CS165 – Computer Security

Attack surface and access control
Oct 26, 2021

Security Problems

- … could be anywhere in a program
  - Given the types of vulnerability we have seen, does that give us any insight into where we should look for vulnerabilities?

Security Problems

- … could be anywhere in a program
  - Given the types of vulnerability we have seen, does that give us any insight into where we should look for vulnerabilities?
    - Software flaw
    - Accessible to an adversary
    - Who can exploit the vulnerability

Security Problems

- … could be anywhere in a program
  - Given the types of vulnerability we have seen, does that give us any insight into where we should look for vulnerabilities?
    - Software flaw
    - Accessible to an adversary
    - Who can exploit the vulnerability
- Typically, we look for software flaws (e.g., control flow hijacking), but let us first consider "adversary accessibility"

Attack Surface

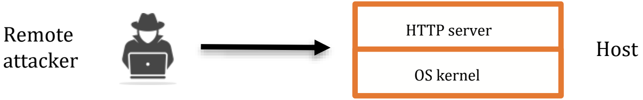
- After Microsoft faced several large-scale vulnerability exploits in the early 2000s
- They began to consider how to prevent such vulnerabilities

Attack Surface

- After Microsoft faced several large-scale vulnerability exploits in the early 2000s
- They began to consider how to prevent such vulnerabilities
- Michael Howard of Microsoft defined the term "attack surface"
  - A program's attack surface consists of the entry points that are accessible to an adversary
- Entry point: where **untrusted inputs** come in
- Example attack surfaces?

Example attack surface?

- Relative to a threat model



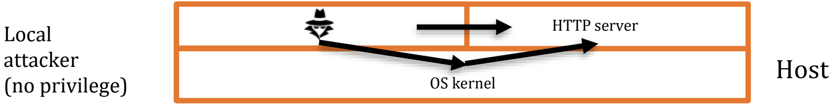
Remote attacker

HTTP server

OS kernel

Host

Goal: execute code remotely



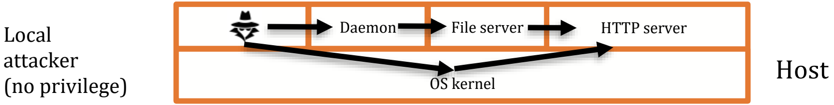
Local attacker (no privilege)

HTTP server

OS kernel

Host

Goal: change the behavior of HTTP server



Local attacker (no privilege)

Daemon

File server

HTTP server

OS kernel

Host

Goal: change the behavior of HTTP server

Anything a target depends on (directly or indirectly) should be included

Attack surface of autonomous vehicle

GSM baseband (3G/4G/5G)	WiFi Internet	V2V networking	Third-party cloud (cmd distros)	Firmware update	GPS	Far
Bluetooth	Wireless key entry	Charging station	Camera	Lidar		Near

Outside

Inside

OBD interface	Infotainment (Android)	RTOS (Linux)	Industrial control system	USB, SDCARD interface		User-facing
Transmission Steering Brake subsystem	Power subsystem	CAN bus	Electrical subsystem	Anti-theft Tire pressure monitoring		Internal components

Relative Attack Surface Metric (system)

- Howard proposed the notion of a relative attack surface quotient (RASQ) metric
  - The idea is that we can use the metric to compare systems -- which has larger relative attack surface
- The metric lists a set of entry points that you should be concerned about minimizing as a system distributor

Relative Attack Surface Metric (system)

- Open (TCP/UDP) sockets - descriptors
- Open RPC endpoints - descriptors
- Open named pipes - descriptors
- Services - daemons
- Services running by default - daemons
- Services running as SYSTEM (or root) - daemons
- Active Web handlers – web server components
- Active ISAPI filters – web server add-ins
- Dynamic web pages – files
- Executable vdirs – directories for scripts

Relative Attack Surface Metric (system)

- Enabled accounts – accounts
- Enabled accounts in admin group – accounts
- Null sessions to pipes and shares – anonymous connections allowed
- Guest account enabled – accounts (special)
- Weak ACLs in FS – files allowing “full control” to everybody
  - “Full control” is the moral equivalent of UNIX rwxrwxrwx permissions
- Weak ACLs in Registry – registry keys that allow “full control” to everybody
- Weak ACLS on shares – Directories that can be shared by remote users that allow “full control” to everybody
- VBScript, JavaScript, Active X enabled – applications enabled to execute Visual Basic Script, JavaScript or Active X controls

Relative Attack Surface Metric (system)

- Essentially, you would count the number of unsafe instances
  - Also combined with weights per item, but numeric weights that are meaningful are often hard to predict effectively
- Windows systems saw a gradual reduction in attack surface metric values in the 2000s
  - But, attacks kept coming, exploiting new vulnerabilities
- Can we say something about programs individually with respect to their attack surfaces?

