Week 7: Lecture A
Access Control & Isolation

Tuesday, October 3, 2023
Announcements

- **Project 2: AppSec** released
  - **Deadline:** Thursday, October 19th by 11:59PM

### Project 2: Application Security

**Deadline: Thursday, October 19 by 11:59PM.**

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies. You may optionally work alone, or in teams of at most two and submit one project per team. If you have difficulties forming a team, post on Piazza’s Search for Teammates forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University’s Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else’s solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). Don’t risk your grade and degree by cheating!

Complete your work in the CS 4440 VM—we will use this same environment for grading. You may not use any external dependencies. Use only default Python 3 libraries and/or modules we provide you.

### Helpful Resources

- The CS 4440 Course Wiki
- VM Setup and Troubleshooting
- Terminal Cheat Sheet
- GDB Cheat Sheet

### Table of Contents:

- Helpful Resources
- Introduction
- Objectives
- Start by reading this!
  - Setup Instructions
  - Important Guidelines
- Part 1: Beginner Exploits
  - Target 0: Variable Overwrite
  - Target 1: Execution Redirect
  - What to Submit
- Part 2: Intermediate Exploits
  - Target 2: Shellcode Redirect
  - Target 3: Indirect Overwrite
  - Target 4: Beyond Strings
  - What to Submit
- Part 3: Advanced Exploits
  - Target 5: Bypassing DEP
  - Target 6: Bypassing ASLR
  - What to Submit
- Part 4: Super L33T Pwnage
  - Extra Credit: Target 7
  - Extra Credit: Target 8
  - What to Submit
- Submission Instructions
Project 2 Progress Update

- Working on Targets 0–2: 0%
- Working on Targets 3–4: 0%
- Working on Targets 5–6: 0%
- Finished!: 0%
- Haven’t started :( 0%
Announcements

See Discord for meeting info!

www.utahsec.com
Questions?
Last time on CS 4440...

Automated Bug-Finding
Fuzz Testing
Symbolic Execution
Exploitation

Common Vulnerabilities
- 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
Proactive Vulnerability Discovery

Static Analysis:
Proactive Vulnerability Discovery

Static Analysis:

- Analyze program **without running it**

- **Challenges:**
Proactive Vulnerability Discovery

Static Analysis:

- Analyze program **without running it**

**Challenges:**
- **False negatives** (vulnerabilities missed)
- **False positives** (results are unusable)

As code size grows, **analysis speed drops**
Stefan Nagy

Proactive Vulnerability Discovery

Static Analysis:

- Analyze program **without running it**

  **Challenges:**
  - False negatives (vulnerabilities missed)
  - False positives (results are unusable)
  
  As code size grows, **analysis speed drops**

Dynamic Testing:
Static Analysis:
- Analyze program **without running it**

  **Challenges:**
  - False negatives (vulnerabilities missed)
  - False positives (results are unusable)
  As code size grows, **analysis speed drops**

Dynamic Testing:
- Analyze program **by executing it**

  **Advantages:**
Proactive Vulnerability Discovery

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

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

The space of possible program behaviors

Source: https://blog.trailofbits.com/2020/10/22/lets-build-a-high-performance-fuzzer-with-gpus/
Why do we need feedback in fuzzing?
Why do we need feedback in fuzzing?
Feedback-driven Fuzzing

Inputs

Program

???
Feedback-driven Fuzzing

Inputs

Program

Execute and Collect Feedback
(e.g., code coverage)
Feedback-driven Fuzzing

Inputs

Program

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

Interesting!
(new code)

???
Feedback-driven Fuzzing

Inputs → Program → Collect Feedback (e.g., code coverage) → Interesting! (new code) → Inputs

Program → Collect Feedback → Uninteresting (no new code) → Program
Feedback-driven Fuzzing

Inputs → Program → Execute and Collect Feedback (e.g., code coverage) → Interesting! (new code) → Crashes (SEGFAULT) → Uninteresting (no new code) → Inputs
Types of Feedback-driven Fuzzers

Black-box

Grey-box

White-box
Types of Feedback-driven Fuzzers

Black-box
Zero Introspection

Grey-box
Some Introspection

White-box
High Introspection
Types of Feedback-driven Fuzzers

- **Black-box**: Zero Introspection
- **Grey-box**: Some Introspection
- **White-box**: High Introspection

The spectrum ranges from ineffective to inefficient.
Types of Feedback-driven Fuzzers

