


# The beheading of Madame de Maintenon

0x41con 2024

# About me

@jonpalmisc

- Florida man
- Low level & security enjoyer
  - (De)compilers, operating systems, software cracking, ...
- Also interested in: electric guitar, typography, probably other things ...
- Present: iOS shenanigans at 
- Past: Binary Ninja developer at Vector 35

# Before we begin

## What's this talk?

- Florida man beheads beloved disassembler mascot
- This is a brief recount of the journey

# Before we begin

## Why do this?

- I thought it would be funny
- EOF

# Before we begin

## Why do this? (actually)

- IDA's headless mode isn't truly headless
- Limited to what Hex-Rays allows you to do
- TL;DR: I was annoyed you can't use IDA as a library

# Before we begin

## The goal

- Analyze files headlessly
- Standalone IDAPython REPL
- Use IDA more or less as a library
- Avoid “cheating” or lame hacks
- Freedom from Hex-Rays

# The dirty work

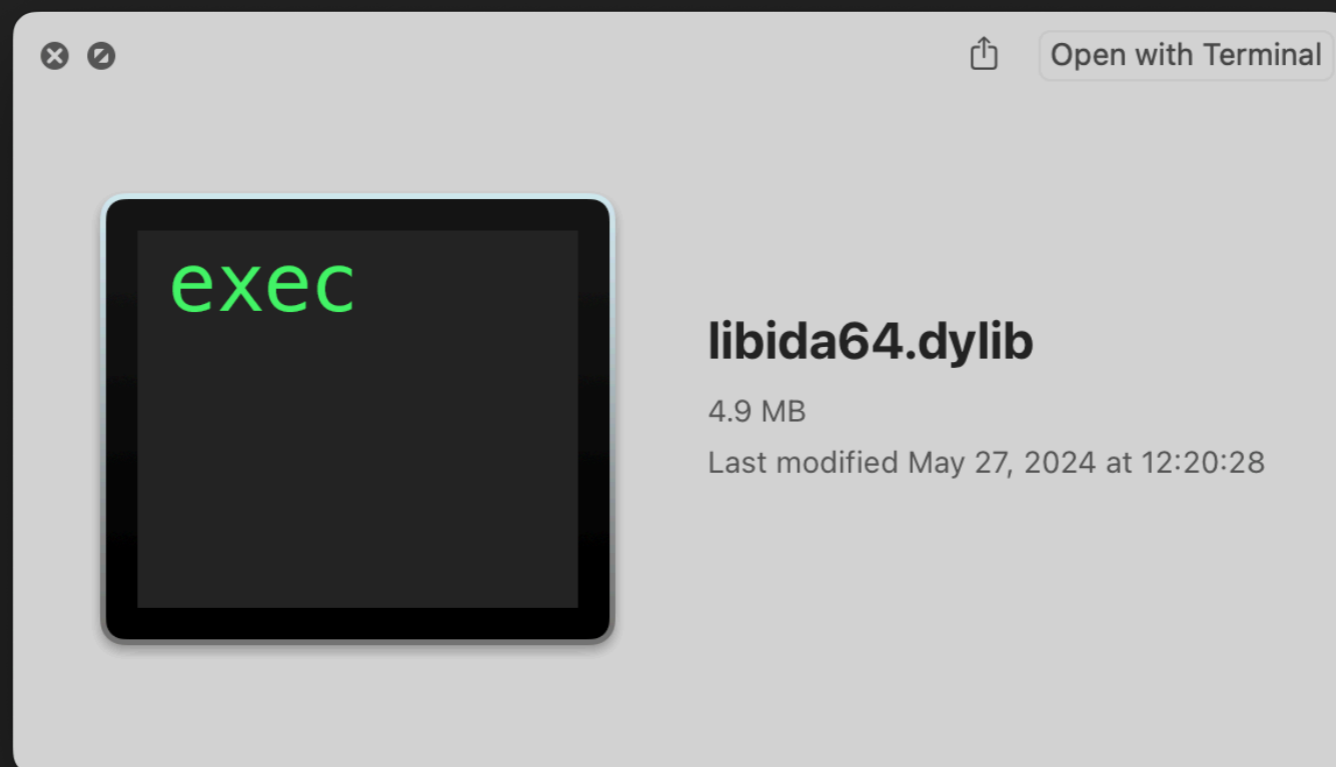
## Project timeline

- Sometime 2022: First attempt, got a half-working prototype
- February 2024: Second attempt, it works?
- May 2024: “Headless IDA eta son” — Hex-Rays

# The dirty work

## The big question

- Knew IDA's analysis is implemented in a shared library
  - Not all functions documented in SDK
- Can I make headless out of this?

