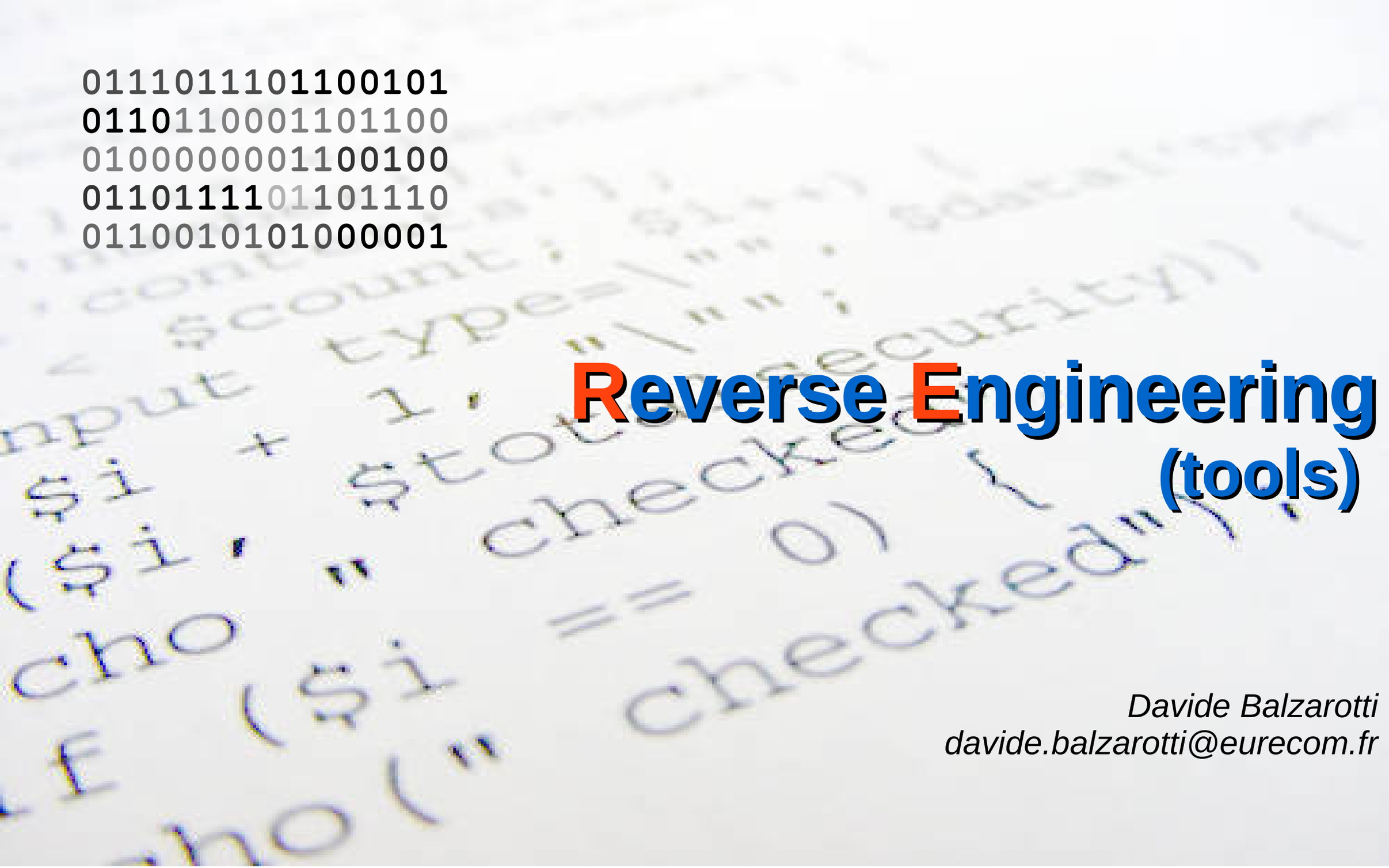
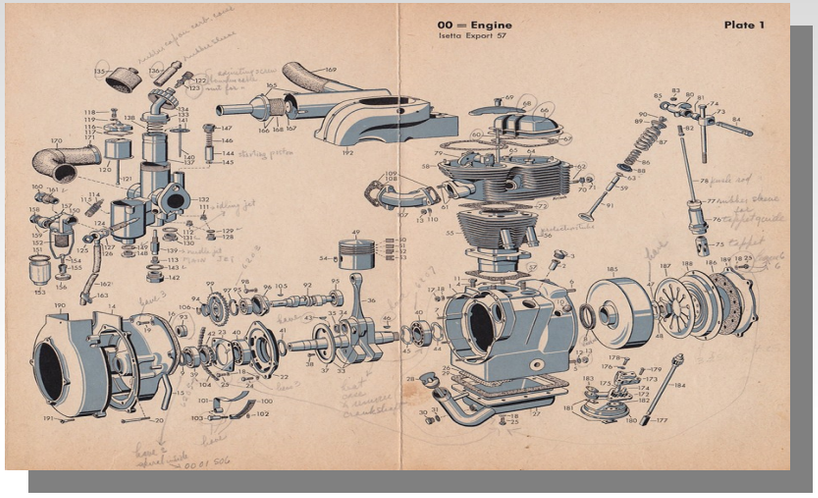


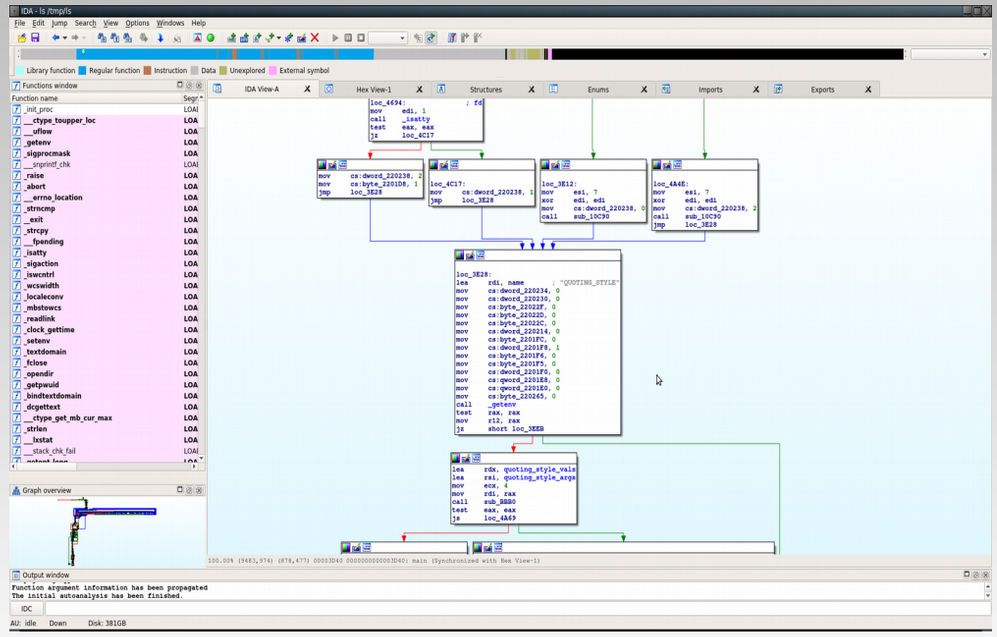
Reverse Engineering (tools)