Program Attack Surface

- Can we say something about programs **individually** with respect to their attack surfaces?
- What do we need to identify to determine the adversary-accessible entry points of a program?

Program Attack Surface

- Can we say something about programs **individually** with respect to their attack surfaces?
- What do we need to identify to determine the adversary-accessible entry points of a program?
  - Identify the relevant subset of system resources that can be used in an attack (are or could be controlled by an adversary)
  - Identify when such resources may be used by the program (**program entry points**)
- Is it possible to compute such information?

Program Entry Points

- What’s a program entry point?
- Programs obtain information from external sources (e.g., files and network sockets), and the program statements that access such external sources are entry points
  - What’s an example of an entry point?

Program Entry Points

- What’s a program entry point?
- Programs obtain information from external sources (e.g., files and network sockets), and the program statements that access such external sources are entry points
  - What’s an example of an entry point?
    - System calls provide the sources for gaining most external information
    - But, for attack surfaces, we focus on the statements that a program makes to the individual library/system calls

System Calls as Attack Surface

- Why should we use system calls for the attack surface?

System Calls as Attack Surface

- Why should we use system calls for the attack surface?
  - A Program has to use library calls to access external resources
  - Wrappers in libraries, e.g., fopen() vs. open()
  - Program statements that invoke each call
    - Only a subset of these may be adversary accessible
- E.g., consider the “open” system call
  - May be invoked via “open” or “fopen” library call
  - fopen(input_pathname, …) vs. fopen(“/bin/sh”)

System Calls as Attack Surface

- Why should we use system calls for the attack surface?
  - A Program has to use library calls to access external resources
  - Wrappers in libraries, e.g., fopen() vs. open()
  - Program statements that invoke each call
    - Only a subset of these may be adversary accessible
- E.g., consider the “open” system call
  - May be invoked via “open” or “fopen” library call
  - fopen(input_pathname, …) vs. fopen(“/bin/sh”)
- E.g., consider the “read” system call
- How many system calls access adversary-controlled data?

System Calls as Attack Surface

- Which system calls should constitute a program’s attack surface?

System Calls as Attack Surface

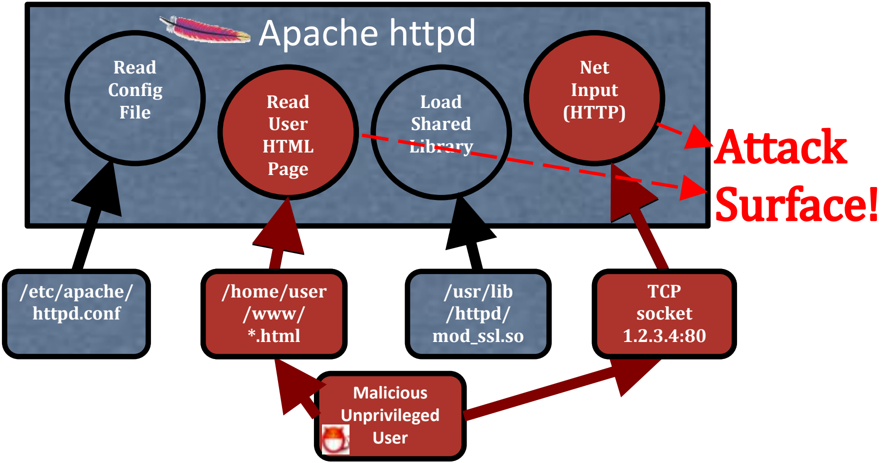
- Which system calls should constitute a program’s attack surface?
  - All of them
    - At some point, any call may access adversary-controlled data
    - So test them all

System Calls as Attack Surface

- Which system calls should constitute a program’s attack surface?
  - All of them
    - At some point, any call may access adversary-controlled data
    - So test them all
  - Only ones that actually may access adversary-controlled resources
    - Only need to test a subset of such each program’s entry points to evaluate the attack surface
    - How do you determine which may access adversary-controlled resources?

