Week 1: Lecture A

Course Introduction

Monday, January 8, 2024
Reminders

- Be sure to join the course **Canvas** and **Piazza**
  - See links at top of course page
  - [cs.utah.edu/~snagy/courses/cs5963/](cs.utah.edu/~snagy/courses/cs5963/)

- Trouble accessing? See me after class!
  - Or email me at: **snagy@cs.utah.edu**
Today’s Class

- Welcome to CS 5963/6963 😊

- Course Overview

- What is software testing?
  - How does it work?
  - Why do we use it?

- Ethics and Academic Integrity
About Me

Stefan Nagy
Assistant Professor, KSoC

Co-founder and Co-director:

SSG UTAH SOFTWARE SECURITY GROUP
SCHOOL OF COMPUTING | THE UNIVERSITY OF UTAH

Places I’ve been:

University of Utah, 2022–now
Virginia Tech, Ph.D. 2016–2022
Univ. of Illinois, B.S. 2012–2016

cs.utah.edu/~snagy
twitter.com/snagycs
@snagy@infosec.exchange
Our work: systems and software security, binary analysis, fuzzing
Course Overview
What brought you here?
Course Goals

- Help you become **better researchers**
- Expose you to **different perspectives**
- Experience with **state-of-the-art tools**
- Get course credit so you can graduate?
- All while learning about **software testing**
Course Components

- Reading & evaluating research
  - Contextualize
  - Pros vs. cons
  - Contribution
  - Summarizing
  - Identify assumptions
Course Components

- Reading & evaluating research
  - Contextualize
  - Pros vs. cons
  - Contribution
  - Summarizing
  - Identify assumptions

- Conducting & presenting research
  - Identify an open problem and solve it
  - Develop new tooling and release it
  - Evaluate and disseminate your work
  - Help society by finding security bugs
Course Format

- **Meetings:** Mondays & Wednesdays at 1:25 – 2:45 PM
- **Locations:** WEB L114 (class), MEB 3446 (office hours)
  - Office hours held from 2:45 – 3:30 PM following lecture
- **20 – 30 min:** instructor-led lecture on topic of the day
  - Slides will be posted on the course website Schedule
- **40 – 50 min:** student-led paper presentation & discussion
  - One or two papers per day related to the lecture topic
Course Website

cs.utah.edu/~snagy/courses/cs5963

CS 5963/6963: Applied Software Security Testing

This special topics course will dive into today’s state-of-the-art techniques for uncovering hidden security vulnerabilities in software. Introductory fuzzing exercises will provide hands-on experience with industry-popular security tools such as AFL+ and AddressSanitizer, culminating in a final project where you’ll work to hunt down, analyze, and report security bugs in a real-world application or system of your choice.

This class is open to graduate students and upper-level undergraduates. It is recommended you have a solid grasp over topics like software security, systems programming, and C/C++.

Learning Outcomes: At the end of the course, students will be able to:

- Design, implement, and deploy automated testing techniques to improve vulnerability on large and complex software systems.
- Assess the effectiveness of automated testing techniques and identify why they are well- or ill-suited to specific codebases.
- Distill testing outcomes into actionable remediation information for developers.
- Identify opportunities to adapt automated testing to emerging and/or unconventional classes of software or systems.
- Pinpoint testing obstacles and synthesize strategies to overcome them.
- Appreciate that testing underpins modern software quality assurance by discussing the advantages of proactive and post-deployment software testing efforts.
Schedule

- **Weeks 1 – 3:** Course Intro & Systems Research 101

- **Weeks 4 – 9:** Fundamentals of Software Fuzzing
  - Three (relatively easy) labs
  - Semester Project begins on Week 6

- **Weeks 10 – 12:** Emerging Enhancements in Fuzzing

- **Weeks 13 – 16:** New Frontiers & Project Presentations
Grading

- **10%** – Attendance & Paper Discussions
- **10%** – Paper Presentations (one per student)
- **15%** – Lab 1: Beginner Fuzzing
- **15%** – Lab 2: Crash Triage
- **15%** – Lab 3: Harnessing
- **35%** – Final Project
Attendance & Participation

