# Introduction to Information Security


מרצים:

Dr. Eran Tromer: tromer@cs.tau.ac.il

Prof. Avishai Wool: yash@eng.tau.ac.il

מתרגלים:

Itamar Gilad (itamargi@post.tau.ac.il)

Nir Krakowski (nirkrako@post.tau.ac.il)

# Today

- Reverse Engineering 101
- IDA (!)
- Binary patching 101

- More tools

# Reverse Engineering

- What does the following code do:
  - LEA EDX, [address to “Hello, world!”]
  - MOV ECX, 12

  MYLOOP:
  - PUSH EDX
  - CALL printf
  - ADD ESP, 4
  - LOOP MYLOOP

# Reverse Engineering

- What is it?
  - Using the binary to recreate any knowledge needed
- Why?
  - Recreating lost platforms (ReactOS)
  - 'Secret' algorithms (Encryption, trade secrets, etc.)
  - Hidden features (and hidden backdoors)
  - Internal structures & implementation details
  - Bugs / Vulnerabilities that only exist in the binary
  - you name it!

# So, what's the problem?

- Compiling is like a one-way function.
- Information is lost, and we *often* loose access to –
  - Variable and function names
  - Comments
- What do we still have -
  - Import and export names (relations between modules)
  - Structure of parameters to functions.
  - Starting point
  - Hard-coded strings
  - Constants

# RE Process

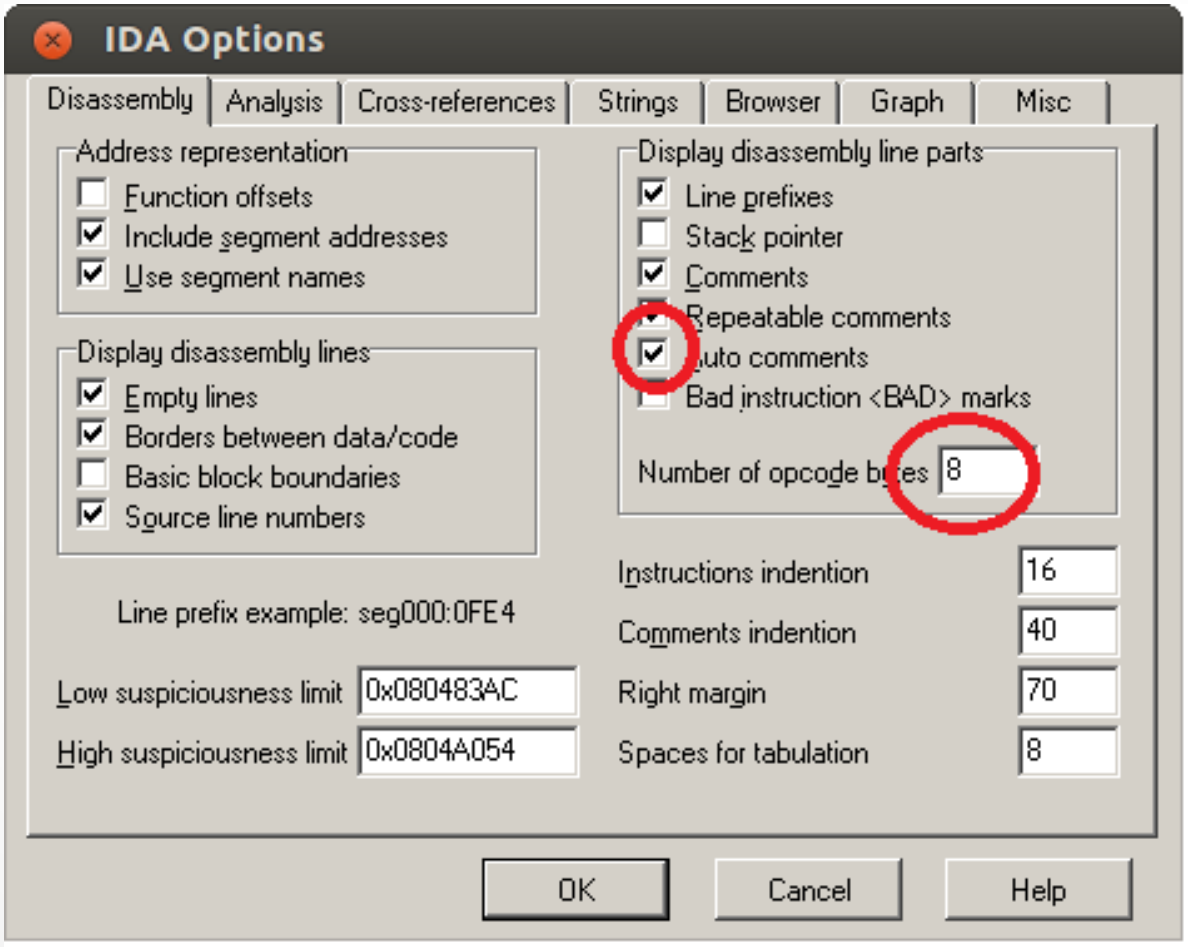
- **Our objectives –**
  - Find the most interesting piece of code in the least amount of time
  - Understand what it does and how
  - Find weaknesses and figure out how to exploit them
- Use leads –
  - Strings, UI
  - Dynamic debugging, breakpoints.
  - Library and system functions
- Interpret the assembled code by using intelligent guesses –
  - Context-based
  - Code is written by people using regular code conventions
  - Code is written in an upper level language, and compilers are usually pretty predictable