Davide Balzarotti
davide.balzarotti@eurecom.fr

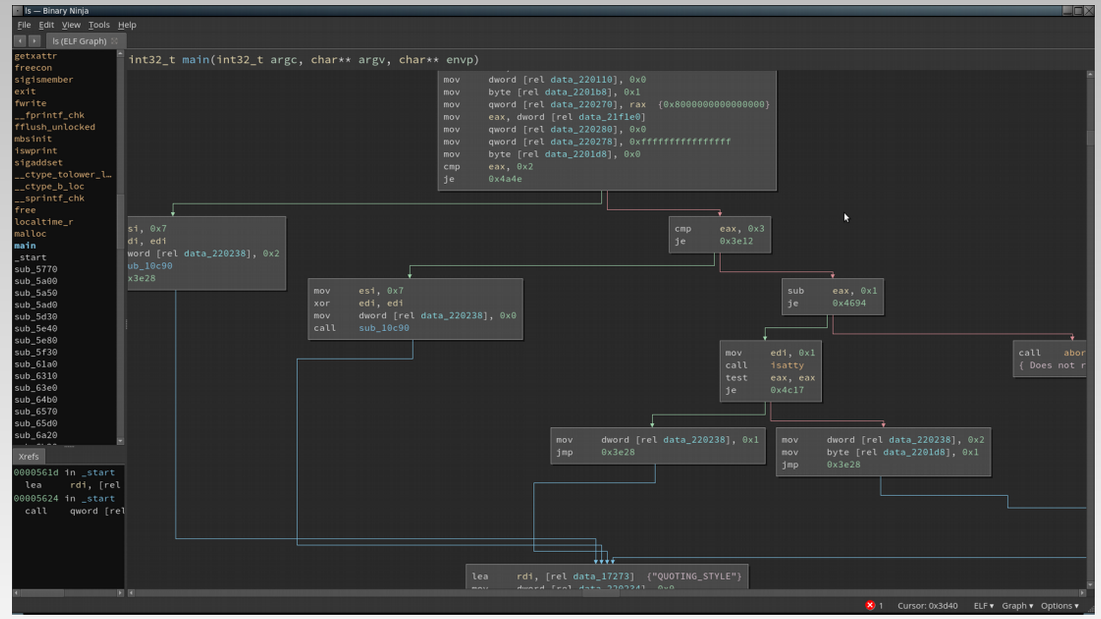


- ✔ Assembly 101
- ✔ Disassembly algorithms
  - → Linear sweep and recursive
  - → Detecting function prologues
- ✔ Decompilation
- ✔ Language Constructs
  - ✔ Assembly and C
  - ✔ Assembly and C++
- ✔ Limits of Static Analysis
  - ✔ General limitations
  - ✔ Anti-disassembly
  - ✔ Packing