- **Requirement 1:** Show up to class
  - Contact me about absences in advance
Attendance & Participation

- **Requirement 1:** Show up to class
  - Contact me about absences in advance

- **Requirement 2:** Participate during other students’ presentations
  - Ask thoughtful questions
  - Understand the science
  - **Help your classmates learn**
Paper Presentations

- **Two paper presentations** per lecture, followed by **5–10 minute discussions**
Paper Presentations

- **Two paper presentations** per lecture, followed by **5–10 minute discussions**

- **Audience:** you are **not** required to read the paper
  - ... but you are required to **participate** in the discussion!
**Paper Presentations**

- **Two paper presentations** per lecture, followed by **5–10 minute discussions**

- **Audience**: you are **not** required to read the paper
  - ... but you are required to **participate** in the discussion!

- **Presenters**: your job is to **teach us the paper**
  - Summarizing
  - Contextualize
  - Pros vs. cons
  - Contributions
  - Key assumptions
  - Prepare a short slide deck (you can get “inspired” from existing presentations)
  - 15 – 20 minute presentation (with a 5–10 minute audience discussion to follow)
Paper Presentations

- **Signup sheet** available on course website (must use **UofU gcloud** account)
- **38 fuzzing papers** from top venues in security, software engineering, and some workshops
- Choose one paper by **Monday, January 22**

---

**CS 5963/6963: Applied Software Security Testing**

This special topics course will dive into today’s state-of-the-art techniques for uncovering hidden security vulnerabilities in software. Introductory fuzzing exercises will provide hands-on experience with industry-popular security tools such as AFL++ and AddressSanitizer, culminating in a final project where you’ll work to hunt down, analyze, and report security bugs in a real-world application or system of your choice.

This class is open to graduate students and upper-level undergraduates. It is recommended you have a solid grasp over topics like software security, systems programming, and C/C++.

**Learning Outcomes:** At the end of the course, students will be able to:

- Design, implement, and deploy automated testing techniques to improve vulnerability on large and complex software systems.
- Assess the effectiveness of automated testing techniques and identify why they are well- or ill-suited to specific codebases.
- Distill testing outcomes into actionable remediation information for developers.
- Identify opportunities to adapt automated testing to emerging and/or unconventional classes of software or systems.
- Propose testing strategies to overcome them.
- Appreciate that testing underpins modern software quality assurance by discussing the advantages of proactive and post-deployment software testing efforts.
Hands-on Labs

- Three (relatively easy) labs to be completed **solo**
  - **Lab 1**: Beginner fuzzing
  - **Lab 2**: Crash triage
  - **Lab 3**: Target harnessing
Hands-on Labs

- Three (relatively easy) labs to be completed solo
  - **Lab 1**: Beginner fuzzing
  - **Lab 2**: Crash triage
  - **Lab 3**: Target harnessing

- Paced with the introductory content from Weeks 4–9
  - Apply the techniques you’ve learned in class
  - Get familiar with state-of-the-art tools like **AFL** and **ASAN**
  - **Deliverables**: a short report (1–3 pages) of what you’ve learned
Hands-on Labs

- Three (relatively easy) labs to be completed solo
  - Lab 1: Beginner fuzzing
  - Lab 2: Crash triage
  - Lab 3: Target harnessing

- Paced with the introductory content from Weeks 4–9
  - Apply the techniques you’ve learned in class
  - Get familiar with state-of-the-art tools like AFL and ASAN
  - Deliverables: a short report (1–3 pages) of what you’ve learned

- Designed to prepare you for the Semester Final Project
Objective: **uncover new bugs in a real-world program**

Team up in groups of 1 – 4

Select an “interesting” target program of your choice; e.g.:
- Popular applications
- Nintendo emulators
- Old computer games
- MacOS Rosetta
- **GET CREATIVE!**

**Figure out how to fuzz** your target, **find bugs**, and **responsibly disclose them**

**Deliverables:** a report, disclosure of bugs, and open-source your team’s fuzzer
Objective: uncover new bugs in a real-world program

Team up in groups of 1 – 4

Select an “interesting” target program for your investigation:
- Popular applications
- Nintendo emulators
- Old computer games
- MacOS Rosetta
- GET CREATIVE!

