Introduction to Information Security

Administration
About Us

• Website: https://course.cs.tau.ac.il/infosec16

• Nir Krakowski
  • nir.kra@gmail.com

• Dan Gittik
  • dan.gittik@gmail.com

• Yuval Lewi

• Recitations on Sunday at 16:00-17:00 and 17:00-18:00
• Reception hour on Sunday at 18:00 until last customer
About the Course

- The course is really, really, really hard
  - Information security includes many different disciplines
  - Some you'll have to master by yourself (or with Google)
About the Course

• The course is really, really, really hard
  • Information security includes many different disciplines
  • Some you'll have to master by yourself (or with Google)
• On the other hand, it's also really cool

• Weekly homework assignments
  • 75% of the best grades will constitute 35% of the final grade
  • Copying (from other students or resources) is strictly forbidden

• The lectures and recitations are not always synchronized – this is by design
• Please fill in the course questionnaire and read the submission instructions
Introduction to Information Security

Assembly x86
Assembly

- Translates directly to machine code
  - Instruction
    - Opcode – what to do
    - Operand – what to do it on (can be missing, single, or multiple)

- It's weird, cumbersome, platform-dependent – but eventually, everything comes down to it
Assembly

- Opcodes
  - Expressions (MOV, ADD, SUB, XOR, CMP, ...)
  - Flow Control (JMP, CALL, RET, JZ, JNZ, JE, JNE, ...)
  - And many more...

- Operands
  - Constants
  - Registers
  - Memory Addresses (in brackets – [address])
Registers

- EAX, EBX, ECX, EDX, ESI, EDI – general purpose
  - EAX return values
  - ECX loop counters
  - ESI source
  - EDI destination

- EIP Instruction Pointer (points to the next instruction)
- ESP Stack Pointer (points to the stack head)
- EBP Base Pointer (points to the stack frame)

- E_X (0-31 bits), _X (0-15 bits), _L (0-7 bits), _H (8-15 bits)
Example

MOV EAX, 1
MOV EBX, 2
ADD EAX, EBX  # EAX is now 3
Example

```
MOV ECX, 10
_MY_LOOP:
...
# Happens 10 times
DEC ECX
CMP ECX, 0
JNE _MY_LOOP
...
```
The Stack

- Consider int \( x = 1 \), long \( y = 2 \) and short \( z = 3 \)
- Advance ESP by 14 and use \([ESP+10]\), \([ESP+2]\) and \([ESP]\), respectively

```
int x = 1
long y = 2
short z = 3
```

```assembly
SUB ESP, 14
MOV [ESP+10] , 1
MOV [ESP+2] , 2
MOV [ESP] , 3
```
Function Calls

• When a function is called...
  • Push the arguments and the address of the next instruction unto the stack, and jump to it
• When a function returns...
  • Pop the return address off the stack and jump back to where we left off
  • The caller should "clear" the arguments off the stack

```
f(1, 2);

ESP  | 0x110 | 2
ESP  | 0x10C | 1
ESP  | 0x108 | EIP
ESP  | 0x104 |

return;

ESP  | 0x100 |

PUSH 2
PUSH 1
CALL f  # PUSH EIP, JMP f
ADD ESP, 8

RET    # POP EIP, JMP EIP
```
Function Calls

- But each function uses the stack, too!
  - In the function prologue, push EBP and set it to ESP – that's the function base (or frame)
  - In the function epilogue, set ESP to EBP ("release" it's stack), pop EBP, and you're golden
- The function arguments are often accessed as [EBP+8], [EBP+12], etc.
- Either the caller or the callee should "back up" some or all of the registers

```
  | 0x110 | 2 |
  | 0x10C | 1 |
  | 0x108 | EIP |
  | 0x104 | EBP |
  | 0x100 |
```

```
PUSH EBP
MOV EBP, ESP
ESP
EBP
ESP
ESP
EBP
PUSH EBP
MOV ESP, EBP
POP EBP
```