COMP6447

Intro by Adam

Course Structure

- Lectures weekly
  - 2 hours long (Recorded + live streamed with questions on Echo360/moodle)
  - Small break in middle for break or discussion if in person
  - Recommend **coming in-person**, can chat after lecture about interesting stories, etc
  - Feel free to stop me at anytime for questions
- Tutorial labs every week after lecture
  - 1 hour - Tutor walking through an example of what happened in lecture
  - 1 hour - Students playing around with/applying what they learnt in the lecture
- Wargames weekly
- https://webcms3.cse.unsw.edu.au/COMP6447/22T2/outline

Some more admin

- Where to find us/get help
  - Slack
    - seceduau.slack.com
    - join #comp6447
  - cs6447@cse.unsw.edu.au - for personal matters
- Self learning
  - A lot of hard/niche content in this course
  - We can only teach/cover the fundamentals.
  - Burden is on the student to experiment with techniques taught in lectures/tutorials.
    - All advanced exploitation techniques are based on the fundamentals we teach
    - The weekly wargames are a good example of the difficulty of challenges you might see in the exam
    - We won’t release solutions to all wargames

Wargames

2-5 challenges every week applying what you learnt in the week lecture + tutes

Solve remotely, and find a flag on the system to prove you solved it

Not meant to be an examination (hence the low percentage of final grade)

- Meant to help you practice

Adam’s top tips

- Don’t just copy flags (you can probably find solutions from friends, etc easily, not worth it in long run)
- Don’t leave them till the last minute (i know, cliche..)
    - Some challenges will require days. (past student experiences)
- Ask in class forum / tutors / Adam / friends for advice / help.
    - **Don’t ask specific questions about a specific challenge.**

Wargames cont…

First wargame is already released.

Not hacking challenges, just some practice to learn how to use tooling (will be covered in tutorials)

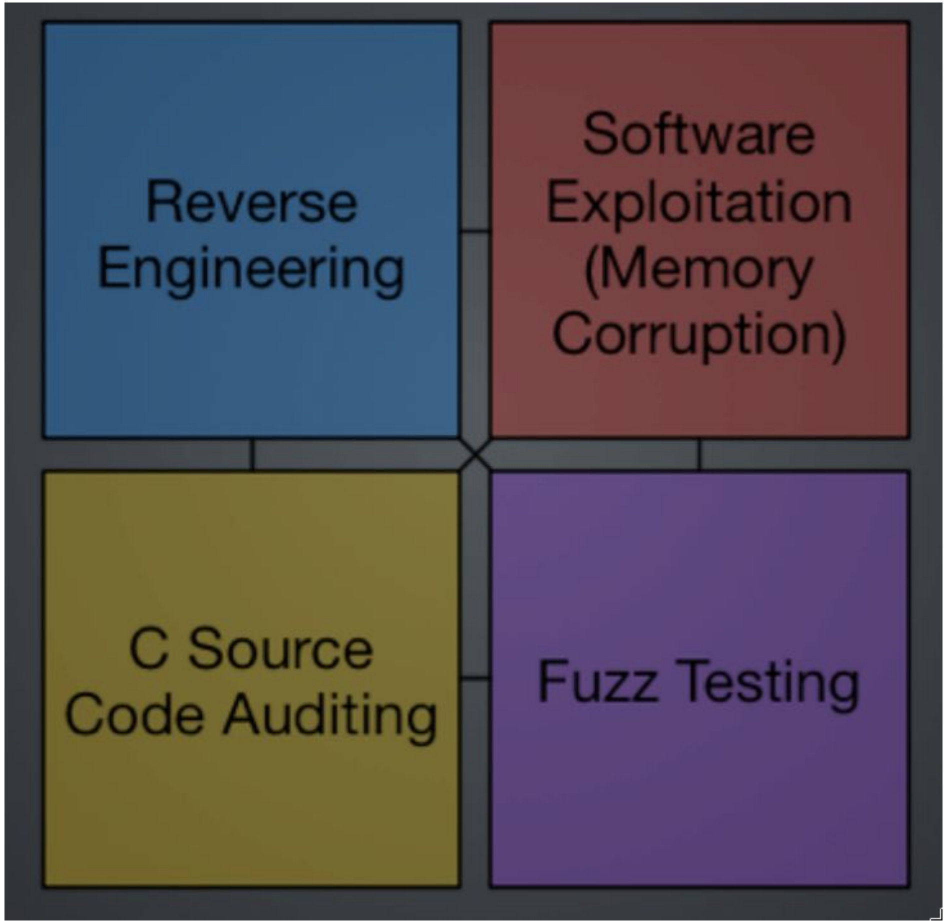
Tutorials will cover tooling and setting up stuff this week

Assessments

- Weekly Wargames
  - 30%
- Mid-term Exam
  - Week 5
  - 10%
- Major Assignment
  - Fuzzer group assignment (midpoint component due week 7)
  - due Sunday 17:59, end of week 10
  - 20%
- Final Exam
  - during final exam period
  - 40%

what even is 6447?