- Black-box: Zero Introspection
- Grey-box: Some Introspection
- White-box: High Introspection
Coverage-guided Fuzzing

- **Code coverage:** program regions exercised by each test case

- **Horse racing analogy:** “breed” (mutate) only the “winning” (coverage-increasing) inputs
  - New coverage? **Keep and mutate the input**
  - Old coverage? **Discard it and try again**

- Most fuzzing today is **coverage-guided**
  - Good balance of performance and precision
Code Coverage Metrics

- Program represented as **control-flow graphs (CFG)**
  - Directed graph encompassing all program paths
  - Basis of virtually all software analysis techniques

- Various coverage metrics in use today
  - **Instructions**: units that make up basic blocks
  - **Basic blocks**: nodes of the program’s CFG
  - **Edges**: transitions between basic blocks
  - **Hit counts**: frequencies of basic blocks
  - **Paths**: sequences of edges
Impact of Code Coverage

```c
void top(char input[4]) {
    if(input[0] == 'b') {
        if(input[1] == 'a') {
            if(input[2] == 'd') {
                if(input[3] == '!') {
                    OVERFLOW();
                }
            }
        }
    }
}
```
void top(char input[4]) {
  if(input[0] == 'b')
    if(input[1] == 'a')
      if(input[2] == 'd')
        if(input[3] == '!')
          OVERFLOW();
}
void top(char input[4]) {
    if(input[0] == 'b') {
        if(input[1] == 'a') {
            if(input[2] == 'd') {
                if(input[3] == '!') OVERFLOW();
            }
        }
    }
}

Estimated Mutations Required

$(2^8)^4 = 4,294,967,296$

$4 \times (2^8) = 1,024$
Model-agnostic Input Generation

- Brute-force your way to valid inputs
  - Bit and byte “flipping”
  - Addition and subtraction
  - Inserting random chunks
  - Inserting dictionary “tokens”
  - Splicing two inputs together

- The good: super fast
  - Incorporating feedback like coverage enables you to **synthesize valid inputs** (eventually)

```html
<html><header><title>Hello</title></header><body>World<br/></body></html>
```

```xml
<a> <a/> </a> = 'a'
```
Follow a pre-defined input **specification**
- Pre-defined input grammars
- Dynamically-learned grammars
- Domain-specific generators

**The good:** many more valid inputs
- Model-agnostic inputs are often discarded because they fail basic input sanity checks
- Valid inputs = **higher code coverage**

```xml
XML_GRAMMAR: Grammar = {
  "<start>": ["<xml-tree>",
    "<xml-tree>": ["<text>",
      "<xml-open-tag><xml-tree><xml-close-tag>",
      "<xml-openclose-tag>"],
    "<xml-tree><xml-tree>",
    "<xml-open-tag>": ["<id>"],
    "<xml-openclose-tag>": ["<id/>", "<xml-attribute>"],
    "<xml-close-tag>": ["</id>",
    "<xml-attribute": ["<id><id>"],
    "<xml-close-tag>": ["<id>", "<xml-attribute> <xml-attribute>"]],
    "<id>": [
      "<letter>", "<id><letter>",
    "<text>": ["<text><letter_space>"],
    "<letter_space>": string.ascii_letters + string.digits + "\" + "'" + ","],
    "<letter_space>": string.ascii_letters + string.digits + "\" + "'" + ","],
}
```
Taint Tracking

- Track input bytes’ flow throughout program
  - Identify input “chunks” that affect program state
    - Chunks that affect branches
    - Chunks that flow to function calls

- Mutate these chunks
  - Random mutation
  - Insert fun or useful tokens

- The good: finding vulnerable buffers, solving branches

0. def f(x, y):
1.   if (x > y):
2.     ...........

<table>
<thead>
<tr>
<th>11</th>
<th>11</th>
<th>11</th>
<th>Bytes that comprise X</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>Bytes that comprise Y</td>
</tr>
</tbody>
</table>

Mutate!

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
Symbolic Execution

0. `def f (x, y):`
1. `if (x > y):`
2. `x = x + y`
3. `y = x - y`
4. `x = x - y`
5. `if (x - y > 0):`
6. `assert false`
7. `return (x, y)`

Possible path constraints:
- `(A > B)` and `(B-A > 0)` = unsatisfiable
- `(A > B)` and `(B-A <= 0)` = satisfiable
- `(A <= B)` = satisfiable
if($x^3 == 1881672302290562263$)
    OVERFLOW();
}
if($x^3 == 1881672302290562263)$

OVERFLOW(); // $x = 1234567$

Estimated Mutations Required

Good luck!

