CY 5770 Software Vulnerabilities and Security

4 Credits

Instructor

Dr. Ziming Zhao
Email: z.zhao@northeastern.edu
Office: Room 513, 177 Huntington Ave, Boston, MA

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.

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) x86 and x86-64 assembly and reversing engineering with Intel syntax
    - (f) Linux file permissions
    - (g) Set-UID programs
    - (h) ELF file format, .plt, .got, lazy binding
    - (i) Memory map of a Linux process (x86 and x86-64)
    - (j) System calls

   (k) Environment variables
   (l) Linux shell commands and tricks, piping
   (m) Tools: `gcc`, `gdb`, `pwndbg`, `objdump`, `ltrace`, `strace`, `env`, `readelf`, `/proc/PID/maps`, `pmap`, `tmux`, `IDA Pro`, `Ghidra`, `binary ninja`, `tee`

2. Stack-based buffer overflow
   (a) C local and global variables
   (b) C calling conventions (x86 and x86-64)
   (c) How stack works
   (d) Overflow local variables
   (e) Overflow return addresses on stack
   (f) Injecting in local buffers, in environment variables, in program arguments
   (g) Frame pointer attack (overwrite saved EBP/RBP)
   (h) Return-to-libc attacks
   (i) 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) Position Independent Executable (PIE)
   (g) Address space layout randomization (ASLR) and how to bypass it
   (h) FORTIFY_SOURCE
   (i) Seccomp (syscall firewall)
   (j) 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) Non-printable, non-alphanumeric shellcode
   (d) Research development: English shellcode, DNA shellcode

5. Format string vulnerability
   (a) C function with variable arguments
   (b) Difference with C++ function overloading

    (c) Format string, e.g., %n
    (d) Information leakage with format string vulnerability (memory read)
    (e) Format string vulnerability for memory write, e.g., .got, .fini
    (f) RELRO
    (g) Control-flow bending for Turing-complete printf-oriented programming

6. Heap security and exploitation
    (a) What is heap? Dynamic allocator and its interfaces, e.g., `malloc()`, `free()`
    (b) tcache, chunks and metadata
    (c) Use after free (UAF) vulnerabilities
    (d) Double free vulnerabilities
    (e) Metadata corruption
    (f) Safe-linking

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) Hardware-supported CFI
    (f) 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 1 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	10	1	10
Anonymous Course Evaluation Bonus	2	12	24
Total			1024

Table 1: Grades Breakdown

Grading Chart

Table 2 presents the grading chart.

5770 (Undergraduate)		5770 (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

Students are expected to have a background in the C programming language. Solid background from the following classes helps significantly: (1) Computer Organization; (2) Introduction to Computer Security; and (3) 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.
- Per Larsen, Ahmad-Reza Sadeghi. The Continuing Arms Race: Code-Reuse Attacks and Defenses.

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.
- N Carlini, A Barresi, M Payer, D Wagner, TR Gross. Control-Flow bending: On the effectiveness of Control-Flow integrity. USENIX Security Symposium, 2015

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 NEU's academic integrity principles, regardless of whether instruction is in-person or remote. As an institution of higher learning, NEU 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 NEU's tradition of academic excellence. The academic integrity policy is available at `https://osccr.sites.northeastern.edu/academic-integrity-policy/`

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 NEU Office of Student Conduct and Conflict Resolution. 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 college and university have policies to upgrade the penalty, which is independent of the course policy.

Syllabus Update

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