- Teach about hacking and hacking history
- Try to teach things that will be useful when you start in the workforce



Job Types?

Penetration Testing

Relatively brief assessments of a particular product or environment - frequently based on network scanning and/or analysis of a web or mobile app - e.g. test a new website built on apache and PHP.

Red Teaming

More in-depth, open ended penetration testing - without a limited scope but often with an objective - i.e. compromise any web servers or employees of KFC with the goal of stealing their secret recipe.

Vulnerability Research

Not focused on implementation or deployment of a technology - focused on the actual technology itself - i.e. finding vulnerabilities in the apache and PHP code, which can be used to attack every apache installation.

Terminology

Bug – A programming error or oversight that can be triggered to produce incorrect results, or cause it to behave in unintended ways

Threat - A person or group able to do harm or performing some hostile action

Vulnerability – A software or system weakness which allows the system to be used in a manner which would be considered a breach of the intended or extended result.

Exploit – A tool or code created to exploit one or more vulnerabilities for some purpose. In reality, good exploits for some vulnerabilities take weeks or months of high-skill development

Shellcode – Executable code that we get our exploit to run

Types of vulnerabilities

- Design / Architecture vulns
  - Major mistakes at a high level, ie: how components interact with each other, ie: wifi deauth
  - Hard to fix, require total rewrites
- Implementation vulns
  - Minor mistakes in implementation (off by one errors, forgetting to seed rand())
  - Software engineers making mistakes, misreading documentation, etc

Where can we find vulnerabilities

- File formats
  - Exploits in complex file formats which are decoded by complex software, ie: videos, images, docs, pdfs, anti viruses (hehe)
- Remote
  - Exploits that can compromise a system over a network
  - Ie: web server, ssh, etc
- Client side
  - Ie: web browser
- Local (priv esc)
  - Exploiting software running as a higher privilege on the same machine. Ie: attacking the kernel

Exploit Mitigations (why we teach old stuff)

Modern computers have a lot of exploit **mitigations**.

- We start of by learning how to exploit vulnerabilities that are not practically exploitable on modern systems
- There are lots of modern technologies which make exploits harder to write / getreliable, and sometimes they even make it impossible
- We need to disable a lot of these to teach exploit writing. In this course, we start with none enabled, and then we enable them again one-by-one to teach how to bypass them
- Jumping straight to exploitation on a modern OS would be like teaching primary school students university mathematics
    - You are all primary school students?

What do people do with exploits?

So you’ve found a bug? What should you do with it

- Firstly –whatever the hell you want; don’t let anyone tell you what to do with it

Vulnerabilities

- Public (full) Disclose
- Coordinated (“responsible”) vendor disclosure
- (responsible) non-disclosure

Exploits:

- Simple PoC, public disclose (ie: metasploit)
- Medium quality - commercial exploit pack (ie: celebrite)
- High Quality - fame, $$$

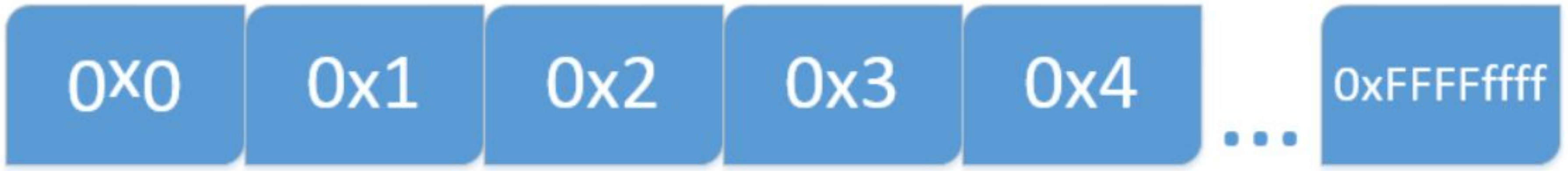
DO NOT MAKE THE MISTAKE OF ASSUMING THAT “Good intentions” WILL KEEP YOU SAFE.