Figure out how to fuzz your target, find bugs, and responsibly disclose them.

Deliverables: a report, disclosure of bugs, and open-source your team’s fuzzer

5-minute project proposal on Feb. 28

Final presentations at semester’s end

You have full creative liberty—get creative and fuzz something fun!
Key Dates

- **Jan. 15**: No class (MLK Jr. Day)
- **Jan. 22**: Select one paper to present
- **Feb. 07**: Lab 1 due
- **Feb. 14**: Lab 2 due
- **Feb. 19**: No class (President’s Day)
- **Feb. 28**: Lab 3 due
- **Feb. 28**: 5-minute project proposals
- **Mar. 04 & 06**: No class (Spring Break)
- **Apr. 17 & 22**: Final project presentations
Lateness Policy

- Assignments will be posted on course website
  - See cs.utah.edu/~snagy/courses/cs5963/assignments

- Due by **11:59 PM** on the specified deadline date
  - Late assignments will **not** be accepted

- If you are sick / traveling / abducted by aliens...
  - Try to keep me posted and we will figure something out
Course Materials

- No textbook is required for this course

- Some excellent resources on fuzzing are:
  - *The Fuzzing Book* by Zeller, Gopinath, Böhme, Fraser, and Holler
  - *Fuzzing Against the Machine* by Antonio Nappa and Blazquez

- Other general computer security textbooks:
  - *Introduction to Computer Security* by Goodrich and Tamassia
  - *Security Engineering* by Ross Anderson

- These are linked on the course syllabus
  - [cs.utah.edu/~snagy/courses/cs5963/]
No Exams
Questions?
A Brief Overview of Software Testing
Our world depends on software...

- Personal Technology
- Infrastructure & Industry
- Military and Government
... and software security is a *nightmare*
... and software security is a **nightmare**

Amnesty says NSO's Pegasus used to hack phones of Palestinian rights workers

'A cyber-attack disrupted my cancer treatment'

Cyber-attack hits UK internet phone providers

Solarwinds hackers are targeting the global IT supply chain, Microsoft says

Janesville school district hit by ransomware attack

New York subway hacked in computer breach linked to China
Why is software insecure?

- Modern applications accept many sources of input:
  - Files
  - Arguments
  - Environment variables
  - Network packets
Why is software insecure?

- Modern applications accept many sources of input:
  - Files
  - Arguments
  - Environment variables
  - Network packets

- Developer mistakes create **software bugs**
  - Pointer mismanagement, bounds checking, etc.
Why is software insecure?

- Modern applications accept many sources of input:
  - Files
  - Arguments
  - Environment variables
  - Network packets

- Developer mistakes create software bugs
  - Pointer mismanagement, bounds checking, etc.

- Many bugs are exploitable by attackers
  - Denial of service, info leakage, code execution
Software Security Vulnerabilities

- Denial of Service: 16%
- Code Execution: 25%
- Overflow: 6%
- Cross Site Scripting: 13%
- Directory Traversal: 8%
- Bypass Something: 5%
- Gain Information: 4%
- Gain Privilege: 3%
- Memory Corruption: 16%
- SQL Injection: 2%
- File Inclusion: 2%
- Cross Site Request Forgery: 2%
- HTTP Response Splitting: 0%

Source: cvedetails.com
Software Security Vulnerabilities

- **WH:** $100+ billion in annual cybersecurity damages
- **NIST:** 25 vulnerabilities per every 1,000 lines of code
- **NASA:** 1–100 million lines of code in modern software
- **DHS:** 80% of attacks exploit unknown vulnerabilities

We need effective, scalable approaches for vetting all software and systems
Proactive Vulnerability Discovery

Static Analysis:
- Analyze program **without running it**
- Accuracy a major concern
  - **False negatives** (vulnerabilities missed)
  - **False positives** (results are unusable)
- As code size grows, **speed drops**

