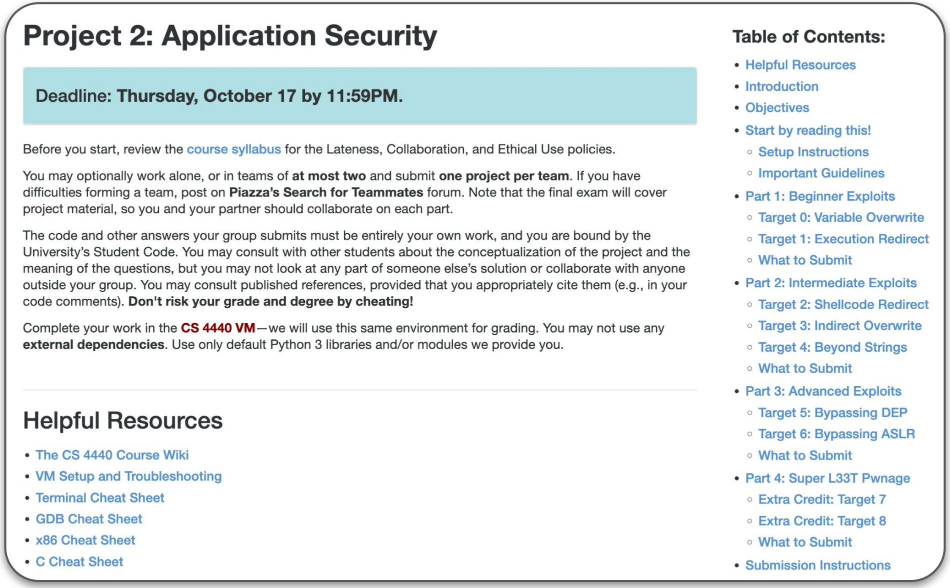
Week 7: Lecture A

Access Control & Isolation

Tuesday, October 1, 2024

Announcements

- **Project 2: AppSec** released
  - **Deadline:** Thursday, October 17th by 11:59PM



Project 2: Application Security

Deadline: Thursday, October 17 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
- x86 Cheat Sheet
- C Cheat Sheet

Table of Contents:

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

- **Project 1** grades **and regrades** are now available on **Canvas**

- **Statistics:**
  - Average score: **100%**
  - Last year's average: **85%**

- **Fantastic job!**

Announcements

See Discord for meeting info!

utahsec.cs.utah.edu

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**

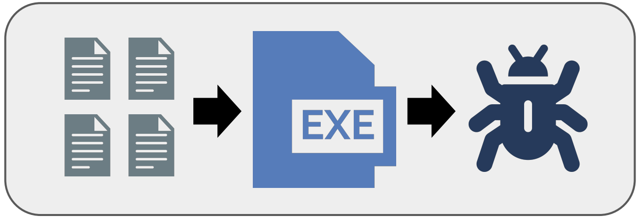
Proactive Vulnerability Discovery

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:

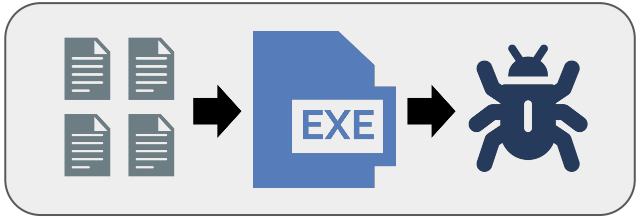


Proactive Vulnerability Discovery

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:**

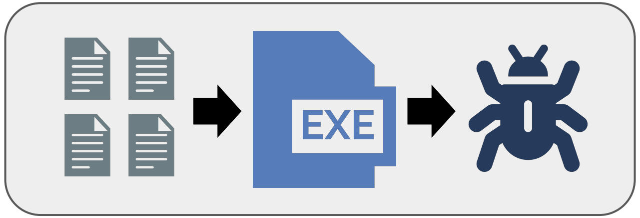
Proactive Vulnerability Discovery

Static Analysis:

- Analyze program **without running it**
- **Challenges:**
  - **False negatives** (vulnerabilities missed)
  - **False positives** (results are unusable)

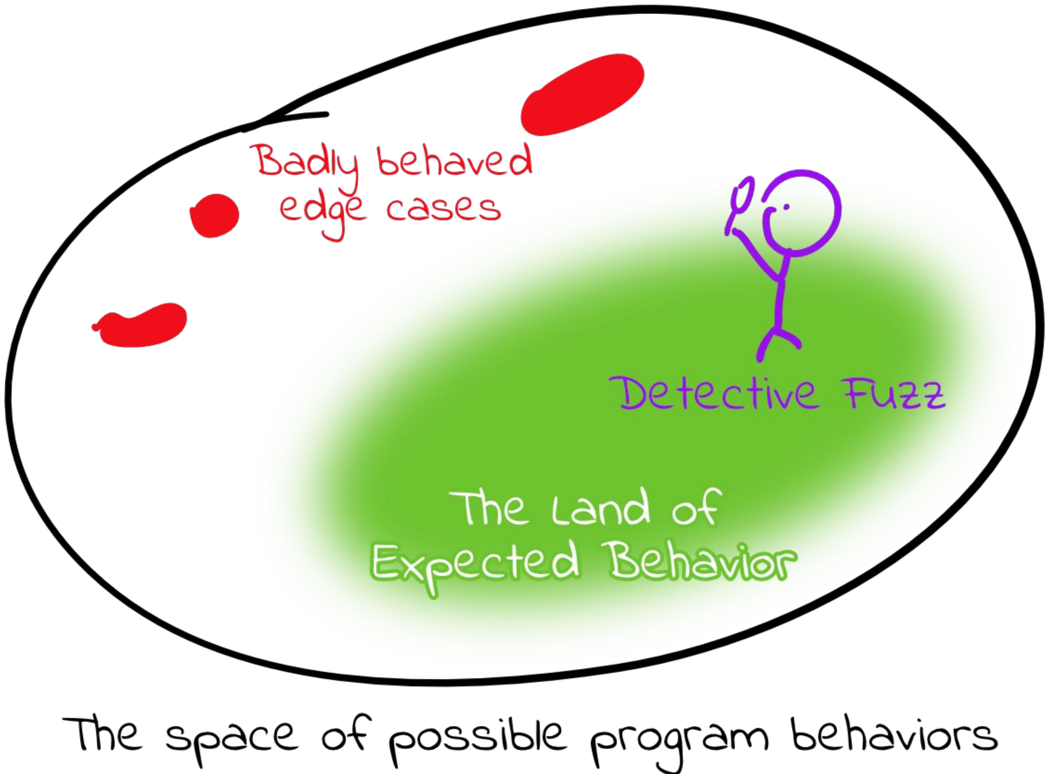
  As code size grows, **analysis speed drops**

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**

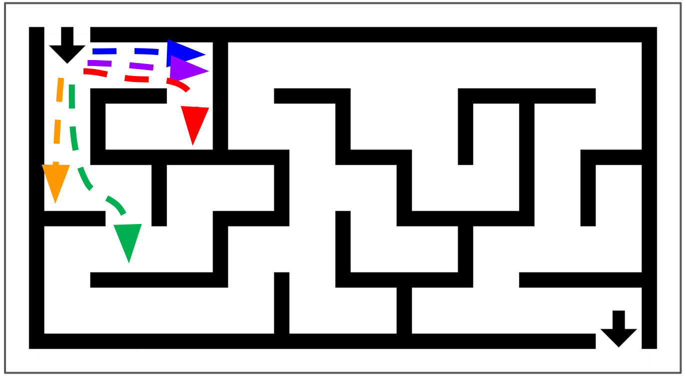
Finding Bugs with Fuzzing

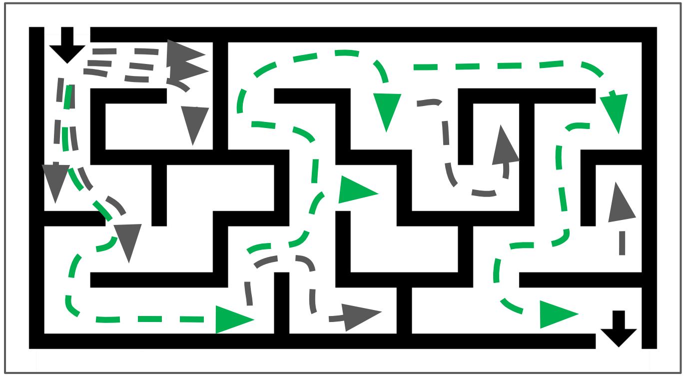


Source: https://blog.trailofbits.com/2020/10/22/lets-build-a-high-performance-fuzzer-with-gpus/

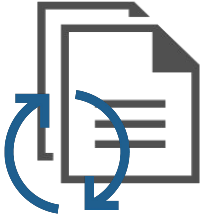
Why do we need feedback in fuzzing?

Why do we need feedback in fuzzing?





Feedback-driven Fuzzing

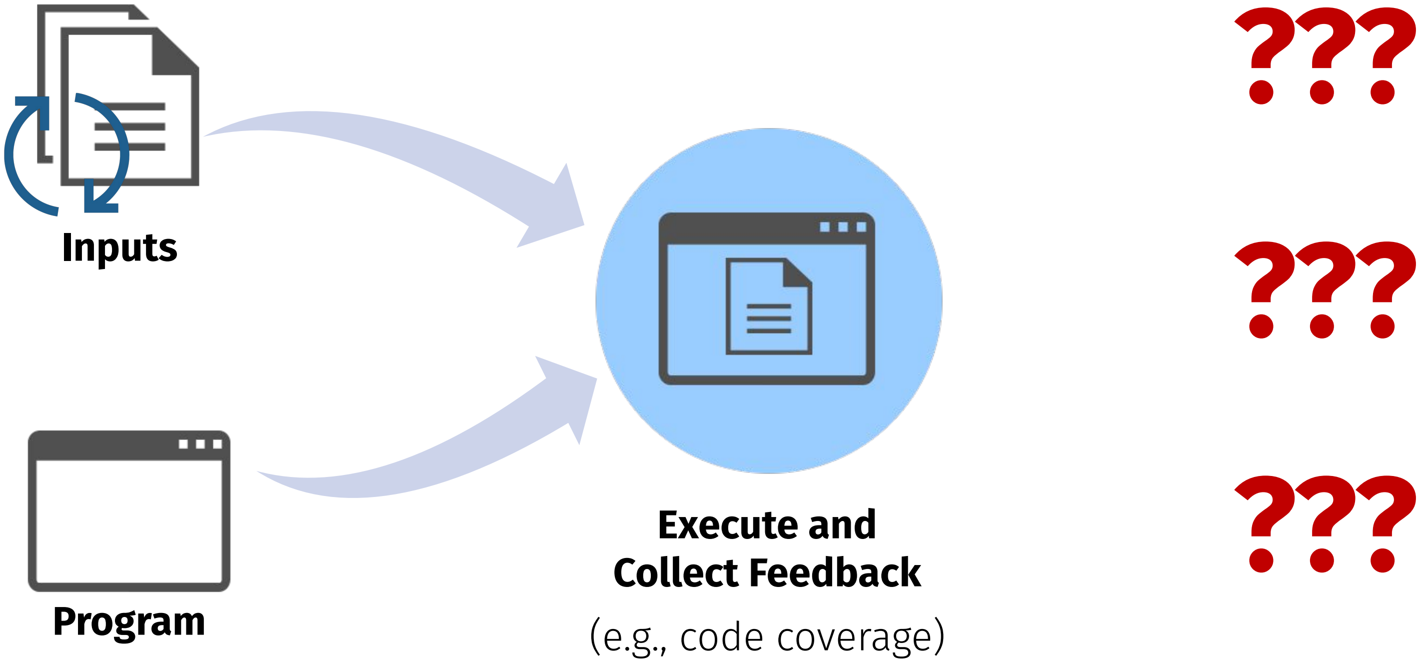


Inputs

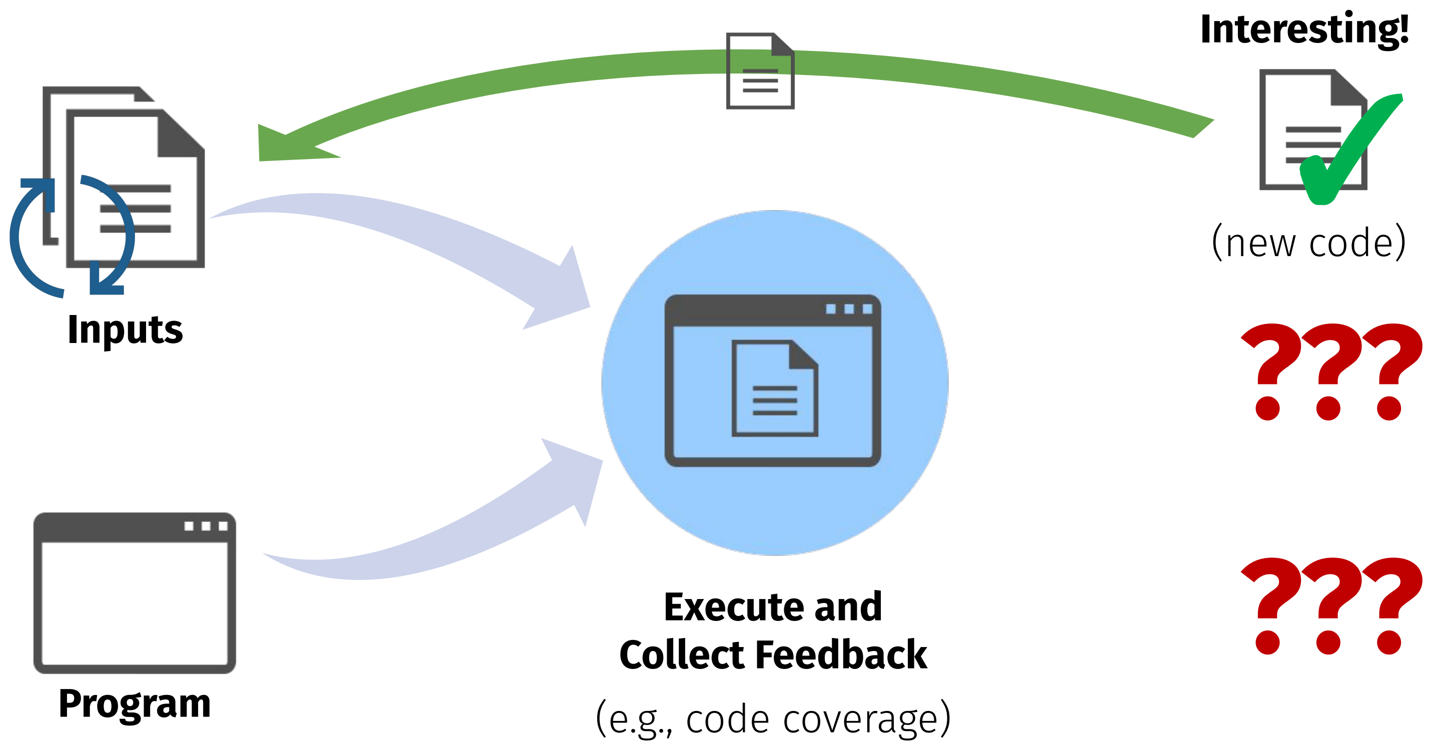
???