Apologies

- I will rant a fair bit about memory today
- This is really important raw theory, and there isn’t much practical component for this introductory stuff. This is required knowledge before we start reversing + exploiting stuff next week
- A lot of you will know all of this, sorry :(

Thinking about memory

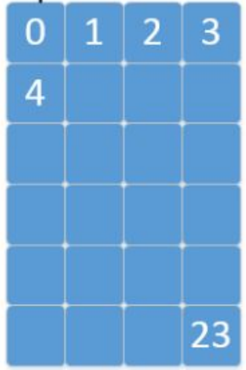
- CPUs, at a low level just see a long strip of bytes as memory
- CPUs have a few registers for super duper fast access, but these are really small and not considered part of memory
- Memory is just a range of values CPUs can access.



More on memory

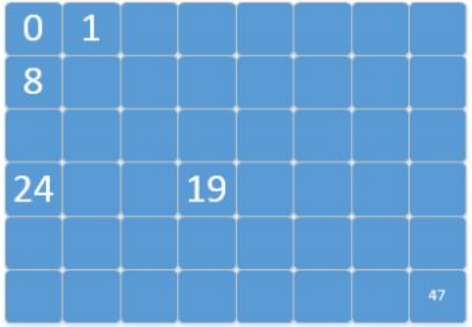
- RAM is 1d. But thinking about it in 2d can be useful.
- Memory is often not accessed 1 byte at a time.
    - We care about data structures, not bytes

Top of 32 bit stack

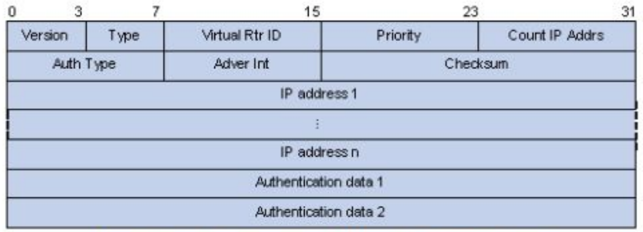


Bottom of 32 bit stack

Stacks are often visualized or thought of with a width of a pointer (ie. 32-bit stacks are 4 bytes wide, visually, 64-bit, 8)



Bottom of 64 bit stack



Network Protocols (this is VRRP) are often drawn 4 bytes wide, for convenience.

Types in memory

- C and C++ use a bunch of several core types to do work. But **all of them** are just integers of various sizes.

__usually__

- char => 1 byte
- short => 2 bytes or more
- int => normally 4 bytes
- long => at least 4 bytes, at least as long as int
- pointers => normally 4 bytes, or 8 depending on architecture (just spicy ints)
- structs => a sequence of different variations of above types

Signedness

- Won’t cover signedness, but you should google and learn how negative numbers are stored in memory, it might be helpful when reversing
  - 2’s compliment

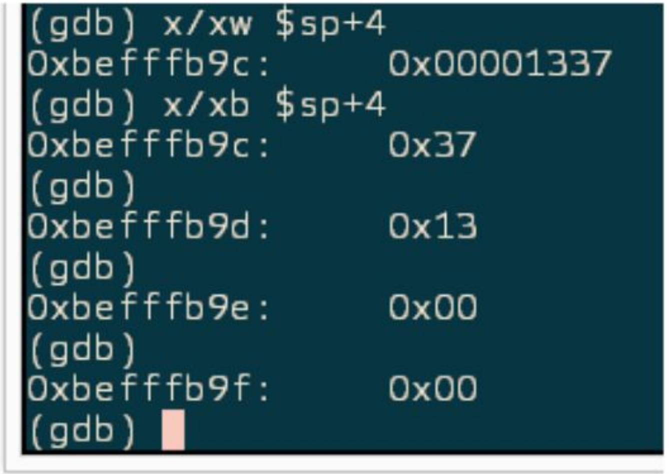
Pointers

- Pointers aren’t special or magic.
- Pointers are just another integer, where the value is the address of something else