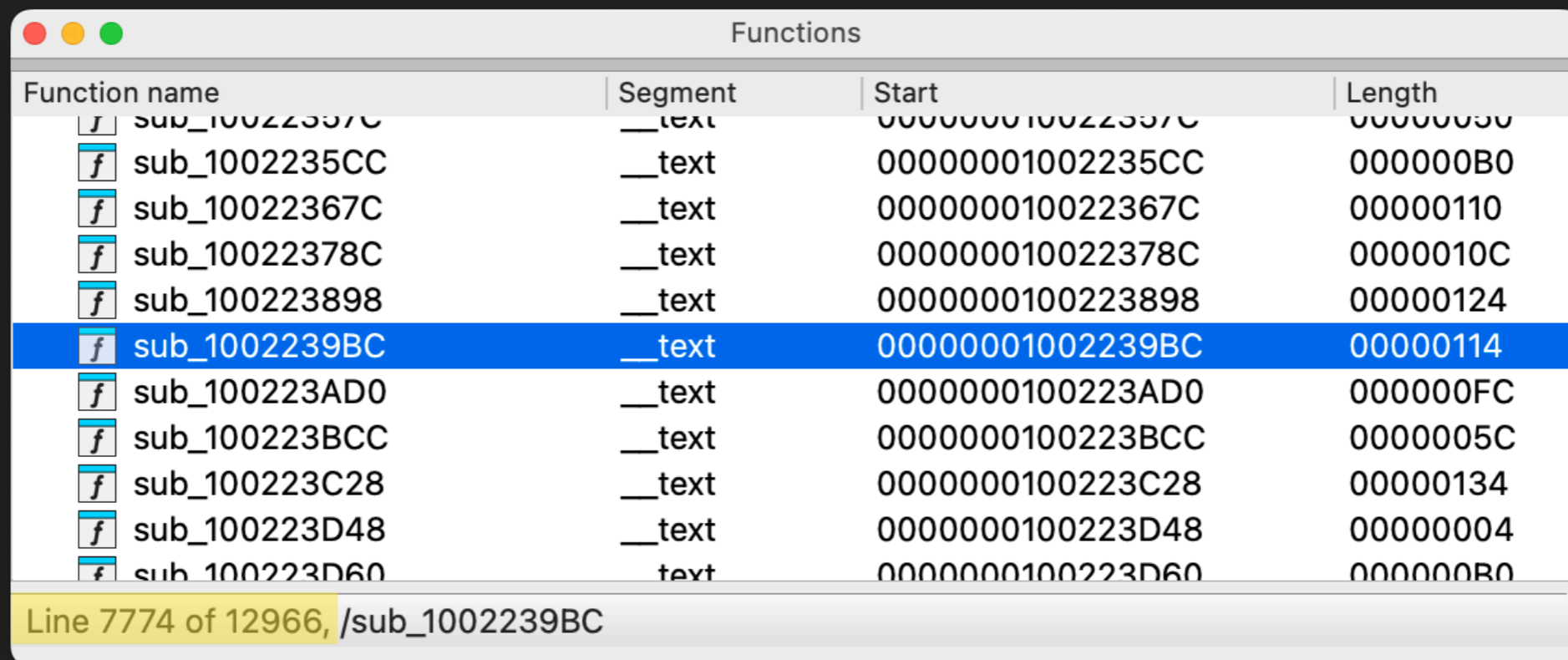




# The dirty work

## A moment of self-reflection

- open `ida64 -a ida64`
  - Some progress made, still lots of code to look at...



The screenshot shows the 'Functions' window in IDA Pro. It displays a table of functions with columns for 'Function name', 'Segment', 'Start', and 'Length'. The function `sub_1002239BC` is highlighted in blue. Below the table, the status bar shows 'Line 7774 of 12966, /sub\_1002239BC'.

