Agenda – Security (Chapter 9)

Introduction
Example: Buffer overflow exploits
Types of threats
Protection Domains
Cryptography
- Secret-key
- Public-key
- One-way functions
- Digital signatures
- Certificate Authorities

Authentication (passwords)
End-to-end encryption tools
Malware

Additional Reading:

Hacks, Leaks, and Revelations -- Micah Lee
Hackers – Steven Levy
Cyberpunk – Hafner and Markoff
The Cuckoo's Egg – Clifford Stoll
The Jargon File
Data and Goliath – Bruce Schneier

Preface

Early (Unix systems) security: challenge is to keep users from reading or tampering with others' files

- Protection flags were enough

Today's challenges are complexity/internet driven:

- Single-user devices can still be stolen or broken into
- Identities can be spoofed with a stolen device or password
- Ease-dropping on network messages
- What else?

Easiest way to build a secure system is to keep it simple. Today's systems are far from simple!

Hacker Terminology

Within early cs communities, a **hacker** is a term of respect for someone who is good with computers and programming

A **white hat** is someone who breaks into systems in order to expose problems so they can be fixed.

A **black hat** is someone who breaks into for financial gain or to attack individuals or organizations.

- **doxing** refers to posting personal information such as home addresses and phone numbers for the purposes of bullying and intimidation
- A **zero-day** is a vulnerability or security hole in a computer system unknown to its owners, developers or anyone capable of mitigating it.
- Wannabe black hats and hackers are called **script-kiddies**

Organizations perform **red team**/blue **team** exercises to try to find problems before black hat hackers do. The **red team** attacks. The blue **team** defends.

Terminology

When a bug is a security bug, we call it a **vulnerability**

Inputs that trigger the bug to achieve a desired purpose is called an **exploit**.

Demo and Example: Buffer Overflow exploit

- 50% of security incidents reported at CERT (see cert.org) are due to buffer overflow attacks
- Modern compilers and languages include features for boundary checking making these exploits harder

Real World Exploits: Buffer Overflow

Recall: **Buffer** is another term for array.

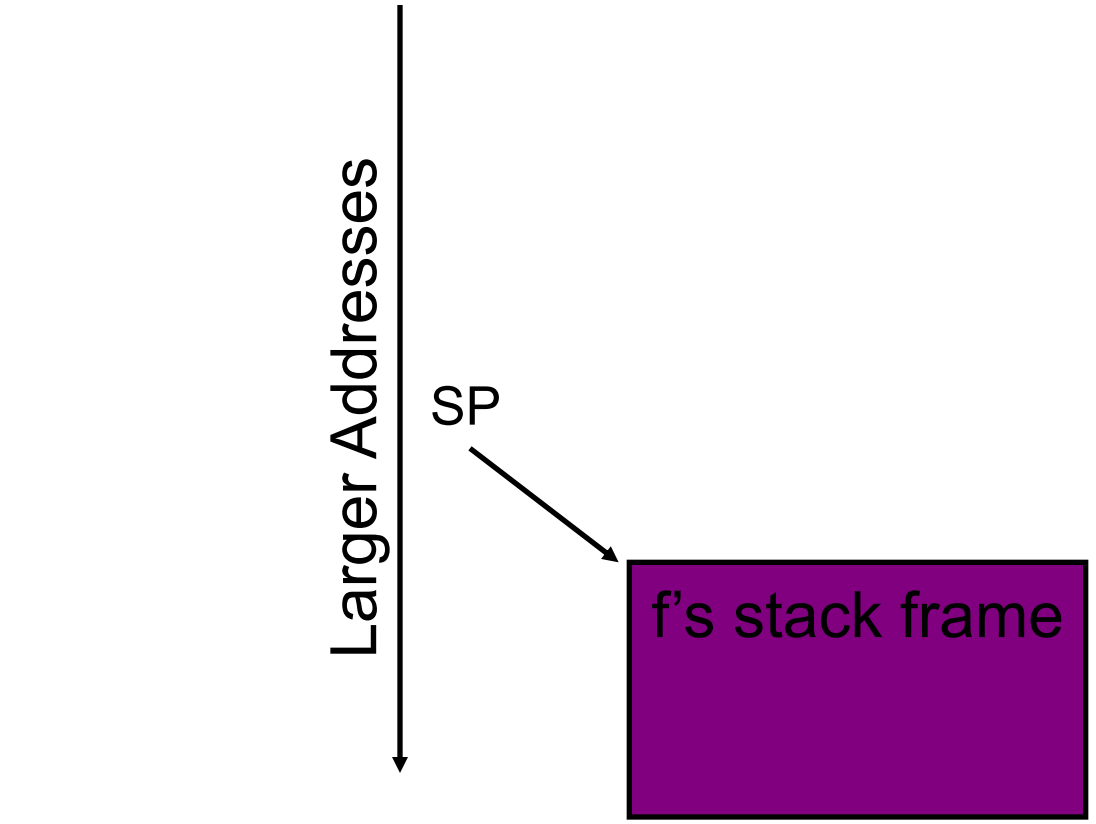
Buffer overflow occurs when we try to access memory outside of the bounds of an array.

