CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: Obrian 109
Time: Monday, Wednesday 5:00PM-6:20PM


新
年
快
樂
HAPPY NEW YEAR
2022
YEAR OF THE TIGER

First off, Logistics!

Classes are recorded and released publicly on YouTube
You can join Zoom remotely at
https://buffalo.zoom.us/j/96496986365?pwd=WUczY2xOTVE2bm9lcElOSnRLYllrUT09

Have a notebook in front of you
Bring your own laptop
From the second class, have the hacking environment ready

https://zzm7000.github.io/teaching/2022springcse410510/index.html

We have an online CTF platform for this class. A virtual machine will be provided if needed.

Feel free to interrupt me and ask questions
Wear a face mask!

Instructor and Teaching Assistant

Dr. Ziming Zhao
Assistant Professor, CSE
Director, CyberspAce seCuriTy and forensIcs Lab (CactiLab)

Email: zimingzh@buffalo.edu
http://zzm7000.github.io
http://cactilab.github.io
Office hours: Wednesday 3:30 PM - 4:30 PM or by appointment
https://buffalo.zoom.us/j/95299258797?pwd=QlBhbjJIUlM5WmlETmFtOE5qT1Z5dz09

Teaching assistant: Md. Armanuzzaman Tomal
Office hours: Friday 3:30 PM - 4:30 PM or by appointment
https://buffalo.zoom.us/j/95299258797?pwd=QlBhbjJIUlM5WmlETmFtOE5qT1Z5dz09

About CactiLab

Research areas:

- Embedded system and software security (Arm Cortex-M, Cortex-A, RISC-V, FPGA, etc.)
- Security in/with machine learning/deep learning
- Autonomous driving security
- Formally verify the security properties of crypto protocols and system code
- Blockchain security
- IoT hacking/CTF platforms (Roblox for hacking)

We need students at all levels for funded research, volunteer work, independent study, etc.

Students

Graduate (Master, PhD) - CSE 510 (3-credit)
Undergraduates (Sophomore, junior, senior) - CSE 410 (3-credit)