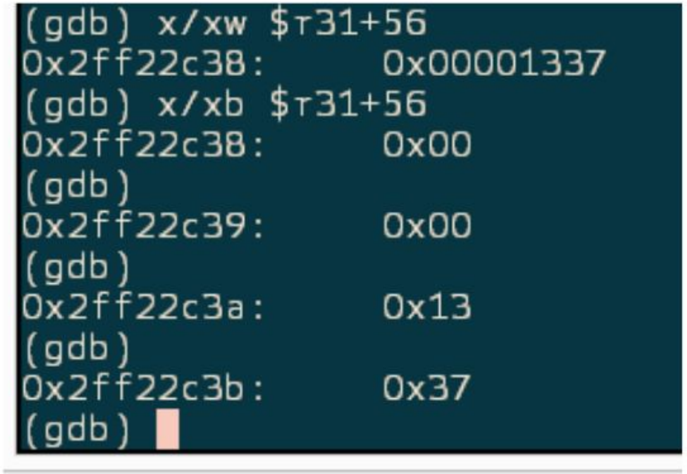
Endianness

- Each architecture has Endianness
  - Big or little endian (MIPS, PowerPC vs x86)
  - x86 is little endian
- Some architectures let you pick/switch the endianness during runtime (ARM)
- Little endian is more prevalent for reasons (laziness)
  - Used to be called “Wrong-endian” because you can be a bit more dodgy with your types and things won’t break
- Endianness only applies to memory. Not to registers

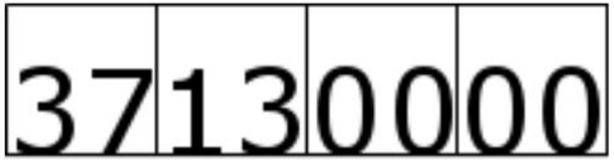
Graphically



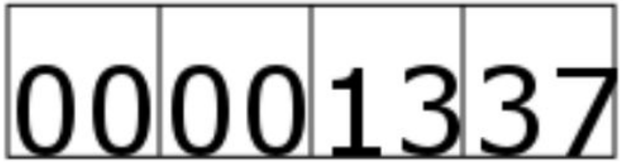
```
(gdb) x/xw $sp+4
0xbefffb9c:     0x00001337
(gdb) x/xb $sp+4
0xbefffb9c:     0x37
(gdb)
0xbefffb9d:     0x13
(gdb)
0xbefffb9e:     0x00
(gdb)
0xbefffb9f:     0x00
(gdb)
```



```
(gdb) x/xw $r31+56
0x2ff22c38:     0x00001337
(gdb) x/xb $r31+56
0x2ff22c38:     0x00
(gdb)
0x2ff22c39:     0x00
(gdb)
0x2ff22c3a:     0x13
(gdb)
0x2ff22c3b:     0x37
(gdb)
```



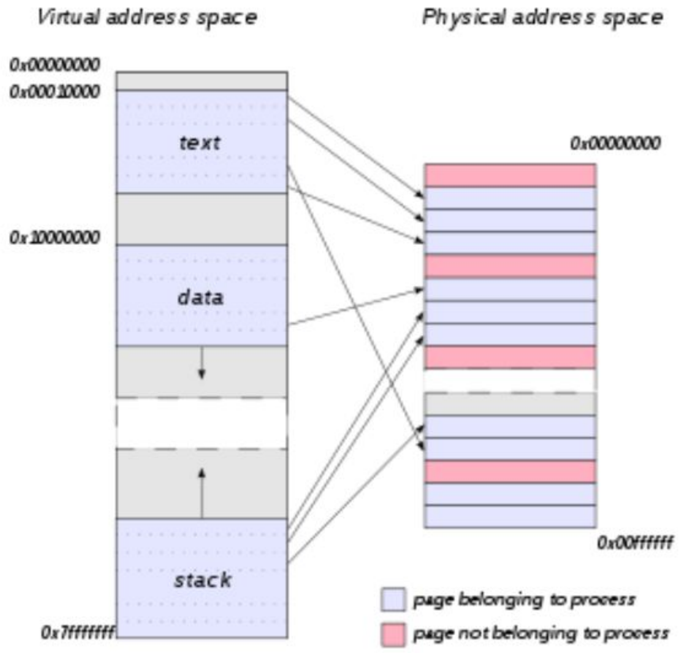
Little-endian
0x00001337
(Linux, ARMLE)



Big-endian
0x00001337
(AIX, POWER7)

ADDRESS SPACE LAYOUT

- We only worry about virtual addresses
- Virtual Address Spaces are split into different regions
- Each segment has set permissions
  - .text (code) usually r-x
  - .data (global variables) usually rw-
  - .rodata (constants) usually r–



Second half of lecture

Next part of this lecture is

- Not examinable
- ie: no need to take notes

History of hacking / exploitation (<insert meme>)