# IDA

- The Interactive Dis-Assembler (**IDA**) is the most popular reverse engineering tool
  - Version 5.0 is free-ware and that is what we'll use.
- IDA does several things automatically:
  - Disassemble x86 binary code into human readable format
  - Identifies ELF headers (executable file formats)
  - Signature based recognition for library functions and compiler tricks
  - Creates code graph by basic blocks
  - Code and data **xrefs** (references to memory addresses, functions)
- Provides a good environment for research:
  - Adding comments (';')
  - Renaming labels: code blocks, variables, function names, structures. ('n')
  - Change interpretation of binary data (code->data, data->code, data type change, etc.)

https://www.hex-rays.com/products/ida/support/freefiles/IDA_Pro_Shortcuts.pdf

# IDA Options


IDA Options
Disassembly
Analysis
Cross-references
Strings
Browser
Graph
Misc
Address representation
Function offsets
Include segment addresses
Use segment names
Display disassembly lines
Empty lines
Borders between data/code
Basic block boundaries
Source line numbers
Line prefix example: seg000:0FE4
Low suspiciousness limit
0x080483AC
High suspiciousness limit
0x0804A054
Display disassembly line parts
Line prefixes
Stack pointer
Comments
Repeatable comments
Auto comments
Bad instruction <BAD> marks
Number of opcode bytes
8
Instructions indention
16
Comments indention
40
Right margin
70
Spaces for tabulation
8
OK
Cancel
Help

# IDA Demo

- [Helllo World]

# IDA Demo

- [stricmp]