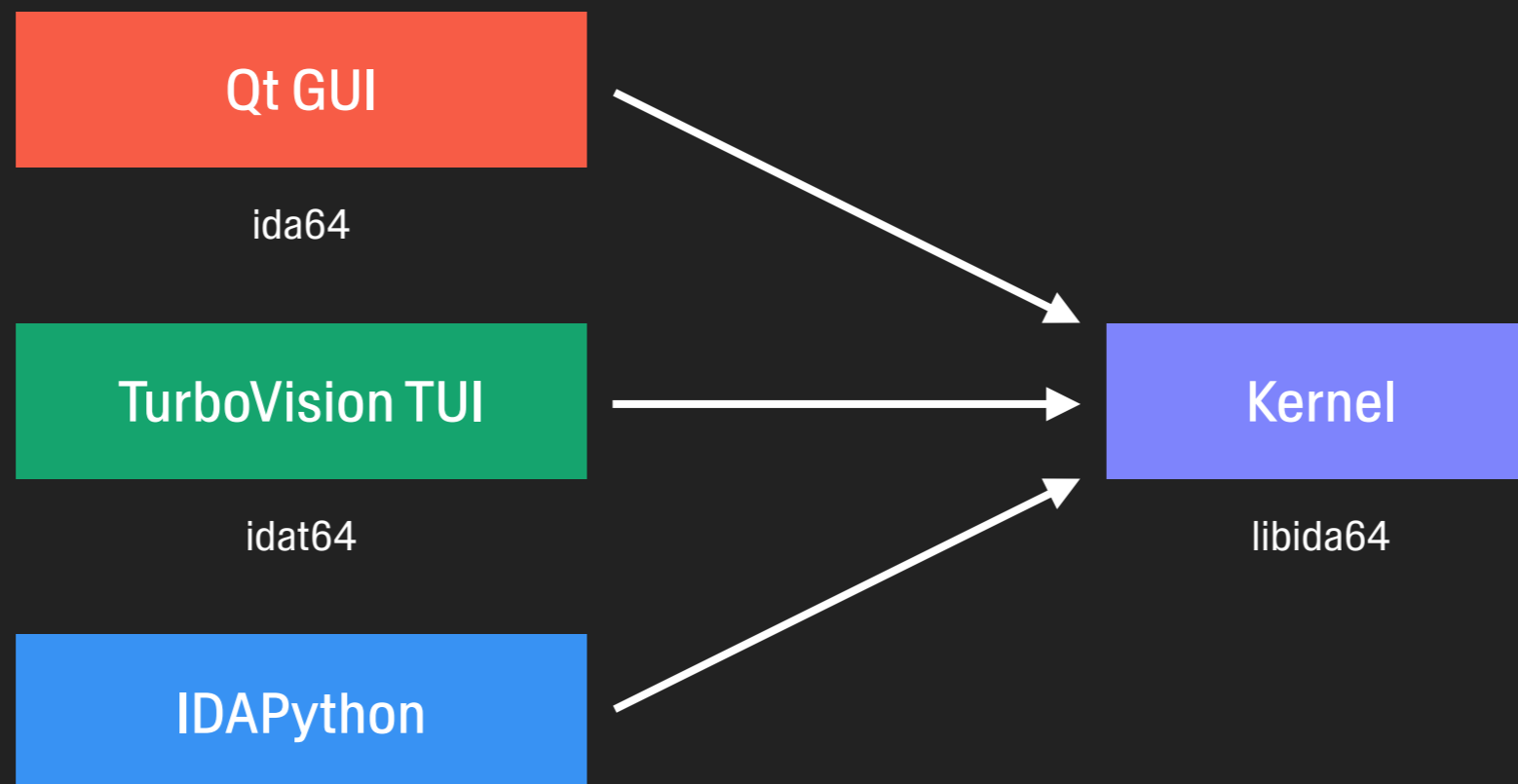
Function name	Segment	Start	Length
<code>sub_10022357C</code>	<code>__text</code>	<code>000000010022357C</code>	<code>00000030</code>
<code>sub_1002235CC</code>	<code>__text</code>	<code>00000001002235CC</code>	<code>000000B0</code>
<code>sub_10022367C</code>	<code>__text</code>	<code>000000010022367C</code>	<code>00000110</code>
<code>sub_10022378C</code>	<code>__text</code>	<code>000000010022378C</code>	<code>0000010C</code>
<code>sub_100223898</code>	<code>__text</code>	<code>0000000100223898</code>	<code>00000124</code>
<code>sub_1002239BC</code>	<code>__text</code>	<code>00000001002239BC</code>	<code>00000114</code>
<code>sub_100223AD0</code>	<code>__text</code>	<code>0000000100223AD0</code>	<code>000000FC</code>
<code>sub_100223BCC</code>	<code>__text</code>	<code>0000000100223BCC</code>	<code>0000005C</code>
<code>sub_100223C28</code>	<code>__text</code>	<code>0000000100223C28</code>	<code>00000134</code>
<code>sub_100223D48</code>	<code>__text</code>	<code>0000000100223D48</code>	<code>00000004</code>
<code>sub_100223D60</code>	<code>text</code>	<code>0000000100223D60</code>	<code>000000B0</code>

Line 7774 of 12966, /sub\_1002239BC

# The dirty work

## Aside: On IDA's app architecture

- Analysis (kernel) separated from UI
- Good choice, but not without some funny quirks...