Usually this results in a segmentation fault; however, buffer overflows can be exploited by hackers to make a program execute in a manner different from what was intended. This is called a **buffer overflow exploit.**

How a buffer exploit works

```
f() {
  g();
}

g() {
  int x;
  // more local
  // variables
  ...
}
```

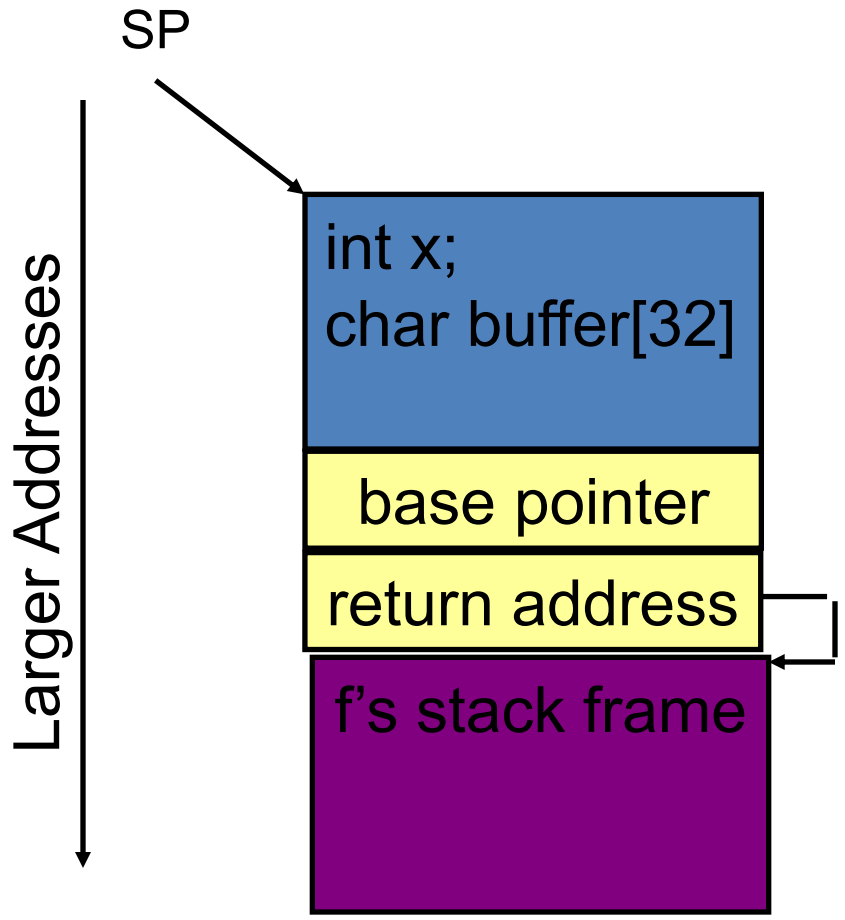


Before calling g

How a buffer exploit works

```
f() {
   g();
}

g() {
   int x;
   char buffer[32];
   scanf(" %s", buffer);
}
```