# Binary patching

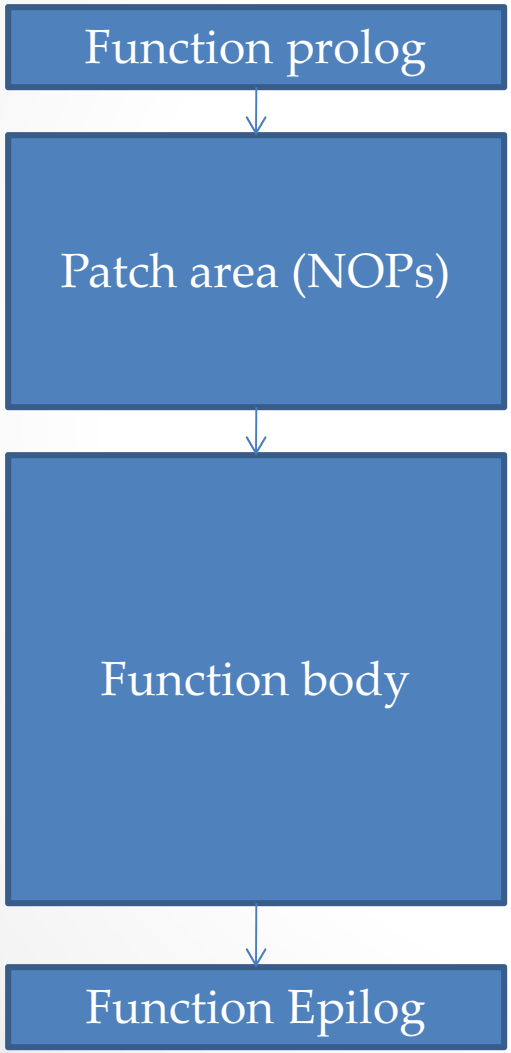
- What?
  - Changing instructions/data/metadata in the "production" binary
- Why?
  - You lost the source code
  - You never had the source code
  - Small changes that would be easier to test on their own
  - Hot patching
  - And many more

# Binary patching example

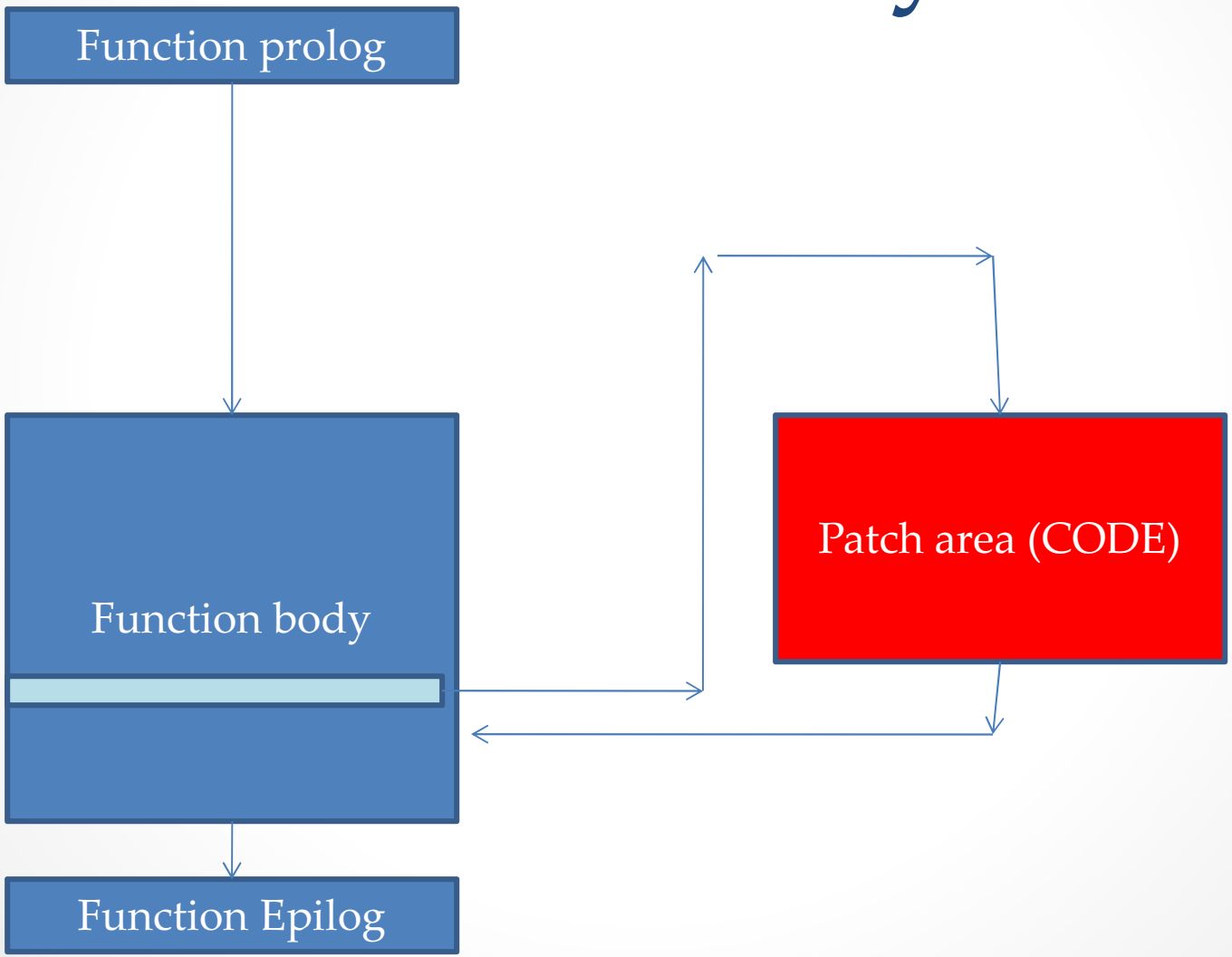
```
int verify_login(char * username, char * password)
{
        if ((0 == strcmp(username, “root”)) &&
            (0 == strcmp(password, “my_pass”)) {

                return 0;
        }
        else {
                return 1;
        }
}
```