Solves instantly
Feedback-driven Fuzzing vs. Symbolic Execution

```
if(A^3 + B^3 + C^3 == 3)
  OVERFLOW();
}
```

Estimated Mutations Required

???

???
if($A^3 + B^3 + C^3 == 3$)

OVERFLOW();

// $A = 5699368212219623807203$
// $B = -569936821113563493509$
// $C = -472715493453327032$

Estimated Mutations Required

Good luck!

Good luck!
Input Generation Trade-offs

- Model-agnostic Fuzzing:
  - Advantages: ???
Input Generation Trade-offs

- **Model-agnostic Fuzzing:**
  - **Advantages:** great on *simple, easy-to-solve branches*; attains **really fast** speed
  - **Challenges:** ???
Input Generation Trade-offs

- **Model-agnostic Fuzzing:**
  - **Advantages:** great on *simple, easy-to-solve branches*; attains *really fast* speed
  - **Challenges:** need a lot of luck to solve *multi-byte conditionals, checksums*

- **Model-guided Fuzzing:**
  - **Advantages:** ???
Input Generation Trade-offs

- **Model-agnostic Fuzzing:**
  - Advantages: great on simple, easy-to-solve branches; attains really fast speed
  - Challenges: need a lot of luck to solve multi-byte conditionals, checksums

- **Model-guided Fuzzing:**
  - Advantages: more valid inputs leads to higher coverage earlier on
  - Challenges: ???
Input Generation Trade-offs

- **Model-agnostic Fuzzing:**
  - **Advantages:** great on simple, easy-to-solve branches; attains really fast speed
  - **Challenges:** need a lot of luck to solve multi-byte conditionals, checksums

- **Model-guided Fuzzing:**
  - **Advantages:** more valid inputs leads to higher coverage earlier on
  - **Challenges:** out of luck if specification is not defined or hard-to-define

- **White-box Generation:**
  - **Symbolic Execution Advantages:** ???
  - **Taint Tracking Advantages:** ???
Input Generation Trade-offs

- **Model-agnostic Fuzzing:**
  - **Advantages:** great on simple, easy-to-solve branches; attains really fast speed
  - **Challenges:** need a lot of luck to solve multi-byte conditionals, checksums

- **Model-guided Fuzzing:**
  - **Advantages:** more valid inputs leads to higher coverage earlier on
  - **Challenges:** out of luck if specification is not defined or hard-to-define

- **White-box Generation:**
  - **Symbolic Execution Advantages:** precise solving of multi-byte conditionals
  - **Taint Tracking Advantages:** easily identifies key data chunks, branch constraints
  - **Challenges:** ???
Input Generation Trade-offs

- **Model-agnostic Fuzzing:**
  - **Advantages:** great on simple, easy-to-solve branches; attains really fast speed
  - **Challenges:** need a lot of luck to solve multi-byte conditionals, checksums

- **Model-guided Fuzzing:**
  - **Advantages:** more valid inputs leads to higher coverage earlier on
  - **Challenges:** out of luck if specification is not defined or hard-to-define

- **White-box Generation:**
  - **Symbolic Execution Advantages:** precise solving of multi-byte conditionals
  - **Taint Tracking Advantages:** easily identifies key data chunks, branch constraints
  - **Challenges:** far too heavyweight to deploy on all generated inputs; closed-source code
Impact of Automated Bug-finding

New Vulnerabilities Reported Per Year

Source: cvedetails.com
Impact of Automated Bug-finding

New Vulnerabilities Reported Per Year

Source: cvedetails.com
Questions?
Interested in automated bug finding?

  - Everything you’d ever want to know about fuzzing for finding security bugs!
  - Course project: team up to fuzz a real program (of your choice), and find and report its bugs!
  - [https://cs.utah.edu/~snagy/courses/cs5963/](https://cs.utah.edu/~snagy/courses/cs5963/)
Interested in automated bug finding?


CS 5963 - 001 Applied S/W Secur Test
Class Number: 18228  Instructor: NAGY, STEFAN  Component: Special Topics  Type: In Person  Units: 3.0
Requisites: Yes  Wait List: No  View Feedback

This class will prepare students to become effective software testers capable of automating vulnerability discovery in today's large and complex software systems. This course will cover the fundamental design considerations behind today's state-of-the-art software testing tools, and equip students with the know-how to soundly evaluate their results and effectiveness. Students will team up to target a software or system of their choice, and devise their own testing strategies to find new vulnerabilities in it, analyze their severity, and report them to its developers. Prerequisites CS 3505, CS 4400 and CS 4440

Days / Times
MoWe/01:25PM-02:45PM

Locations
WEB L114

Meets With
• CS 6963 001

Stefan Nagy
This time on CS 4440...