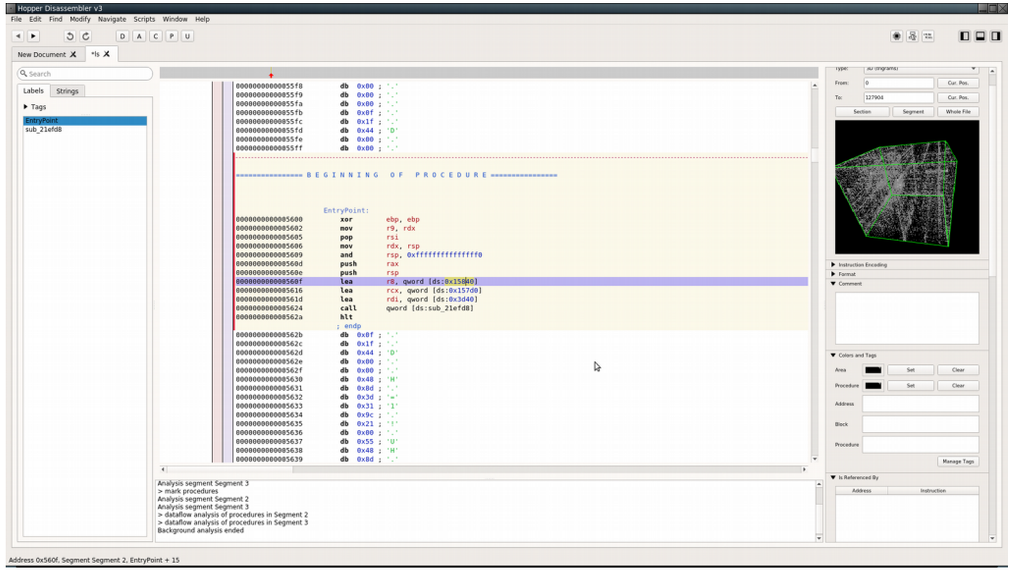
Ida Pro



Binary Ninja



Hopper

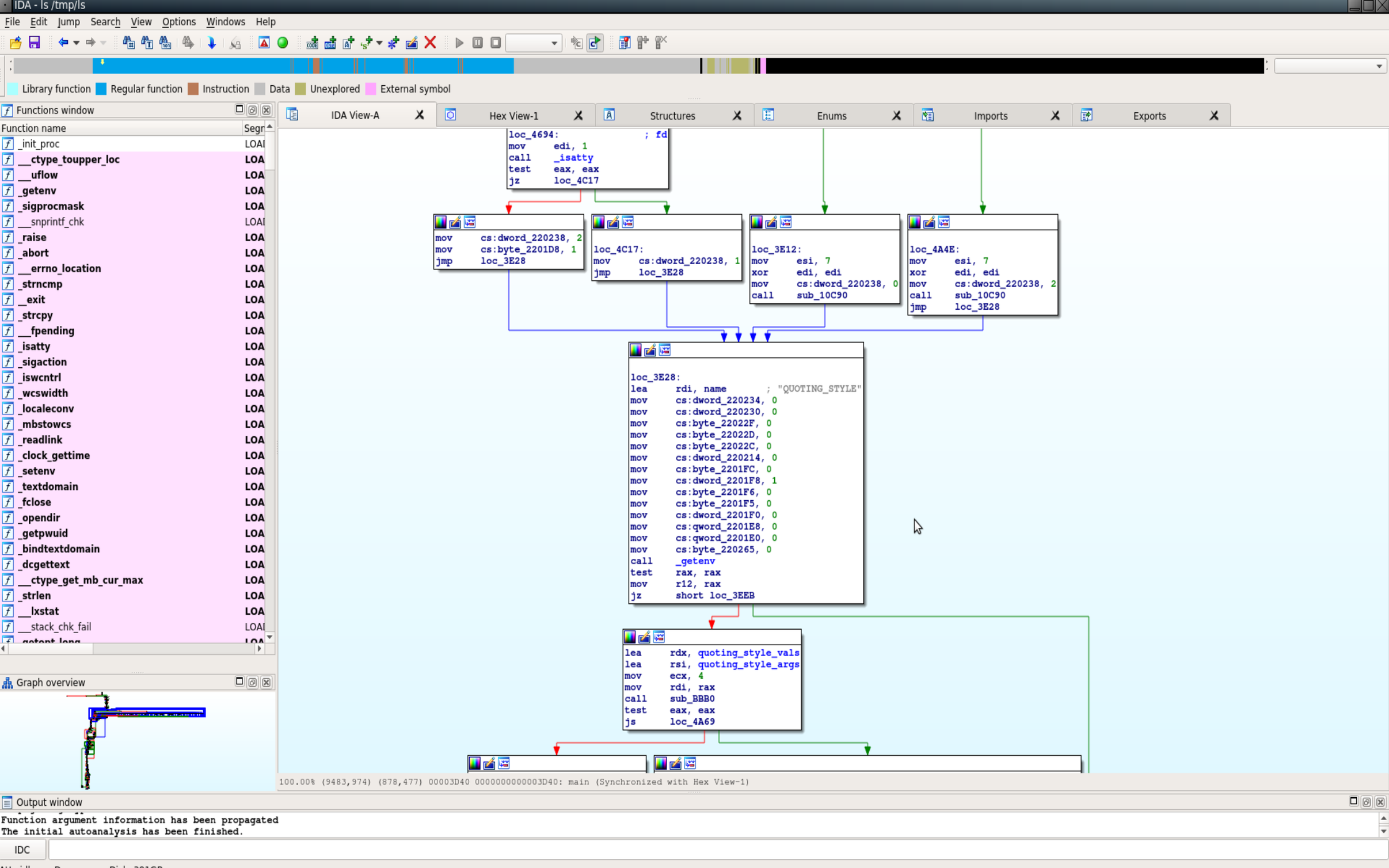


Radare 2

IDA Pro

IDA

- The Interactive Disassembler (IDA) Pro is the best available tool for analyzing program binaries. It provides:
  - an interactive, programmable, extensible, multi-processor, recursive traversal disassembler
  - a graphical interface with several code visualization tools
  - support for more than 60 families of processors and more than 30 file types
- Commercial application
  - One seat license costs 529 / 1019E (and you need to choose one OS)
  - Version 7.0 (with limited features) is free for non-commercial use
  - Evaluation version: only PE & ELF on X86 and ARM
    (you cannot save your work and it will time out after some use)


IDA - ls /tmp/ls
File Edit Jump Search View Options Windows Help
Library function Regular function Instruction Data Unexplored External symbol
Functions window
Function name Segr
_init_proc LOAI
__ctype_toupper_loc LOA
__uflow LOA
_getenv LOA
_sigprocmask LOA
__snprintf_chk LOAI
_raise LOA
_abort LOA
__errno_location LOA
_strncmp LOA
__exit LOA
_strcpy LOA
__fpending LOA
_isatty LOA
_sigaction LOA
_iswcntrl LOA
_wcswidth LOA
_localeconv LOA
_mbstowcs LOA
_readlink LOA
_clock_gettime LOA
_setenv LOA
_textdomain LOA
_fclose LOA
_opendir LOA
_getpwuid LOA
_bindtextdomain LOA
_dcgettext LOA
__ctype_get_mb_cur_max LOA
_strlen LOA
__lxstat LOA
__stack_chk_fail LOAI
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
loc_4694: ; fd
mov edi, 1
call _isatty
test eax, eax
jz loc_4C17
mov cs:dword_220238, 2
mov cs:byte_2201D8, 1
jmp loc_3E28
loc_4C17:
mov cs:dword_220238, 1
jmp loc_3E28
loc_3E12:
mov esi, 7
xor edi, edi
mov cs:dword_220238, 0
call sub_10C90
loc_4A4E:
mov esi, 7
xor edi, edi
mov cs:dword_220238, 2
call sub_10C90
jmp loc_3E28
loc_3E28:
lea rdi, name ; "QUOTING_STYLE"
mov cs:dword_220234, 0
mov cs:dword_220230, 0
mov cs:byte_22022F, 0
mov cs:byte_22022D, 0
mov cs:byte_22022C, 0
mov cs:dword_220214, 0
mov cs:byte_2201FC, 0
mov cs:dword_2201F8, 1
mov cs:byte_2201F6, 0
mov cs:byte_2201F5, 0
mov cs:dword_2201F0, 0
mov cs:qword_2201E8, 0
mov cs:qword_2201E0, 0
mov cs:byte_220265, 0
call _getenv
test rax, rax
mov r12, rax
jz short loc_3EEB
lea rdx, quoting_style_vals
lea rsi, quoting_style_args
mov ecx, 4
mov rdi, rax
call sub_BBB0
test eax, eax
js loc_4A69
100.00% (9483,974) (878,477) 00003D40 0000000000003D40: main (Synchronized with Hex View-1)
Output window
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC

IDA Database

- After IDA opens a file and performs its analysis, it saves all the information in a custom database, stored in four binary files
  - `program.id0` – the b-tree style database
    `program.id1` – flags for each program byte
    `program.nam` – info for IDA name window
    `program.til` – type definition
  - By default, the files are compressed and stored in a single `.idb` file when you quit the program
  - After the database is created, the original binary is useless

RULE #1: *Everything you see and you do, you are working on the database. Changes are NOT automatically reflected to the binary*

IDA Analysis

- Step I: Loader
  - Parse the file and load the required sections at the right virtual address
- Step II: Recursive disassembler
  - A processor module is used to interpret each instruction
  - A number of heuristics are applied to increase coverage
- Step III: Arguments and variable identification
  - IDA tracks the value of the ESP register inside each function, to properly locate and rename each access to local variables and parameters
- Step IV: Datatype analysis
  - Based on the knowledge of the parameters of common library calls, IDA adds comments to the code containing the parameter name

Names & References

- IDA automatically generates a lot of symbolic names
  - `sub_xxxx` – a function at address xxxx
  - `loc_xxxx` – an instruction at address xxxx
  - `byte_ word_ dword_xxxx` – data at address xxxx
  - `var_xx` – local variable at offset EBP-xx
  - `arg_n` – function argument at offset EBP+8+n

- IDA also keeps track of all code and data references
  - Code xrefs model the fact that an instruction is the target of a jmp or call
  - Data xrefs models the fact that a byte is read, written, or referenced by another instruction

Make no mistakes

RULE #2: *There are plenty of things you can do.*
But there is no way to “undo” an operation.