Program

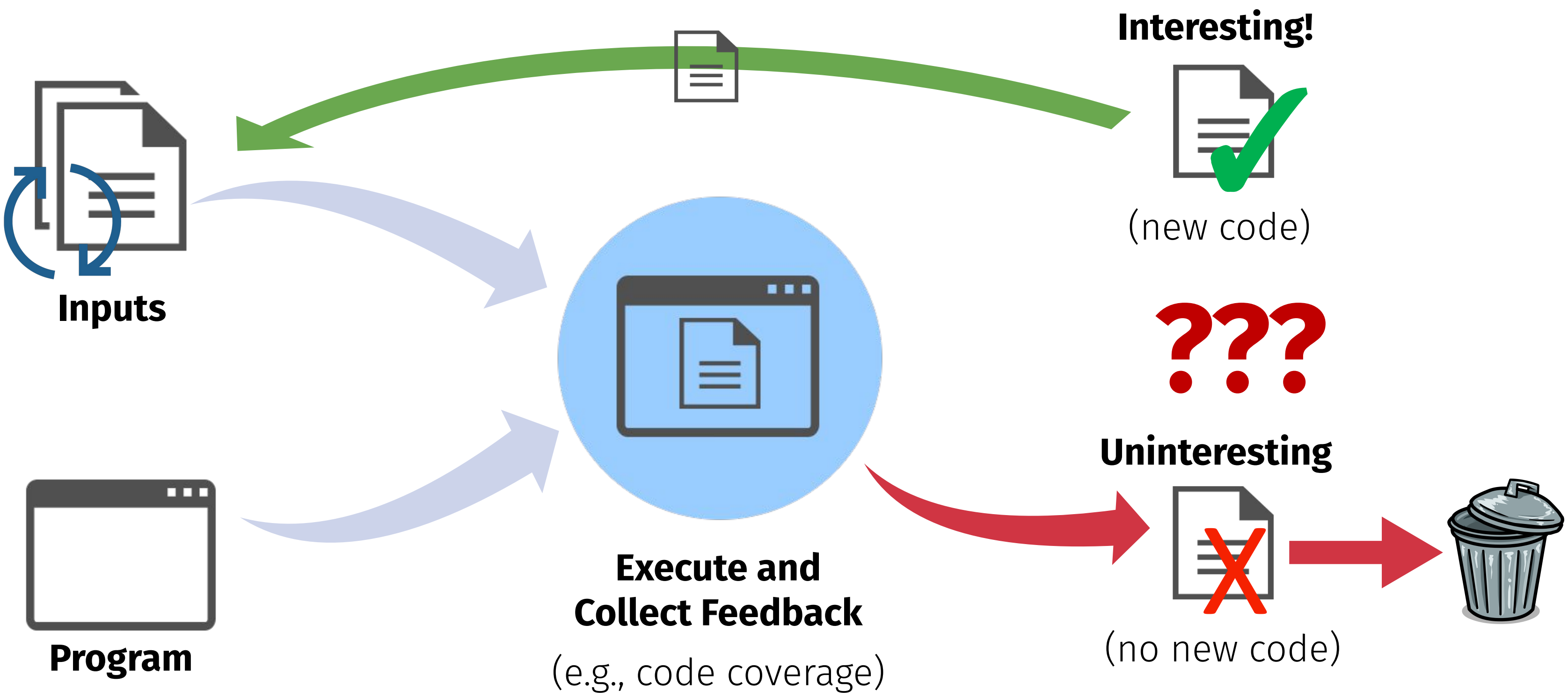
Feedback-driven Fuzzing



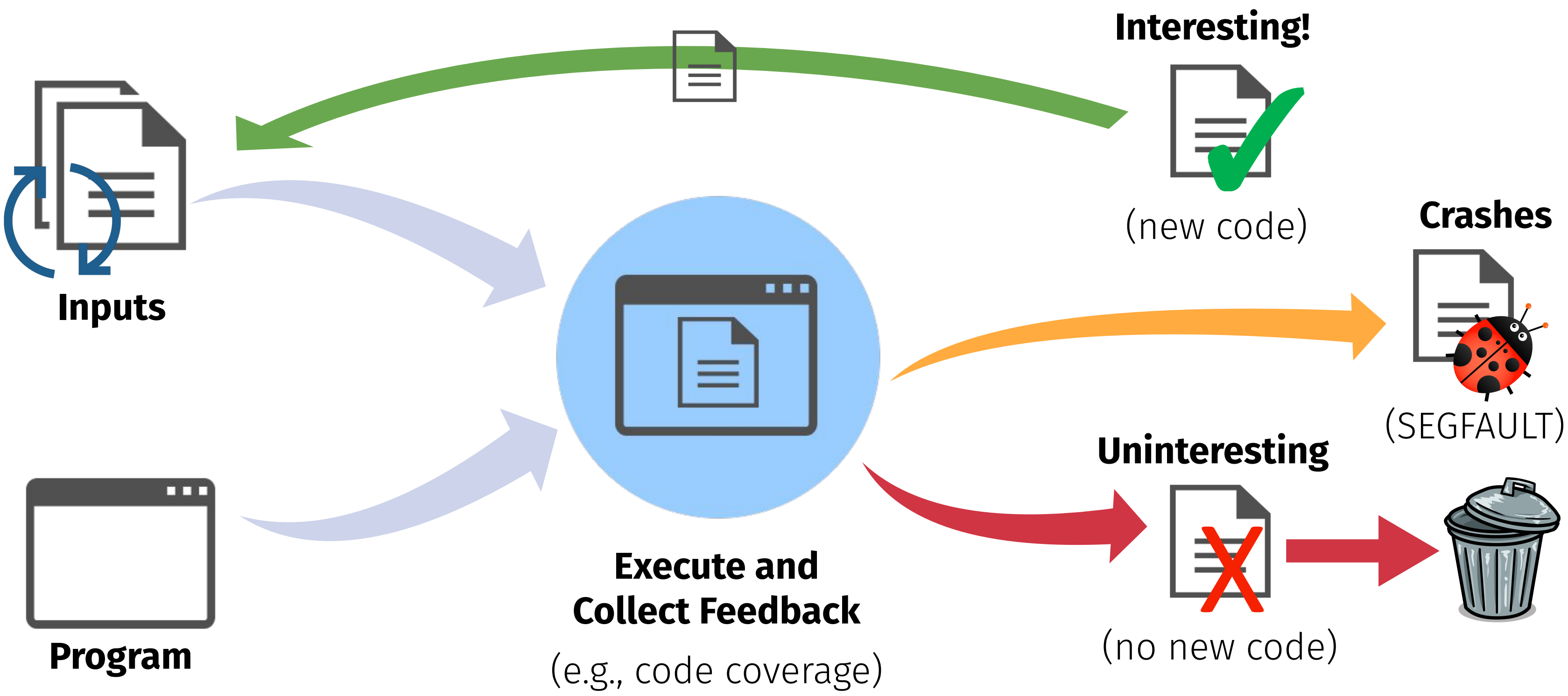
Feedback-driven Fuzzing



Feedback-driven Fuzzing

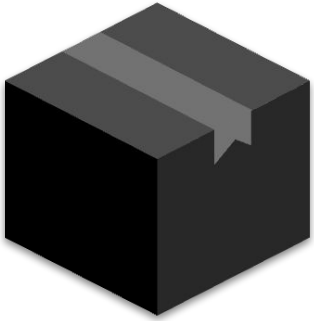


Feedback-driven Fuzzing



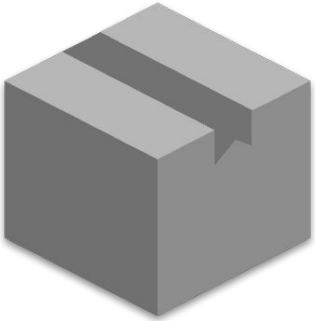
Types of Feedback-driven Fuzzers

Black-box



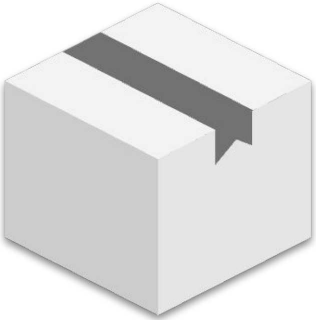
???

Grey-box



???

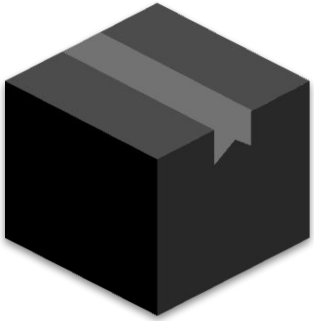
White-box



???

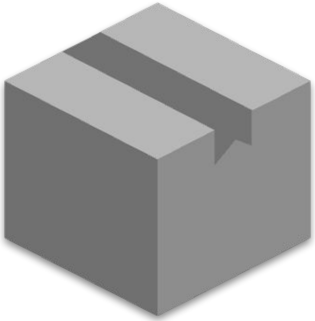
Types of Feedback-driven Fuzzers

Black-box



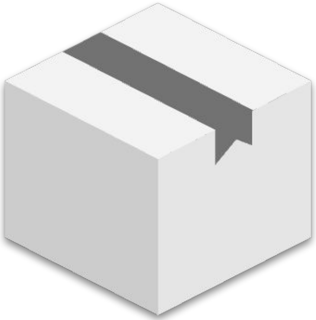
Zero Introspection

Grey-box



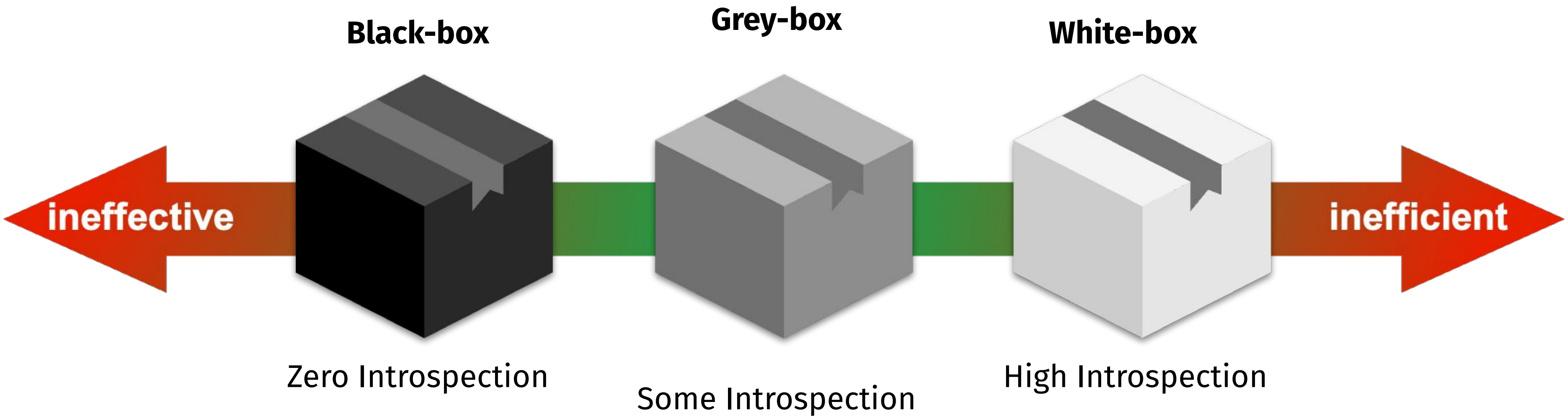
Some Introspection

White-box

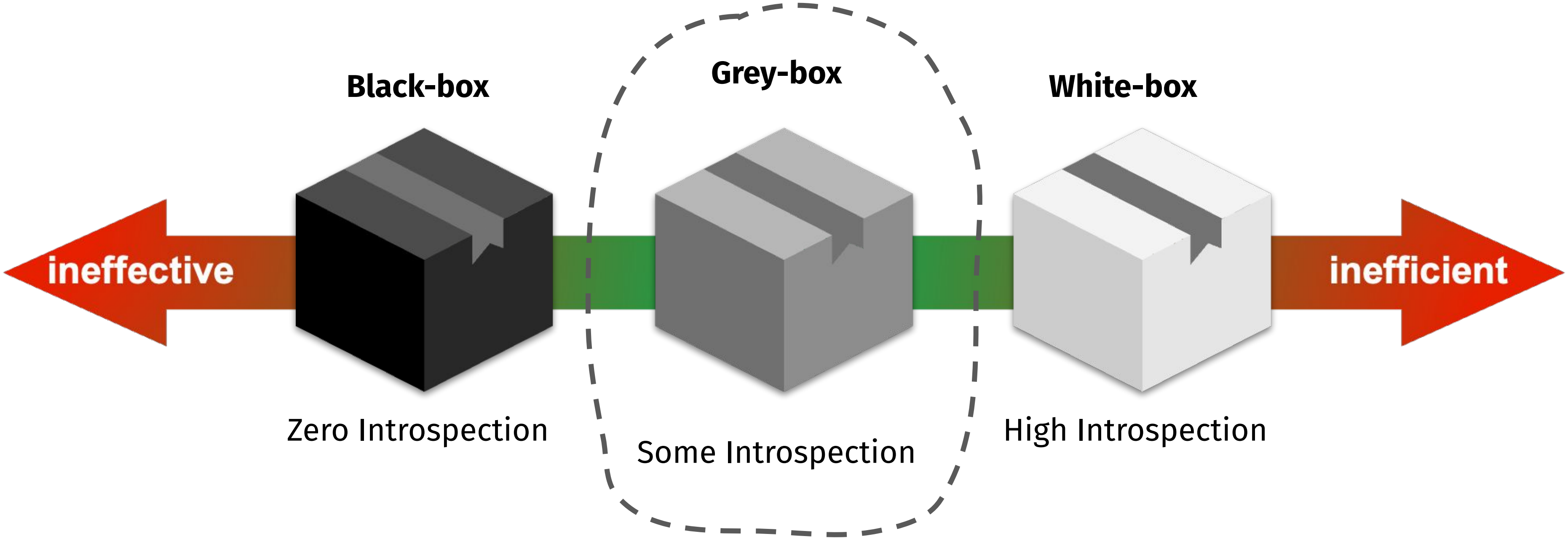


High Introspection

Types of Feedback-driven Fuzzers



Types of Feedback-driven Fuzzers



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

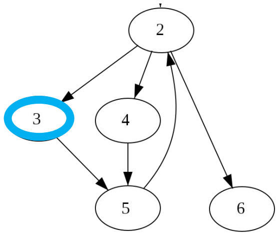
```
177x  function fib(n) {
177x    if (n === 0) {
 34x      return 0
177x    } else if (n === 1) {
 55x      return 1
143x    } else if (n > 1) {
 88x      return fib(n - 1) + fib(n - 2)
        } else {
          thrower()
        }
177x  }
  1x  console.log('fib(10):', fib(10))
```

Code Coverage Metrics

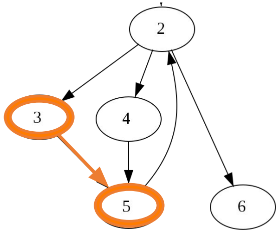
- Program represented as **control-flow graphs (CFG)**
  - Directed graph encompassing all program paths
  - Basis of virtually all software analysis techniques

- 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

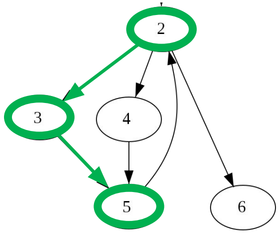
Blocks



Edges



Paths



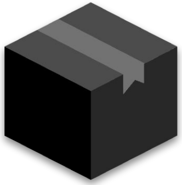
Impact of Code Coverage

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

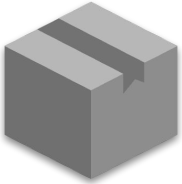
Impact of Code Coverage

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

Estimated Mutations Required



???