Access Control
Permissions
Process Isolation
So far, we’ve talked about **thwarting bugs** by **proactively** discovering them
- E.g., run fuzzing and try to catch all the bugs!
- Hopefully the **attacker** will not beat us to it…

**Question:** how can we redesign our **systems** to prevent software exploits?
Clearly we can’t assume Application Developers will write safe code...
Clearly we can’t assume Application Developers will write safe code...
  - Unless they are alumni of CS 4440 😊

What principles should our safe system design uphold?
Clearly we can’t assume **Application Developers** will write safe code…
- Unless they are alumni of **CS 4440 😊**

What principles should our safe system design uphold?
- Control **who can access** what
- Prevent **applications from spying** on one another
- Implement **safeguards to minimize damage** of attacks
Access Control
Access Control

- **Access Control:** the heart of security on commodity computing systems
- **Goal:** ???
Access Control

- **Access Control:** the heart of security on commodity computing systems

- **Goal:** control which **principles** have **access** to which system **resources**
Access Control: the heart of security on commodity computing systems

Goal: control which principles have access to which system resources
Access Control

- **Access Control:** the heart of security on commodity computing systems

- **Goal:** control which principles have access to which system resources
Access Control

- **Access Control**: the heart of security on commodity computing systems

- **Goal**: control which **principles** have **access** to which system **resources**

**Resources:**
- Files
- Programs
- Peripherals
- Instructions
- ...
Access Control: the heart of security on commodity computing systems

Goal: control which principles have access to which system resources

Access control mechanisms exist at all levels of a modern computer
  - E.g., Hardware, Hypervisor, Operating System, Middleware, Application
Access Control: the heart of security on commodity computing systems

Goal: control which principles have access to which system resources

Access control mechanisms exist at all levels of a modern computer
  - E.g., Hardware, Hypervisor, Operating System, Middleware, Application

Relies on

- Application
- Middleware
- Operating System
- Hypervisor
- Hardware
Access Control

- **Access Control**: the heart of security on commodity computing systems

- **Goal**: control which **principles** have **access** to which system **resources**

- Access control mechanisms exist at **all levels** of a modern computer
  - E.g., Hardware, Hypervisor, Operating System, Middleware, Application
Access Control History

- Wasn’t necessary back in “the day”

- ENIAC
  - The first programmable, electronic, general-purpose digital computer
  - Built in 1945 by U.S. Army / UPenn
  - Access control consisted of just a single user and a single program
Access Control History

- **LEO III**
  - “Lyons Electronic Office”
  - Introduced concept of **multi-tasking**
  - Consisted of a single master program “**Operating System**”
  - Allowed 12 “**application**” programs to be run concurrently
Access Control History

- **PLATO 1 / PLATO 2**
  - Developed by Univ. of Illinois (ILLIAC)
  - Based on a time-sharing computer system, with users and programmers connected to a central mainframe
  - Access control = **multiple users, multi-tasking**
Access Control History

- **Moore’s Law**: number of transistors in an IC doubles about every two years
- **By 1980**: we all have access to computers!
Access Control History

- **Moore’s Law:** number of transistors in an IC doubles about every two years

- **By 1980:** we all have access to computers!
Access Control History

- **Moore’s Law**: number of transistors in an IC doubles about every two years
- **By 1980**: we all have access to computers!
- **Same terrible security ideas**
Bugs are **inevitable** in any complex software system.

- **NIST:** 10–50 bugs per every 1000 code lines

- Many bugs are never found.
Bugs are **inevitable** in any complex software system

**NIST:** 10–50 bugs per every 1000 code lines

Many bugs are never found

Many are found and never reported
- Weaponized by Nation-States, criminals
- What we know as **Zero-Day Exploits**

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

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

Cyber-attack hits UK internet phone providers

Janesville school district hit by ransomware attack

'A cyber-attack disrupted my cancer treatment'

New York subway hacked in computer breach linked to China
Implementing Access Control
Isolating Applications

- **Adversary 1:** exploited user-space process
  - **Targets 2–8** in Project 2 (after your attacks)
Isolating Applications

- **Adversary 1**: *exploited* user-space process
  - Targets 2–8 in Project 2 (after your attacks)

- **Adversary 2**: *malicious* user-space process
  - Spyware app your aunt installed
  - That TikTok app that *you* installed
**Isolating Applications**

- **Adversary 1: exploited** user-space process
  - Targets 2–8 in Project 2 (after your attacks)

- **Adversary 2: malicious** user-space process
  - Spyware app your aunt installed
  - That TikTok app that *you* installed

- **Goal:** protect the system (i.e., all other processes + the OS) from an *evil process*
What specific resources must we protect?
What specific resources must we protect?