- One change can trigger many other ones behind the scene
  - From (most of the) changes, it is possible to *manually* change things back
  - But it can be a painful and long job
- The undo feature will not be implemented in the near future :(
  - But starting from version 6.2, it is possible to save and restore hierarchical snapshots of the database

Demo

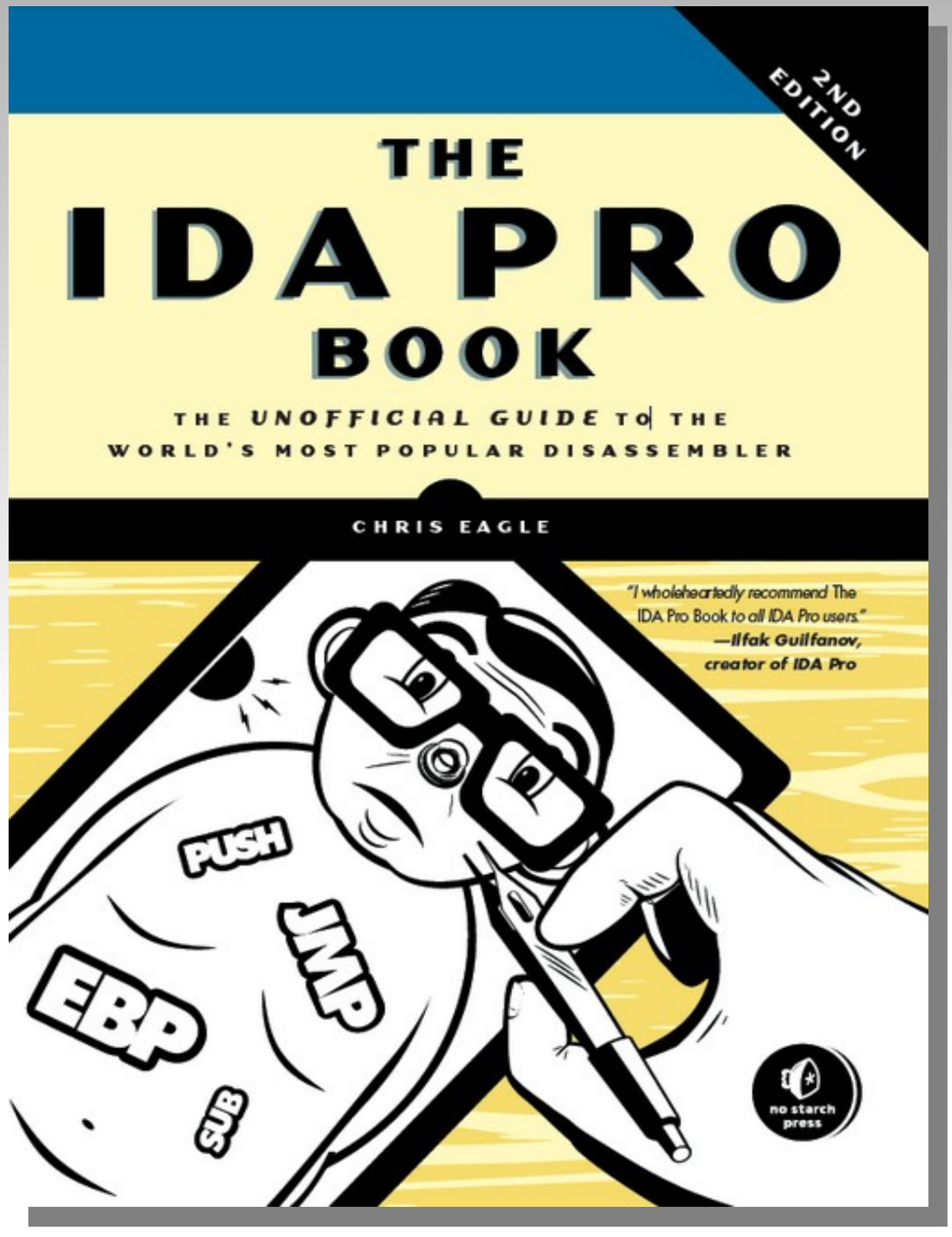
Scripting IDA

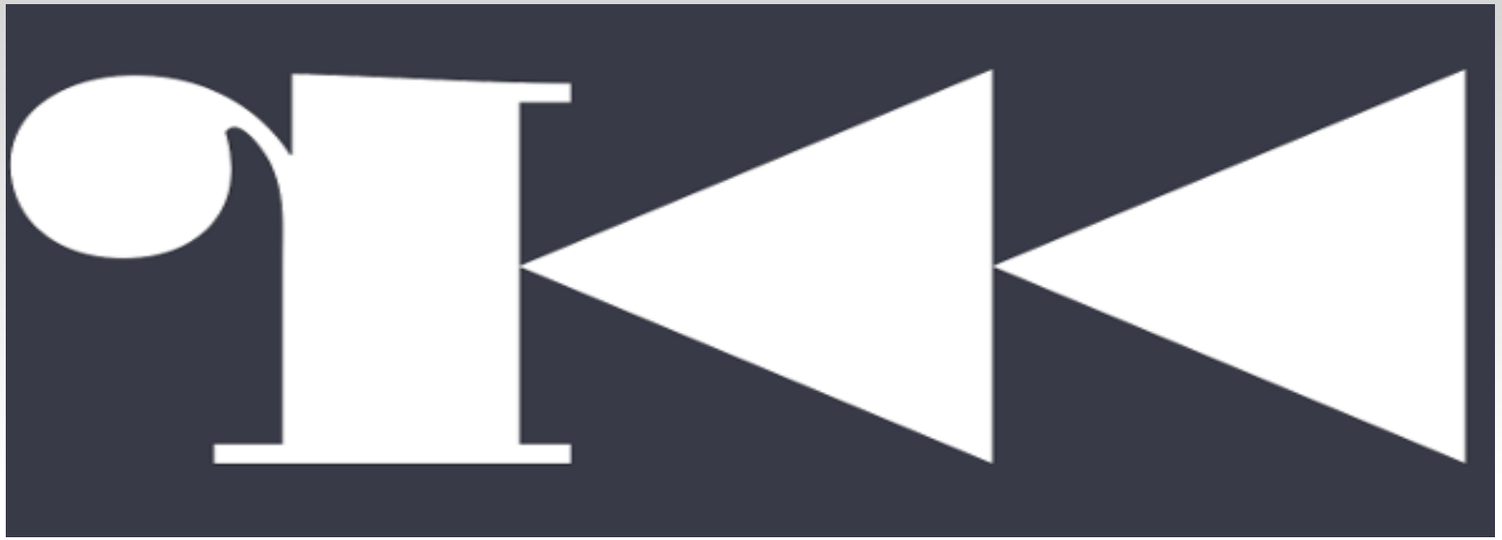
- Two Options:
  IDC scripting language (syntax similar to C) or Python

```
import idc, idaapi, re
from idautils import *

def find_functions_headers(start,end):
    data = idc.GetManyBytes(start,end-start)
    if not data:
        return []
    return  [m.start()+start for m in
                re.finditer("\x55\x89\xe5\x83\xec", data)]

for s in Segments():
    for p in find_functions_headers(SegStart(s), SegEnd(s)):
        if p not in Functions(SegStart(s), SegEnd(s)) :
            print ">>  POSSIBLE Undetected Function at 0x%x <<"%p
```

IDA from the Command Line

- `$ idaq -B file`
  - Run the default analysis and create a text file with the entire disassembly
- `$ idaq -A -c -S"script_full_path" file`
  - Run a given script on the target file
  - Scripts cannot print to the standard output, so the messages need to be redirected to a file


2ND EDITION
THE
IDA PRO
BOOK
THE UNOFFICIAL GUIDE TO THE
WORLD'S MOST POPULAR DISASSEMBLER
CHRIS EAGLE
"I wholeheartedly recommend The IDA Pro Book to all IDA Pro users."
—Ilfak Guilfanov,
creator of IDA Pro
PUSH
JMP
EBP
SUB
no starch press



Radare2

Radare

- Command-line, VIM-like binary analysis framework
- Free and open source (always install from `GIT`)
- Suite of tools:
  - **`radare2`**
  - `r2pm`
  - `rarun2`
  - `ragg2`
  - **`rabin2`**
  - `radiff2`
  - `rax2`
  - `rahash2`
  - **`rasm2`**
  - `rafind2`
  - `r2agent`
  - `rasign2`

Radare

- Command-line, VIM-like binary analysis framework
- Free and open source (always install from `GIT`)
- Suite of tools:
  - **`radare2`**
  - `r2pm`
  - `rarun2`
  - `ragg2`
  - **`rabin2`**
  - `radiff2`
  - `rax2`
  - `rahash2`
  - **`rasm2`**
  - `rafind2`

Support for binary format

```
> rabin2 -L
```

Radare

- Command-line, sort of VIM-like binary analysis framework
- Free and open source (always install from `GIT`)
- Suite of tools:
  - **`radare2`**
  - `r2pm`
  - `rarun2`
  - `ragg2`
  - **`rabin2`**
  - `rax2`
  - `rahash2`
  - **`rasm2`**
  - `rafind2`
  - `r2agent`
  - `gn2`

Disassembler and binary analyzer

```
> rasm2 -L
```

Radare2

- Command-line, sort of VIM-like binary analysis framework
  - Commands are sequences of characters, in which each one as a special meaning
  - A question mark after a command prints a short help message

```
p   // [p]rint command group (alone does not do anything)
p?  // [p]rint the list of commands in the print group
pd  // [p]rint [d]isassemby
pd? // [p]rint the list of commands in the print disassembly group
pdf // [p]rint [d]isassemby [f]unctions
```