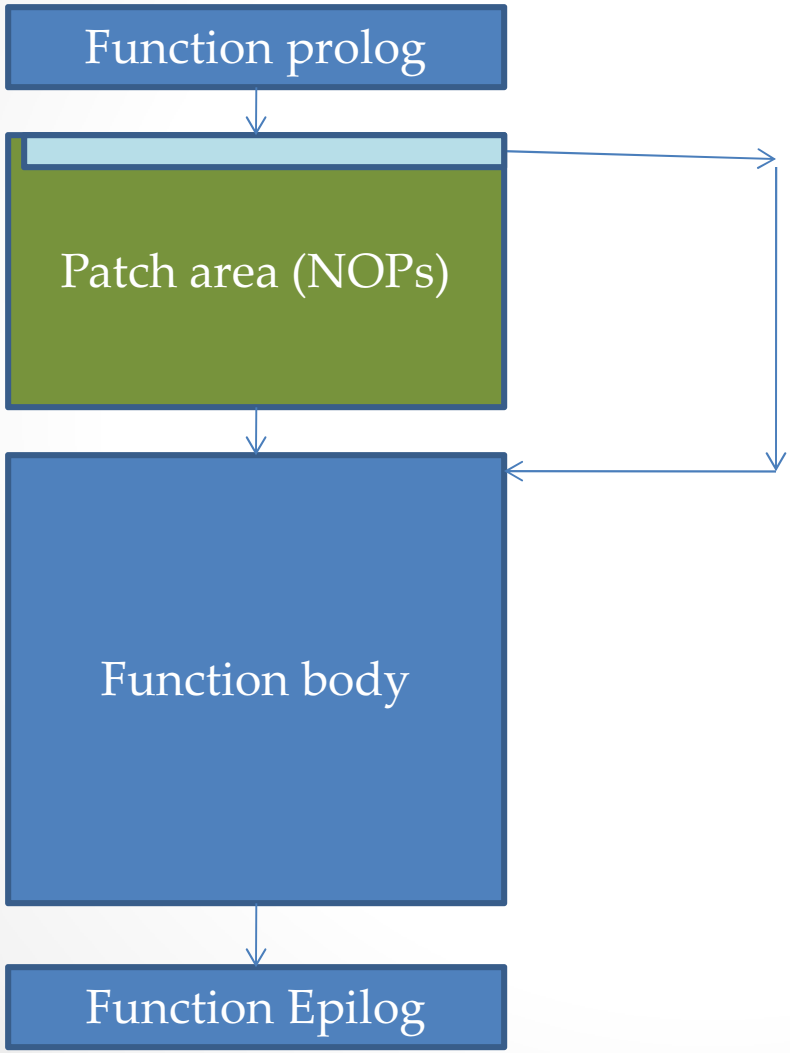
# Patch Layout



# Execution Layout

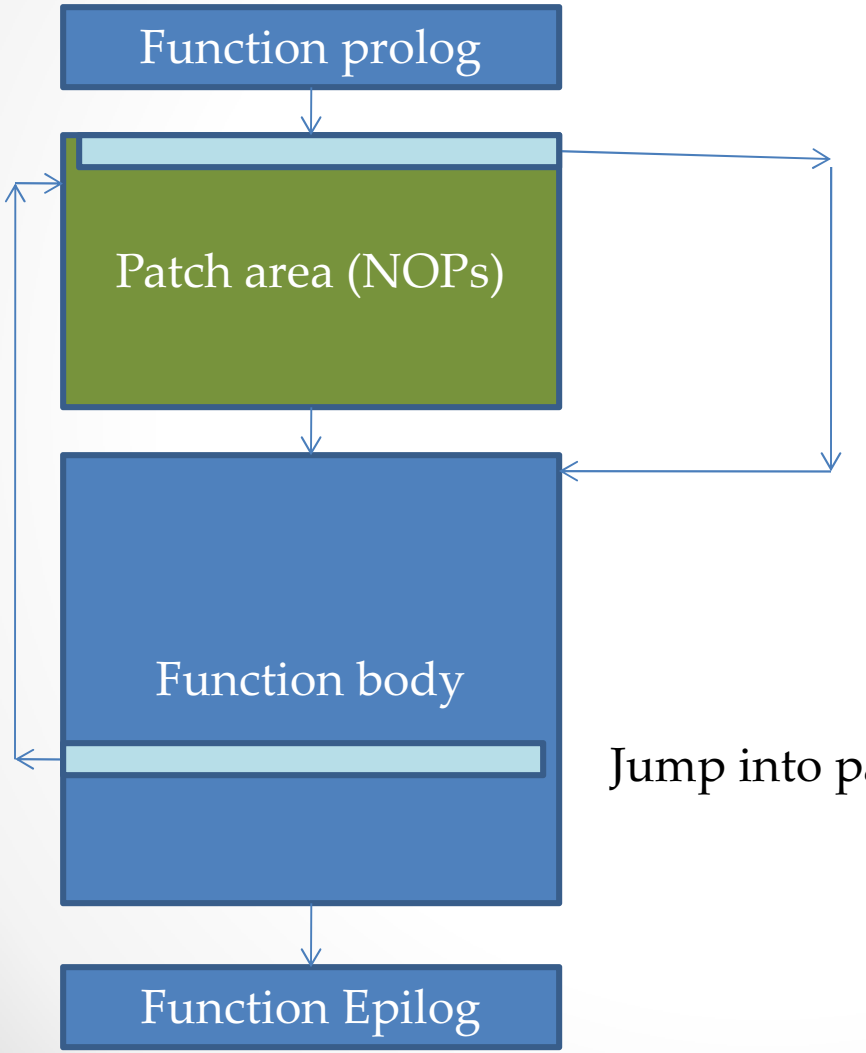


# Patch Layout



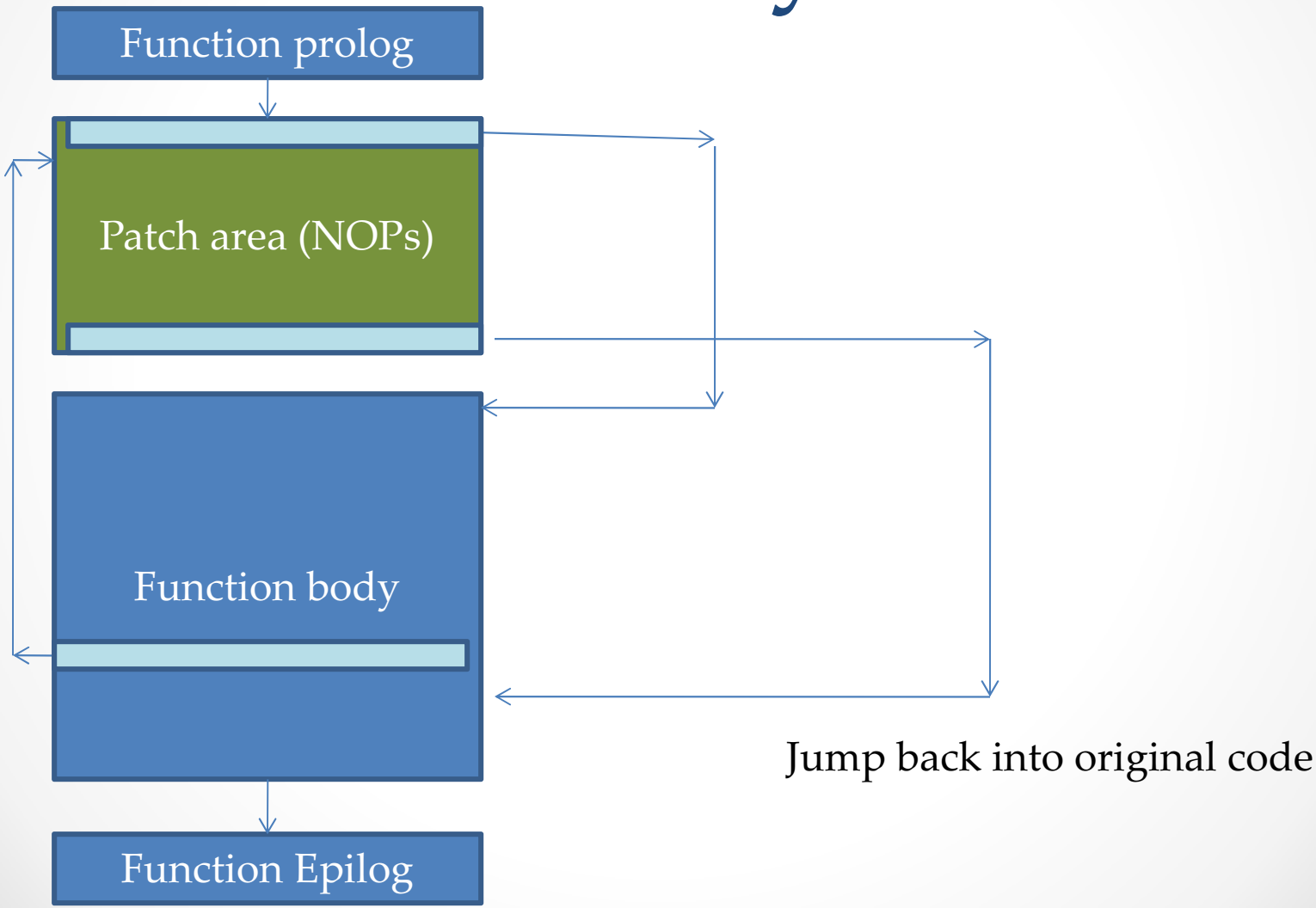

