Week 1A: Course Intro

CS 5963 & CS 6963

Stefan Nagy

University of Utah
I'm recruiting!

My Research: systems and software security, binary analysis, fuzzing

University of Utah
Assistant Prof. (2022–)

Virginia Tech
Ph.D.

University of Illinois at Urbana-Champaign
B.S.
Course Goals

Help you become better researchers,

Expose you to different perspectives,

Hands-on learning with state-of-the-art tools,

... 

Get course credit,

...

All while learning about software security testing
Course Components

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

- Conducting & presenting research
  - Work together with other students
  - Help society by finding security bugs
Format

- **Meetings:** Tuesday & Thursday 9:10 – 10:30 AM

- **40 – 60 min:** instructor-led lecture

- **20 – 30 min:** student-led paper presentation & discussion
Course Website

https://www.cs.utah.edu/~snagy/courses/cs5963
Schedule

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

- **Weeks 4 – 8:** Fundamentals of Software Fuzzing
  - Three (relatively easy) labs
  - Semester Project begins on Week 6

- **Weeks 9 – 12:** Emerging Enhancements in Fuzzing

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

- 10% – Attendance & Paper Discussion
- 10% – Paper Presentations
- 15% – Lab 1
- 15% – Lab 2
- 15% – Lab 3
- 35% – Team Project
Attendance & Participation

- **Requirement 1:** Show up to every 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

- **Audience:** not required to read the paper
  - Required to participate in discussion

- **Presenters:** your job is to teach us the paper
  - Contextualize
  - Pros vs. cons
  - Contribution
  - Summarizing
  - Identify assumptions
  - **Prepare a short slide deck** (get “inspired” from existing presentations)
  - **10 – 20 minute presentation** (with 10 minute audience discussion to follow)
Hands-on Labs

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

- Paced with the introductory content from Weeks 3–8
  - Apply the techniques you’ve learned in class
  - Get familiar with state-of-the-art tools
  - Deliverables: a report what you’ve learned

- Designed to prepare you for the Semester Team Project
Semester Team Project

- **Objective:** _uncover new bugs in a real-world program_
- Team up in groups of 3 – 4
- Select an “interesting” target program of your choice; e.g.:
  - Popular applications
  - Nintendo emulators
  - Old computer games
  - MacOS Rosetta
  - The SoC’s TA Portal
  - ...
  - _GET CREATIVE!_
- **Figure out how to fuzz** your target, **find bugs**, and **disclose them**
- **Deliverables:** a report, disclosure of bugs, and open-source your team’s fuzzer
Semester Team Project

- **Objective:** uncover new bugs in a real-world program

- **Team:** form groups of 3 – 4

- **Select** an “interesting” target program of your choice:
  - Popular applications
  - Nintendo emulators
  - Old computer games
  - MacOS Rosetta
  - The SoC’s TA Portal
  - ...
  - **GET CREATIVE!**

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

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

**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.
Semester Team Project

Objective: uncover new bugs in a real-world program

Team up in groups of 3 – 4

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

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

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

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 goal is to help devs & users, have fun, and learn!
Key Dates

- Aug. 23  First class
- Sep. 06  Sign up for paper presentations
- Sep. 20  Lab 1 due
- Sep. 27  Lab 2 due
- Oct. 06  Lab 3 due
- Oct. 09 – 16 No class (Fall break)
- Oct. 18 & 20 Semester project proposals
- Nov. 21 – 27 No class (Thanksgiving break)
- Dec. 06 & 08 Team project presentations
No Exams
Meeting Location?
Questions?
Software Security Background
Our world depends on software...

Personal Technology

Infrastructure & Industry

Military and Government
... and software security is a **nightmare**

**New Vulnerabilities Reported Per Year**

Source: cvedetails.com
... 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

New York subway hacked in computer breach linked to China

Janesville school district hit by ransomware attack
Software Security Vulnerabilities

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

Source: cvedetails.com
Common Security Errors

- Missed initialization check
- Free’d pointers not NULL’d
- Unchecked memory writes

Consequences

- Use uninitialized memory
- Use non-owned memory
- Overflowing a data buffer

Attacker Exploitation

- Software denial of service
- Leak sensitive information
- Inject & run arbitrary code

Race against time to find & fix vulnerabilities before they are exploited
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
Software Security Testing
Software Security Testing

Research Areas
- Input generation
- Runtime feedback
- Bookkeeping
- Harnessing
- Bug oracles
- Crash deduplication
- Property testing
- Differential testing
- Root cause analysis
- Bug exploitability
- Patch testing
- Directed fuzzing
Testing Open- vs. Closed-source Software

**Open Source:**
- Publicly-available source codebase
- Achieves security by *transparency*
- Semantic richness facilitates high-performance, effective fuzzing

**Closed Source:**
- Distributed as a precompiled binary
- Opaque to everyone but its developer
- Upwards of 10x slower fuzzing speeds
- Forced to rely on crude techniques
Testing Open- vs. Closed-source Software

- Global market size over $240 billion
- 85% contains critical vulnerabilities
- 89% of the most exploited software

Closed Source:
- Distributed as a precompiled binary
- Opaque to everyone but its developer
- Upwards of 10x slower fuzzing speeds
- Forced to rely on crude techniques
Testing Open- vs. Closed-source Software

Closed Source:
- Distributed as a precompiled binary
- Opaque to everyone but its developer
- Upwards of 10x slower fuzzing speeds
- Forced to rely on crude techniques

My research

Countermeasures

Weird Interfaces

OS Asymmetries

Less Precision
Questions?