- **Memory**
  - Code and data of Operating System
  - Code and data of other processes
What specific resources must we protect?

- **Memory**
  - Code and data of Operating System
  - Code and data of other processes

- **Files, Directories, and Metadata**
  - The sudo-ers files
  - Your HOME directory
  - Program-specific file descriptors
What specific resources must we protect?

- **Memory**
  - Code and data of Operating System
  - Code and data of other processes

- **Files, Directories, and Metadata**
  - The sudo-ers files
  - Your HOME directory
  - Program-specific file descriptors

- **The Network**
  - Other systems on the same network
What specific resources must we protect?

- **Memory**
  - Code and data of Operating System
  - Code and data of other processes

- **Files, Directories, and Metadata**
  - The sudo-ers files
  - Your HOME directory
  - Program-specific file descriptors

- **The Network**
  - Other systems on the same network

- **External Devices and Peripherals**
  - Your USB drive that contains a pirated copy of Super Bowl LVII
How should we protect them?
How should we protect them?

- **Principle of Least Privilege**
  - “In a particular abstraction layer of a computing environment, every module (e.g., process, user, or program) must be able to access **only the information and resources that are necessary**”
How should we protect them?

- **Principle of Least Privilege**
  - “In a particular abstraction layer of a computing environment, every module (e.g., process, user, or program) must be able to access only the information and resources that are necessary”

- In other words, apps should mind their own business!

- Critical design consideration for protecting data and functionality from faults and malicious behavior
Access Control Matrix

- Conceptual model that specifies the rights each entity (row) has for each resource (column)

- Entity rights:
  - **R** = Read
  - **W** = Write
  - **X** = Execute
**Access Control Matrix**

- Conceptual model that specifies the rights **each entity (row)** has for **each resource (column)**

**Entity rights:**
- **R** = Read
- **W** = Write
- **X** = Execute

<table>
<thead>
<tr>
<th></th>
<th>OS</th>
<th>Accounting Program</th>
<th>Accounting Data</th>
<th>Insurance Data</th>
<th>Payroll Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>RX</td>
<td>RX</td>
<td>R</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Alice</td>
<td>RX</td>
<td>RX</td>
<td>R</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>Sam</td>
<td>RWX</td>
<td>RWX</td>
<td>R</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>Accounting Program</td>
<td>RX</td>
<td>RX</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
</tr>
</tbody>
</table>
Access Control Matrix

- Conceptual model that specifies the rights each entity (row) has for each resource (column)

- **Entity rights:**
  - R = Read
  - W = Write
  - X = Execute

<table>
<thead>
<tr>
<th></th>
<th>OS</th>
<th>Accounting Program</th>
<th>Accounting Data</th>
<th>Insurance Data</th>
<th>Payroll Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>R X</td>
<td>R X</td>
<td>R</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Alice</td>
<td>R X</td>
<td>R X</td>
<td>R</td>
<td>R W</td>
<td>R W</td>
</tr>
<tr>
<td>Sam</td>
<td>R W X</td>
<td>R W X</td>
<td>R</td>
<td>R W</td>
<td>R W</td>
</tr>
<tr>
<td>Accounting Program</td>
<td>R X</td>
<td>R X</td>
<td>R W</td>
<td>R W</td>
<td>R W</td>
</tr>
</tbody>
</table>
Implementing Access Control

How can we implement AC matrices on real systems?

Answer: **Access Control Lists**
- Generalization of UNIX file system permissions
- Stored with file system object as metadata (object-centric)

- Compactly and efficiently encodes access to an object via the subject’s (user or group) system rights

**Capabilities:** subject centered alternative to ACLs
- For each subject, store list of objects and permissions
Implementing Access Control

- How can we implement AC matrices on real systems?
- **Answer:** *Access Control Lists*
  - Generalization of UNIX file system permissions
  - Stored with file system object as metadata (object-centric)
- Compactly and efficiently encodes access to an object via the subject's *(user* or *group*) system rights
- **Capabilities:** subject centered alternative to ACLs
  - For each subject, store list of objects and permissions

How to completely **remove** user **Bob**?
Implementing Access Control

- How can we implement AC matrices on real systems?