Divert execution around patch area

# Patch Layout

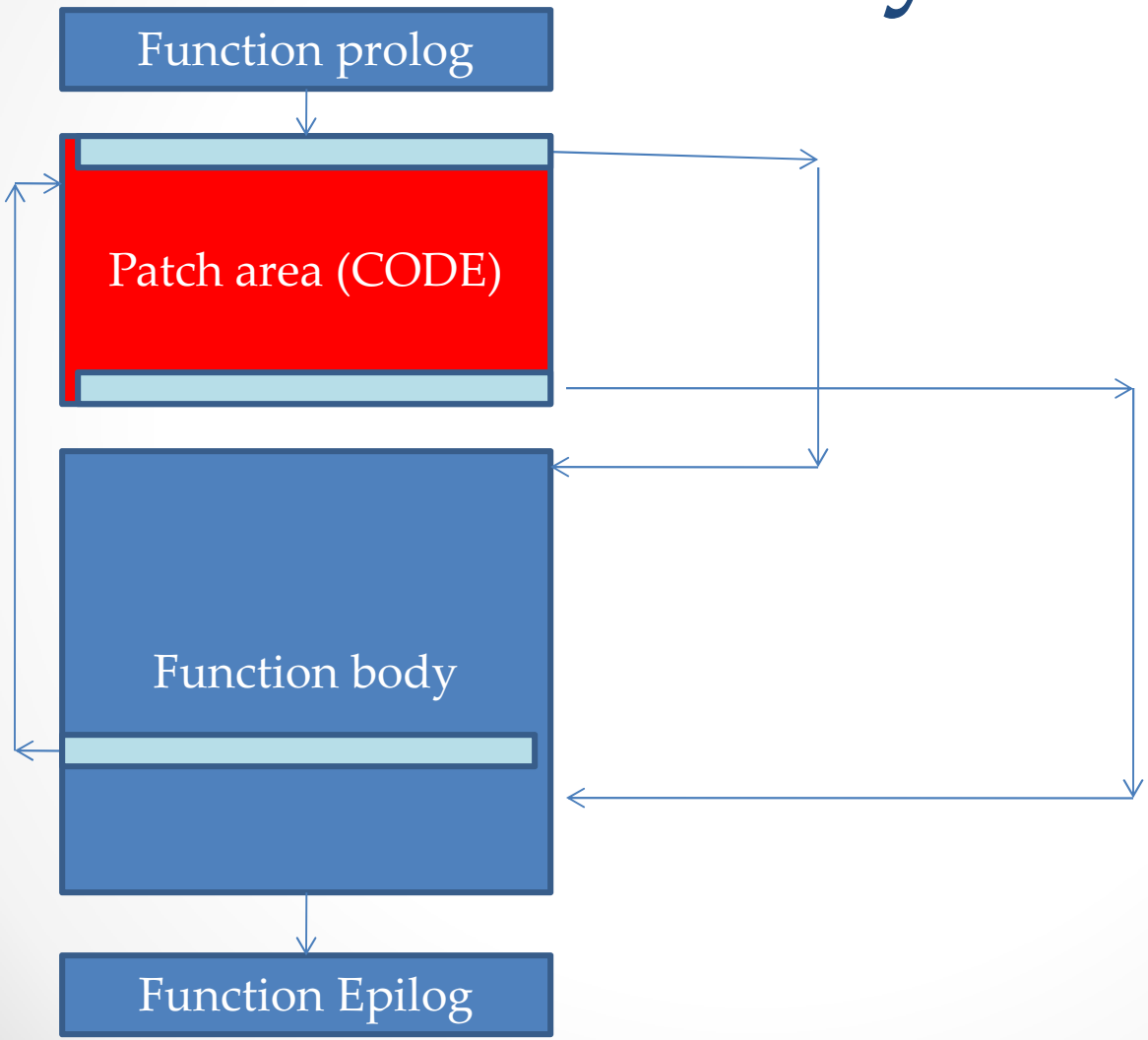



Jump into patch area

# Patch Layout



# Patch Layout



# New tools!

va_to_offset.py – A tool to map a virtual address (as you see in IDA) to a file offset

patch_util_gcc.py – A script that lets you patch a binary by using simple text files with (bare) assembly instructions

# This week's exercise

- First reverse engineering task
- First binary patching task
- It isn't hard – but please start early and contact us if you have any trouble with the setup