All are invited to slack ***cacti-workspace***, ***#ubcse410510-spring2022***

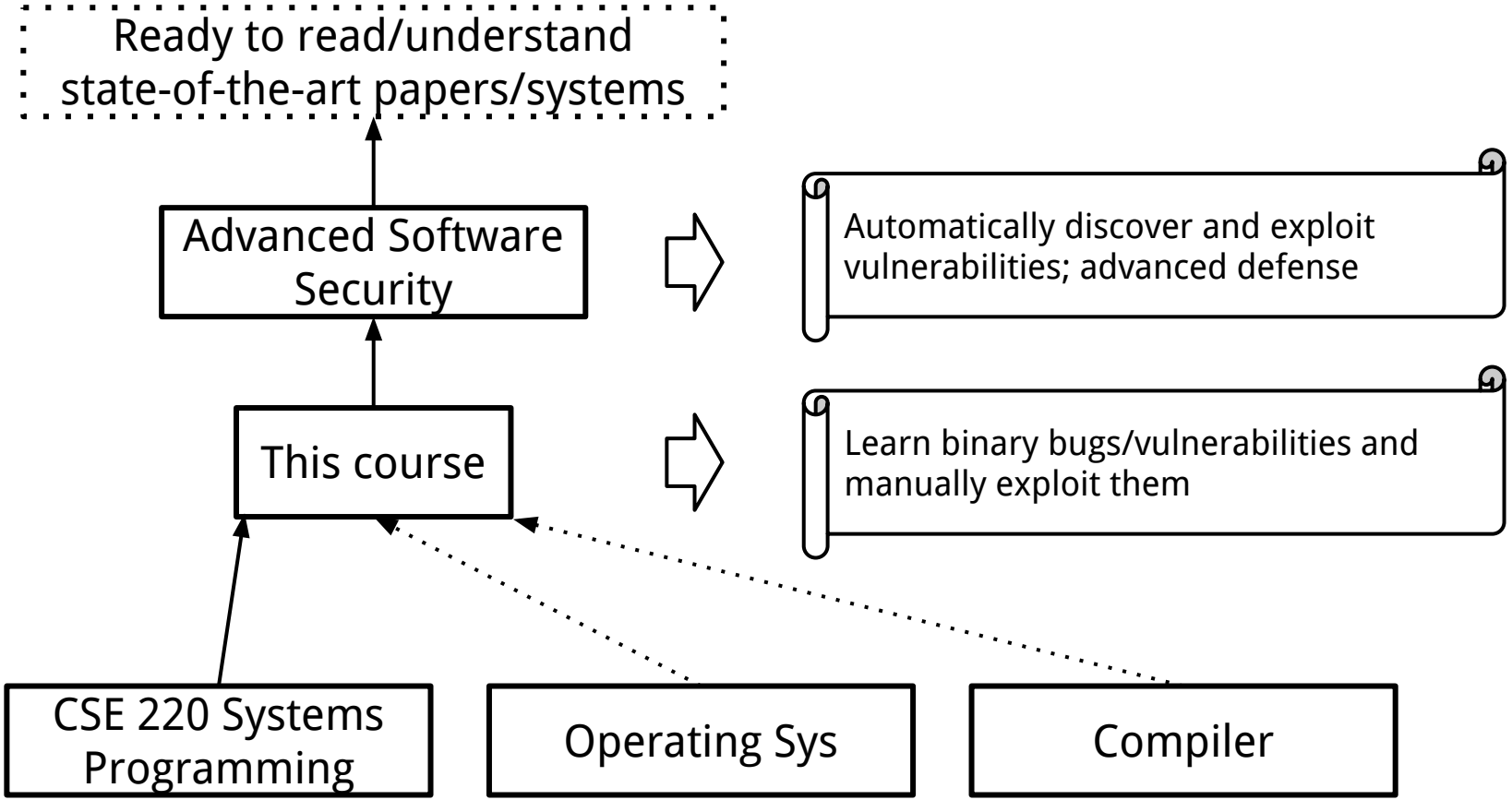
Course Goals

To provide you with good understanding of the **theories**, **principles**, **techniques** and **tools** used for binary software and system hacking and defense.

By software and system, I mean native software, binary, most likely developed in C/C++. The security of web software, Java, Python are out of the scope.

You will study, in-depth, binary reverse engineering, vulnerability classes, vulnerability analysis, exploit/shellcode development, defensive solutions, etc., to understand how to crack and protect **native** software. You will get your hands dirty.

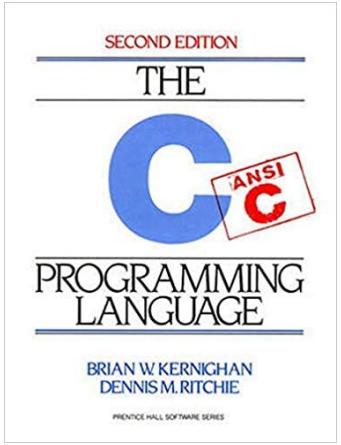
If you want to be a system/software security guy …



This week's Agenda

1. Class overview and logistics
2. Background knowledge
   a. Compiler, linker, loader
   b. x86 and x86-64 architectures and ISA
   c. Linux file permissions
   d. Set-UID programs
   e. Memory map of a Linux process
   f. System calls
   g. Environment and Shell variables
   h. Basic reverse engineering

Prerequisites



The real prerequisite:
The C Programming Language

Classes that will help you understand this class:
CSE 220 Systems Programming
CSE 421 Introduction to Operating Systems
CSE 521 Operating Systems

Other skills:
Reverse engineering (Using objdump, IDA Pro, Ghidra, etc.)
Debugging (GDB, pwngdb)
Google, reading, self-learning, getting hands dirty

Topics

Binary attack and defense using x86 and x86-64 as examples. Discover **vulnerabilities**. Develop **exploits**. Memory corruption attacks.

1. Stack-based buffer overflow
2. Defenses against stack-based buffer overflow
3. Shellcode development
4. Format string vulnerabilities
5. Heap-based buffer overflow
6. Integer overflow
7. Return-oriented programming
8. …