- Answer: **Access Control Lists**
  - Generalization of UNIX file system permissions
  - Stored with file system object as metadata (object-centric)

- Compactly and efficiently encodes access to an object via the subject’s (user or group) system rights

- **Capabilities**: subject centered alternative to ACLs
  - For each subject, store list of objects and permissions

---

**How to completely remove user Bob?**

**Revoke** all of his permissions!
Modern Permissions Schemes
File System Permissions

- **Users**: `uid`, 32-bit integer, every file has one
- **Groups**: `gid`, 32-bit integer, every file has one

<table>
<thead>
<tr>
<th>R</th>
<th>W</th>
<th>X</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
File System Permissions

- **D** = Directory
- **R** = read files in D
- **W** = write a file in D
- **X** = access a file in D if you know its path

- **Last number** = D’s total subdirectories

```
drwxrwxr-x 4 cs4440 cs4440 .
drwxr-xr-x 17 cs4440 cs4440 ..
drwxrwxr-x  2 cs4440 cs4440 bin
-rw-rw-r--  1 cs4440 cs4440 build.py
-rwxrwxr-x  1 cs4440 cs4440 build.sh
-rw-rw-r--  1 cs4440 cs4440 cookie
-rw-------  1 cs4440 cs4440 core
-rwxrwxr-x  1 cs4440 cs4440 helper.c
drwxrwxr-x  2 cs4440 cs4440 __pycache__
-rwxrwxr-x  1 cs4440 cs4440 shellcode.py
```
File System Permissions

- **D** = Directory
- **R** = read files in D
- **W** = write a file in D
- **X** = access a file in D if you know its path
- **Last number** = D’s total subdirectories

```bash
drwxrwxr-x  4 cs4440 cs4440 .
drwxr-xr-x 17 cs4440 cs4440 ..
drwxrwxr-x  2 cs4440 cs4440 bin
-rw-rw-r--  1 cs4440 cs4440 build.py
-rwxrwxr-x  1 cs4440 cs4440 build.sh
-rw-rw-r--  1 cs4440 cs4440 cookie
-rw--------  1 cs4440 cs4440 core
-rwxrwxr-x  1 cs4440 cs4440 helper.c
drwxrwxr-x  2 cs4440 cs4440 __pycache__
-rwxrwxr-x  1 cs4440 cs4440 shellcode.py
```
File System Permissions

- **D** = Directory
- **R** = read files in D
- **W** = write a file in D
- **X** = access a file in D if you know its path

- **Last number** = D’s total subdirectories

```
-drwxrwxr-x  4 cs4440 cs4440  .
drwxr-xr-x  2 cs4440 cs4440  bin
-drwxrwxr-x  2 cs4440 cs4440  .
- rw-rw-r--  1 cs4440 cs4440  build.py
- rwxrwxr-x  1 cs4440 cs4440  build.sh
- rw-rw-r--  1 cs4440 cs4440  cookie
- rw-------  1 cs4440 cs4440  core
- rwxrwxr-x  1 cs4440 cs4440  helper.c
-drwxrwxr-x  1 cs4440 cs4440  __pycache__
-drwxrwxr-x  2 cs4440 cs4440  shellcode.py
```
File System Permissions

- **D** = Directory
- **R** = read files in D
- **W** = write a file in D
- **X** = access a file in D if you know its path
- **Last number** = D’s total subdirectories

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
File System Permissions

- **D** = Directory
- **R** = read files in D
- **W** = write a file in D
- **X** = access a file in D if you know its path
- **Last number** = D’s total subdirectories

```plaintext
drwxrwxr-x 4 cs4440 cs4440 .
drwxr-xr-x 17 cs4440 cs4440 ..
drwxrwxr-x 2 cs4440 cs4440 bin
-rw-rw-r--  1 cs4440 cs4440 build.py
-rwxrwxr-x  1 cs4440 cs4440 build.sh
-rw-rw-r--  1 cs4440 cs4440 cookie
-rw-------- 1 cs4440 cs4440 core
-rwxrwxr-x  1 cs4440 cs4440 helper.c
drwxrwxr-x 2 cs4440 cs4440 __pycache__
-rwxrwxr-x  1 cs4440 cs4440 shellcode.py
```
**File System Permissions**