# The dirty work

## Finding the path forward

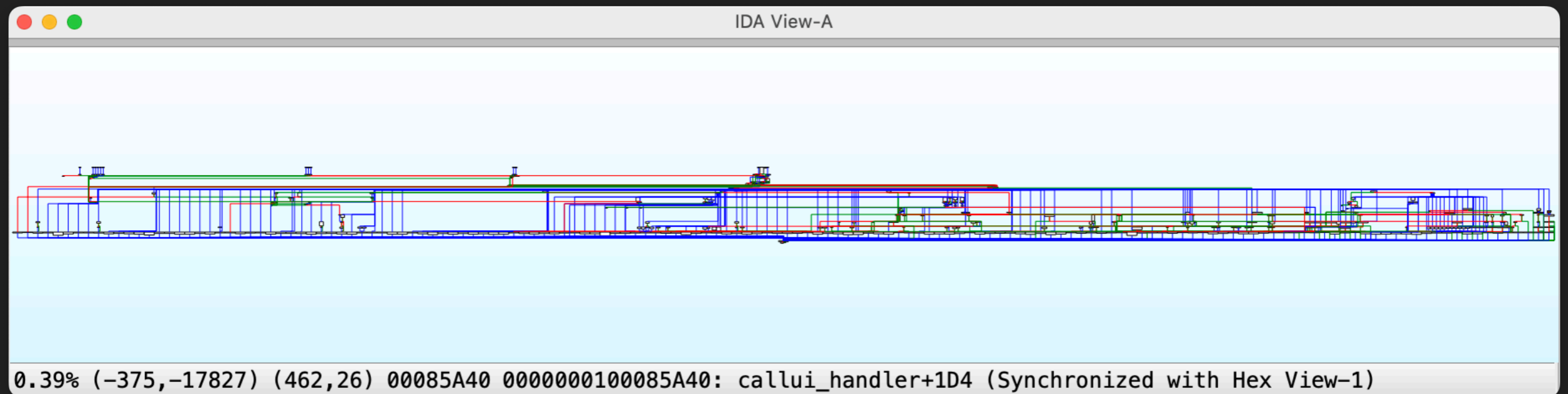
- Both frontends have an early call to 'init\_kernel'
- Absent from SDK headers...
  - Takes pointer to function defined by caller

```
Pseudocode-A
50     qword_10019DFA8 = v31;
51     v31[0] = off_10018EFA8;
52     qatexit(sub_1000DEFA4);
53     dword_1001A0450 = a1;
54     qword_1001A0458 = init_kernel(some_func, &dword_1001A0450);
55     if ( dword_1001A0450 < 2 )
56     {
57         v28 = 0;
58         v29 = 0;
59     }
60     else
61     {
62         v28 = 0;
63         v29 = 0;
64         v2 = 1LL;
```

# The dirty work

## A curious function pointer

- Similar functions passed by both frontends
  - Takes an integer and variadic arguments
  - World's biggest (nested) switch statement



# The dirty work

## A curious function pointer (cont.)

- Pointer saved by kernel ("callui"), used a lot

The image shows a debugger window titled "Pseudocode-A" displaying assembly-like pseudocode. Line 49 is highlighted in green and shows the assignment `callui = user_func_ptr;`. Below it, lines 50-55 show an `if` statement and its body. A second window titled "xrefs to \_callui" is overlaid on the bottom right, showing a list of cross-references to the `callui` symbol. The table below is a transcription of the data in that window.

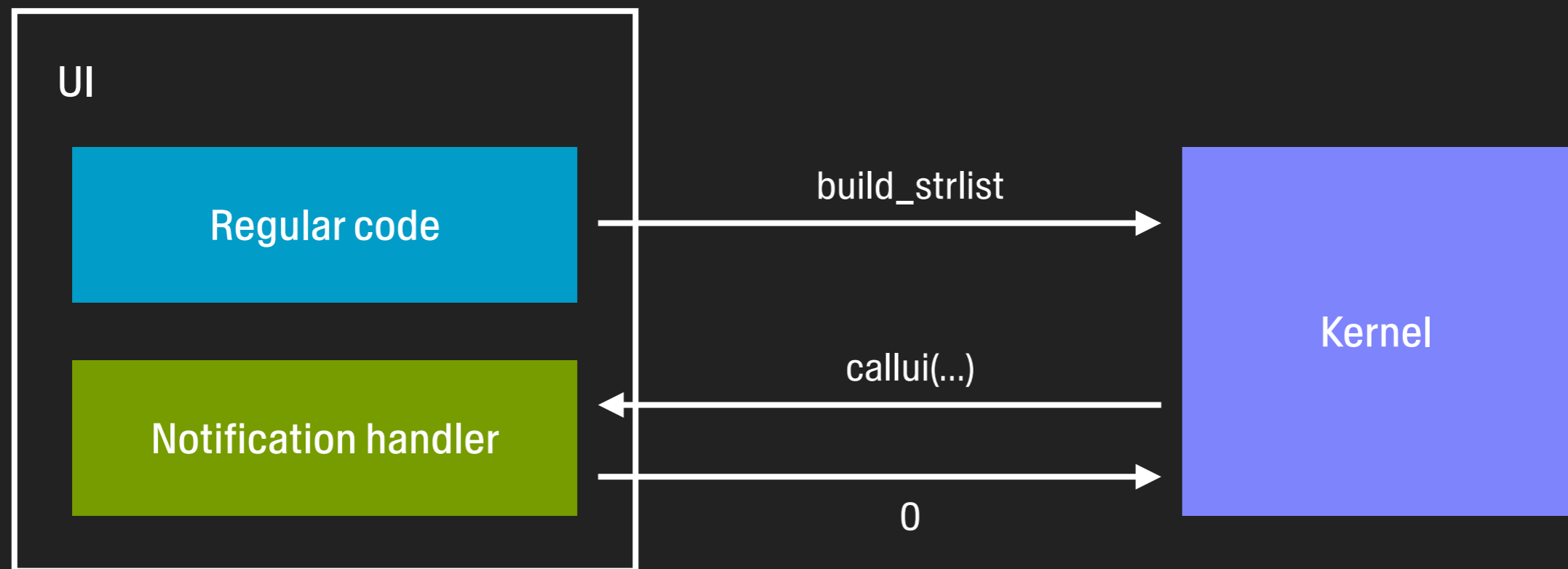
Direction	Type	Address	Text
Up	o	sub_326514+64	ADRP X8, #_callui@PAGE
Up	r	sub_326514+6C	LDR X8, [X8,#_callui@PAGEOFF]
Up	o	sub_3265F0+70	ADRL X22, _callui
Up	r	sub_3265F0+78	LDR X8, [X22]; sub_1F418
Up	r	sub_3265F0+9C	LDR X9, [X22]; sub_1F418
Up	o	sub_326738+68	ADRP X9, #_callui@PAGE
Up	r	sub_326738+70	LDR X9, [X9,#_callui@PAGEOFF]
Up	o	sub_326818+BC	ADRP X9, #_callui@PAGE