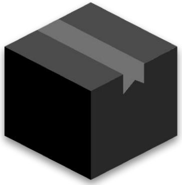


???

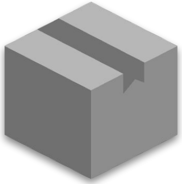
Impact of Code Coverage

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

Estimated Mutations Required



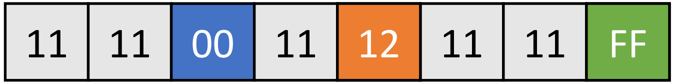
$(2^8)^4$ = **4,294,967,296**



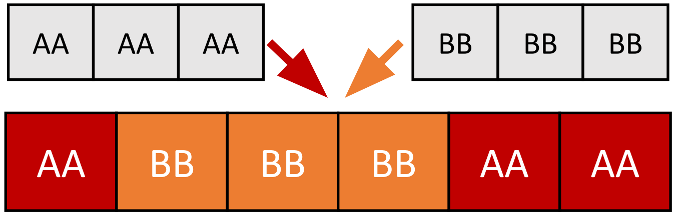
$4*(2^8)$ = **1,024**

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



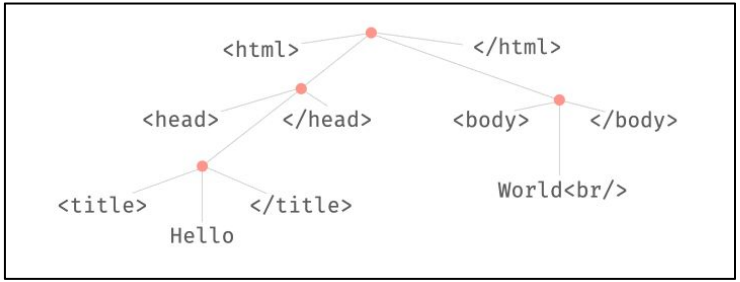
- **The good:** super fast
  - Incorporating feedback like coverage enables you to **synthesize valid inputs** (eventually)



Model-guided Input Generation

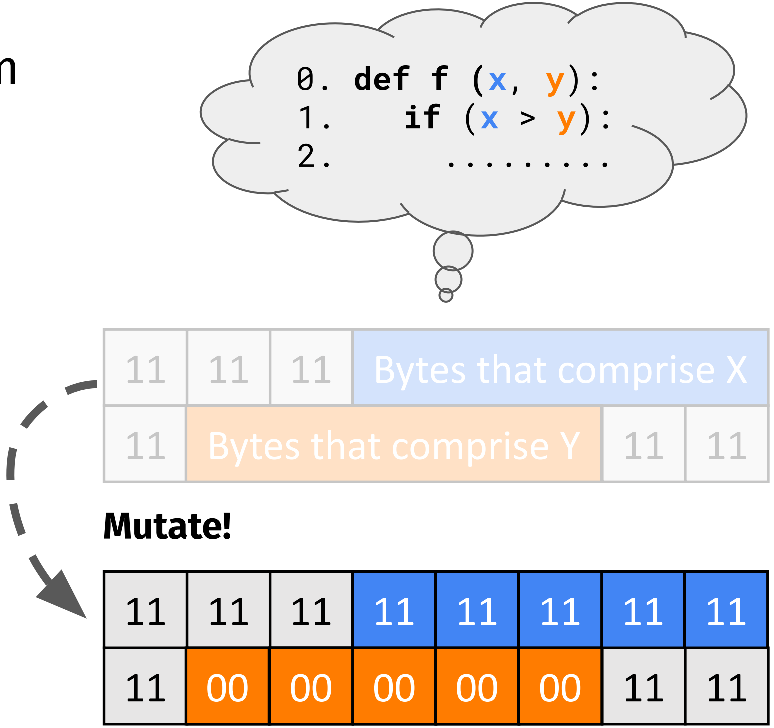
- Follow a pre-defined input **specification**
  - Pre-defined input grammars
  - Dynamically-learned grammars
  - Domain-specific generators
- **The good:** many more valid inputs
  - Model-agnostic inputs are often discarded because they fail basic input sanity checks
  - Valid inputs = **higher code coverage**

```
XML_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>>", "<<id> <xml-attribute>>"],
    "<xml-openclose-tag>": ["<<id>/>", "<<id> <xml-attribute>/>"],
    "<xml-close-tag>":     ["</<id>>"],
    "<xml-attribute>":     ["<id>=<id>", "<xml-attribute> <xml-attribute>"],
    "<id>":                ["<letter>", "<id><letter>"],
    "<text>":              ["<text><letter_space>", "<letter_space>"],
    "<letter>":            srange(string.ascii_letters + string.digits +
                                  "\"" + "'" + "."),
    "<letter_space>":      srange(string.ascii_letters + string.digits +
                                  "\"" + "'" + " " + "\t"),
}
```



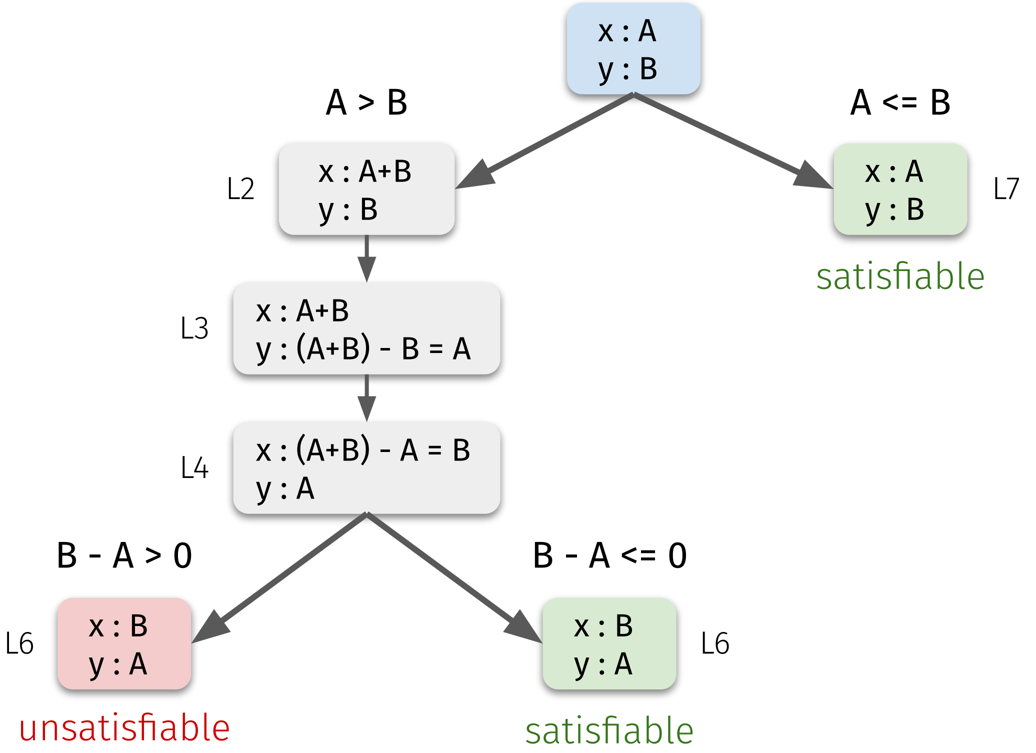
Taint Tracking

- 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



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)
```



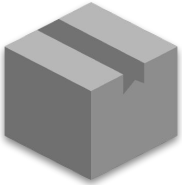
Possible path constraints:

- (A > B) and (B-A > 0) = unsatisfiable
- (A > B) and (B-A <= 0) = satisfiable
- (A <= B) = satisfiable

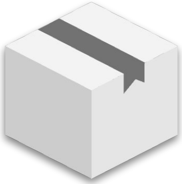
Feedback-driven Fuzzing vs. Symbolic Execution

```
if(x^3 == 18816723022905622263)
    OVERFLOW();
}
```

Estimated Mutations Required



???

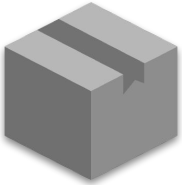


???

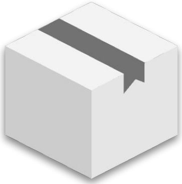
Feedback-driven Fuzzing vs. Symbolic Execution

```
if(x^3 == 1881672302290562263)
    OVERFLOW(); // x = 1234567
}
```

Estimated Mutations Required



Good luck!

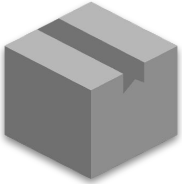


Solves instantly

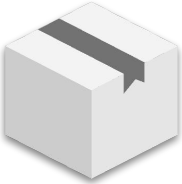
Feedback-driven Fuzzing vs. Symbolic Execution