- **D** = Directory
- **R** = read files in D
- **W** = write a file in D
- **X** = access a file in D if you know its path
- **Last number** = D’s total subdirectories

<table>
<thead>
<tr>
<th>Permissions</th>
<th>Owner</th>
<th>Group</th>
<th>Others</th>
<th>Size</th>
<th>File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>drwx</td>
<td>cs4440</td>
<td>cs4440</td>
<td>.</td>
<td>4</td>
<td>build.py</td>
</tr>
<tr>
<td>drwx</td>
<td>cs4440</td>
<td>cs4440</td>
<td>..</td>
<td>17</td>
<td>build.sh</td>
</tr>
<tr>
<td>drwx</td>
<td>cs4440</td>
<td>cs4440</td>
<td>bin</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>cs4440</td>
<td>cs4440</td>
<td>cookie</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>cs4440</td>
<td>cs4440</td>
<td>core</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-rw--------</td>
<td>cs4440</td>
<td>cs4440</td>
<td>helper.c</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>drwx</td>
<td>cs4440</td>
<td>cs4440</td>
<td><strong>pycache</strong></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>cs4440</td>
<td>cs4440</td>
<td>shellcode.py</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
File System Permissions

- First three represent **Owner’s privileges**

```
-rwXrwxrwx 1 root  cs4440 target6
-rwxrwXr-x 1 cs4440 cs4440 target6.c
-rwxrwxrwx 1 root  cs4440 target7
-rwxrwXr-x 1 cs4440 cs4440 target7.c
-rwxrwxrwx 1 root  cs4440 target8
-rw-rw-r-- 1 cs4440 cs4440 target8.c
-rw-rw-r-- 1 cs4440 cs4440 tmp
```
File System Permissions

- First three represent **Owner’s privileges**
- Next three represent **Group’s privileges**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Owner</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rwxrwxrwx</td>
<td>root</td>
<td>cs4440</td>
<td>target6.c</td>
</tr>
<tr>
<td>rwxrwxr-x</td>
<td>cs4440</td>
<td>cs4440</td>
<td>target6.c</td>
</tr>
<tr>
<td>rwxrwxrwx</td>
<td>root</td>
<td>cs4440</td>
<td>target7.c</td>
</tr>
<tr>
<td>rwxrwxr-x</td>
<td>cs4440</td>
<td>cs4440</td>
<td>target7.c</td>
</tr>
<tr>
<td>rwxrwxrwx</td>
<td>root</td>
<td>cs4440</td>
<td>target8.c</td>
</tr>
<tr>
<td>rw-rw-r--</td>
<td>cs4440</td>
<td>cs4440</td>
<td>tmp</td>
</tr>
<tr>
<td>rw-rw-r--</td>
<td>cs4440</td>
<td>cs4440</td>
<td>tmp</td>
</tr>
</tbody>
</table>
File System Permissions

- First three represent **Owner’s privileges**
- Next three represent **Group’s privileges**
- Last three represent **everyone else**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Owner</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rw-rw-r--</td>
<td>1 root</td>
<td>cs4440 target8.c</td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>1 root</td>
<td>cs4440 tmp</td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>1 cs4440</td>
<td>cs4440 target6.c</td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>1 cs4440</td>
<td>cs4440 target7.c</td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>1 cs4440</td>
<td>cs4440 target8.c</td>
</tr>
<tr>
<td>-rw-rw-r--</td>
<td>1 cs4440</td>
<td>cs4440 tmp</td>
</tr>
</tbody>
</table>
Permission Puzzles

1. No permissions?
   - ??? ??? ???

2. Read, Write, Exec only for owner?
   - ??? ??? ???

3. Execute for all?
   - ??? ??? ???

4. Owner can read, write, & exec; Group can only read; and all others have no permissions.
   - ??? ??? ???

Stefan Nagy
Permission Puzzles

1. No permissions?

2. Read, Write, Exec only for owner?

3. Execute for all?

4. Owner can read, write, & exec; Group can only read; and all others have no permissions.
Process Permissions

- Every process has one **uid**, up to many **gids**
- Actions: **create**, **kill**, **debug** (ptrace)
- Login process (**uid**=0, **root**)
  - Checks (username, password) tuple
  - Changes uid to user's value (via setuid)
  - Start's user's shell (/bin/sh)
    - Processes now run as current user!

