069
37 int32_t function_401000(int32_t a1, int32_t a2, int32_t a3, int32_t a4) {
38     int32_t result = g5; // 0x401006
39     g5 = 0;
40     int32_t v1 = 0; // esi
41     if (a2 > 0) {
42         int32_t v2 = 0; // 0x40104a
43         int32_t v3 = 0; // 0x401050
44         int32_t v4 = g3; // 0x401012
45         int32_t v5 = 0; // 0x40101a
46         int32_t v6 = a1; // 0x401035
47         while (true) {
48             int32_t v7 = (int32_t)*(char *) (v6 + v5) | v4 & -256; // 0x401012
49             int32_t v8; // 0x40101d26
50             int32_t v9; // 0x40104a28
51             int32_t v10; // 0x40105f
52             int32_t v11; // 0x40101224
53             if (v7 == 223) {
54                 // 0x40101a
55                 v10 = v5 + 1;
56                 v9 = v2;
57                 v8 = v7;

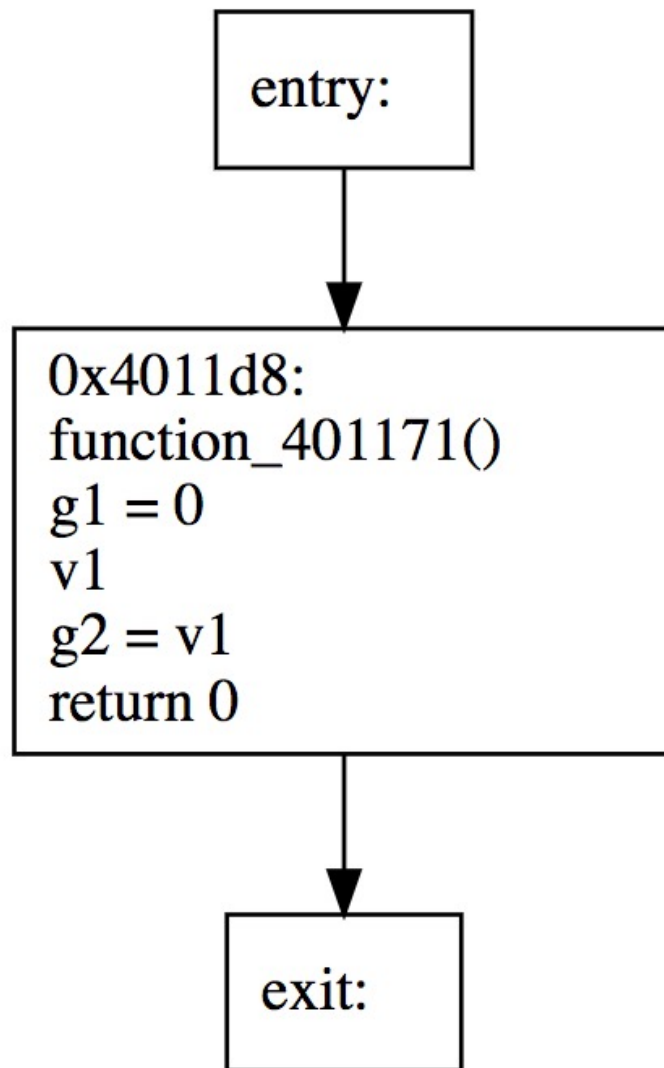
```

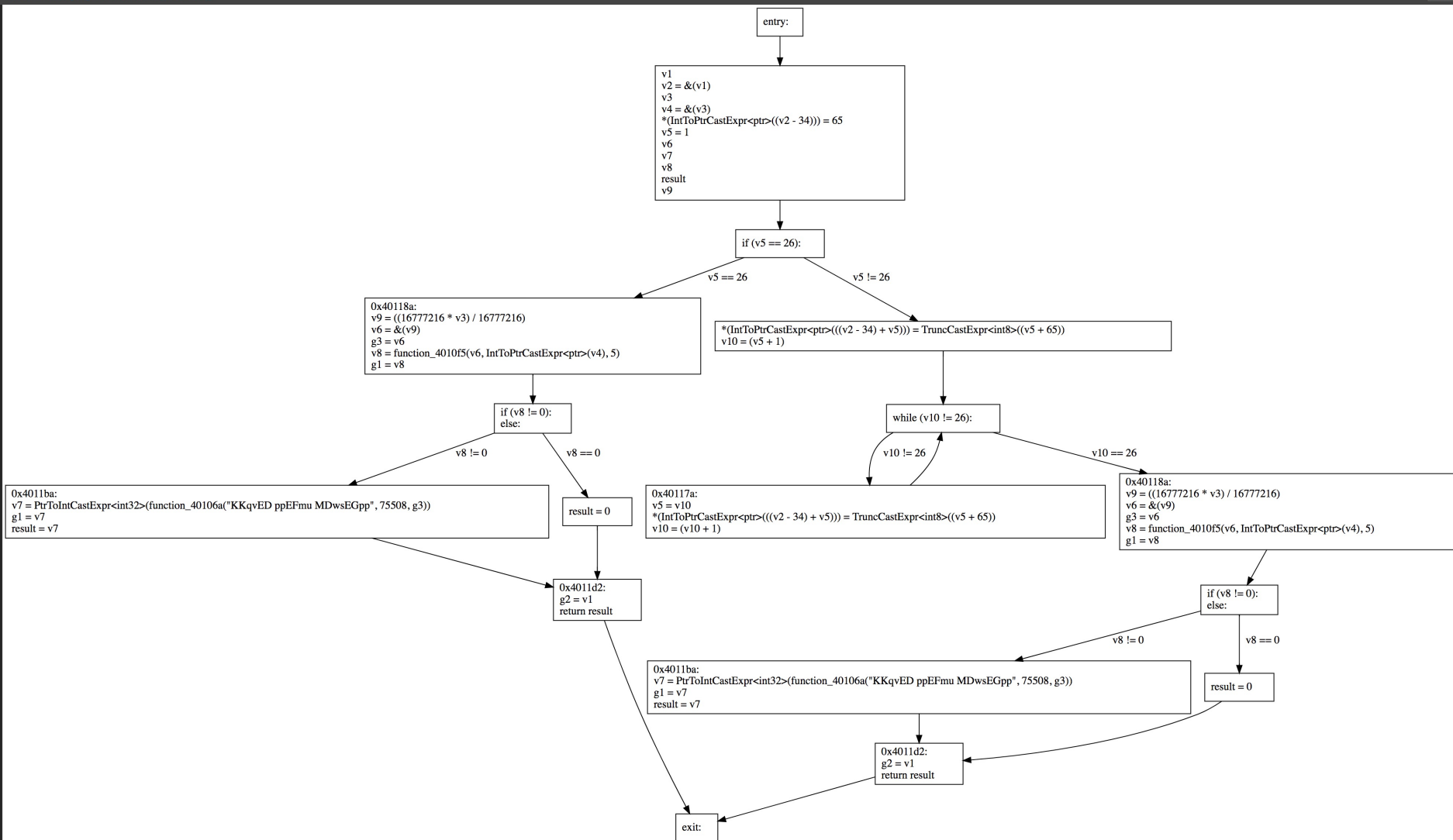
Call graph of the module.

Control-flow graph for entry_point()

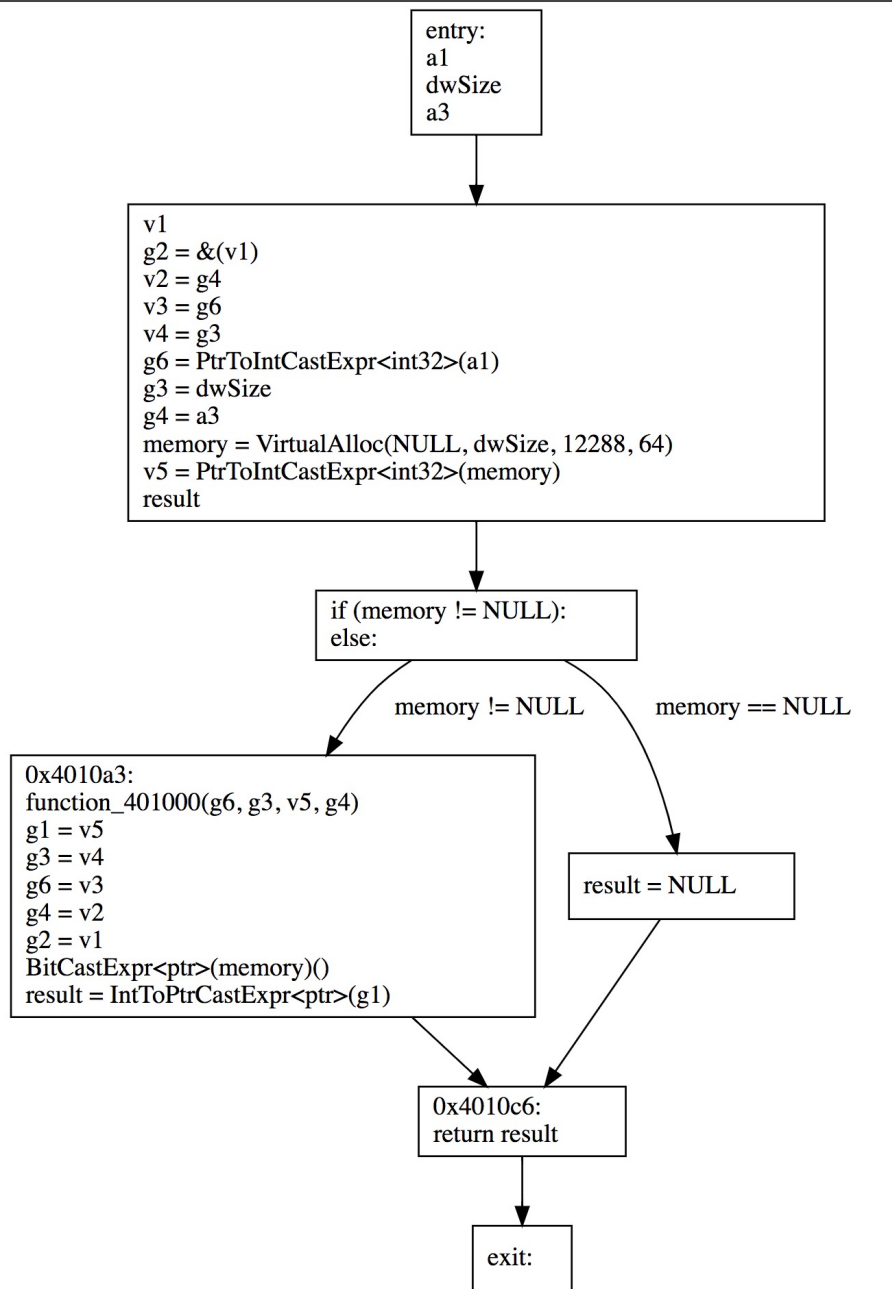
Close



Control-flow graph for function 'entry_point()'.



Control-flow graph for function 'function_401171()'.



Control-flow graph for function 'function_40106a(*a1, dwSize, a3)'.

Analysis Tool - Hopper