```
if(A^3 + B^3 + C^3 == 33)
    OVERFLOW();
}
```

Estimated Mutations Required



???



???

Feedback-driven Fuzzing vs. Symbolic Execution

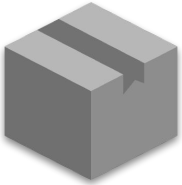
```
if(A^3 + B^3 + C^3 == 33)
    OVERFLOW();
}
```

A = 8,866,128,975,287,528

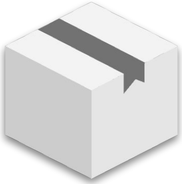
B = -8,778,405,442,862,239

C = -2,736,111,468,807,040

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**

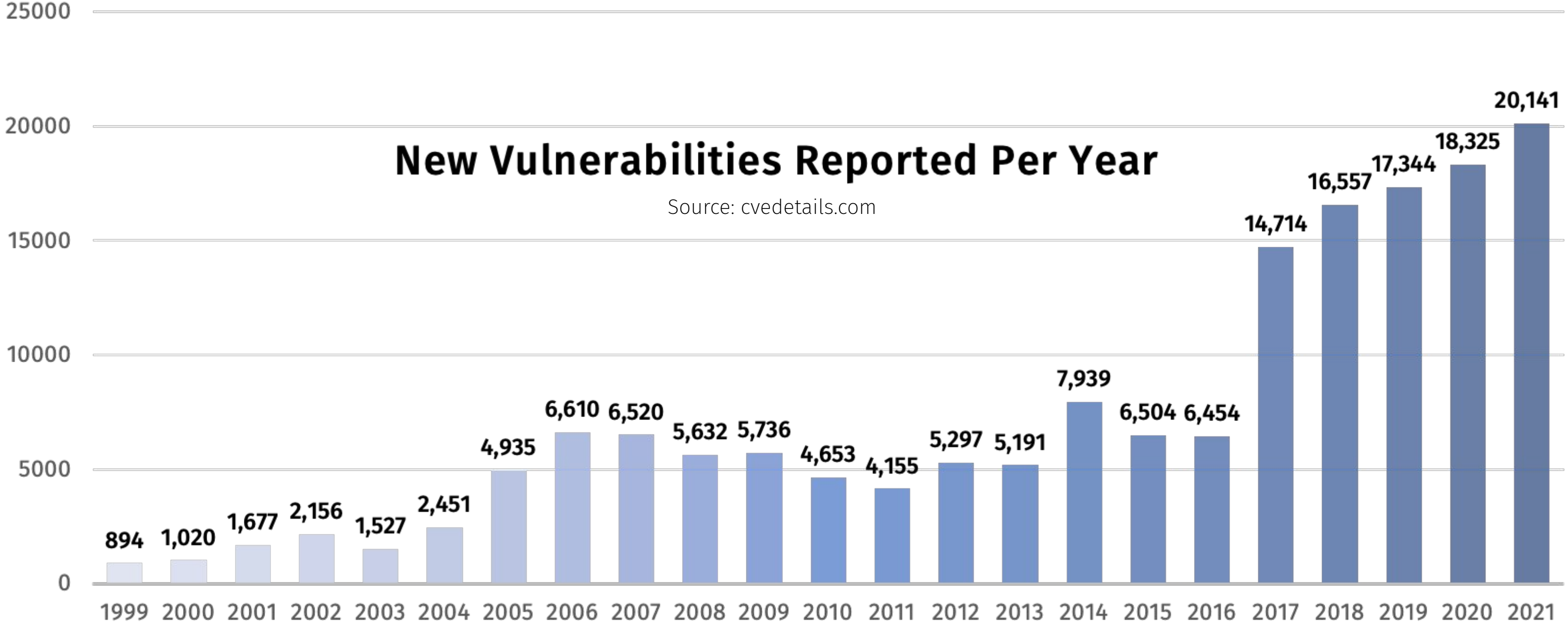
- **Model-guided Fuzzing:**
  - **Advantages:** more valid inputs leads to **higher coverage earlier on**
  - **Challenges:** out of luck if specification is **not defined** or **hard-to-define**

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

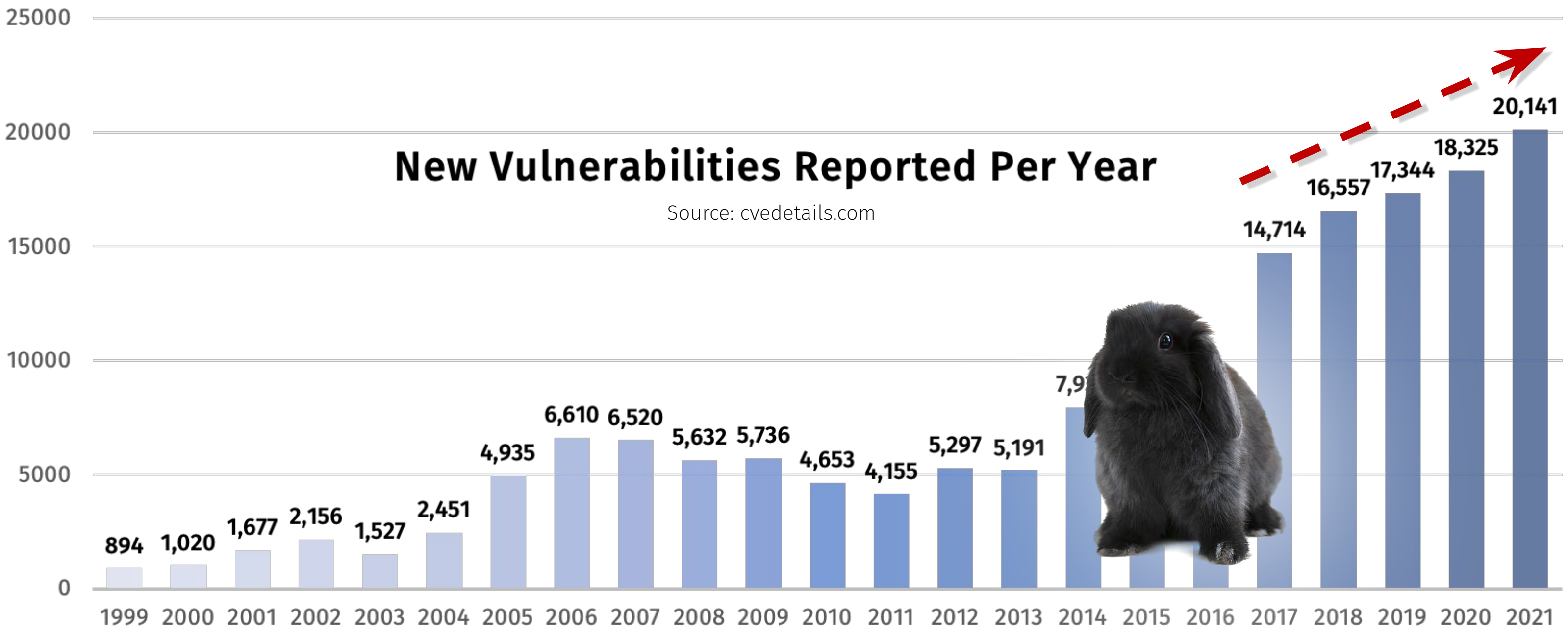
Input Generation Trade-offs

- **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 Fuzzing



Impact of Fuzzing



Questions?

Interested in fuzzing?

- **Spring 2025: CS 5963/6963: Applied Software Security Testing**
  - **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/

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. Projects will provide hands-on experience with real-world security tools like AFL++ and AddressSanitizer, culminating in a final project where **you'll team up to hunt down, analyze, and report security bugs in a real 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++.

Professor

Stefan Nagy

Interested in fuzzing?

CS 5963 - 001 Applied S/W Secur Test

Class Details

Class Number: 14578 | Instructor: NAGY, STEFAN | Component: Special Topics | Type: In Person | Units: 3.0

Requisites: Yes | Wait List: No | View Feedback

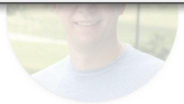
This class will prepare students to become effective software testers capable of automating vulnerability discovery in today's large and complex software systems. This course will cover the fundamental design considerations behind today's state-of-the-art software testing tools, and equip students with 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

Food for Thought

- 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?

Principles of a Safe System

- Clearly we can't assume **Application Developers** will write **safe code**...

Principles of a Safe System

- 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?

Principles of a Safe System

- 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

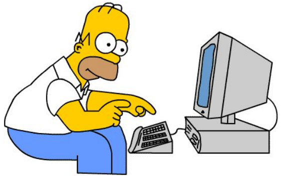
Access Control

Access Control

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

- **Goal: ???**

Root Admin



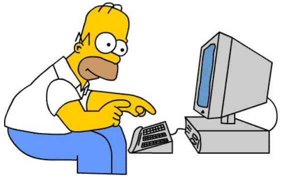
Non-root Users

Access Control

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

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

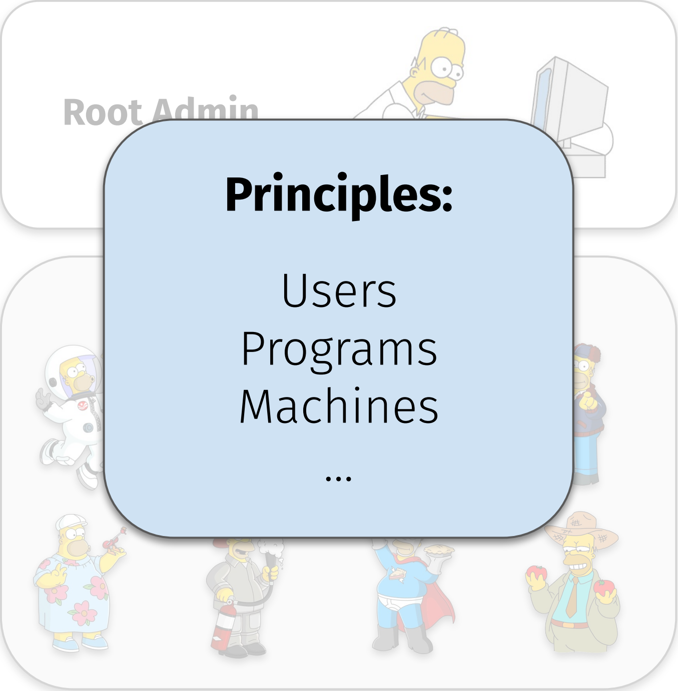
Root Admin



Non-root Users

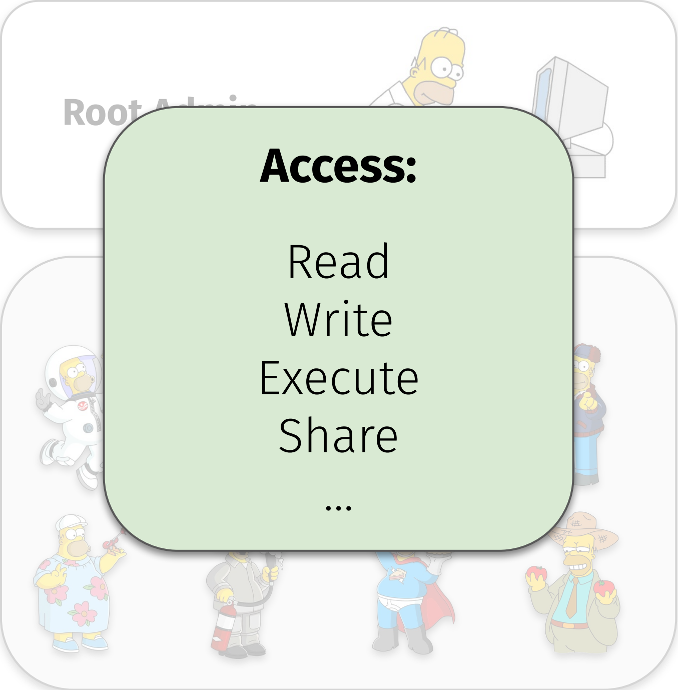
Access Control

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



Access Control

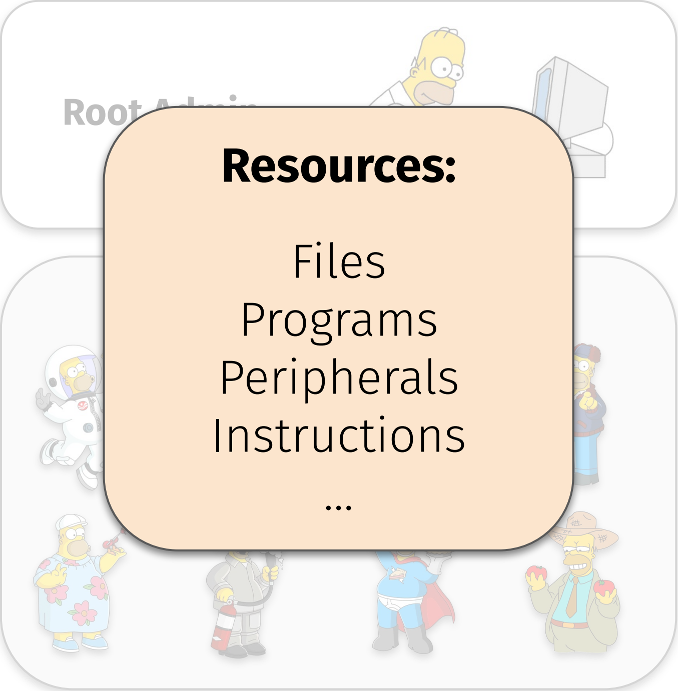
- **Access Control:** the heart of security on commodity computing systems
- **Goal:** control which **principles** have **access** to which system **resources**



Access Control

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