Dynamic Testing:
- Analyze program **by executing it**
- Better accuracy: **no false positives**
  - Execution reveals only what exists
  - Program crashed? You found a bug!
- Capable of very **high throughput**
Proactive Vulnerability Discovery

- Widely deployed in industry today:

- Over 36,000 errors in 550 codebases
- Over 18,000 errors in Google Chrome
- Over 11,000 errors in Linux’s kernel

Dynamic Testing:

- Analyze program by executing it
- Better accuracy: no false positives
  - Execution reveals only what exists
  - Program crashed? You found a bug!
- Capable of very high throughput
Key Approach: **Fuzz Testing**
Key Approach: **Fuzz Testing**
Key Approach: **Fuzz Testing**

- **Inputs**
- **Program**
- **Execute and Collect Feedback**
  (e.g., code coverage)
Key Approach: **Fuzz Testing**

- Inputs
- Program
- Execute and Collect Feedback
  (e.g., code coverage)

Interesting! (new code)
Key Approach: **Fuzz Testing**

- **Inputs**
- **Program**
- **Execute and Collect Feedback** (e.g., code coverage)
- **Interesting!** (new code)
- **Uninteresting** (no new code)

Stefan Nagy
Key Approach: **Fuzz Testing**

- **Inputs**
- **Program**
- **Execute and Collect Feedback**
  (e.g., code coverage)
- **Interesting!**
  (new code)
- **Crashes**
  (SEGFAULT)
- **Uninteresting**
  (no new code)
Key Approach: **Fuzz Testing**

The space of possible program behaviors

Source: https://blog.trailofbits.com/2020/10/22/lets-build-a-high-performance-fuzzer-with-gpus/
Key Approach: **Fuzz Testing**

Google: We've open-sourced ClusterFuzz tool that found 16,000 bugs in Chrome

New fuzzer finds 26 USB bugs in Linux, Windows, macOS, and FreeBSD

Fuzzing continues to remain today's most **popular** and **successful** software security testing approach

Source: https://blog.trailofbits.com/2020/10/22/lets-build-a-high-performance-fuzzer-with-gpus/
My Research: Extending Fuzzing’s Reach

Closed-source Binaries
Linux Binaries, Firmware Windows, MacOS Binaries Obfuscated Executables

Code Dev/Analysis Tools
Compilers, Debuggers Language Transpilers Binary Analysis Tools

Complex Codebases
Applications, Kernels Software Product Lines Heterogeneous Software

Can closed-source code be fuzzed as well as open-source?

Where do these tools fail? How can we find their bugs?

What code aren’t we fuzzing? Are there bugs we are missing?

Prior Work:
Fast Coverage Tracing
Fast Process Execution

Ongoing Work:
Fuzzing Decompilers
Fuzzing Transpilers

Ongoing Work:
Configuration Fuzzing
Automated Harnessing
Topics in this Course

SOFTWARE QUALITY ASSURANCE

FUNCTIONAL
SMOKE
SECURITY
INSTALLATION
REGRESSION
COMPATIBILITY
EXPLORATORY
VOLUME
LOAD
LOCALIZATION
STRESS
USABILITY
Topics in this Course

- Input generation
- Runtime feedback
- Optimization
- Harnessing
- Sanitizers
- Bug oracles
- Property testing
- Differential testing
- Bug reporting
- Deduplication
- Root cause analysis
- Severity analysis
Questions?
Ethical Considerations
NOTE: **Under no circumstances** may you **exploit or misuse** any bugs that you find (e.g., zero-day vulnerabilities) for unauthorized access or other illegal activity.

Violations of this policy will be referred to Student Conduct.
NOTE: Under no circumstances may you exploit or misuse any bugs that you find (e.g., zero-day vulnerabilities) for unauthorized access or other illegal activity.

Violations of this policy will be referred to Student Conduct.

Our goals in this course are to help devs & users, have fun, and learn!
Questions / Professor AMA
Next time on CS 5963/6963...

Research 101: Ideas