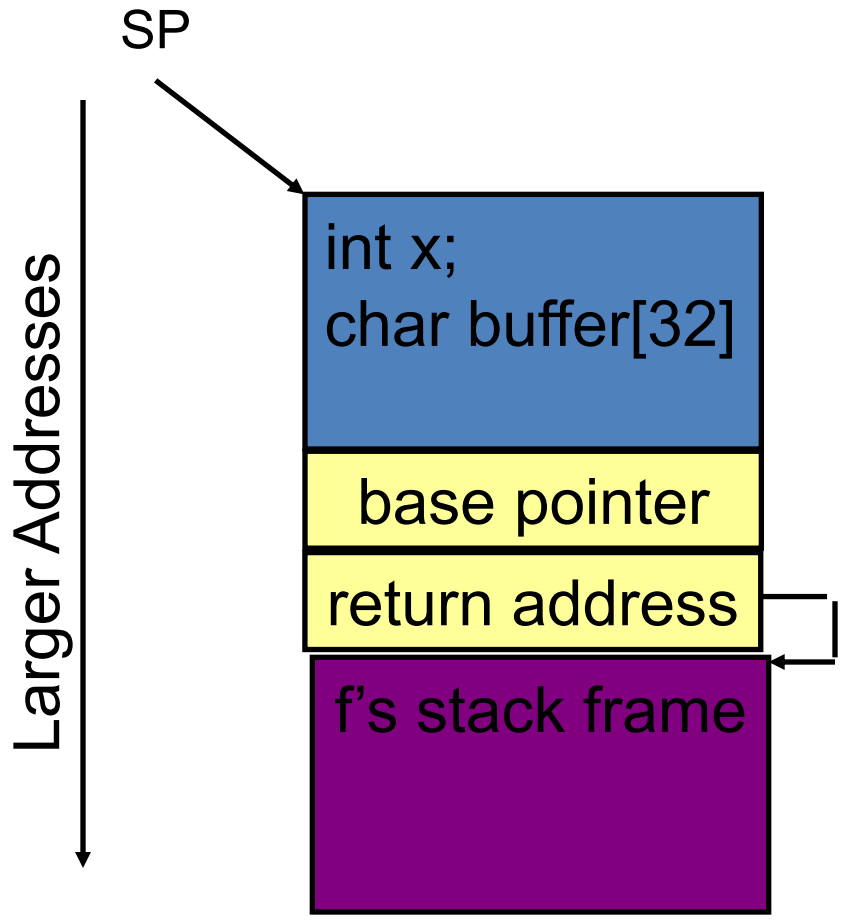


After calling g

How a buffer exploit works

```
f() {
   g();
}

g() {
   int x;
   char buffer[32];
   scanf(" %s", buffer);
}
```

