VIRGINIA TECH · DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Software Fuzzing for Security

ECE 4974 · Independent Study

3 Credit Hours **Pass / Fail** Instructor: Prof. Binoy Ravindran · binoy@vt.edu

COURSE DESCRIPTION

Software fuzzing is among the most effective and widely deployed techniques for automatically discovering security-critical defects in software, including memory-safety errors, crashes, and undefined behavior. This independent study introduces the student to modern coverage-guided fuzzing and to the dynamic sanitization techniques that make fuzzing campaigns effective at surfacing latent vulnerabilities.

The student will study the principles behind fuzzing and sanitization, gain hands-on expertise with industry-standard tools — the AFL++ fuzzer and the LLVM sanitizers AddressSanitizer (ASan) and UndefinedBehaviorSanitizer (UBSan) — and apply them to real-world target programs, including programs selected from Google's OSS-Fuzz project. The study culminates in a fuzzing campaign in which the student instruments a target, runs and tunes the fuzzer, triages discovered crashes, and analyzes and reports the results.

EXPECTED TIME COMMITMENT & DELIVERABLES

As a 3-credit-hour study, students should expect to devote approximately 9–12 hours per week across the term, including a weekly one-hour meeting with the instructor to review progress, discuss readings, and plan the next milestone. Significant deliverables are:

- **Weekly progress reports / lab notebook** documenting reading, environment setup, experiments, and results.
- **Three hands-on lab milestones:** (1) building and instrumenting a target with ASan and UBSan; (2) standing up and running an AFL++ coverage-guided campaign; (3) crash triage, deduplication, and root-cause analysis.
- **A capstone fuzzing campaign** on an instructor-approved open-source target, with a written technical report and a short oral presentation of findings.

PREREQUISITES

Junior or senior standing in Computer Engineering or Computer Science (or instructor permission). Working proficiency in C/C++ and the Linux command line is assumed, along with basic familiarity with the compilation and build process (compilers, linkers, makefiles). Prior coursework in operating systems or computer organization is helpful but not required.

STUDENT LEARNING OUTCOMES

Upon successful completion of this independent study, the student will be able to:

- **Specify** software fuzzing techniques, including coverage-guided greybox fuzzing, mutation strategies, seed corpus design, and instrumentation.
- **Implement** software fuzzing campaigns using industry-standard tools including AFL++ (target instrumentation, harness construction, corpus management, and campaign tuning).
- **Implement** address- and behavior-sanitization techniques using the LLVM sanitizers AddressSanitizer (ASan) and UndefinedBehaviorSanitizer (UBSan).
- **Analyze** the results of fuzzing and sanitization campaigns — interpreting coverage and crash data, triaging and deduplicating findings, and performing root-cause analysis of detected vulnerabilities.

READING LIST & RESOURCES

There is no required textbook. Readings are drawn from tool documentation, foundational papers, and curated online material distributed through Canvas:

- AFL++ documentation and the AFL++ paper (Fioraldi et al., "AFL++: Combining Incremental Steps of Fuzzing Research"): aflplus.plus
- LLVM AddressSanitizer (ASan) documentation: clang.llvm.org/docs/AddressSanitizer.html
- LLVM UndefinedBehaviorSanitizer (UBSan) documentation: clang.llvm.org/docs/UndefinedBehaviorSanitizer.html
- Manès et al., "The Art, Science, and Engineering of Fuzzing: A Survey," IEEE TSE 2021: doi.org/10.1109/TSE.2019.2946563
- Google fuzzing tutorials and guidance: github.com/google/fuzzing
- Google OSS-Fuzz project (continuous fuzzing of open-source software; source of target programs): github.com/google/oss-fuzz

GRADING — PASS / FAIL

This course is graded on a Pass/Fail basis. A grade of **Pass** requires satisfactory completion of all deliverables, consistent weekly engagement, and a capstone report and presentation judged to meet expectations. The components below are weighted to guide the assessment of satisfactory progress:

COMPONENT	WEIGHT	BASIS
Weekly reports & engagement	20%	Lab notebook, meeting participation, and steady progress.
Lab milestones (3)	45%	Sanitizer instrumentation; AFL++ campaign; crash triage & analysis.
Capstone campaign & report	25%	Fuzzing campaign on an approved target with written technical report.
Final presentation	10%	Oral presentation and discussion of methodology and findings.

TENTATIVE SCHEDULE — 12 WEEKS

WEEKS	TOPIC / ACTIVITY	MILESTONE
1	Software vulnerabilities & memory safety; introduction to fuzzing; environment and toolchain setup.	Toolchain ready
2–3	Dynamic sanitization: building and instrumenting targets with ASan and UBSan; reading sanitizer reports.	Lab 1
4–5	Coverage-guided greybox fuzzing principles; instrumentation, mutation, and feedback.	—
6–7	AFL++ in depth: harness construction, seed corpus design, persistent mode, campaign tuning; combining AFL++ with sanitizers.	Lab 2
8–9	Crash triage, deduplication, coverage analysis, and root-cause analysis of detected defects.	Lab 3
10–11	Capstone fuzzing campaign on an approved open-source target; iterative tuning and analysis.	Report draft
12	Final technical report and oral presentation of findings.	Final deliverable