Program Attack Surface

- Program system calls accessible to an adversary



System TCB Attack Surface

- Only 13.8% of total entry points for Linux system services were accessible to adversaries at all
  - Only 3.8% for read/write operations
  - Listing all entry points as attack surface would be a huge over-approximation

Total Entrypoints	Accessible to Adversaries	Potentially Vulnerable (overt permissions)	Previously Known Bugs
2138	295	81	35

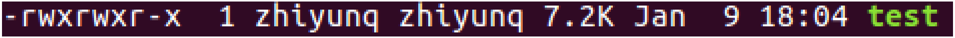
  - Found via runtime testing with Linux package test suites – lower bound

An example - E-voting

- Who are the principals?
  - Voters, Admins, Talliers, Others
- Who are adversaries?
- Which commands may be threatened (attack surface)?
  - Start Program (by admins)
  - Submit a vote (by voter)
  - Count votes (by tallier)

Access control

- Give “users” permissions to access “resources”

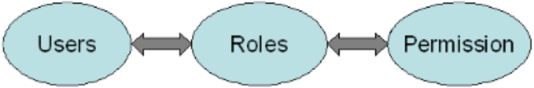


```
-rwxrwxr-x  1 zhiyunq zhiyunq 7.2K Jan  9 18:04 test
```

- Attack surface computed with respect to a **threat model**
  - Local attacker (unprivileged)
  - Local attacker (system privilege)
  - Local attack (with root)
  - Remote attacker
- … **and an access control policy**
  - What can a (unprivileged/system/root) user do on a system? What files can the user write? What services can they contact?

- Why do we need access control?
  - Hint: think of it from the attack surface point of view
  - (Android story)

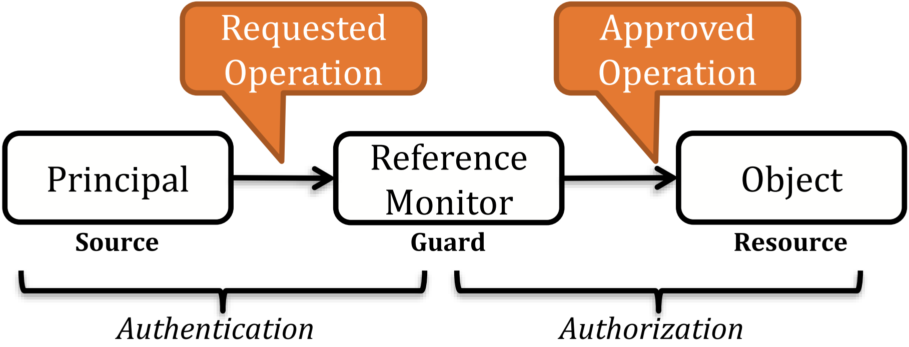
- Challenge: how do we know if our policy is good enough? The right policy? The right implementation?



Users

Roles

Permission


Requested Operation
Approved Operation
Principal
Reference Monitor
Object
Source
Guard
Resource
Authentication
Authorization

Policies and Mechanisms

- Policy says what is, and is not, allowed
  - This defines "security" for the site/system/*etc*.
- Mechanisms enforce policies
- Composition of policies
  - If policies conflict, discrepancies may create security vulnerabilities

Policies and Mechanisms (example)

- Policy says what is, and is not, allowed
  - A file should be readable by only the root user

- Mechanisms enforce policies
  - Kernel checks through file open() syscall.