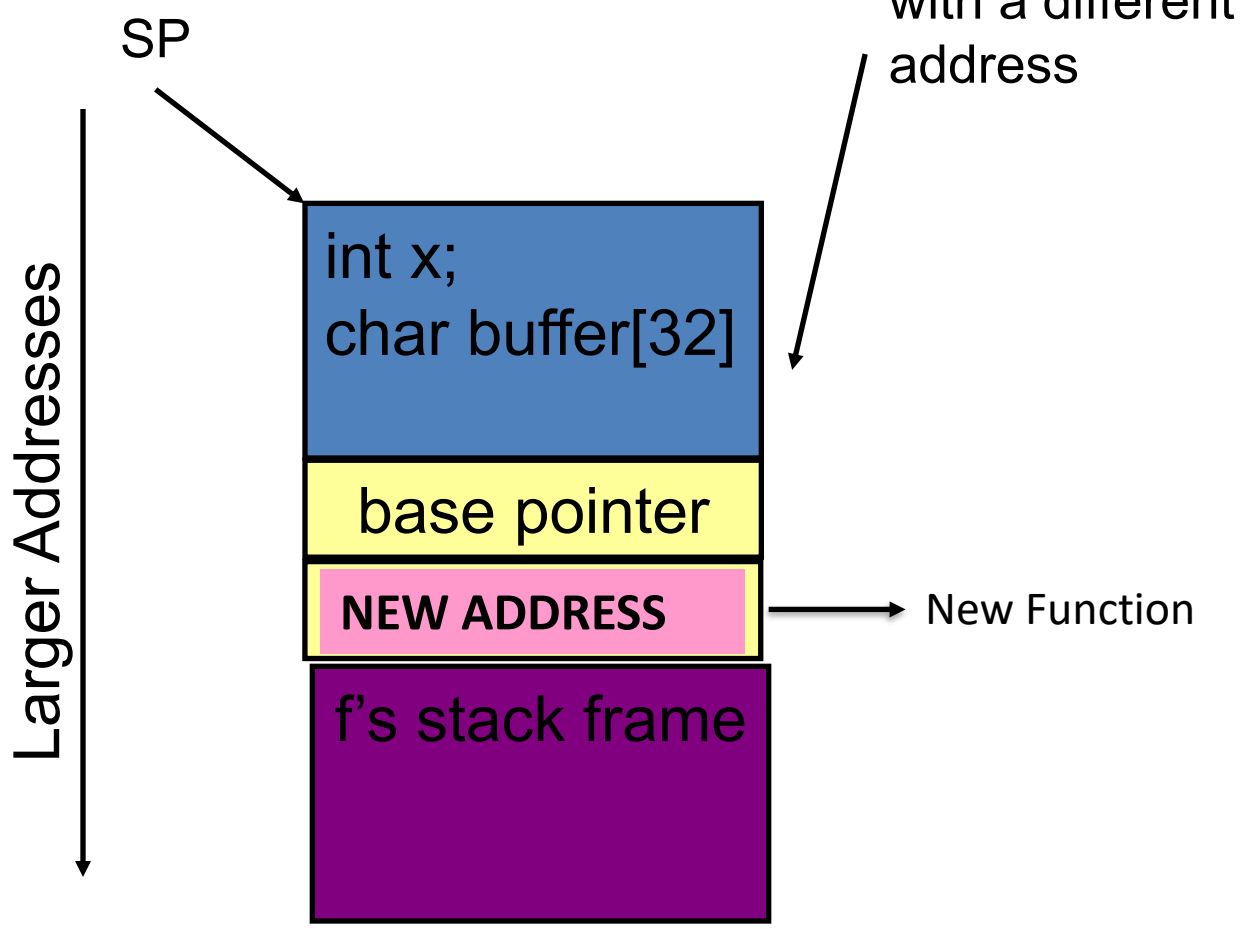


After calling g

How a buffer exploit works

```
f() {
   g();
}

g() {
   int x;
   char buffer[32];
   scanf(" %s", buffer);
}
```



After calling g

Demo: Exploiting buffer vulnerabilities

```
void endGame(void){
  printf("You win!\n");
  exit(0);
}

int main(){
  int guess, secret, len, x=3;
  char buf[12];
  printf("Enter secret number:\n");
  scanf("%s", buf);
  guess = atoi(buf);
  secret=getSecretCode();
  if (guess == secret)
    printf("You got it right!\n");
  else{
    printf("You are so wrong!\n");
    return 1;
  }
  printf("Enter the secret string to win:\n");
  scanf("%s", buf);
  guess = calculateValue(buf, strlen(buf));
  if (guess != secret){
    printf("You lose!\n");
    return 2;
  }
  endGame();
  return 0;
}
```