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

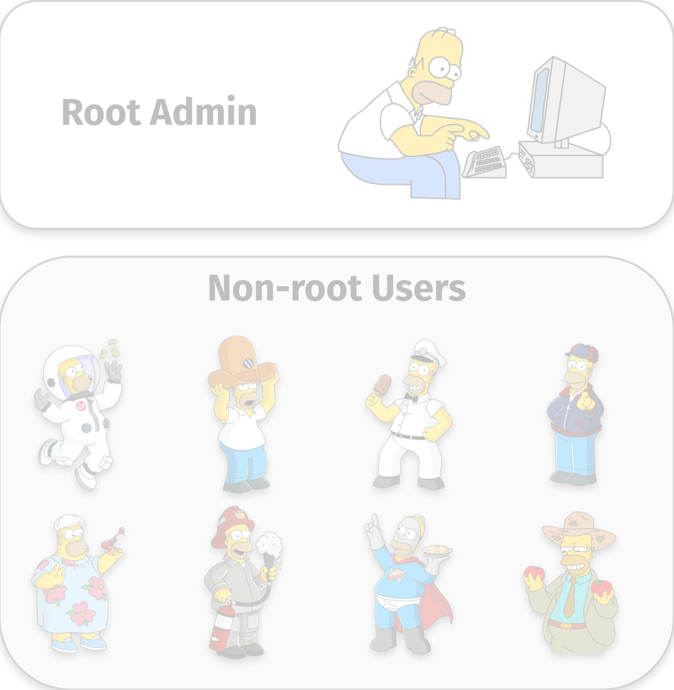


Access Control

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

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

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

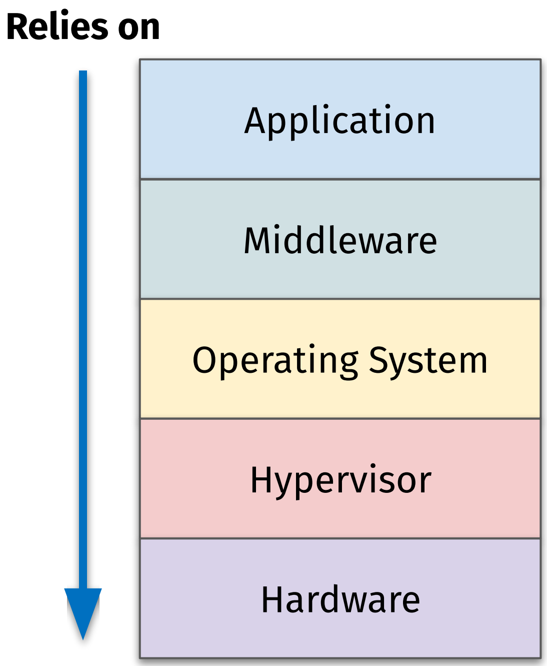


Access Control

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

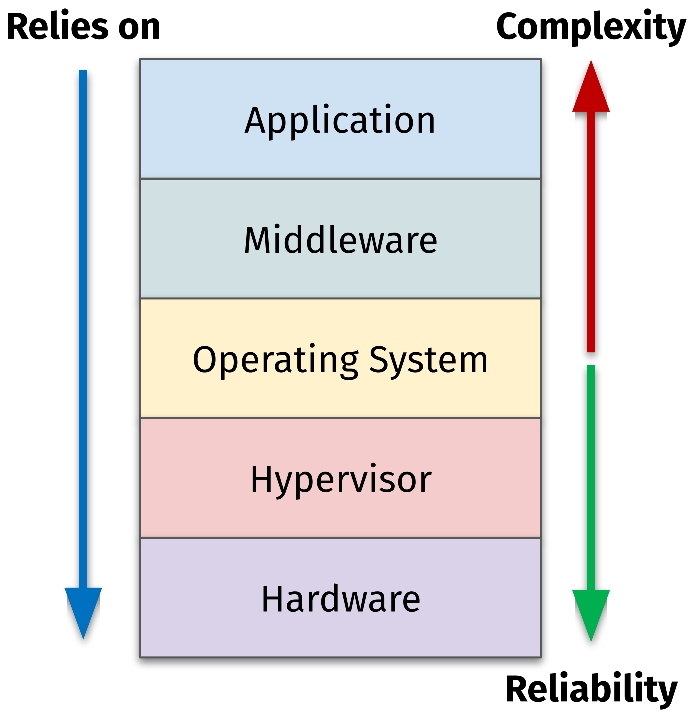
- **Goal:** control which **principles** have **access** to which system **resources**

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



Access Control

- **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"

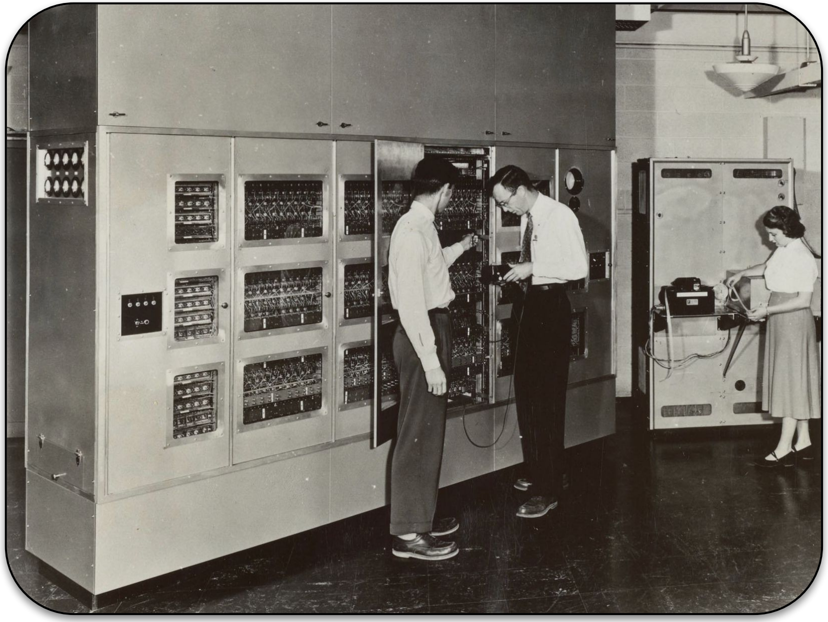
- **ENIAC**
  - The first programmable, electronic, general-purpose digital computer
  - Built in 1945 by U.S. Army / UPenn
  - Access control consisted of just a **single user** and a **single program**

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

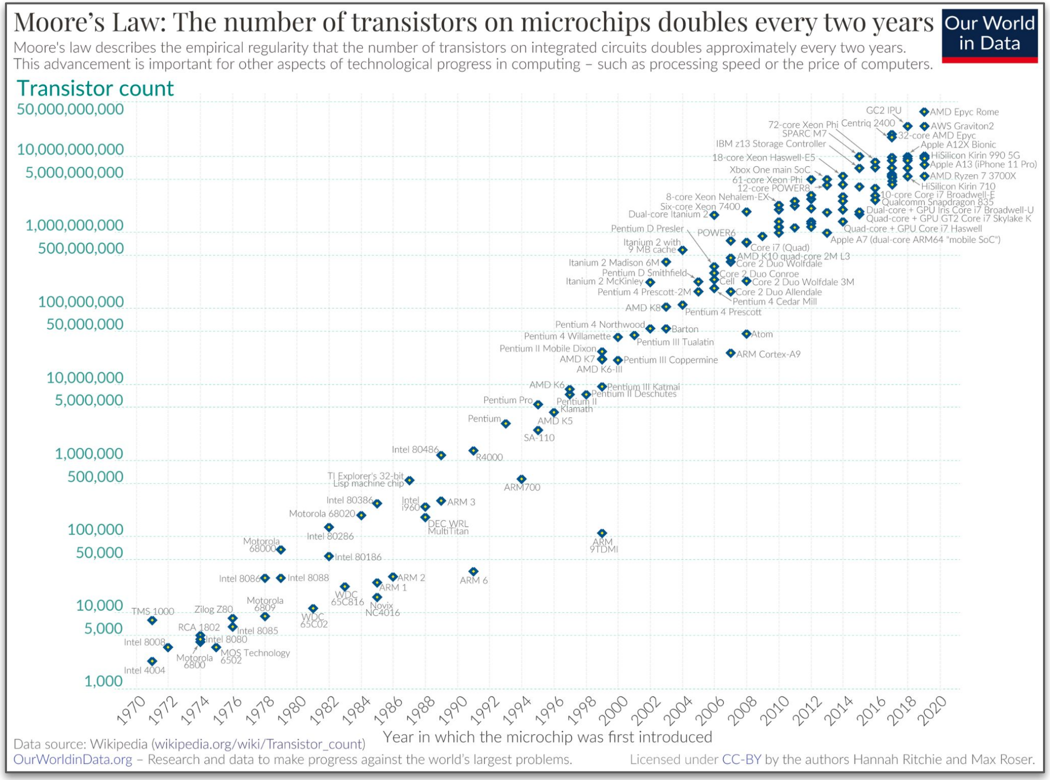
Access Control History

- **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

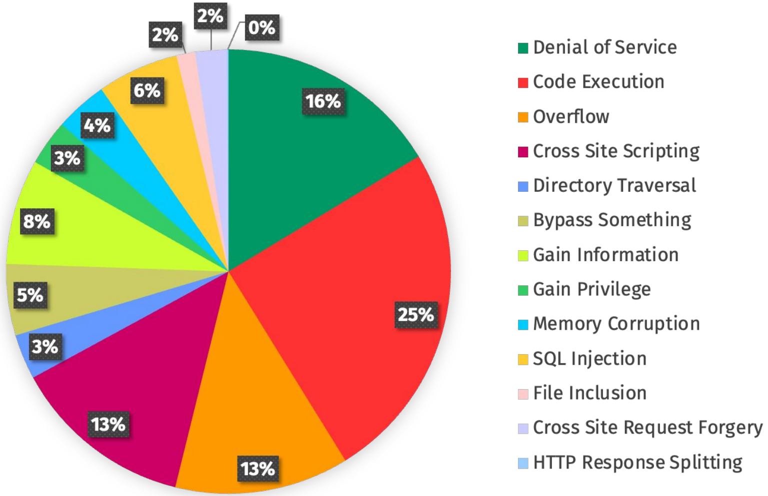
- **By 1980:** we all have access to computers!

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**

Prevention and Detection

- Bugs are **inevitable** in any complex software system
- **NIST: 10–50 bugs** per every **1000 code lines**
- Many bugs are never found



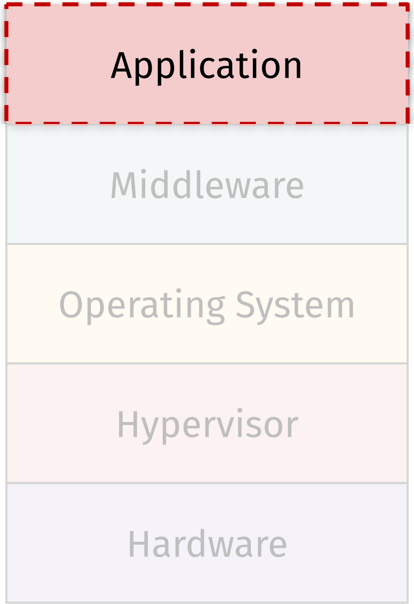
Prevention and Detection

- 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**

Implementing Access Control

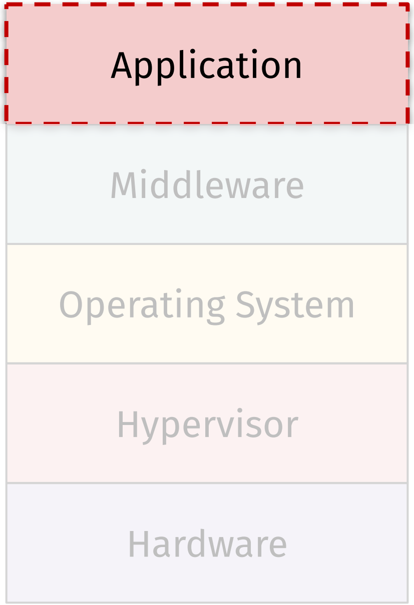
Isolating Applications

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



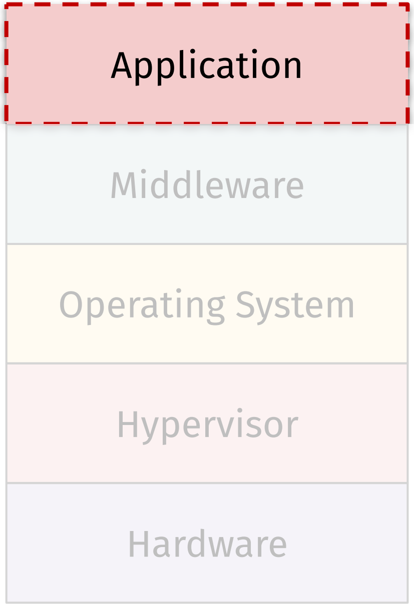
Isolating Applications

- **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

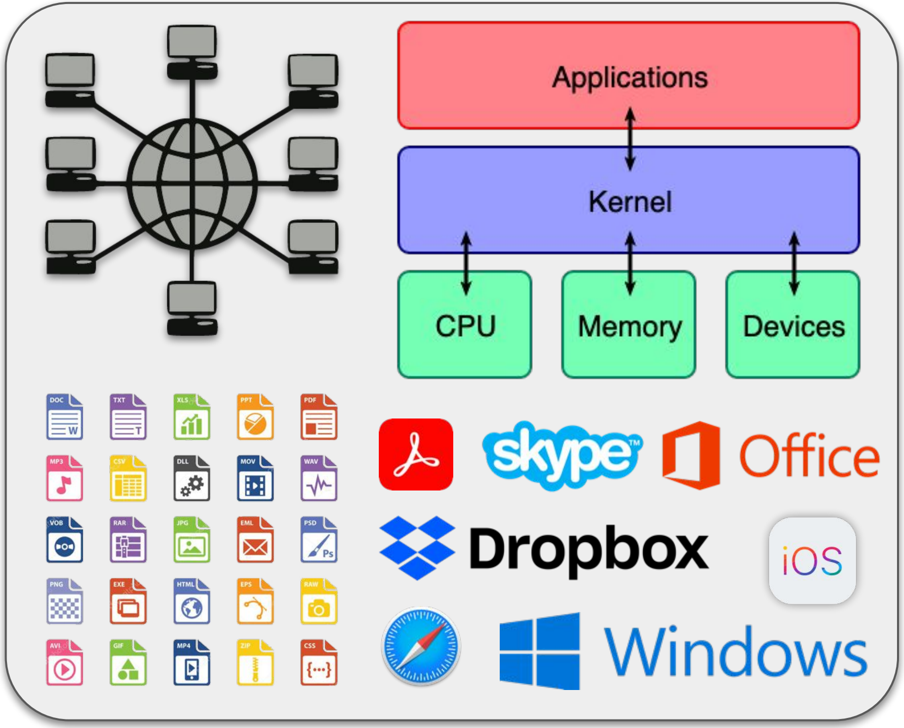
- **Adversary 1:** **exploited** user-space process
  - **Targets 2–8** in Project 2 (after your attacks)
- **Adversary 2:** **malicious** user-space process
  - Spyware app your aunt installed
  - That TikTok app that ***you*** installed
- **Goal:** **protect the system** (i.e., all other processes + the OS) from an **evil process**



What specific resources must we protect?

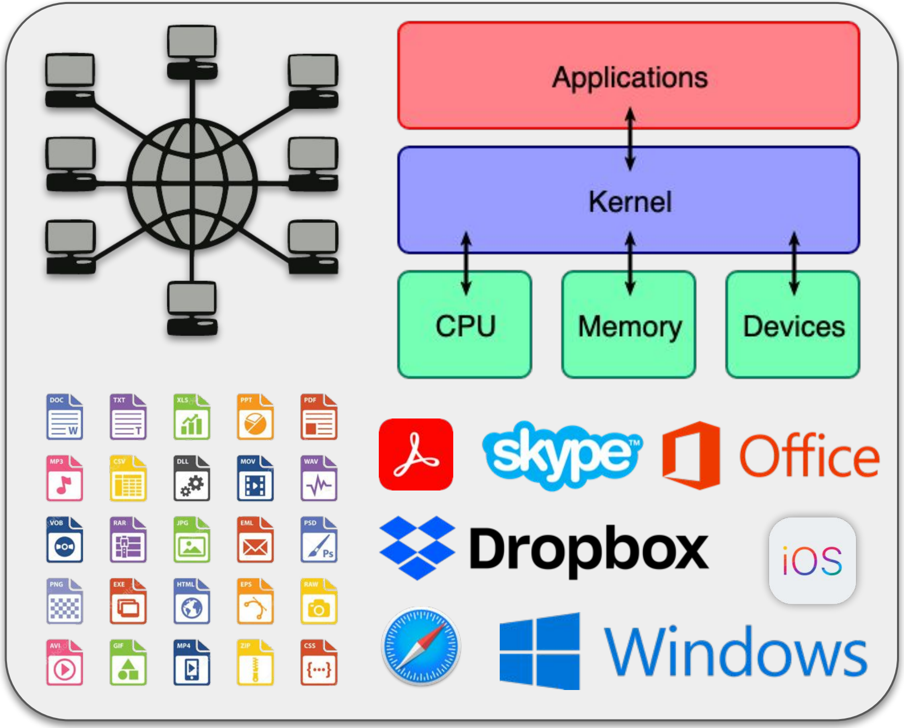
What specific resources must we protect?

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



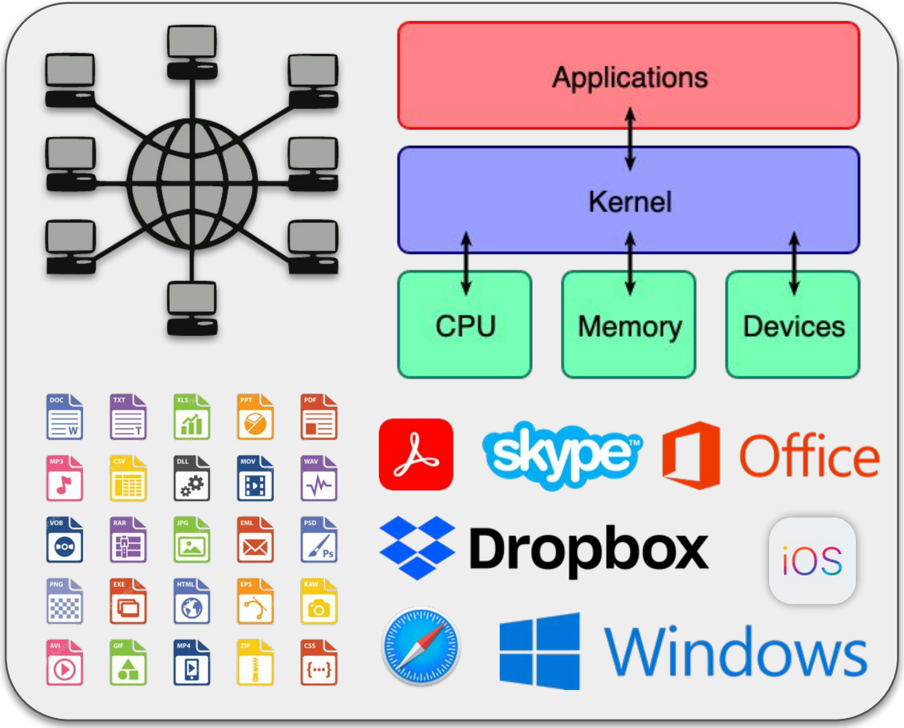
What specific resources must we protect?

- **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



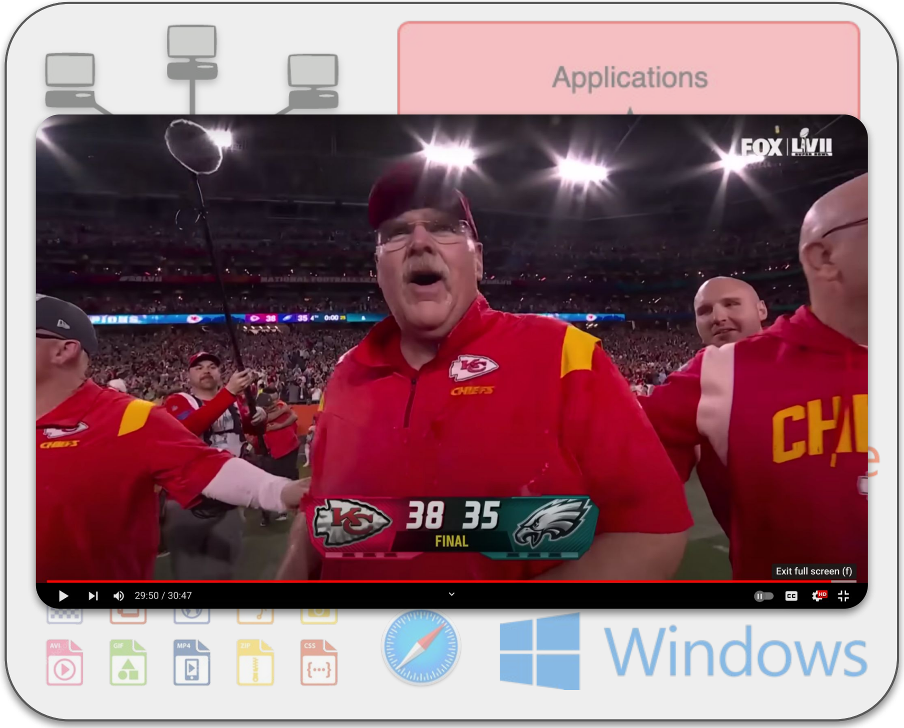
What specific resources must we protect?

- **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

	OS	Accounting Program	Accounting Data	Insurance Data	Payroll Data
Bob	R X	R X	R	–	–
Alice	R X	R X	R	R W	R W
Sam	R W X	R W X	R	R W	R W
Accounting Program	R X	R X	R W	R W	R W

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

	OS	Accounting Program	Accounting Data	Insurance Data	Payroll Data
Bob	R X	R X	R	–	–
Alice	R X	R X	R	R W	R W
Sam	R W X	R W X	R	R W	R W
Accounting Program	R X	R X	R W	R W	R W

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

R	W	X	O
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

cs4440@cs4440: ~/targets

File Actions Edit View Help

cs4440@cs4440: ~/targets