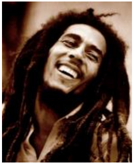
Agenda

- Access Control Matrix
  - Overview
  - Access Control Matrix Model
  - Protection State Transitions
    - Commands
    - Conditional Commands
- Foundational Results

NEXT

Alice and Bob

- Standard names for “agents” in a security or crypto scenario
- Also known as “A” and “B”



An Access Control Scenario

- Alice:

```
1. New Secret foo
```

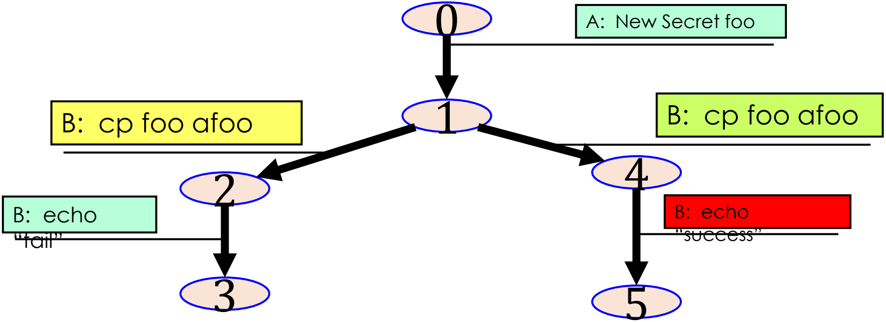
- Bob:

```
2. If (cp foo afoo)
3.    then echo "success"
4. else echo "fail"
```

Intent:

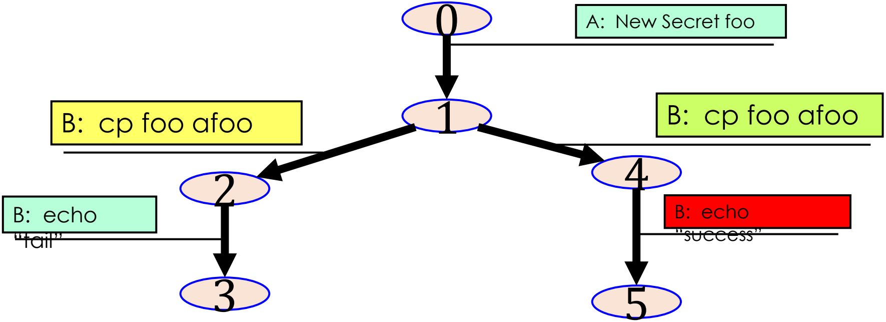
- Bob’s **cp** is attempting to violate Alice’s expected access policy
- If **cp** succeeds then the principle of **confidentiality** is not satisfied

Characterizing the Violation



Basic Abstraction: States and Transitions

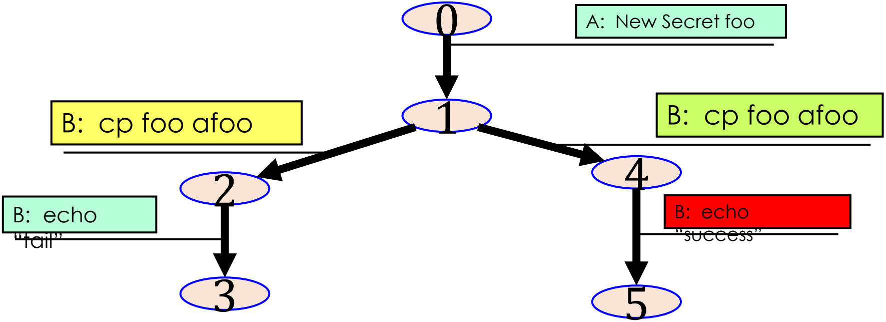
Characterizing the Violation



Basic Abstraction: States and Transitions

Q: What are the States?

Characterizing the Violation



Basic Abstraction: States and Transitions

Q: What are the States?

Q: What determines if we reach State 2 or 4 from State 1?

States

- State of a system
  - A collection of the current values of all memory locations, storages, registers, etc.
  - A subset of this collection that deals with protection is the protection state of the system

Secure and non-Secure States

Characterize states in a system as “Secure” and “non-Secure”

A system is **Secure** if every transition maps Secure states to Secure states

Consequence: In the scenario, security is compromised if Alice’s “New secret foo” yields a state in which Bob can access foo.

Protection States

- An abstraction that focuses on security properties
  - ✓ Primarily interested in characterizing Safe states
  - ✓ Goal is to prove that all operations in the system preserve "security" of the protection state
  - ✓ **Access Control Matrix** is our first Protection State model

Questions