Low level attacks
Disassembler and debugger

Mario Alviano

University of Calabria, Italy

A.Y. 2016/2017

Disassembler

- It is instructive to disassemble simple programs

```
objdump -M intel -D <binary-file>
```

Example

- Try `variable.c`
- Try `array.c`
- Try `if.c`
- Try `while.c`
- Try `for.c`

Different compilers produce different assembly!

Debugger setup

- We are going to use `gdb` (GNU Debugger)
- Add `set disassembly intel` to `~/.gdbinit`
- Add the flag `-g` to `gcc` to compile with extra debugging information

Try on `for.c`

- Run the debugger passing a binary file as first argument
- Try the `list` command
- Try `disassemble main`

Breakpoints and watchpoints

- Breakpoints break execution before specific instructions
- Watchpoints break execution when variables are changed

Example

- Add a breakpoint to `main` function
- Add a watchpoint to variable `a`
- Watch `a` with condition `a = 5`

Register inspection

- `info registers`
- `info register <register>`
  e.g., `info register eip`

Memory inspection

`x/`*nfu* *addr*

- *n*: number of units (default is 1)
- *f*: display format (default is `x`)
  - `o`: octal
  - `x`: hexadecimal
  - `u`: unsigned decimal
  - `t`: binary
  - `i`: instruction
- *u*: size of unit (default is `w`)
  - `b`: byte
  - `h`: halfword, 2 bytes
  - `w`: word, 4 bytes
  - `g`: giant, 8 bytes
- *addr*: can be a register (`$eip`), an address (`0x8048416`), or a variable (`&i`)

Little-endian vs big-endian machines

- Compare `x $eip` and `x/4b $eip`
- If bytes are in reverse order, your machine is little-endian
- Take this into account when exploiting

More on GDB

Assembly stepping

- `step` and `next` work on C instructions
- Add `i` suffix to execute one assembly instruction
- That is, use `stepi` and `nexti`

GDB Cheat Sheet

`http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf`

Use assembly in C/C++

- Use instruction `__asm__(<assembly-code-here>)`
- Compile with `-masm=intel`

Example

Try `find_start.c`

Disable protection mechanisms

- Disable Address Space Layout Randomization (ASLR):
  `sudo bash -c 'echo 0 > /proc/sys/kernel/randomize_va_space'`
  (default value is 2)
- Compile with `-fno-stack-protector` to disable canaries
- Compile with `-z execstack` to enable executable stack

Buffer overflow

- Essentially, writing after the last element of an array
- Target EIP to control execution of the running program

Example

1. Try `buffer.c`
2. Try `buffer2.c`

Core dump

- Activate core dump generation with `ulimit -c unlimited`
- Analyze core with `gdb -q -c core`

Overflow into the return address

- Try `overflow.c`
- Function `gets()` does not bound its argument
- Find the address of function `unlinked_code`, say 0x0804845b
- Try the following:

```
for i in $(seq 30 50); do
    echo $i;
    python -c "print('A'*$i + '\x5b\x84\x04\x08')" | a.out;
done
```

Serial codes

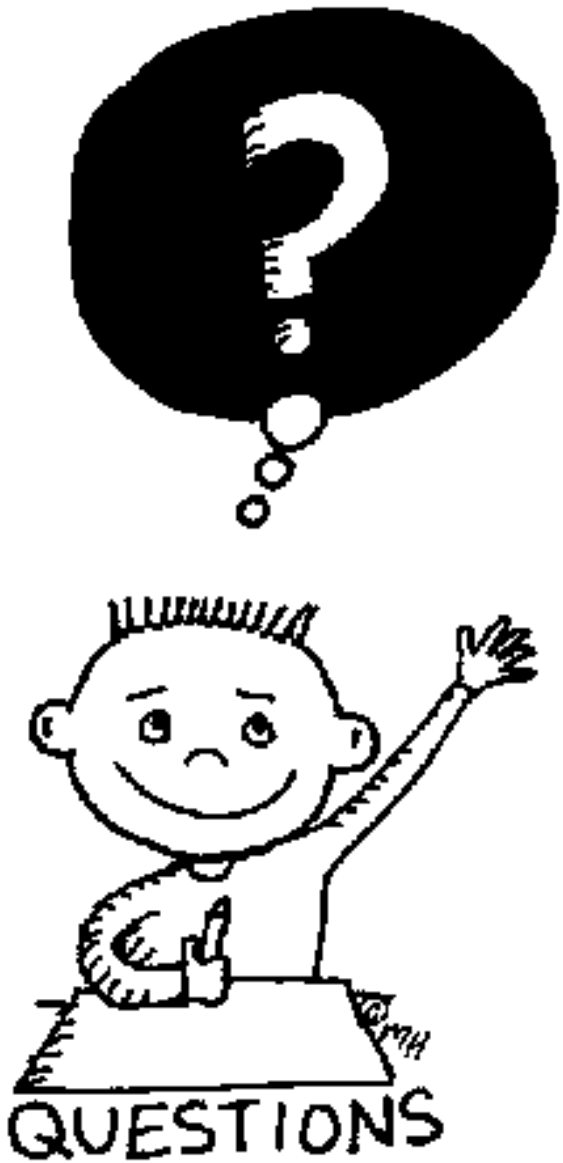
- Try `serial.c`
- Identify the address of `call do_valid_stuff()`, say 0x08048618

Exercise 1

Can you force EIP to be 0x08048618?

Exercise 2

- `'d'*8 + 'DD'` is a valid serial code
- Can you provide a different, valid serial code?



END OF THE LECTURE