CSE 418/518: Software Security

Fall 2023
3 Credits
Meeting once a week; 2 hours 50 minutes

Instructor

Dr. Ziming Zhao
Email: zimingzh@buffalo.edu
Office: 338B, Davis Hall
Office hours: Monday 3:30 PM - 4:30 PM or by appointment

Course Description

This course is designed to provide students with good understanding of the theories, principles, techniques and tools used for software and system hacking and hardening. Students will study, in-depth, binary reverse engineering, vulnerability classes, vulnerability analysis, exploit and shellcode development, defensive solutions, etc. to understand how to crack and protect native software. In particular, this class covers offensive techniques including stack-based buffer overflow, heap security, format string vulnerability, return-oriented programming, etc. This class also covers defensive techniques including canaries, shadow stack, address space layout randomization, control-flow integrity, seccomp, etc. A key part of studying security is putting skills to the test in practice. In this class the progress of students are evaluated by hacking binary challenges.

Learning Outcomes and ABET Mapping (Computing Programs)

The direct learning outcomes of this course include:

1. Identify the aforementioned types of software vulnerabilities at source code and binary code level
2. Ethically exploit the discovered software vulnerabilities
3. Understand existing defensive mechanisms to defeat exploitation

Connections of ABET Computing Programs learning outcomes include:

1. By analyzing the source code and/or binary of a program and looking for its weaknesses, students acquire an ability to analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.

2. By understanding the weaknesses of software, learning ethical hacking, and implement the exploits, students acquire an ability to design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.

3. By articulating their exploitation methodologies and finishing the experiment reports, students acquire an ability to communicate effectively in a variety of professional contexts.

4. By learning the history of real-world cyberspace incidents and their social and legal implications, students acquire an ability to recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.

Topics

1. Background knowledge
   (a) Principles of compiler, linker and loader
   (b) x86 and x86-64 architectures
   (c) x86 and x86-64 instruction sets
   (d) Embedded and IoT architectures, e.g., ARM Cortex-A and Cortex-M
   (e) Assembly and reversing engineering with AT&T and Intel syntax
   (f) Linux file permissions
   (g) Set-UID programs
   (h) ELF file format, PLT, GOT
   (i) Memory map of a Linux process (x86 and x86-64)
   (j) System calls
   (k) Environment variables
   (l) Tools: `gcc`, `gdb`, `pwndbg`, `objdump`, `ltrace`, `strace`, `env`, `readelf`, `/proc/PID/maps`, `pmap`, `tmux`, `IDA Pro`, `Ghidra`, `binary ninja`

2. Stack-based bufferoverflow
   (a) C local and global variables
   (b) C calling conventions (x86 and x86-64)
   (c) How stack works
   (d) Overflow local variables
   (e) Overflow RET; shellcode in exploits; shellcode in environment variable; shellcode in program arguments;
   (f) Frame pointer attack
   (g) Return-to-libc attacks
   (h) Tools: `pwntools`

3. Defenses against stack-based buffer overflow

(a) Base and bound check
(b) Data Execution Prevention (DEP)
(c) Stack Canaries: GCC implementation for x86 and x86-64
(d) Shadow stack: (1) traditional; (2) parallel
(e) SafeStack
(f) Address space layout randomization (ASLR) and how to bypass it
(g) FORTIFY_SOURCE
(h) Tools: `ldd`

4. Developing Shellcode
   (a) Writing, compiling, and testing x86 and x86-64 assembly code
   (b) System calls on Intel (x86 and x86-64)
   (c) Constraints on shellcoding: (1) zero-free shellcode; (2) ASCII-only shellcode; (3) English shellcode
5. Format string vulnerability
   (a) Format string
6. Heap security
   (a) Dynamic allocator, ptmalloc, tcache, `malloc()`, `free()`
   (b) Use after free (UAF) vulnerabilities
   (c) Double free vulnerabilities
7. Integer overflow vulnerability
   (a) Integer overflow vulnerability
8. Return-oriented programming
   (a) ROP
   (b) Blind ROP
   (c) Jump-oriented programming
   (d) Control-Flow Integrity (CFI)
   (e) Tools: `ROPgadget`, `pwntools`
9. Data-oriented attacks
   (a) Direct data manipulation
   (b) Data-oriented programming
   (c) Data-Flow Integrity (DFI)
10. Race conditions
    (a) Race in file systems
    (b) Process and thread
    (c) Race in memory

Grading Policy

Students will be evaluated on their performance on the homework and CTFs. Attendance check will be performed in each class. Table 2 shows the grade breakdown.