```

000000010000c30 db      "/System/Library/Frameworks/AppKit.framework/Versions/C/AppKit"
000000010000c6e db      0x00 ; '.'
000000010000c6f db      0x00 ; '.'
000000010000c70 db      3136 dup (0x00)

; Section _text
; Range: [0x1000018b0; 0x10002e440] (183184 bytes)
; File offset : [6320; 189504] (183184 bytes)
; Flags: 0x80000400
; S_REGULAR
; S_ATTR_PURE_INSTRUCTIONS
; S_ATTR_SOME_INSTRUCTIONS

; ===== BEGINNING OF PROCEDURE =====

start:
00000001000018b0 6A00      push     0x0 ; DATA XREF=0x1000009d0
00000001000018b2 4889E5    mov     rbp, rsp
00000001000018b5 4883E4F0  and     rsp, 0xfffffffffffffff0
00000001000018b9 488B7008  mov     rdi, qword [ss:rbp+8]
00000001000018bd 488D7510  lea    rsi, qword [ss:rbp+0x10]
00000001000018c1 89FA     mov     edx, edi
00000001000018c3 83C201    add     edx, 0x1
00000001000018c6 C1E203    shl     edx, 0x3
00000001000018c9 4881F2    add     rdx, rsi
00000001000018cc 4889D1    mov     rcx, rdx
00000001000018cf EB04     jmp     loc_1000018d5

loc_1000018d1:
00000001000018d1 4883C108  add     rcx, 0x8 ; CODE XREF=start+41

loc_1000018d5:
00000001000018d5 48833900  cmp     qword [ds:rcx], 0x0 ; CODE XREF=start+31
00000001000018d9 75F6     jne     loc_1000018d1

