

CS107, Lecture 21

Reverse Engineering



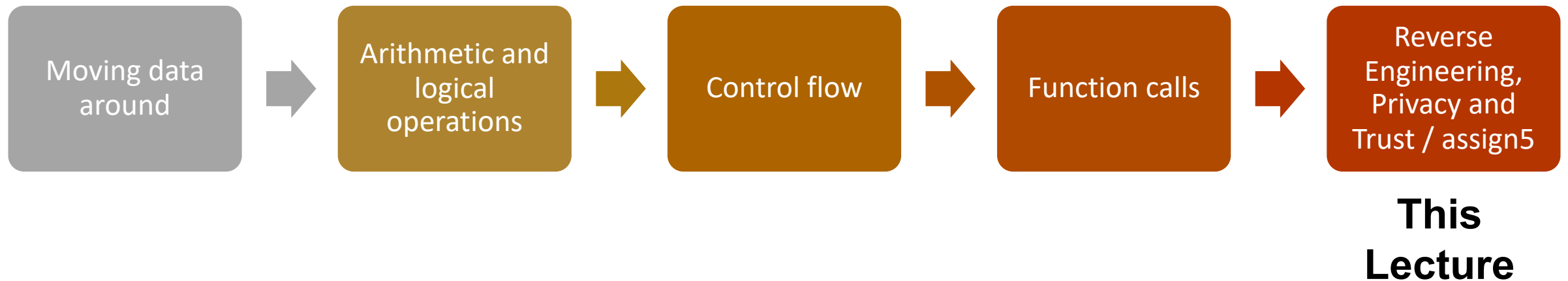
masks recommended

This document is copyright (C) Stanford Computer Science and Nick Troccoli, licensed under Creative Commons Attribution 2.5 License. All rights reserved.

Based on slides created by Cynthia Lee, Chris Gregg, Jerry Cain, Lisa Yan and others.

NOTICE RE UPLOADING TO WEBSITES: This content is protected and may not be shared, uploaded, or distributed. (without expressed written permission)

Learning Assembly



Reference Sheet: cs107.stanford.edu/resources/x86-64-reference.pdf
See more guides on Resources page of course website!

Learning Goals

- Learn how to approach reverse engineering executables
- Understand the requirements and tasks for assign5

Lecture Plan

- **GDB / Function Call Practice:** Recursion
- **Reverse Engineering Practice:** Minivault

```
cp -r /afs/ir/class/cs107/lecture-code/lect21 .
```

Lecture Plan

- **GDB / Function Call Practice: Recursion**
- **Reverse Engineering Practice: Minivault**

```
cp -r /afs/ir/class/cs107/lecture-code/lect21 .
```

Example: Recursion

- Let's look at an example of recursion at the assembly level.
- We'll use everything we've learned about registers, the stack, function calls, parameters, and assembly instructions!
- We'll also see how helpful GDB can be when tracing through assembly.



factorial.c and factorial

gdb tips



<code>layout split</code>	(ctrl-x a: exit, ctrl-l: resize, refresh: refresh, layout reg/asm, focus next)	View C, assembly, and gdb (lab5)
<code>info reg</code>		Print all registers
<code>p \$eax</code>		Print register value
<code>p \$eflags</code>		Print all condition codes currently set
<code>b *0x400546</code>		Set breakpoint at assembly instruction
<code>b *0x400550 if \$eax > 98</code>		Set conditional breakpoint
<code>ni</code>		Next assembly instruction
<code>si</code>		Step into assembly instruction (will step into function calls)

gdb tips



`p/x $rdi`

Print register value in hex

`p/t $rsi`

Print register value in binary

`x $rdi`

Examine the byte stored at this address

`x/4bx $rdi`

Examine 4 bytes starting at this address

`x/4wx $rdi`

Examine 4 ints starting at this address

`finish`

Finish function, return to caller

Lecture Plan

- **GDB / Function Call Practice: Recursion**
- **Reverse Engineering Practice: Minivault**

```
cp -r /afs/ir/class/cs107/lecture-code/lect21 .
```

assign5

You are a security researcher hired to explore potential vulnerabilities and issues at Stanford Bank. **3 core parts:**

- 1. Uncovering ATM software vulnerabilities**
- 2. Demonstrating how a data leak can lead to data aggregation and uncovering of personal information**
- 3. Reverse engineering a secure program – discover 4 passwords needed to gain access to the system**

Minivault

The **minivault** program is practice for part 3, SecureVault (it doesn't share code with SecureVault but is similar reverse-engineering practice).

You must provide correct passwords for 2 stages:

```
./minivault [stage1password] [stage2password]
```

stage1 and **stage2** are 2 functions in minivault, each passed in the password for that stage. Our goal is to get both to return 1, and not 0.

Reverse Engineering Tips

- 1. Run the program live in GDB and step through.** Reading and diagramming by hand is useful, but quickly becomes infeasible with larger programs.
- 2. Break the assembly into chunks**
- 3. Use gdb to verify your hypotheses.**
- 4. Document your knowns and unknowns.** Document and re-verify conflicting assumptions.
- 5. Use compiler explorer to see what code looks like in assembly.**
- 6. Use library functions to your advantage.** If you spot a call to what looks like a library function, it's the real deal.
- 7. When tracing an unknown function, before dissecting its behavior first learn about the input/output of the function and what role it plays.**

Demo: Minivault

Recap

- **GDB / Function Call Practice:**
Recursion
- **Reverse Engineering Practice:**
Minivault

Lecture 21 takeaway: Reverse engineering lets us understand the behavior of a program without seeing its source code. Check out slide 12 for some summarized tips!

```
cp -r /afs/ir/class/cs107/lecture-code/lect21 .
```