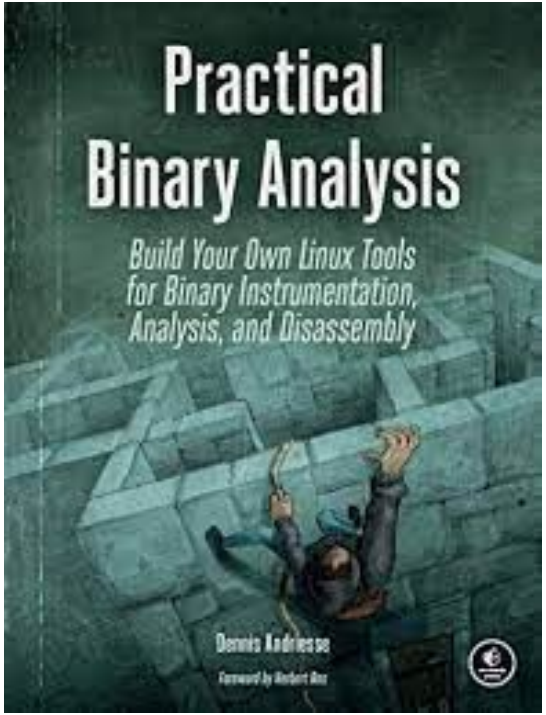
Related Books and Papers

SoK: Eternal War in Memory. IEEE S&P 2013

SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. IEEE S&P 2016

SoK: Shining Light on Shadow Stacks. IEEE S&P 2019

Practical binary analysis: build your own linux tools for binary instrumentation, analysis, and disassembly



The Hacking Environment

http://cse410.cacti.academy/

Only UB students can access this website. If you are off-campus, you need to VPN to connect to UB network to access

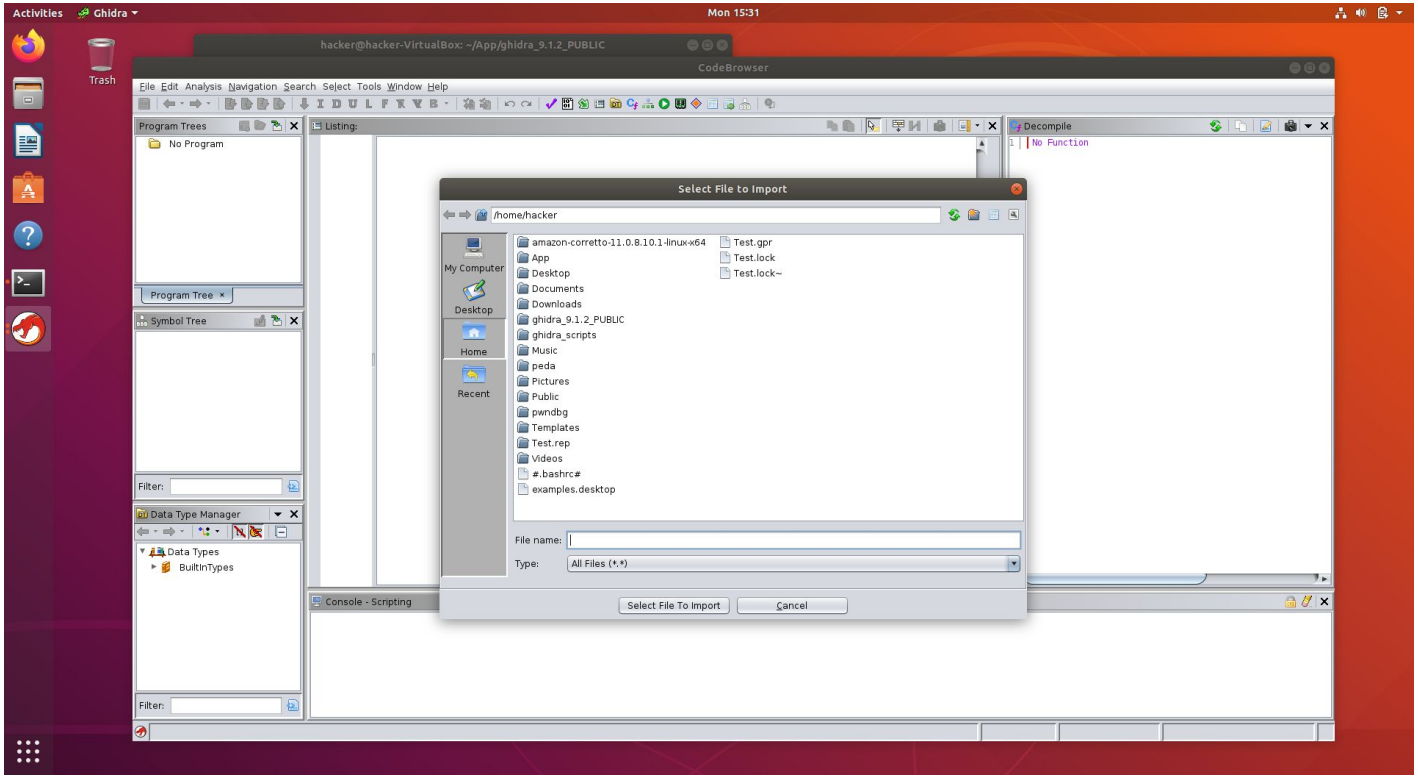
Register an account with your UB username and email address.