```
cs4440@cs4440:~/targets$ users
cs4440
cs4440@cs4440:~/targets$ groups
cs4440 adm cdrom sudo dip plugdev lpadmin sambashare
cs4440@cs4440:~/targets$ ls -la
total 7184
drwxrwxr-x  4 cs4440 cs4440   4096 Feb 15 14:37 .
drwxr-xr-x 17 cs4440 cs4440   4096 Feb 15 14:37 ..
drwxrwxr-x  2 cs4440 cs4440   4096 Feb 15 09:31 bin
-rw-rw-r--  1 cs4440 cs4440   3751 Jan 16 20:03 build.py
-rwxrwxr-x  1 cs4440 cs4440    479 Jan 16 20:03 build.sh
-rw-rw-r--  1 cs4440 cs4440     11 Feb 15 14:36 cookie
```

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

```
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 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 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 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 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

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

Owner

```
-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**

```
                       Owner    Group
-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**
- Last three represent **everyone else**

```
                   Owner   Group
-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
```

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.

??? ??? ???

??? ??? ???

??? ??? ???

??? ??? ???

Permission Puzzles

1. No permissions?

```
--- --- ---
```

2. Read, Write, Exec only for owner?

```
rwx --- ---
```

3. Execute for all?

```
--x --x --x
```

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

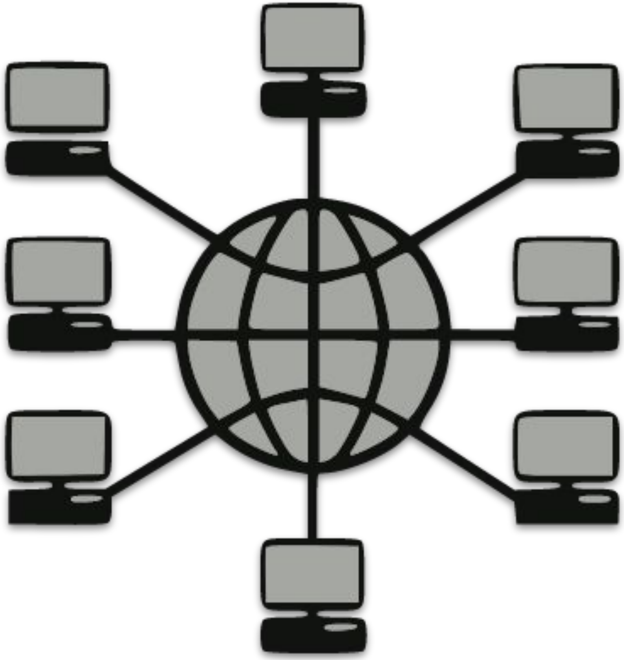
```
rwx r-- ---
```

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`**)

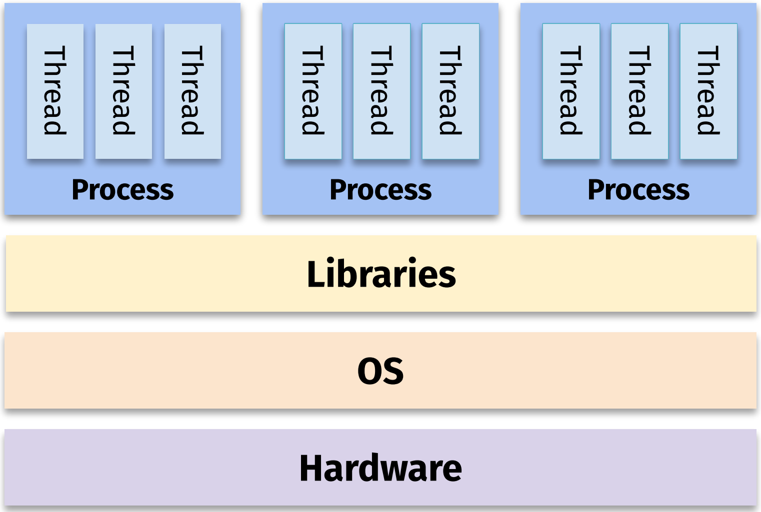


Questions?

Process Isolation

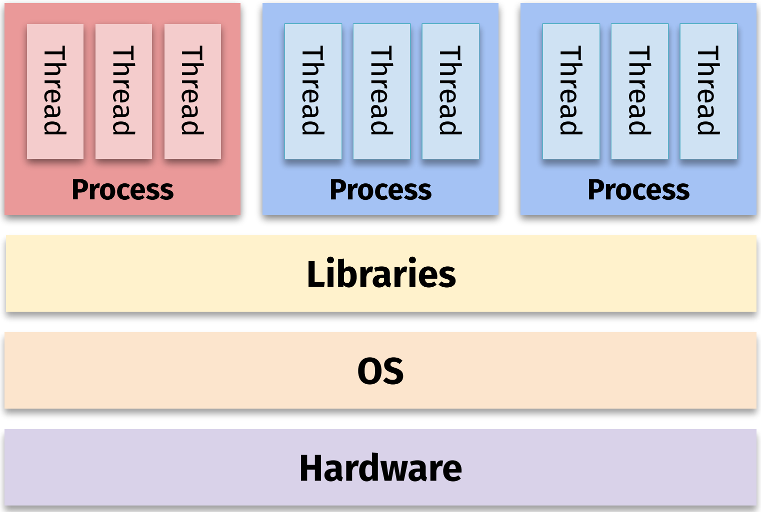
Process Isolation

Goal: ???



Process Isolation

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

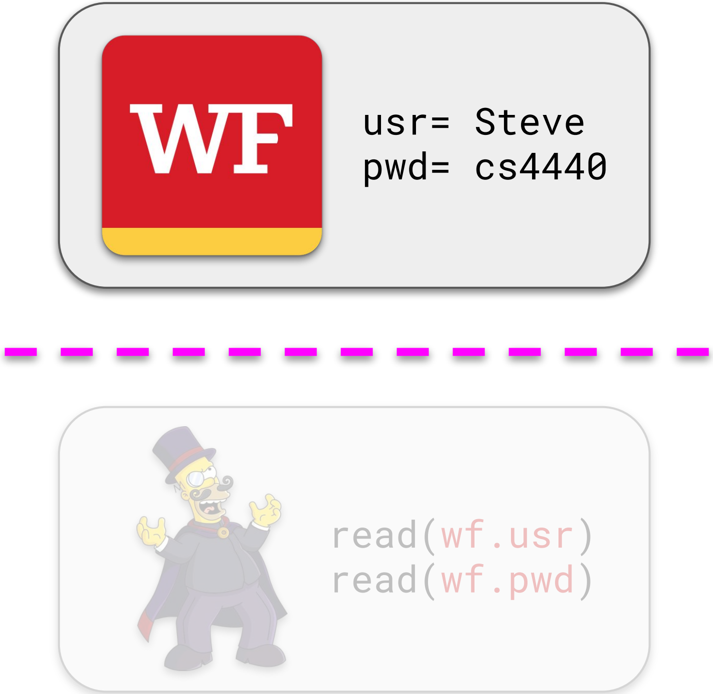
WF

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

```
read(wf.usr)
read(wf.pwd)
```

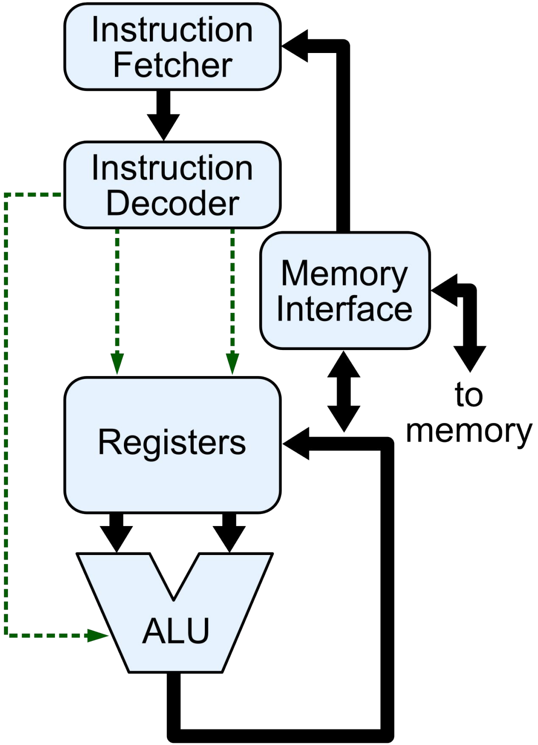
Process Isolation

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



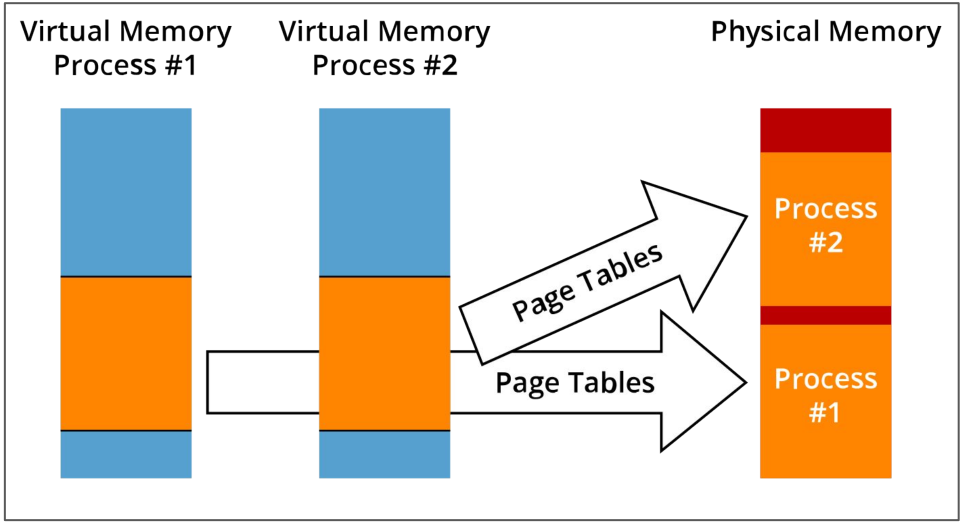
Isolating Process Memory

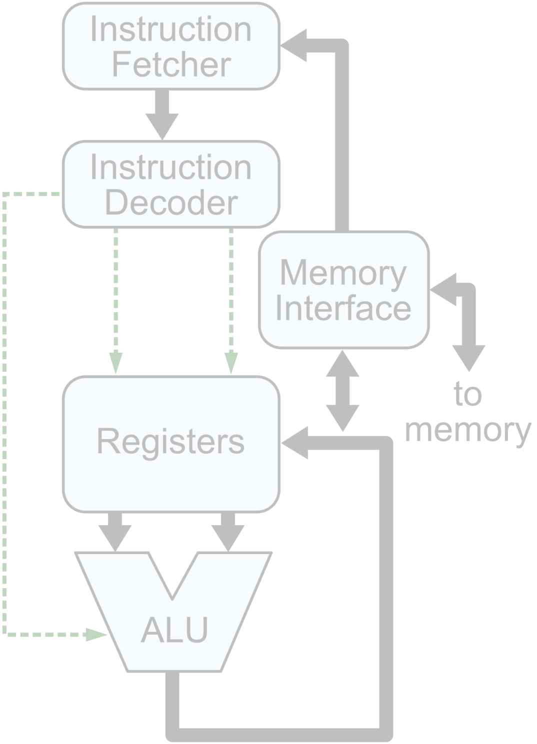
- **Memory Management Unit**
  - Translates virtual memory to physical memory



Isolating Process Memory

- **Memory Management Unit**
  - Translates virtual memory to physical memory
  - Enforce Process-1 **can't access** of **Process-2**'s memory!