00000001000018db 4883C108  add     rcx, 0x8
00000001000018df E80C000000 call   __main
00000001000018e4 89C7     mov     edi, eax ; argument "status" for method
00000001000018e6 E8C5D02000 call   imp__stubs_exit
; endp

00000001000018eb F4     hlt
00000001000018ec db      4 dup (0x90)

; ===== BEGINNING OF PROCEDURE =====

; Variables:
; var_4: -4
; var_8: -8
; var_10: -16

__main:
> analysis section __const
> analysis section __unwind_info
> analysis section __eh_frame
Analysis segment __DATA
> analysis section __program_vars
> analysis section __nl_symbol_ptr
>>> Python Command

```

➔ Commercial debugger, disassembler, and decompiler
➔ \$99

➔ Scriptable

➔ Specializes in Objective-C and Swift (common Mac languages)

➔ Cross platform (Mac, Linux)

Analysis Tool – Radare2

The screenshot displays the Radare2 interface with the following components:

- Functions List:** A list of functions on the left, with 'entry0' selected.
- Disassembler:** The main window showing assembly instructions. The instruction at address 0x4047bf is highlighted: `mov rax, qword [rsp + 0x40]`. Other instructions include `mov rax, qword [rip + 0x218304]`, `mov rdi, qword [rsp + 0x28]`, `lea rsi, qword [rsp + 0x38]`, `mov edx, 1`, `mov rcx, r13`, `add qword [rsp + 0x38], 1`, `mov qword [rip + 0x2182e5], r13`, `mov qword [r13 + 0x20], rax`, `mov qword [r13 + 8], rax`, `call 0x404a70`, `cmp al, 1`, `shb edx, edx`, `and edx, 2`, `add edx, 3`, `jmp 0x404362`, `mov rax, qword [rsp + 0x40]`, `movsxd rcx, r14d`, `mov rdi, qword [rsp + 0x28]`, `shl rcx`, `lea rsi, qword [rsp + 0x38]`, `xor edx, edx`, `add rcx, 0x61bc80`, `mov qword [rcx + 8], rax`, `call 0x404a70`, `xor edx, edx`, `test al, al`, `jne 0x40436b`, `lea rdi, qword [rsp + 0xf0]`, `call 0x40eaa0`, `xor edi, edi`, and `mov r14, rax`.
- Hex Dump:** A column showing the raw bytes of the instructions.
- Registers:** A window at the bottom showing the state of registers: `r15 0x00000000`, `r14 0x00000000`, `r13 0x00000000`, `r12 0x00000000`, `rbp 0x00000000`, `rbx 0x00000000`, `r11 0x00000000`, `r10 0x00000000`, `r9 0x00000000`, `r8 0x00000000`, `rax 0x00000000`, `rcx 0x00000000`, `rdx 0x00000000`, `rsi 0x00000000`, `rdi 0x00000000`, `orax 0x00000000`, `rip 0x00000000`, and `rflags =`.
- Information Panel:** A sidebar on the right with various analysis options like 'file', 'type', 'pic', 'canary', 'nx', 'crypto', 'va', 'root', 'class', 'lang', 'arch', 'bits', 'machine', 'os', 'subsys', 'endian', 'strip', 'static', 'linenum', 'lsyms', 'relocs', 'rpath', 'type', 'os', 'arch', 'bits', 'endian', 'file', 'fd', 'size', 'mode', etc.

- ➔ Free debugger / disassembler
- ➔ Cross platform (Linux, Mac, Windows, Android, ...)
- ➔ Command-line core with optional GUI wrappers - scriptable!
- ➔ Supports wide variety of executable file formats and processor architectures
- ➔ Open source ([GitHub](https://github.com/radareorg/radare2)) and under active development

Analysis Tool – Cutter

Free GUI by same group as Radare

Decompiler

Integrated Ghidra

Debugger / Disassembler

Integrated Radare2

Cross platform

Linux, Windows, Mac

Scriptable / extensible

Open source ([GitHub](https://github.com)) and under active development

Analysis Tools - Misc

- **OllyDbg** – <http://www.ollydbg.de/>
 - Free debugger / disassembler
 - Platform: Windows
 - Development stalled
 - 32-bit version last updated 2013 (stable/functional)
 - Incomplete 64-bit version last updated 2014

- **Windbg**
 - Free debugger / disassembler from Microsoft
 - Platform: Windows
 - Often invoked via plugins from other disassemblers
 - <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/>

Next class: Using
x64dbg and IDA
Pro to dig deep
into malware