Line 688 of 688

Buttons: Help, Search, Cancel, OK

# The dirty work

## A word on UI notifications (callui)



# The dirty work

## Building the guillotine

```
nvim
#include <stdio.h>

extern "C" void *init_kernel(int (*msg)(int, ...), int *ok);

int callui_impl(int msg, ...) {
    printf("msg: %d\n", msg);
    return 0;
}

int main(int argc, char const **argv) {
    int ok = 0;
    init_kernel(callui_impl, &ok);
    return ok;
}
~
~
~
~
~
~
~
~
~
~
[No Name] [+] 13,0-1 All
```

# The dirty work

## Building the guillotine (cont.)

```
jon@Aquila:/private/tmp
; ./hida64
msg: 53
msg: 53
msg: 52
msg: 94

Thank you for using IDA. Have a nice day!

jon@Aquila:/private/tmp
; |
```



# The dirty work

## Sidechannel-driven development

- Treat kernel like black box
- See what messages are sent
  - Reverse response logic in real IDA
  - Reimplement in own code
- Rinse and repeat until kernel works?

# The dirty work

## Sidechannel-driven development (cont.)

- Worked for a bit
  - More responses implemented = kernel loads further
- Got out of hand quick...

```
nvim
enum ui_msg_t {
    /* ... */
    UI_MSG_RESPOND_ZERO,
    UI_MSG_EMPTY_STRING_PTR_SEEMS_TO_WORK,
    UI_MSG_IDK_JUST_RESPOND_1,
    UI_MSG_WTF,
    UI_MSG_COPY_UNLESS_BIG_ARG_OR_IT_WILL_CRASH,
    UI_MSG_THIS_GETS_CALLED_A_LOT,
    /* ... */
};
~
~
~
~
```

# The dirty work

## A comically-timed SDK docs find

- One day, while browsing SDK docs for other answers...

```
nvim include/kernwin.hpp
enum ui_notification_t
{
    ui_null = 0,

    ui_range,          ///< cb: The disassembly range has been changed
                      ///< UI should redraw the scrollbars. See also:
                      ///< \param none
                      ///< \return void

    ui_refresh_choosers, ///< cb: The list (chooser) window contents have
                        ///< UI should redraw them. Please consider requ
                        ///< \param none
                        ///< \return void

    ui_idcstart,       ///< cb: Start of IDC engine work.
                      ///< \param none
                      ///< \return void

    ui_idcstop,        ///< cb: Stop of IDC engine work.
                      ///< \param none
                      ///< \return void
}
```