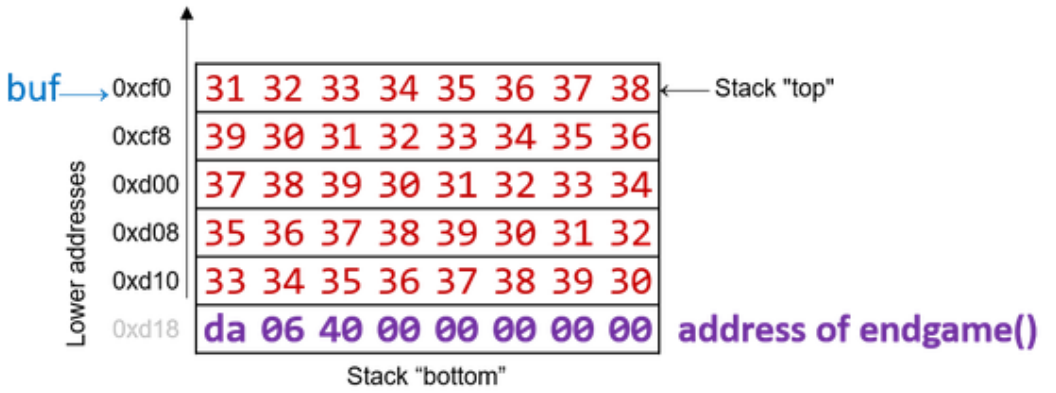
Where are the potential lines where we could overrun a buffer?

Example from *Dive Into Systems*. Try it yourself!

Demo: Exploiting buffer vulnerabilities

```
main:
  <+0>: push   %rbp
  <+1>: mov    %rsp,%rbp
  <+4>: sub    $0x20,%rsp
  <+8>: movl   $0x3,-0x4(%rbp)
  <+15>:mov    $0x400873,%edi
  <+20>:callq  0x400500<printf@plt>
  <+25>:lea    -0x20(%rbp),%rax
  <+29>:mov    %rax,%rsi
  <+32>:movl   $0x400888,%edi
  <+37>:mov    $0x0,%eax
  <+42>:callq  0x400540<scanf@plt>
  <+47>:lea    -0x20(%rbp),%rax
  <+51>:mov    %rax,%rdi
  <+54>:callq  0x400530<atoi@plt>
```

Idea: Overrun the buffer given to *scanf* to overwrite the return address of *main* to go *endgame*.



call stack

Immediately after call to `scanf()`

Registers	
%eax	0x0
%edi	0x400888
%rsi	0xcf0
%rsp	0xcf0
%rbp	0xd10

Memory	
0x400873	"Enter secret number"
0x400888	"%s"

Protecting against buffer overflow: compiler features

Stack randomization randomizes the location of the call stack so hackers can rely on the same memory addresses being re-used when the program is run.

Stack corruption detection uses flags to detect mis-use of the stack

Limited executable regions restricts executable code to only be in certain regions of memory

Defense: Good programming

In C/C++, use length specifiers whenever possible

Table 1. C Functions with Length Specifiers

Instead of:	Use:
`gets(buf)`	`fgets(buf, 12, stdin)`
`scanf("%s", buf)`	`scanf("%12s", buf)`
`strcpy(buf2, buf)`	`strncpy(buf2, buf, 12)`
`strcat(buf2, buf)`	`strncat(buf2, buf, 12)`
`sprintf(buf, "%d", num)`	`snprintf(buf, 12, "%d", num)`

Defense: Good programming

Use languages with built-in protections against memory errors

- JAVA
- Rust
- Link C code against safe version of libc
  - May degrade performance unacceptably

Types of threats

Confidentiality

- Goal: avoid exposure of data
- Examples of potential harms:

Integrity

- Goal: avoid tampering of data
- Examples of potential harms:

Availability

- Goal: ensure services are available
- Examples of potential harms:

Exercise: Think of applications that

requires integrity, availability, but not confidentiality

requires integrity, confidentiality, but not necessarily 24/7 availability?

requires integrity, availability, and confidentiality

Security in operating systems

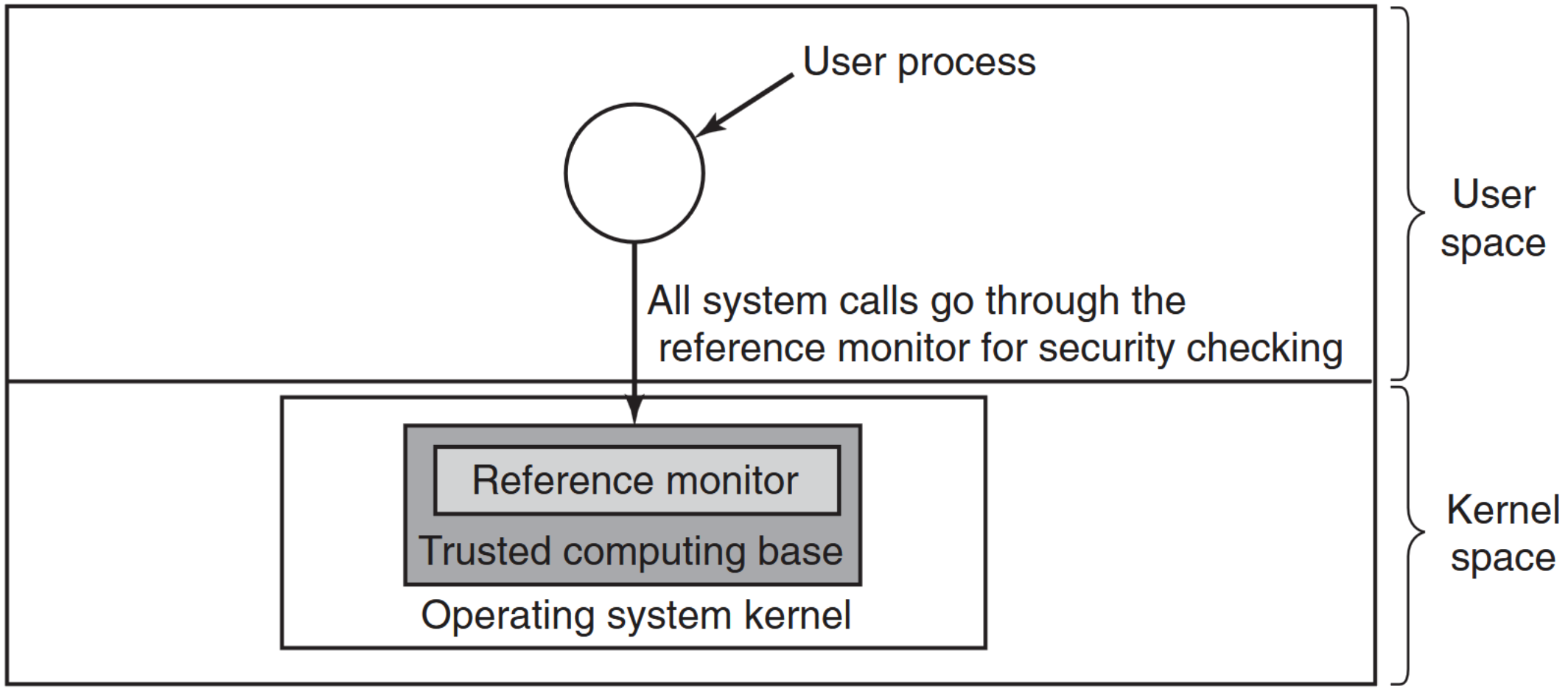
Cryptography involves shuffling a message or file so it can only be read by someone with the key

- Can ensure confidentiality and integrity
- Example application: authentication

Software hardening involves implementing protection mechanisms to programs so that its hard for attackers to make them misbehave

A **trusted system** has formally stated security requirements that the system meets. A trusted systems contains a minimal **Trusted Computing Base (TCB)** that enforces security rules.