Area	No. Items	Points per Item	Points for Area
Homework	14	45	630
Exams (CTFs)	2		360
Midterm Exam (CTF)	1	160	
Final Exam (CTF)	1	200	
Attendance	14	1	14
Anonymous Course Evaluation Bonus	2	10	20
Total			1024

Table 1: Grades Breakdown

Grading Chart

Table 3 presents the grading chart for the undergraduate and graduate portion, respectively..

418 (Undergraduate)		518 (Graduate)	
Points	Grade	Points	Grade
874 -	A	924 -	A
850 - 874	A-	900 - 924	A-
820 - 850	B+	870 - 900	B+
780 - 820	B	830 - 870	B
750 - 780	B-	800 - 830	B-
720 - 750	C+	770 - 800	C+
650 - 720	C	700 - 770	C
550 - 650	D	600 - 700	D
0 - 550	F	0 - 600	F

Table 2: Final Letter Grades

Prerequisite

A prerequisite for undergraduate students is CSE 220 Systems Programming, whereas there is no prerequisite for graduate students. Students are expected to have a background in the C programming language (CSE 220 Systems Programming or equivalent). Solid background from the following classes helps significantly: (1) CSE 341 Computer Organization; (2) CSE 365 Introduction to Computer Security; and (3) CSE 421/521 Operating Systems. The instructor strives to keep this class self-contained.

Textbooks

There is no required textbooks for this course. The instructor will send out book chapters and other reading materials throughout the semester. Some of the teaching materials of this course are based on the following books:

- Randal Bryant, David O'Hallaron. Computer Systems: A Programmer's Perspective, 3rd Edition.
- Dennis Andriesse. Practical Binary Analysis: Build Your Own Linux Tools for Binary Instrumentation, Analysis, and Disassembly.

Papers

Throughout the semester, we will discuss the following papers:

- Laszlo Szekeres, Mathias Payer, Tao Wei, Dawn Song. SoK: Eternal War in Memory. IEEE Security and Privacy 2013.
- Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, Giovanni Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. IEEE Security and Privacy 2016.
- Hovav Shacham. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). ACM CCS 2007.
- Tyler Bletsch, Xuxian Jiang, Vince W. Freeh, Zhenkai Liang. Jump-Oriented Programming: A New Class of Code-Reuse Attack. ACM AsiaCCS 2011.
- Andrea Bittau, Adam Belay, Ali Mashtizadeh, David Mazieres, Dan Boneh. Hacking Blind. IEEE Security and Privacy 2014.

Academic Integrity (AI)

Academic integrity is critical to the learning process. It is your responsibility as a student to complete your work in an honest fashion, upholding the expectations your individual instructors have for you in this regard. The ultimate goal is to ensure that you learn the content in your courses in accordance with UB's academic integrity principles, regardless of whether instruction is in-person or remote. As an institution of higher learning, UB expects students to behave honestly and ethically at all times, especially when submitting work for evaluation in conjunction with any course or degree requirement. Thank you for upholding your own personal integrity and ensuring UB's tradition of academic excellence. The academic integrity policy is available at `https://buffalo.edu/academic-integrity` and `https://engineering.buffalo.edu/computer-science-engineering/information-for-students/graduate-program/cse-graduate-academic-policies/cse-academic-integrity-policy.html`

Course-specific AI Policy

Students are allowed to discuss homework assignments. However, students are NOT allowed to share code, exploits, write-ups, and homework with each other. Plagiarism or any form of cheating in homework or exams (CTFs) is subject to serious academic penalty. All AI violations will be reported to the UB Office of Academic Integrity. There is a zero tolerance policy in this class.

- A minor AI violation of a specific homework assignment (i.e., plagiarism on one question) may result in a `0` on that assignment.
- More serious AI violation may result in downgrade of the final grade and even an `F` or `>F<` on the final grade, e.g., falsification.
- The CSE department has a policy to upgrade the penalty to `F` on the final grade for a second AI violation of any degree.

The instructor takes AI violations very seriously. For example, in Spring 2023, one student received an `F` grade for an AI violation, and four other students received penalties of varying severity.

Syllabus Update

Information in the syllabus may be subject to change with reasonable advance notice.