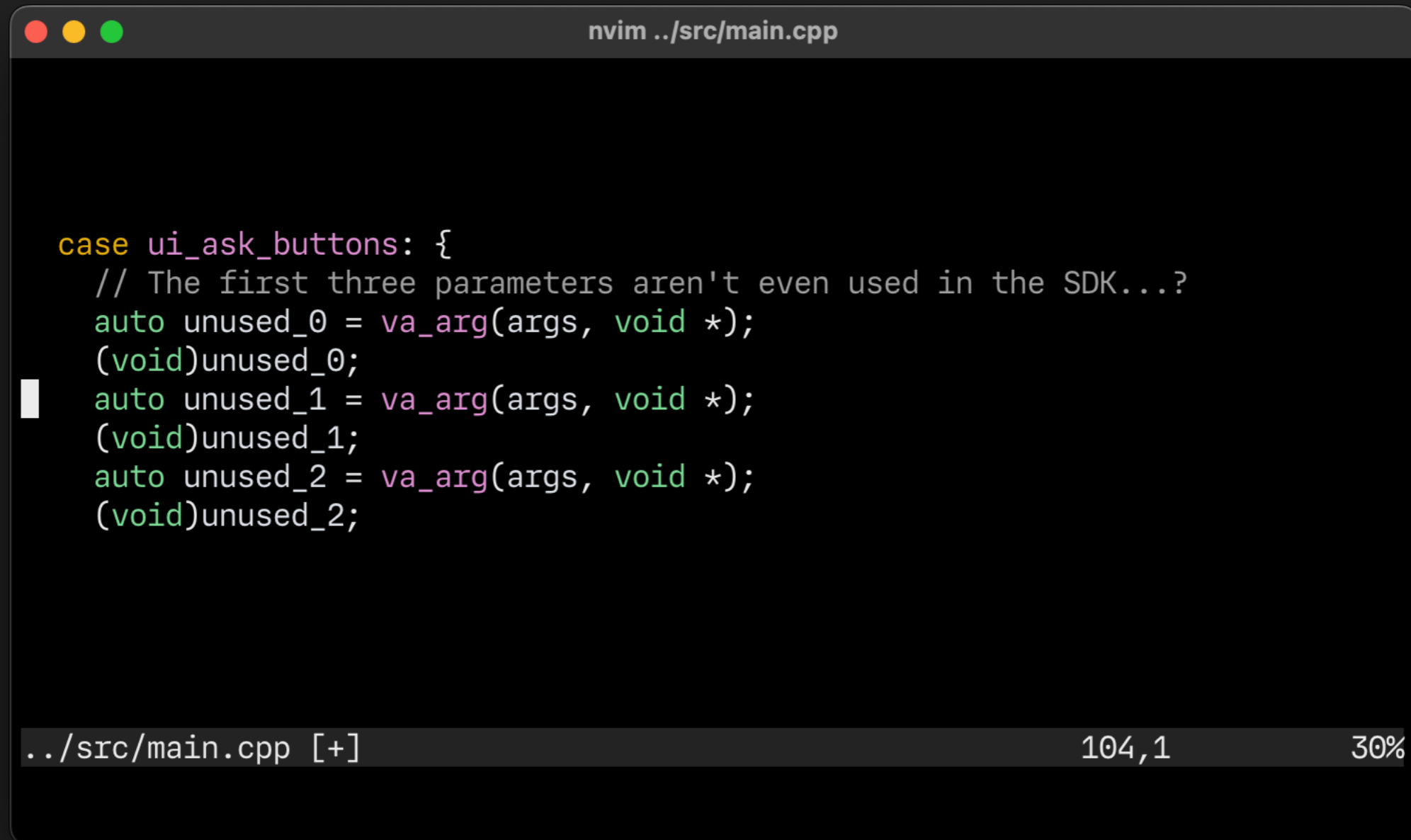
# The dirty work

## The rest of the owl...

- Clean up "callui" handler
- Play more whack-a-mole with internal errors
- Perform animal sacrifice
- Finally analyze a file?