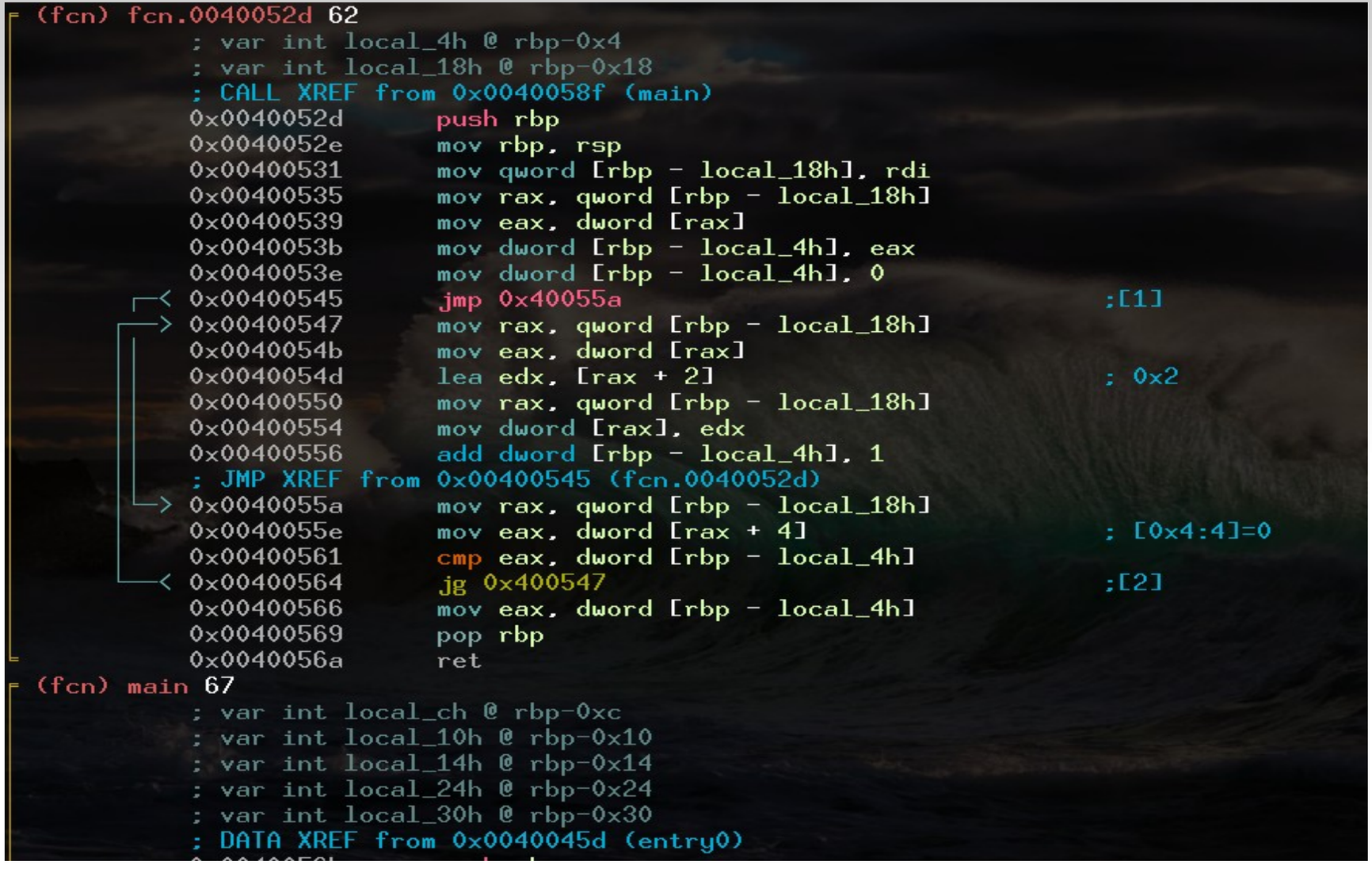
Radare2

- You can use pipes and redirect

```
afl > /tmp/flist  // write the list of functions to a file

afl | grep "foo"  // grep over the functions list

afl ~ foo         // same as above, but using the
                     embedded grep (for portability)
```

- The learning curve is steep
  - There are tons of commands, and new important commits almost every day
  - Several commands may not work (or not as expected)
- It supports many possible ways to visualize the data
  (visual mode, graph mode, web interface, ...)

Visual Mode

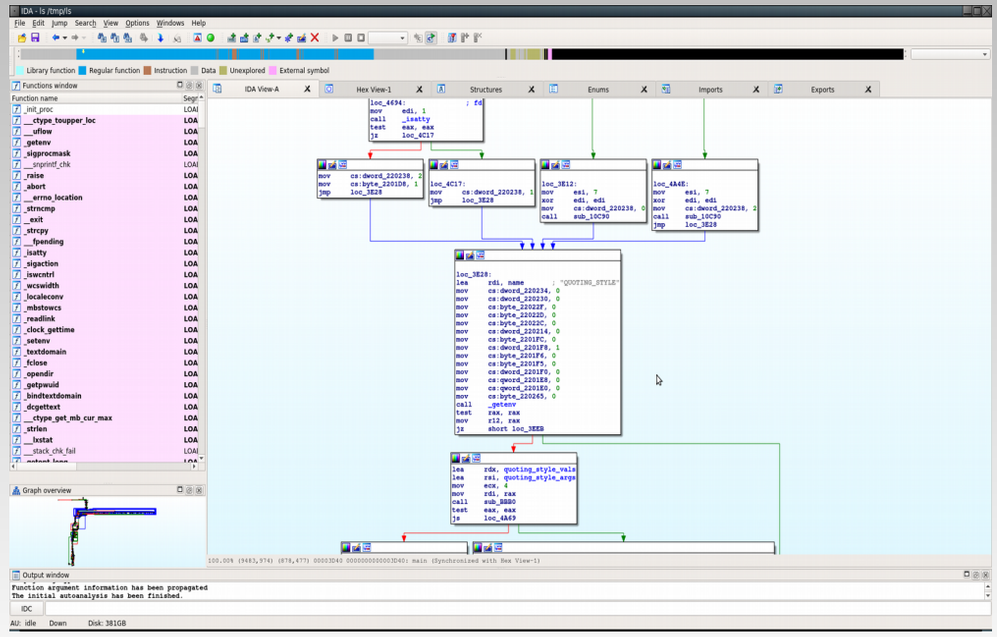

(fcn) fcn.0040052d 62
; var int local_4h @ rbp-0x4
; var int local_18h @ rbp-0x18
; CALL XREF from 0x0040058f (main)
0x0040052d push rbp
0x0040052e mov rbp, rsp
0x00400531 mov qword [rbp - local_18h], rdi
0x00400535 mov rax, qword [rbp - local_18h]
0x00400539 mov eax, dword [rax]
0x0040053b mov dword [rbp - local_4h], eax
0x0040053e mov dword [rbp - local_4h], 0
0x00400545 jmp 0x40055a ;[1]
0x00400547 mov rax, qword [rbp - local_18h]
0x0040054b mov eax, dword [rax]
0x0040054d lea edx, [rax + 2] ; 0x2
0x00400550 mov rax, qword [rbp - local_18h]
0x00400554 mov dword [rax], edx
0x00400556 add dword [rbp - local_4h], 1
; JMP XREF from 0x00400545 (fcn.0040052d)
0x0040055a mov rax, qword [rbp - local_18h]
0x0040055e mov eax, dword [rax + 4] ; [0x4:4]=0
0x00400561 cmp eax, dword [rbp - local_4h]
0x00400564 jg 0x400547 ;[2]
0x00400566 mov eax, dword [rbp - local_4h]
0x00400569 pop rbp
0x0040056a ret
(fcn) main 67
; var int local_ch @ rbp-0xc
; var int local_10h @ rbp-0x10
; var int local_14h @ rbp-0x14
; var int local_24h @ rbp-0x24
; var int local_30h @ rbp-0x30
; DATA XREF from 0x0040045d (entry0)

