CSci 8980-1, Program Analysis for Security
Day 1: Introduction and Logistics

Stephen McCamant
University of Minnesota

Introductions

Outline

Big-Picture Introduction

Course Logistics

Topics Overview

Program Analysis...

- Programs that operate on other programs

...for Security

- There is an adversary
- There is a bad outcome to avoid

Static analysis

- Examine the program itself without executing it
- Over-approximate behavior

Dynamic analysis

- Examine particular program executions
- Under-approximate behavior

Harder to classify

- Model checking
- Symbolic execution

False positives and false negatives

- False positive: claim something that's not really there
  - AKA false alarm, type I error
- False negative: miss something that is really is there
  - AKA type II error

Theoretical limits of program analysis

Every analysis that attempts to check an aspect of program behavior must sometimes either (Rice):

- Give a false positive,
- Give a false negative, or
- Fail to terminate

More analysis terms

- Sound vs. complete
- Weak vs. strong
- Conservative vs. precise
- May vs. must

Binaries and source code

- Source code has various advantages
- But it may not be available

Malware and adversarial software

- *Mal*icious soft*ware*
- What does it mean if the bad guys know the weaknesses of our analysis?

PL and security perspectives

- "Programming languages" cares about:
  - Interesting analysis techniques
  - Program correctness
  - Formalism
- "Security" cares about:
  - Stopping real attacks
  - Robustness against adversaries
  - Applicability to real systems

Outline

Big-Picture Introduction

Course Logistics

Topics Overview

Instructor information

- Stephen McCamant
- Office: 4-225E Keller
- Office hours: Tuesday afternoons TBA, Wednesdays 10-11am
- Email: `mccamant@cs.umn.edu`

Course goals

- Learn about program analysis techniques
- Learn about security applications
- Learn what makes for interesting research

Evaluation components

10% Reading questions
10% Class attendance and participation
10% In-class paper presentation
20% Hands-on assignments
50% Research project

Readings

- Linked from the course web page
- Average of two 10-page papers per class
- Most either public or UMN-licensed
- Take notes while reading
- Bring a copy (to refer to) to class
- Also: optional and historical

Reading questions

- Goal: make sure you read and understand the papers
- One general question per paper
- Average one extra question per class

General questions

- What interesting new thing did you learn?
- What question is raised but not answered?
- Do you disagree with a claim?
- Is something important left out or ambiguous?
- In hindsight, what would you do differently?

Submission logistics

- Email or Moodle? Email
- Due the day before
  - 6pm, midnight, or 6am? midnight
- Late: 50% credit; after 2:30pm: 0

Class participation

- The goal of a seminar is discussion, not lecture
- I expect everyone to contribute
- Aim is not to show off knowledge
  - An interesting question > a straightforward answer

In-class presentation

- One per student, scheduled in advance (about one per week)
- Can also promote an optional or chosen-by-you relevant paper
- Prepare 25 minutes of slides, but expect questions

Hands-on assignments

- Experience actually using these tools
- Done individually
- Mix of using tools, implementation, and questions
- Symbolic execution, binary rewriting, decision procedures

Research project

- Idea: microcosm of research experience
- Formulate a question, answer it, convince others of your results
- Preferred group size of 2

Project topics

- Program analysis for security, or another application of a related analysis technique
- Can use one of our papers as a starting point
- But, must make your own novel contribution

Project goals

- Innovative
- Scholarly
  - Put in context of related work
- Appropriately evaluated
  - Able to convince a skeptic
- Well presented

Project results

- Report: about 10 pages, in the format of a conference paper
- In-class presentation: 25 minutes

Collaboration and cheating

- Principle: learn from each other, but don't substitute another's understanding for your own
- Cardinal sin: taking ideas without acknowledgment

Course web site

- Department web site under `csci8980-pas`
- Also linked from my home page `~mccamant`

Outline

Core techniques

Before spring break: more in depth on techniques with many applications and variations

Dynamic taint analysis

Track at runtime how a program uses confidential or untrusted data

Symbolic execution

Perform logical reasoning about related executions along a single execution path, and explore which such paths are possible

Information flow analysis

Track whether information from one point can affect data at another
Quantitative information flow: measure how much information can flow as a number of bits

SFI and Native Client

Rewrite untrusted code at the instruction level to enforce isolation

CFI and program hardening

Rewrite buggy code to neutralize potential vulnerabilities

Further topics

- After spring break: quicker looks at smaller or less central techniques and problem areas
- We'll need to choose a subset of the following

Test generation

Create tests that reveal vulnerabilities (e.g., better "fuzzing")

Policy inference

Automatically determine what should be protected or how

Side-channel attacks

Attacks that go outside the usual abstractions, such as by using hardware to subvert software protections

Side-channel defenses

Stopping outside-the-box attacks

Dealing with bugs at scale

Can we ever deal with all the bugs in real systems?

Specs and verification

Can we do better if we actually have a machine representation of what a program should do?

Reverse engineering

Can we recover higher-level information given just a binary?

Differential privacy

Tool support for provably protecting statistical disclosures with noise

Programming cryptography

Deriving secure multi-party protocols from a naive implementation

Program obfuscation

Transforming programs so you can't learn much by looking at them

ROP and shellcode

Techniques for bad guys to get their attacks to run under defensive constraints, such as return-oriented programming

Web applications

Security for sever-side and client-side web software

Smartphone applications

Security in new mobile platforms such as iOS and Android