# The dirty work

Some comments to illustrate my experience



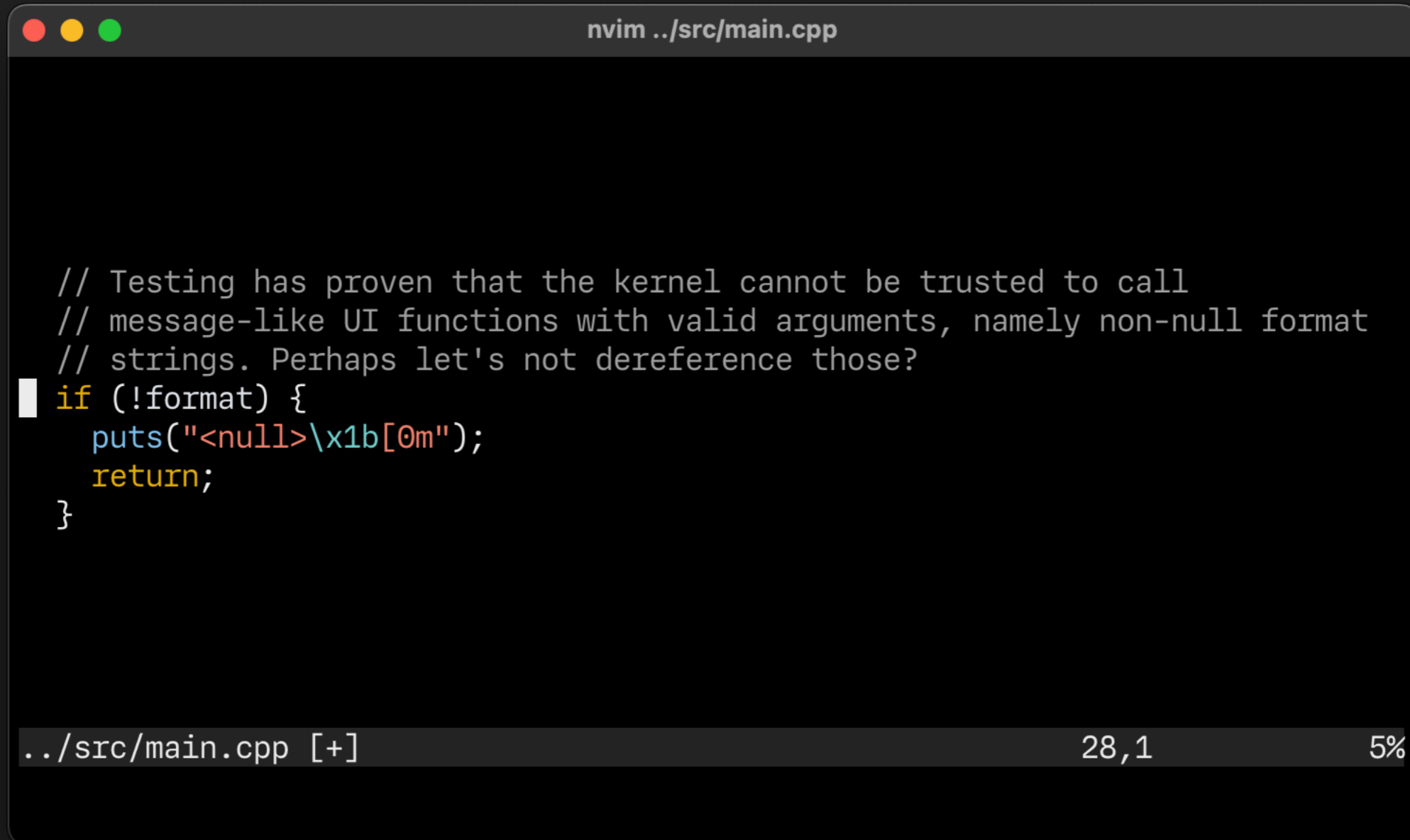
```
nvim ../src/main.cpp

case ui_ask_buttons: {
    // The first three parameters aren't even used in the SDK...?
    auto unused_0 = va_arg(args, void *);
    (void)unused_0;
    auto unused_1 = va_arg(args, void *);
    (void)unused_1;
    auto unused_2 = va_arg(args, void *);
    (void)unused_2;
}

../src/main.cpp [+] 104,1 30%
```

# The dirty work

Some comments to illustrate my experience (cont.)




```
nvim ../src/main.cpp

// Testing has proven that the kernel cannot be trusted to call
// message-like UI functions with valid arguments, namely non-null format
// strings. Perhaps let's not dereference those?
if (!format) {
    puts("<null>\x1b[0m");
    return;
}

../src/main.cpp [+] 28,1 5%
```

# The dirty work

Some comments to illustrate my experience (cont.)