Graph Mode

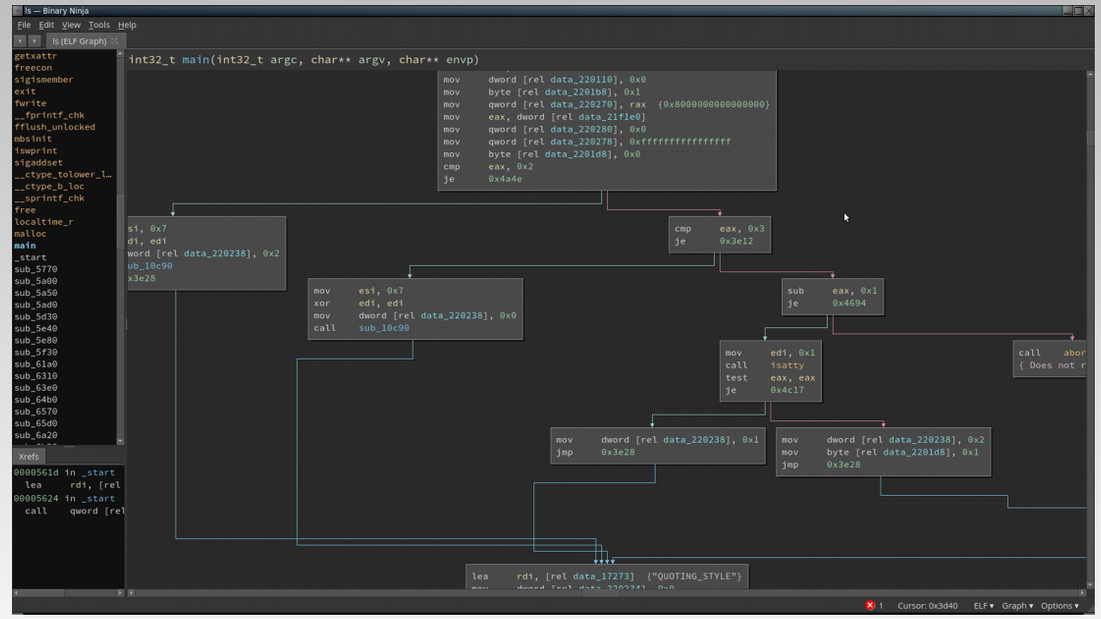

[0x40052d]
(fcn) fcn.0040052d 62
; var int local_4h @ rbp-0x4
; var int local_18h @ rbp-0x1
push rbp
mov rbp, rsp
mov qword [rbp - local_18h],
mov rax, qword [rbp - local_1
mov eax, dword [rax]
...
0x40055a
mov rax, qword [rbp - local_
mov eax, dword [rax + 4]
...
t f
0x400547
mov rax, qword [rbp - local_1
mov eax, dword [rax]
lea edx, [rax + 2]
mov rax, qword [rbp - local_1
...
0x400566
mov eax, dword [rbp - local
...