Security in Operating Systems



Example: the reference monitor accepts all system calls involving security (such as opening files) and determines whether they are allowed or not

OS Security: Protecting Access

A **protection domain** is a set of (object, right) pairs that defines

- what is to be protected
- who is allowed to do what

An object might be a file or device

A right might be permissions like read, write, execute

A domain might be a user (aka **subjects**, **principals**), group, or mode

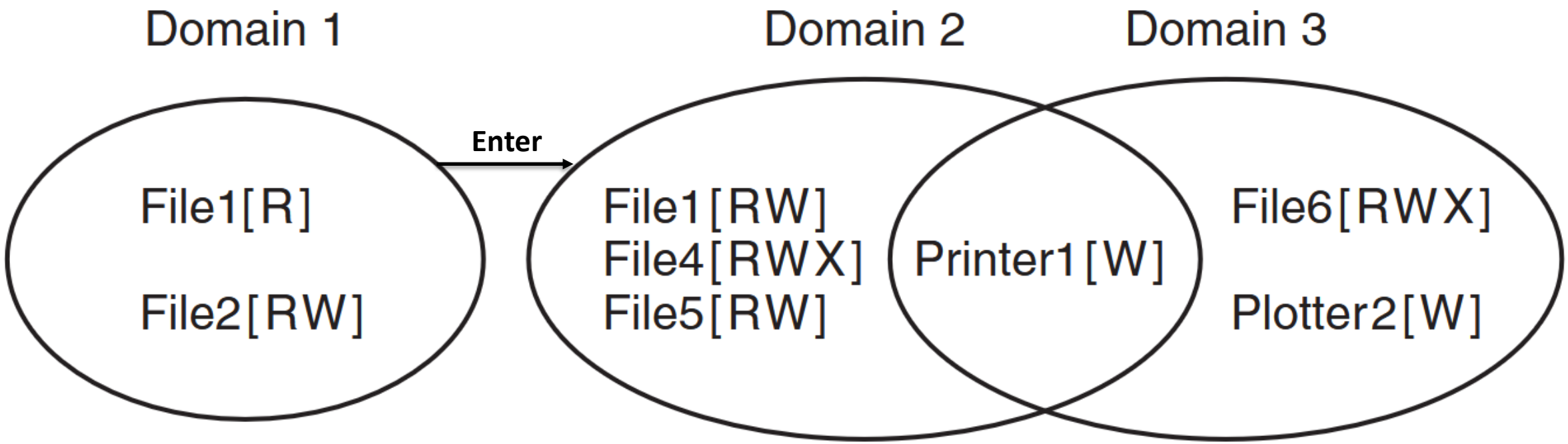
On UNIX, a user's domain is determined by their UID and GID. This determines what resources they have access to

On UNIX, running in kernel mode also affects what resources are available to the process

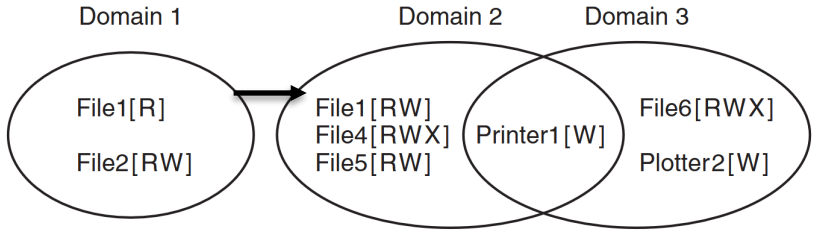
Rule of Thumb: **Principle of Least Authority (POLA)**

Each domain should have the least amount of privileges necessary to do its work. Why?