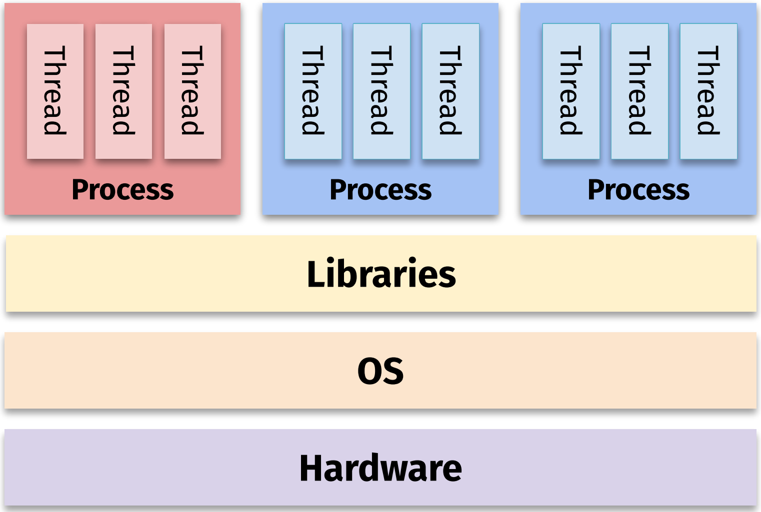




Process Isolation

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

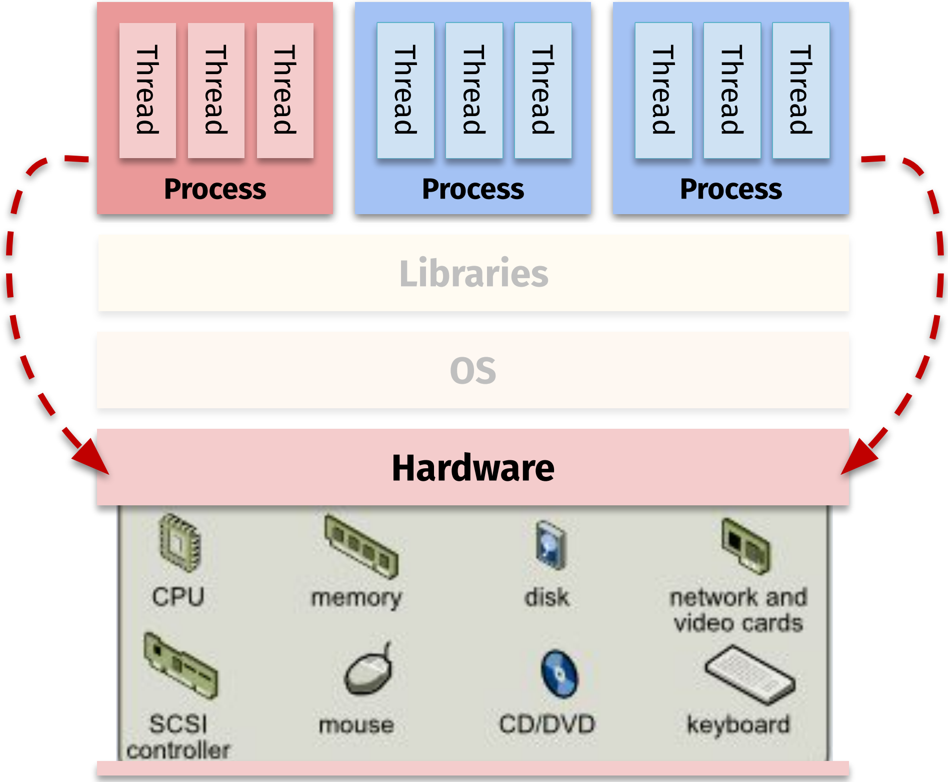
Caveat: ???



Process Isolation

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

Caveat: you must **trust** all potential **isolation bridges**

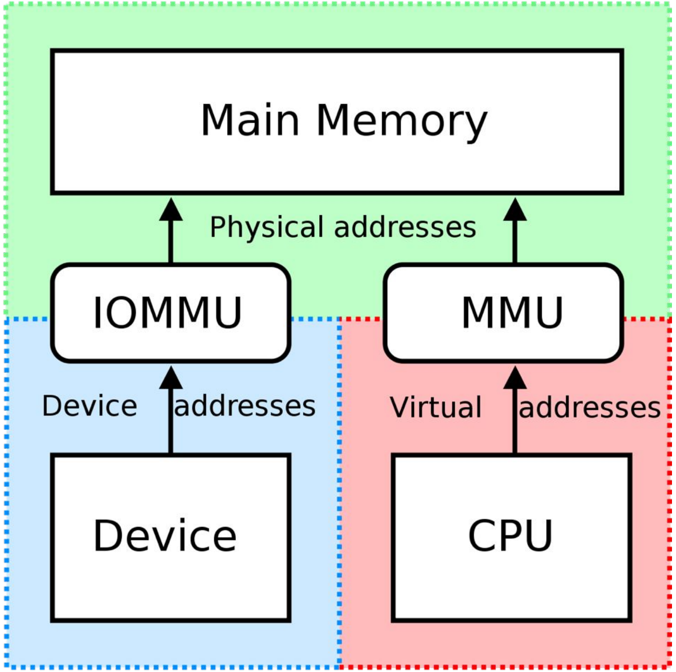


Memory-level Isolation

- What about **malicious peripherals**?
  - Assume plugged-in USB's are hostile!

Memory-level Isolation

- 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



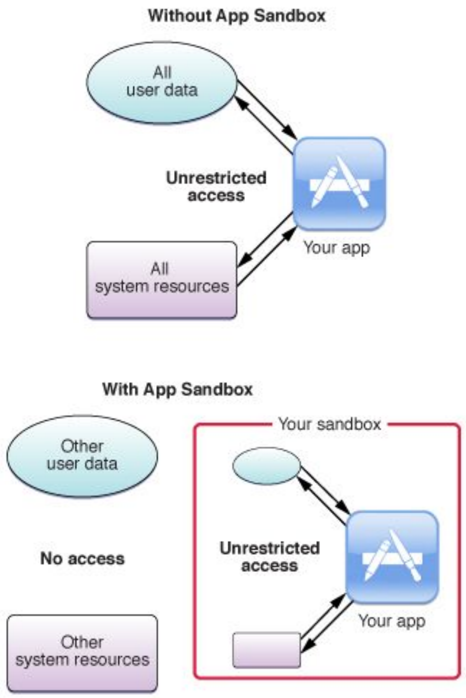
Resource-level Isolation

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

Resource-level Isolation

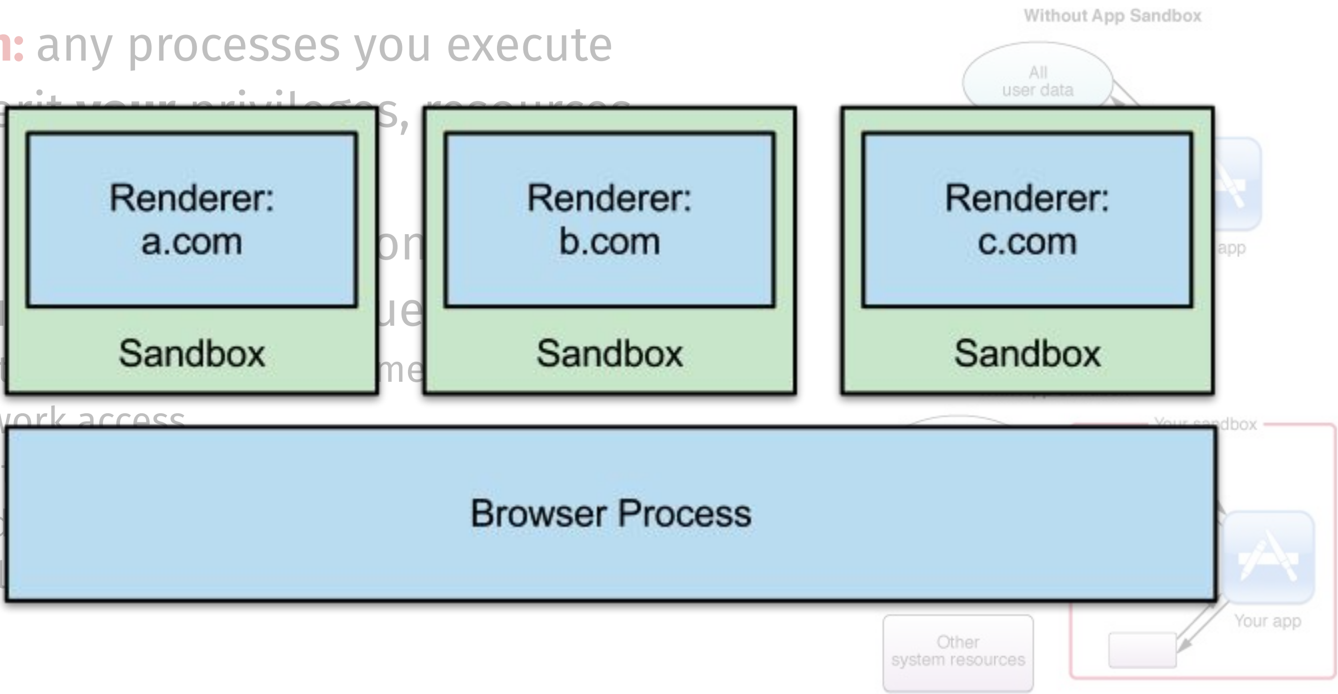
- **Problem:** any processes you execute will inherit **your** privileges, resources

- **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



Resource-level Isolation

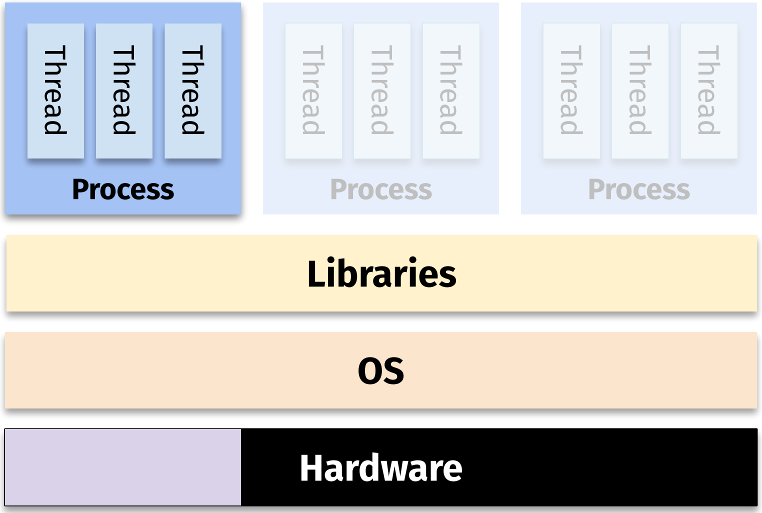
- **Problem:** any processes you execute will inhe... s,
- **Process** ... of resou...
  - Scrat...
  - Network access
  - Abili...
  - Read... disal...



Sandboxing

Goal: give processes the **least privileges**

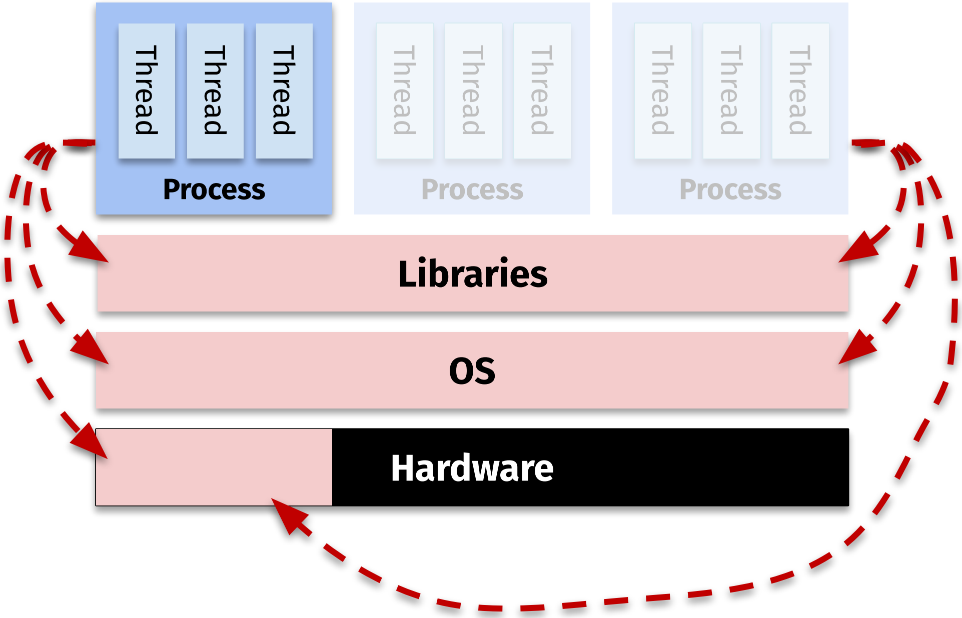
Caveat: ???



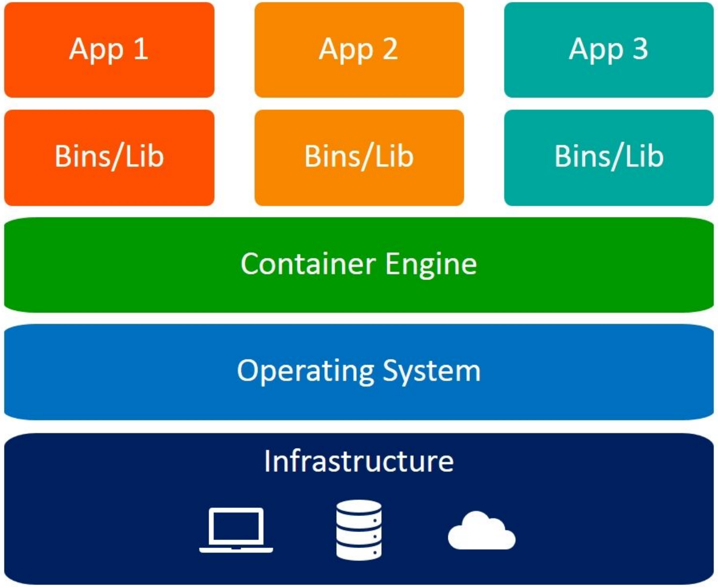
Sandboxing

Goal: give processes the **least privileges**

Caveat: the **trusted computing base** is still very large!