Welcome to CSE410/510 CTF Platform!

CSE410/510 CTF Platform for course practice probelms on 0x86 architecture. The platform was created by Ziming Zhao and members of CactiLab at the University at Buffalo.

The Backup VM

User: CSE610VM pwd: hacker link will be provided later



The Hacking Environment

Intel x86
x86-64, a.k.a amd64
Linux (Ubuntu)

Pwngdb
Pwntools
GDB peda
NSA Ghidra
Binary Ninja

Homework

Reading: book chapter, whitepaper, paper, blog, etc.
Hands-on: hacking, debugging, etc.

Submit before the Wednesday class on UBLearns. We will discuss homework at the beginning of each class.

30% penalty if you submit within 10 mins after class starts. 0 points after 10 mins.

0 points for homework if plagiarising one task is found. No exceptions.

Hacking Assignment Rules

- For each hacking assignment, you will submit your exploit, a simple write-up, and screenshots to show it works
  - Simple write-up:
    - Briefly describe how you solve the challenge
    - Mention who you worked with if any in the write-up

- Discussion is encouraged. But, you cannot share your code, exploits, write-ups to your classmates or post them online.

Exams and Capture-the-Flag (CTF) Hacking

Written midterm: with Midterm CTF
Written final: Take home

Midterm CTF: 2 hours
Final CTF: 3 hours

Grades

Area	No. Items	Points per Item	Points for Area
Homework	14	45	630
Exams	2		160
Written Midterm	1	80	
Written Final	1	80	
CTFs	2		200
Midterm CTF	1	80	
Final CTF	1	120	
Attendance	14	1	14
Anonymous Course Evaluation Bonus	2	10	20
Total			1024

Table 2: Grades Breakdown

Points	Grade
930 -	A
900 - 930	A-
870 - 900	B+
830 - 870	B
800 - 830	B-
770 - 800	C+
700 - 770	C
670 - 700	D+
600 - 670	D
0 - 600	F

Table 3: Final Letter Grades

Academic Integrity

- Discussion is encourage. But, you cannot share your code, exploits to your classmates or post them online.

- The university, college, and department policies against academic dishonesty will be strictly enforced. To understand your responsibilities as a student read: UB Student Code of Conduct.

- Plagiarism or any form of cheating in homework, assignments, labs, or exams is subject to serious academic penalty.

