# CSci 5980/8980 Manual and Automated Binary Reverse Engineering Day 1: Introduction and Logistics

Stephen McCamant University of Minnesota

# Outline

**Big-Picture Introduction** 

**Course Logistics** 

Starting with Compiler Explorer

# **Binaries**

 AKA executables, object code, machine code
 Output of compilation, final form for execution
 Designed for efficiency, not understanding or modification

## Why binaries are inconvenient

- Fewer or no names ("symbols"), no comments
- Control flow flattened into gotos
- Types and structures no longer explicit
- **5** Source  $\rightarrow$  binary mapping is many-to-one
- Numeric addresses make instructions hard to move

#### Why binary analysis is needed

- Inter-operating with proprietary software
- Investigating security vulnerabilities
- Understanding malicious software
- 🖲 (Evil) Attacking vulnerable software
- Evil) Circumventing copy-protection or DRM

#### What is reverse engineering?

- $\mathbf{e}$  In general, artifact ightarrow design
- Binary reverse engineering is figuring things out from a binary that you would normally figure out from source code instead
- Subtasks: recovering any kind of source-like information from a binary

# Inverse operations

- Source to binary is compilation, binary to source is decompilation
- Very challenging to do well, but making slow progress
- Subtask: assembly language to binary is assembly, binary to assembly language is disassembly
  - Core operation is straightforward, but some complications

#### Manual binary R.E.

- Emphasis for first third of class: human decompilation
- Read what binary code does, imagine source code that would do that
- Recognize patterns in compiler-generated binary code
- Understanding code is always ultimately manual

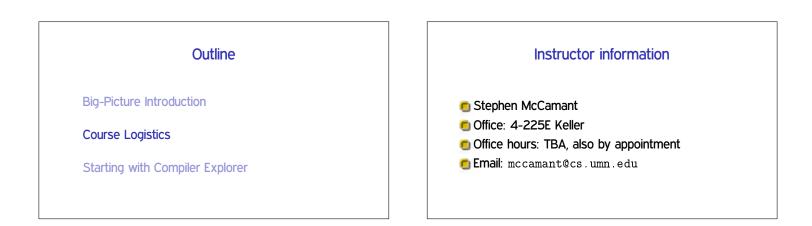
# Automating binary R.E.

Emphasis for second two-thirds of class
 Towards the ideal of automatic decompilation

- Understanding challenges and limitations
- Isolating smaller subtasks
- Tools that can assist human experts

# Some related topics

- Obfuscation: making binaries that are hard to reverse engineer
- Patching: changing the behavior of a binary without having source



# Course style

- Manual phase is a skills class: lecture and demo, practice with homework
- Automatic phase is a seminar: read and discuss research papers
- A large final project turns your skills and knowledge to a practical task

#### **Textbooks**

- Reverse Engineering for Beginners, Yurichev (free, http://beginners.re/)
- Practical Binary Analysis, Andriesse (No Starch Press)

#### Evaluation components (5980)

- 20% Manual R.E. homework (longer)
- 10% Reading questions
- 10% In-class paper presentation (less)
- 10% Hands-on demo assignment
- 50% Final project

#### Evaluation components (8980)

- 10% Manual R.E. homework (less)
- 10% Reading questions
- 20% In-class paper presentations (more)
- 10% Hands-on demo assignment
- 50% Final research project



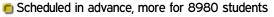
- Decompilation: given binary, you write the source
   Sometimes with an automatic testing framework
- Code cleanup: given ugly C code, rewrite it to be understandable
- Given a binary, find an input that makes it do something special

#### Readings

- Will be linked from the course web page
- Usually one main paper per class
- Most either public or UMN-licensed
- Take notes while reading
- Bring a copy (to refer to) to class
- Also: optional and background

# Reading questions General questions Goal: make sure you read and understand the papers What interesting new thing did you learn? Answer one: a general question selected from list on next slide What question is raised but not answered? Ask one: suggest a question for in-class discussion Is something important left out or ambiguous?

# In-class presentation



- Can also promote an optional or chosen-by-you relevant paper
- Prepare 25 minutes of slides, but expect questions

# Hands-on demo assignment

Experience actually using an existing research tool

Submission logistics

Done individually

Email or Canvas?
Due the day before

9pm? midnight? 3am?

🖲 Late: 50% credit; after 9:45am: 0

- Find existing software, and get it to do something interesting
- Preparation in advance, short writeup, brief in-class demo

# **Final projects**

- Do something new using what you've learned
- Do in groups of 2-3 students
- Must have generalizable value, and be legal and ethical to do and talk publicly about
- For 8980, should be research; for 5980, can be more applied

## **Project results**

- Report: in the format of a conference paper, 6 pages for 5980, 10 pages for 8980
- In-class presentation: 10-15 minutes in one of the last lectures
- Most 5980 projects should also have an electronic deliverable

# 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

- Watch my home page for a site with public information (including these slides)
- We'll also use a Canvas page for homework submissions and grades

#### Outline

**Big-Picture Introduction** 

**Course Logistics** 

Starting with Compiler Explorer