Containers

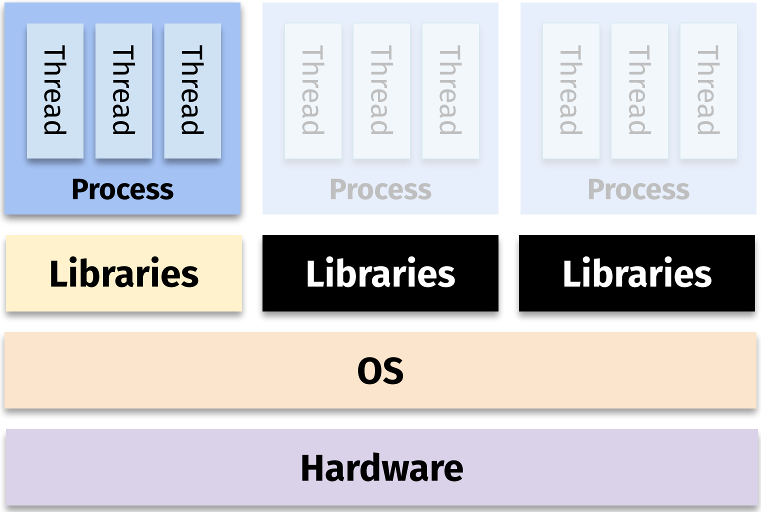


Containers

Containers

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

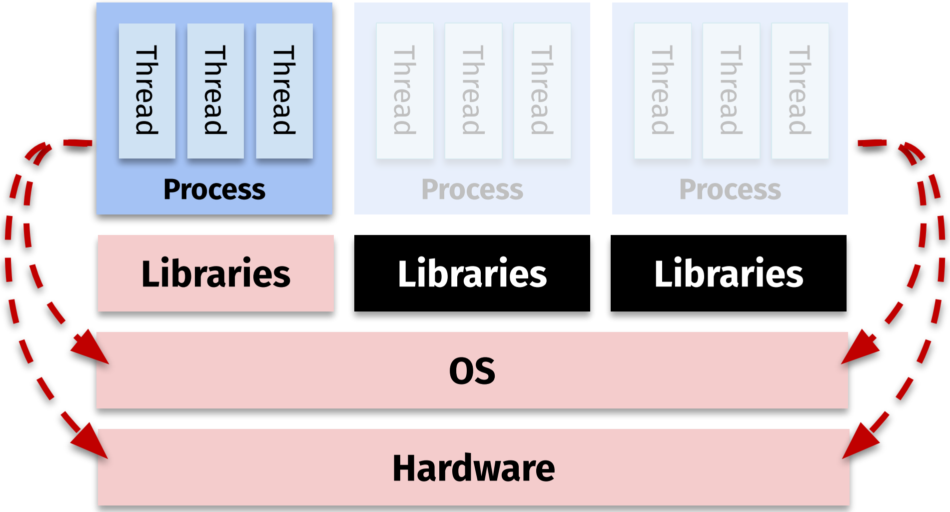
Caveat: ???



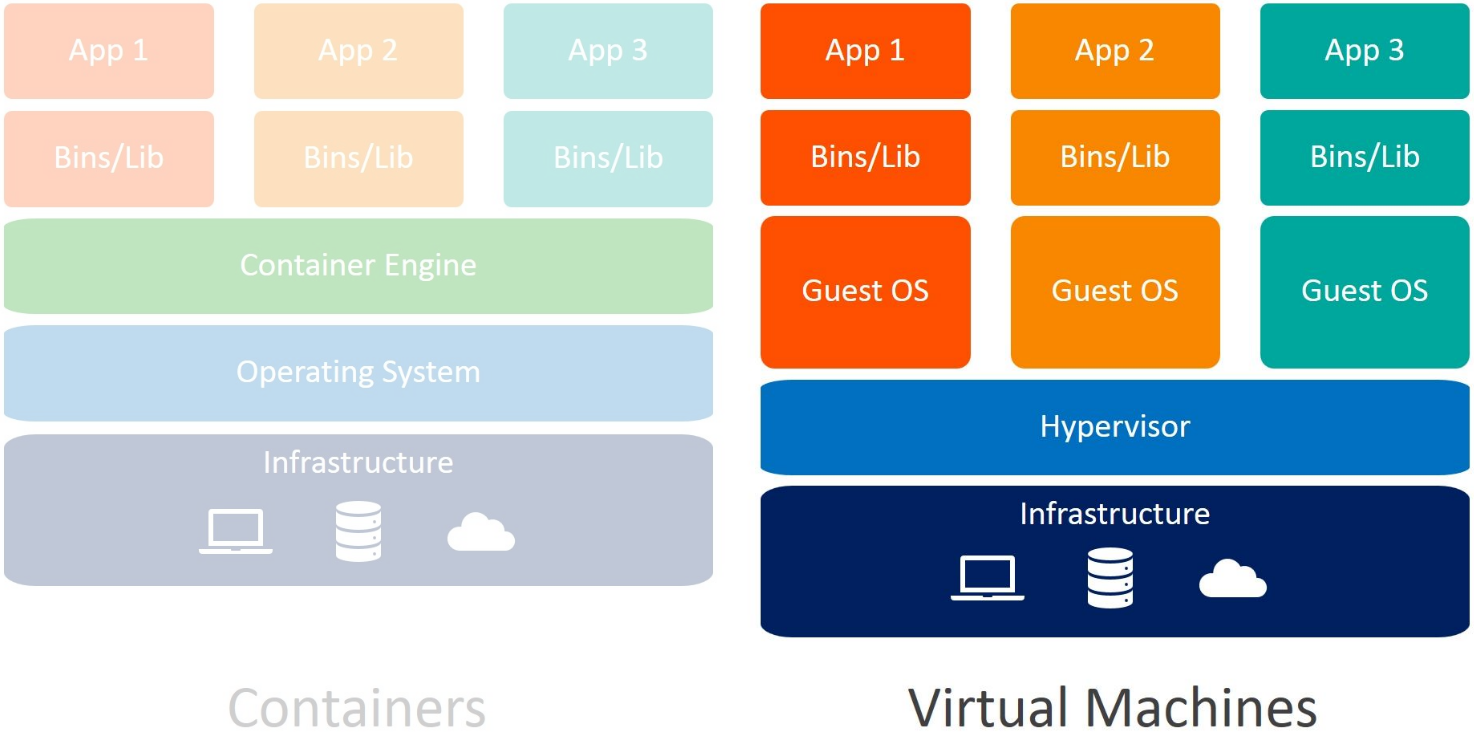
Containers

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

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



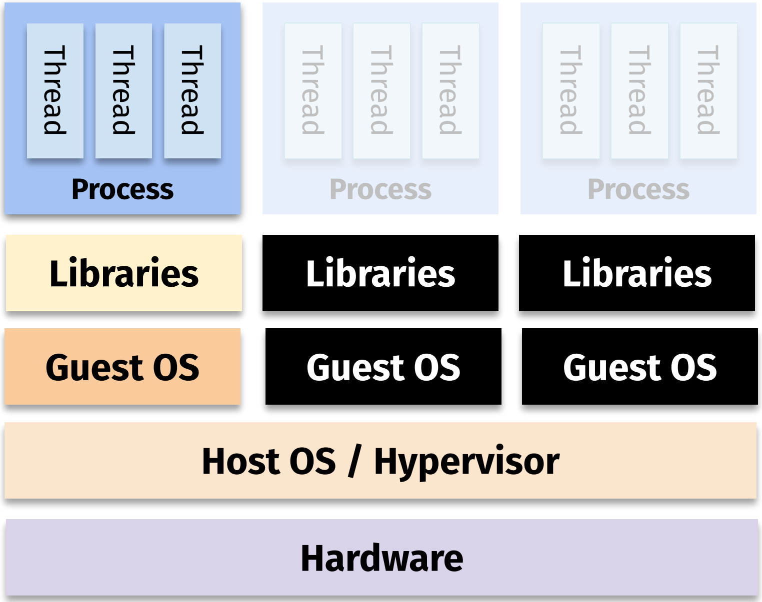
Virtual Machines



Virtual Machines

Goal: completely isolate the **OS**

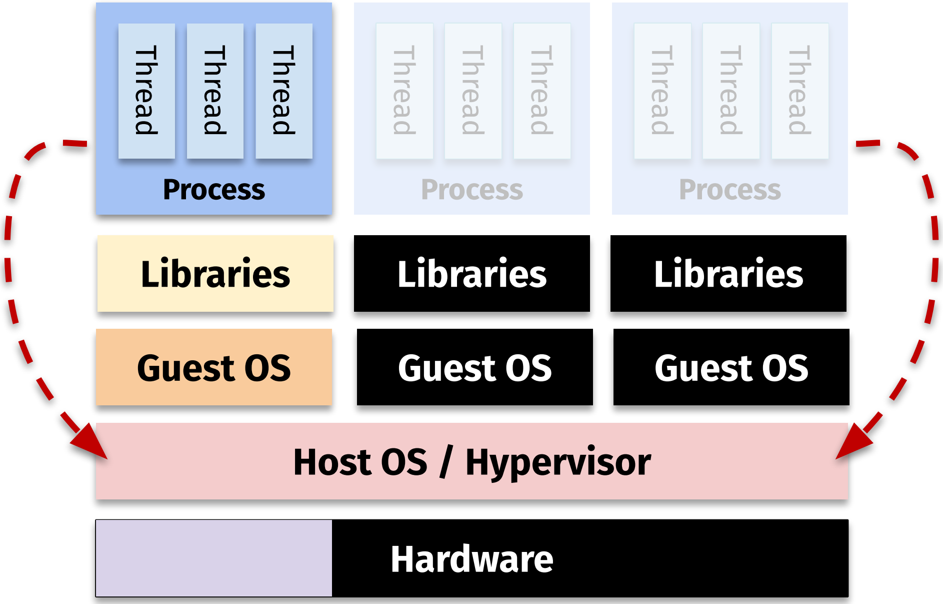
Caveat: ???



Virtual Machines

Goal: completely isolate the **OS**

Caveat: the trusted computing base now the **Hypervisor**

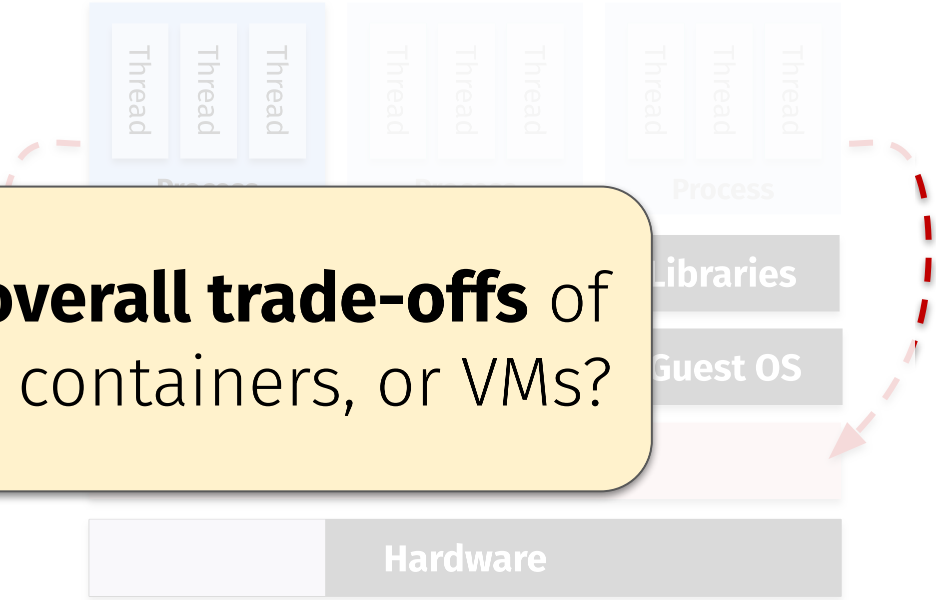


Virtual Machines

Goal: completely is…

Cave… computing base now the **Hypervisor**

So what are the **overall trade-offs** of using sandboxes, containers, or VMs?



Trade-offs of sandboxes/containers/VMs?

Sandboxes are the most secure but also the slowest.	0%
Containers balance speed/security but share the host's kernel.	0%
VMs are faster than containers but offer less isolation.	0%
Containers and VMs offer the same security and performance.	0%
None of the above	

Other Caveats

- **Sandboxes**, **containers**, and **hypervisors** are all software…

Other Caveats

- **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)

Virtual machine escape

3 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In computer security, **virtual machine escape** is the process of a program breaking out of the virtual machine on which it is running and interacting with the host operating system.[1] A virtual machine is a "completely isolated guest operating system installation within a normal host operating system".[2] In 2008, a vulnerability (CVE-2008-0923) in VMware discovered by Core Security Technologies made VM escape possible on VMware Workstation 6.0.2 and 5.5.4.[3][4] A fully working exploit labeled *Cloudburst* was developed by Immunity Inc. for Immunity CANVAS (commercial penetration testing tool).[5] Cloudburst was presented in Black Hat USA 2009.[6]

Previous known vulnerabilities [edit]

- CVE-2007-4993 Xen pygrub: Command injection in grub.conf file.
- CVE-2007-1744 Directory traversal vulnerability in shared folders feature for VMware
- CVE-2008-0923 Directory traversal vulnerability in shared folders feature for VMware
- CVE-2008-1943 Xen Para Virtualized Frame Buffer backend buffer overflow.
- CVE-2009-1244 Cloudburst: VM display function in VMware
- CVE-2011-1751 QEMU-KVM: PIIX4 emulation does not check if a device is hotpluggable before unplugging[7]
- CVE-2012-0217 The x86-64 kernel system-call functionality in Xen 4.1.2 and earlier
- CVE-2014-0983 Oracle VirtualBox 3D acceleration multiple memory corruption
- CVE-2015-3456 VENOM: buffer-overflow in QEMU's virtual floppy disk controller
- CVE-2015-7504 QEMU-KVM: Heap overflow in pcnet_receive function.[8]
- CVE-2015-7835 Xen Hypervisor: Uncontrolled creation of large page mappings by PV guests
- CVE-2016-6258 Xen Hypervisor: The PV pagetable code has fast-paths for making updates to pre-existing pagetable entries, to skip expensive re-validation in safe cases (e.g. clearing only Access/Dirty bits). The bits considered safe were too broad, and not actually safe.
- CVE-2016-7092 Xen Hypervisor: Disallow L3 recursive pagetable for 32-bit PV guests
- CVE-2017-5715, 2017-5753, 2017-5754: The Spectre and Meltdown hardware vulnerabilities, a cache side-channel attack on CPU level (Rogue Data Cache Load (RDCL)), allow a rogue process to read all memory of a computer, even outside the memory assigned to a virtual machine
- CVE-2017-0075 Hyper-V Remote Code Execution Vulnerability
- CVE-2017-0109 Hyper-V Remote Code Execution Vulnerability
- CVE-2017-4903 VMware ESXi, Workstation, Fusion: SVGA driver contains buffer overflow that may allow guests to execute code on hosts[9]

Questions?

Next time on CS 4440...

Security in Practice: Malware