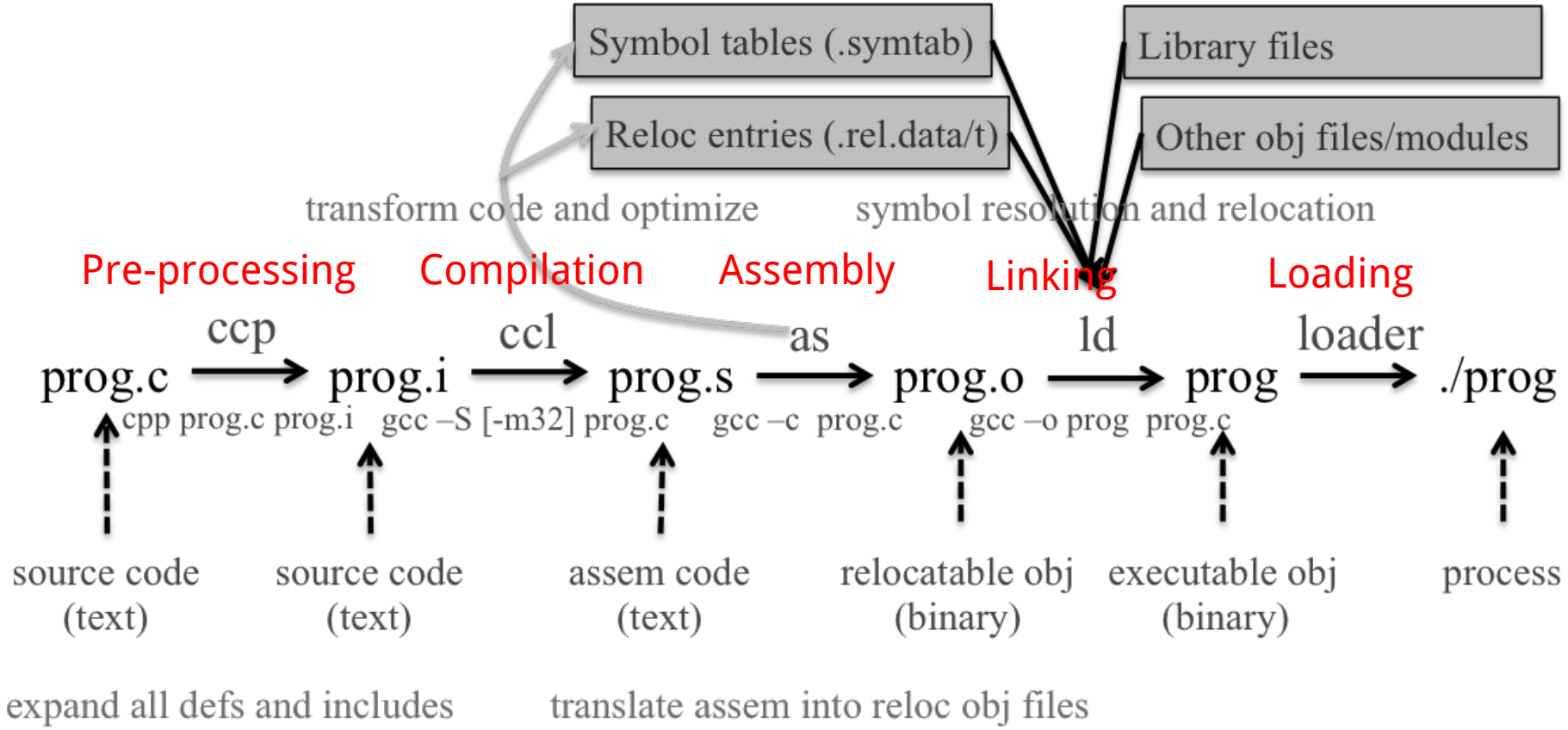
- Any violation of the academic integrity policy will result in a 0 on the homework, lab or assignment, and even an **F** or **>F<** on the final grade. And, the violation will be reported to the Dean's office.

Ethical Hacking

- Do not attempt to violate the law.
- If you discover real-world vulnerabilities using the knowledge you learn from this class, report the vulnerabilities responsibly.

Background Knowledge: Compiler, linker, and loader

From a C program to a process



Loading and Executing a Binary Program on Linux

Validation (permissions, memory requirements etc.)

Operating system starts by setting up a new process for the program to run in, including a virtual address space.

The operating system maps an interpreter into the process’s virtual memory.

Interpreter, e.g., /lib/ld-linux.so in Linux

The interpreter loads the binary into its virtual address space (the same space in which the interpreter is loaded).

It then parses the binary to find out (among other things) which dynamic libraries the binary uses.

The interpreter maps these into the virtual address space (using *mmap* or an equivalent function) and then performs any necessary last-minute relocations in the binary's code sections to fill in the correct addresses for references to the dynamic libraries.

Compiling a C program behind the scene (code/add)

add.c

```
#include "add.h"

#define BASE 50

int add(int a, int b)
{ return a + b +
BASE;}
```

add.h

```
#ifndef ADD_H
#define ADD_H

int add(int, int);

#endif
```

main.c

```
/* This program has an integer overflow vulnerability. */
#include "add.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define USAGE "Usage: add a b\n"

int main(int argc, char *argv[])
{
  int a = 0;
  int b = 0;

  if (argc != 3)
    {
      printf(USAGE);
      return 0;}

  a = atoi(argv[1]);
  b = atoi(argv[2]);
  printf("%d + %d = %d\n", a, b, add(a, b));
}
```

```
gcc -Wall -save-temps -P -m32 -O2 add.c main.c -o add
```

```
gcc -Wall -save-temps -P -O2 add.c main.c -o add64
```