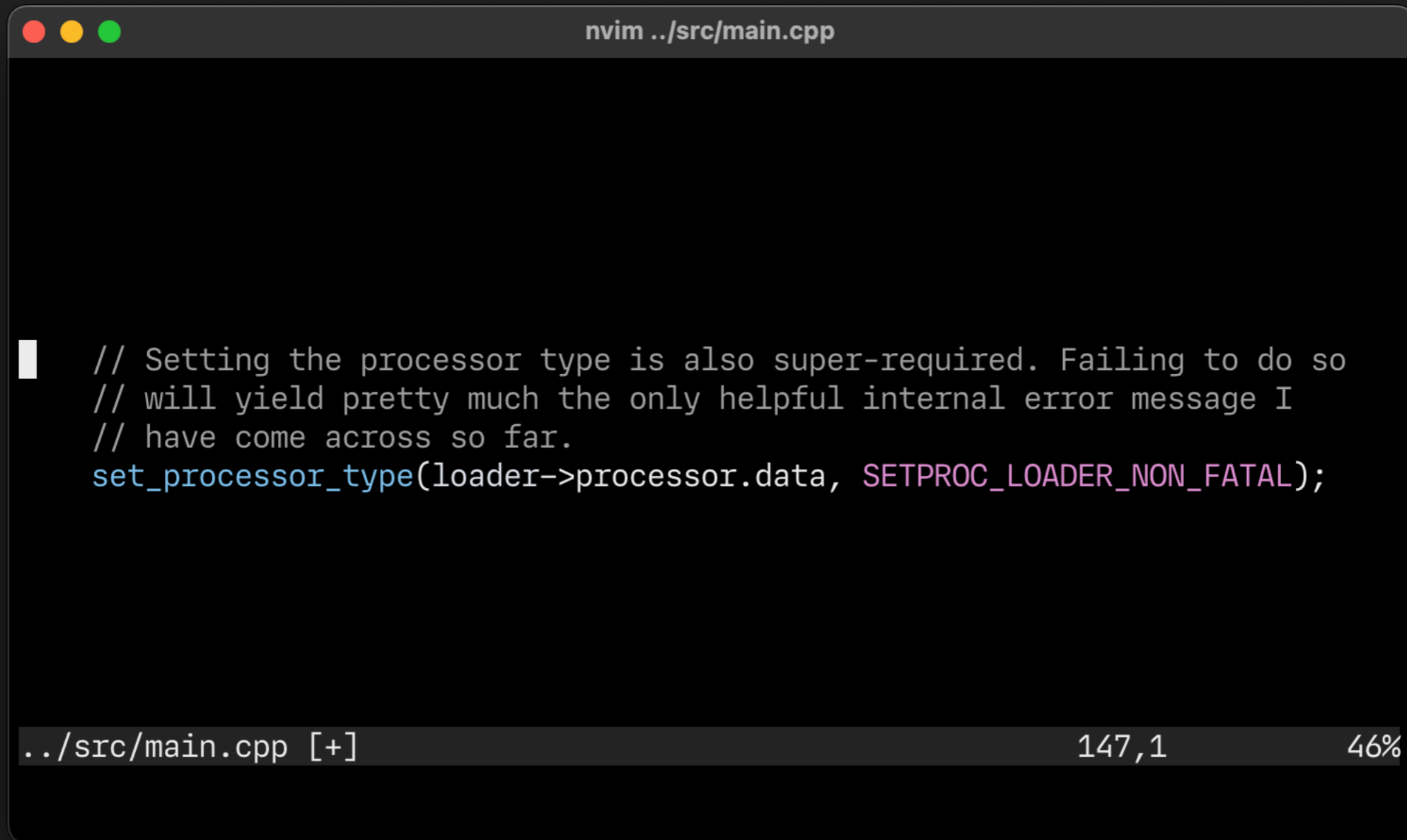
```
nvim ../src/main.cpp

// When asked, the UI is supposed to copy its version string to the
// buffer provided by the kernel. Sometimes the buffer size provided by
// the kernel is extremely large. In those cases, the version should not
// be copied. That will make IDA crash.
if (buffer_size < 0xc00) {
    strncpy(buffer, version_str, buffer_size);
    result.i32 = 3;
}

../src/main.cpp [+] 202,1 59%
```

# The dirty work

Some comments to illustrate my experience (cont.)



```
nvim ../src/main.cpp

// Setting the processor type is also super-required. Failing to do so
// will yield pretty much the only helpful internal error message I
// have come across so far.
set_processor_type(loader->processor.data, SETPROC_LOADER_NON_FATAL);

../src/main.cpp [+] 147,1 46%
```





**Time check: demo?**

# Free at last

But for what?

- Well...
  - What was all that effort for?

# Free at last

Heavy is the head that calls the kernel

- IDA batch mode is faster than GUI IDA
  - Headless IDA faster than batch mode?
  - Well...

# Free at last

Heavy is the head that calls the kernel (cont.)

- Up to 40% faster than unpatched batch mode\*
  - Still up to 25% faster after applying optimization patches



# Free at last

## Alternative interfaces

```
rm -f keybagd.i64 && ./hida64.sh keybagd

IDA is analysing the input file...
You may start to explore the input file right now.
r2ida> aa
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
r2ida> s 0x100014D64
r2ida> pdf
sub_100014D64:
    PACIBSP
    SUB     SP, SP, #0x20
    STP     X29, X30, [SP,#0x10+var_s0]
    ADD     X29, SP, #0x10
    ADRL   X8, unk_100026F69
    CMP     X0, #0
    CSEL   X0, X8, X0, EQ; int
    ADD     X8, X29, #0x10
    STR     X8, [SP,#0x10+var_8]
    ADD     X2, X29, #0x10; arguments
    BL     sub_100014DA0
    LDP     X29, X30, [SP,#0x10+var_s0]
    ADD     SP, SP, #0x20 ; ' '
    RETAB
```

# Free at last

## Many other uses

- Interactive headless IDAPython
- Easier plugin development & debugging
- Build tooling on top of IDA
- Much more...

**One more thing...**





- Had to be there in person for this slide :)
- And if you were there: you don't remember anything...

# Thank you!

- Questions?
  - Find me after if you want to chat more :)
  
- Twitter: [@jonpalmisc](https://twitter.com/jonpalmisc)
- Mastodon: [@jonpalmisc@infosec.exchange](https://infosec.exchange/@jonpalmisc)