Demo

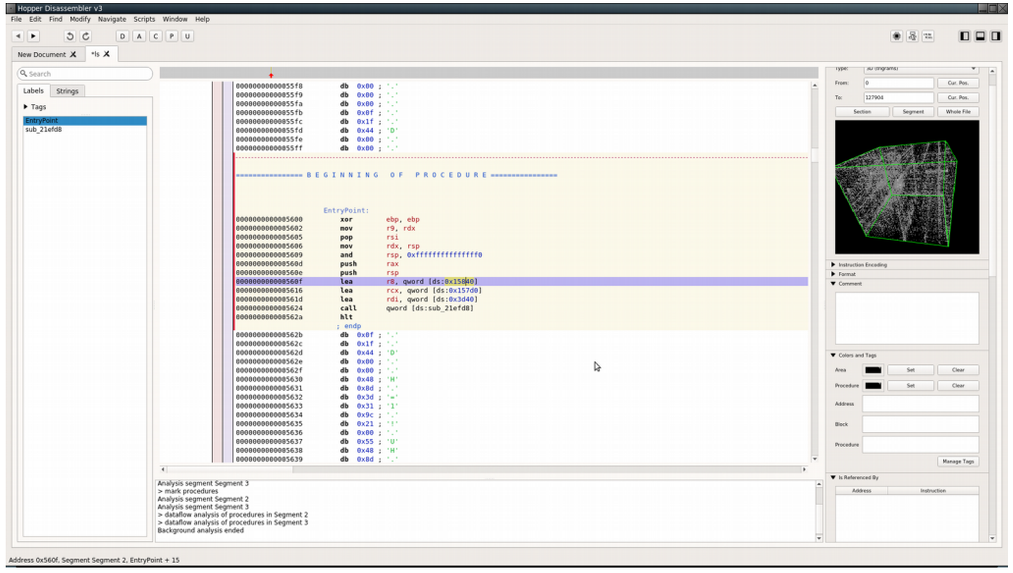
Ida Pro



Binary Ninja



Hopper

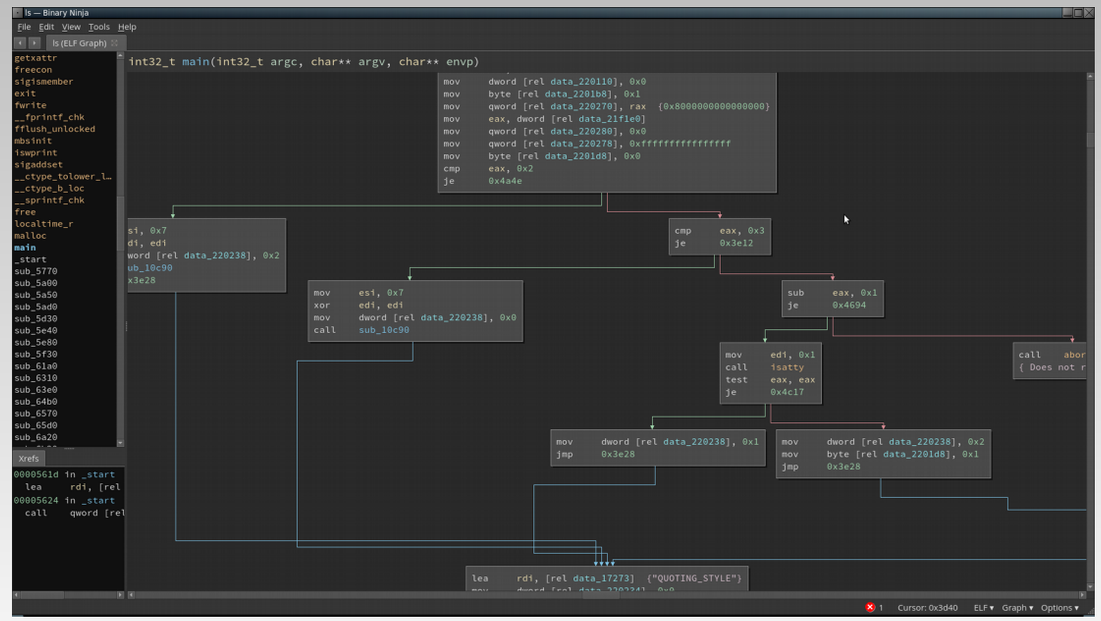


Radare 2

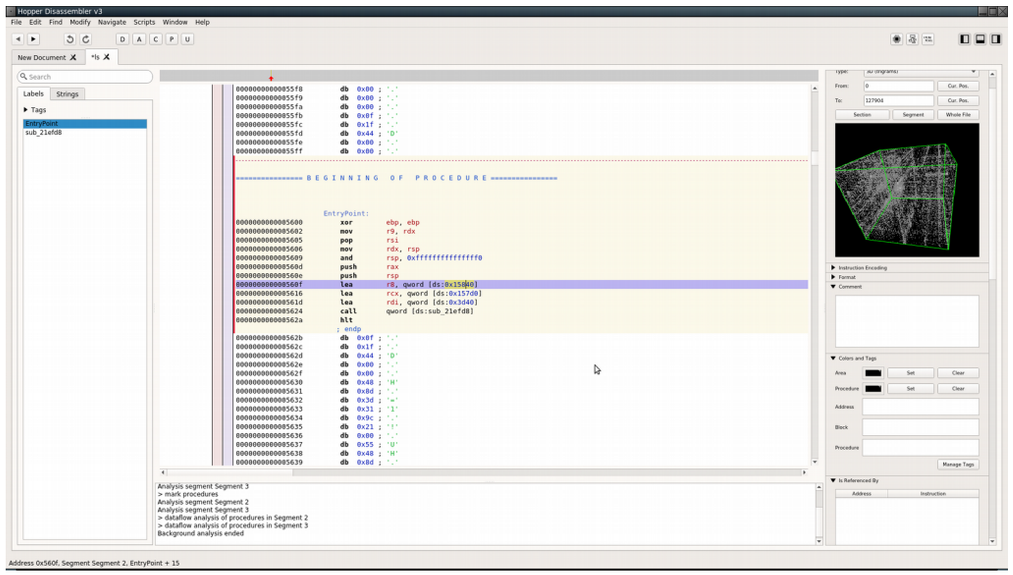
Ida Pro

+ De-facto standard & still the best
+ Best decompiler
+ Free version
- Full version very expensive

Binary Ninja



Hopper



Radare 2

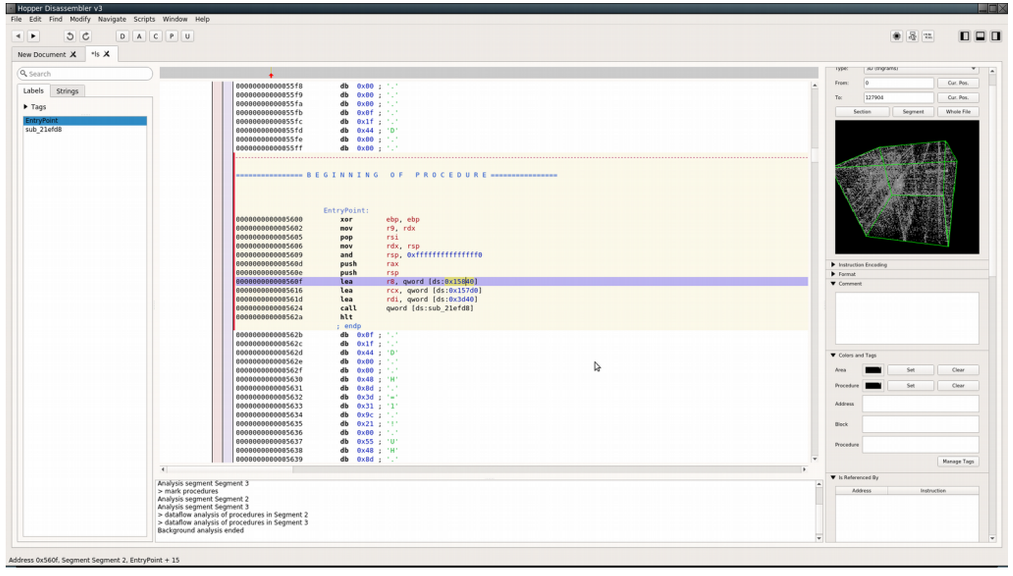
Ida Pro

+ De-facto standard & still the best
+ Best decompiler
+ Free version
- Full version very expensive

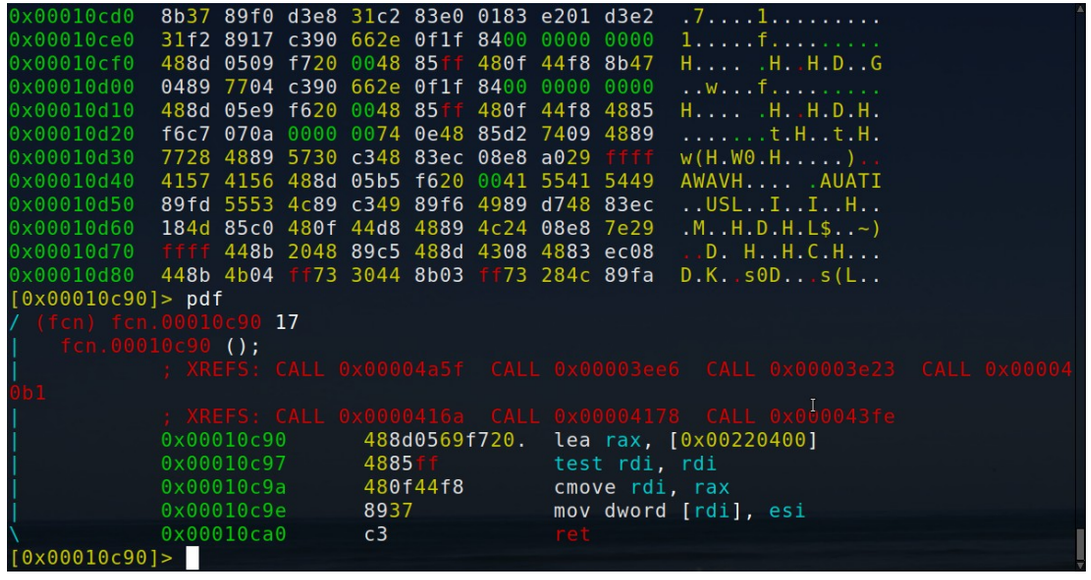
Binary Ninja

+ Personal license 149$
+ Great usability
- No debugging support

Hopper



Radare 2



Ida Pro

+ De-facto standard & still the best
+ Best decompiler
+ Free version
- Full version very expensive

Binary Ninja

+ Personal license 149$
+ Great usability
- No debugging support

Hopper

+ Affordable (99Euro) decompiler
- CFG non-interactive view

Radare 2

Ida Pro

+ De-facto standard & still the best
+ Best decompiler
+ Free version
- Full version very expensive

Binary Ninja

+ Personal license 149$
+ Great usability
- No debugging support

Hopper

+ Affordable (99Euro) decompiler
- CFG non-interactive view

Radare 2

+ Free and open source
+ 1000 functions...
- ..100 of which do not work
- Still cant find an efficient workflow