- **setuid** binaries
  - Program runs with **uid** of owner (e.g., **root**)
  - Not the parent process!
  - Examples: /bin/su, /bin/sudo
Network Permissions

- **Connect**
  - Liberal permissions

- **Listen**
  - Liberal permissions
  - Ports below 1024 reserved for **system**
    - Requires special permissions!

- **Read/write data**
  - As long as you have a file descriptor!

- **Send/receive raw packets**
  - Must be associated with an existing connection
  - Otherwise `uid=0 (root)`
Process Isolation
Goal: ???
**Goal:** minimize damage by **isolating** every process.
Process Isolation

- We can’t just rely on permission schemes
  - Assume attackers can (and will) bypass them

- **Security Goal:** prevent **cross-process** memory access or memory corruption

```
usr= Steve
pwd= cs4440
```

```
read(wf.usr)
read(wf.pwd)
```
We can’t just rely on permission schemes
- Assume attackers can (and will) bypass them

**Security Goal:** prevent *cross-process* memory access or memory corruption

**Memory Management Unit**
- Hardware that acts as gatekeeper of memory
- Translates virtual memory to physical memory
Isolating Process Memory

- **Memory Management Unit**
  - Translates virtual memory to physical memory
Isolating Process Memory

- Memory Management Unit
  - Translates virtual memory to physical memory
  - Enforce Process-1 cannot access of Process-2's memory!
**Goal:** minimize damage by isolating every process

**Caveat:** ???
**Goal:** minimize damage by **isolating** every process

**Caveat:** you must **trust** all potential isolation bridges
What about **malicious peripherals**?
- Assume plugged-in USB’s are hostile!
What about malicious peripherals?
- Assume plugged-in USB’s are hostile!

Solution: the Input/Output (IO) MMU
- Same idea as MMU, but extended to devices
- IO means “input” / “output” devices; e.g.:
  - Network
  - Keyboard
  - USB stick
  - Graphics cards
  - ...
  - Anything that uses a device driver
- **Problem:** any processes you execute will inherit **your** privileges, resources
Resource-level Isolation

- **Problem**: any processes you execute will inherit *your* privileges, resources

- **Process Sandbox**: tight, controlled set of resources to execute guest programs
  - Scratch space on disk and memory
  - Network access
  - Ability to inspect the host system or
  - Read from input devices are usually disallowed or heavily restricted
Problem: any processes you execute will inherit your privileges, resources.

Process Sandbox: a tight, controlled set of resources to execute guest programs.
- Scratch space on disk and memory
- Network access
- Ability to inspect the host system or
- Read from input devices are usually disallowed or heavily restricted

Resource-level Isolation
Sandboxing

Goal: give processes the least privileges

Caveat: ???
**Goal:** give processes the **least privileges**

**Caveat:** the **trusted computing base** is still very large!
Goal: make libraries, middleware specific to each process

Caveat: ???
**Goal:** make **libraries**, **middleware** specific to each process

**Caveat:** the trusted computing base is now the **OS** and **HW**
Virtual Machines

Containers

Virtual Machines

App 1
App 2
App 3

Bins/Lib
Bins/Lib
Bins/Lib

Container Engine

Operating System

Infrastructure

App 1
App 2
App 3

Bins/Lib
Bins/Lib
Bins/Lib

Guest OS
Guest OS
Guest OS

Hypervisor

Infrastructure

Stefan Nagy
**Virtual Machines**

**Goal:** completely isolate the OS

**Caveat:** ???
Goal: completely isolate the OS

Caveat: the trusted computing base now the Hypervisor
Caveats

- Sandboxes, containers, and hypervisors are all software...
Sandboxes, containers, and hypervisors are all software... with **vulnerabilities** too!

**CVE-2022-0185 in Linux Kernel Can Allow Container Escape in Kubernetes**

Last week, a new high-severity CVE was released that affects the Linux kernel. This vulnerability provides an opportunity for an attacker who has access to a system as an unprivileged user to escalate those rights to root. To do this, the attacker must have a specific Linux capability, `CAP_SYS_ADMIN`, which reduces the risk of breakout in some container cases. But in many Kubernetes clusters, it’s likely that an attacker could exploit this issue.

At the moment, there is no public exploit code for this issue. However, one of the researchers who found it has posted a proof of concept showing a container breakout, and it’s expected that exploit code will be released soon.

Integer overflow in Skia in Google Chrome prior to 112.0.5615.137 allowed a remote attacker who had compromised the renderer process to potentially perform a sandbox escape via a crafted HTML page. (Chromium security severity: High)
Questions?
Next time on CS 4440...

Security in Practice: Malware