Example: Protection Domain



Example: Protection Domain



Represent the protection domains from the previous slide using a **protection matrix**

	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
Domain 1											
2											
3											

Protection Domain Implementation

Protection domain matrices are large and sparse: better to store only non-empty values

Two Approaches

- Access Control Lists (ACL)
- Capability List (C-List)

Protection Domain: Access Control Lists

Idea: Each object stores the domains that can reference it

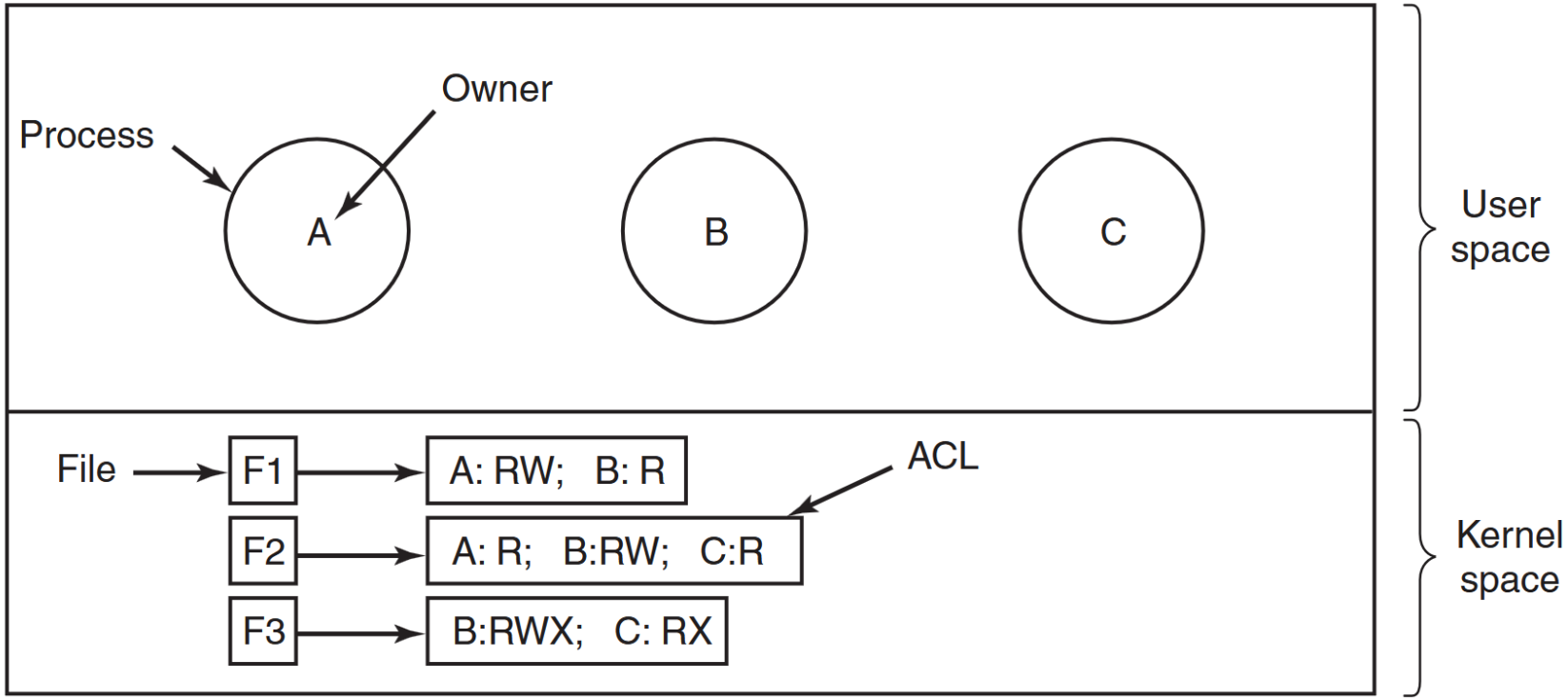


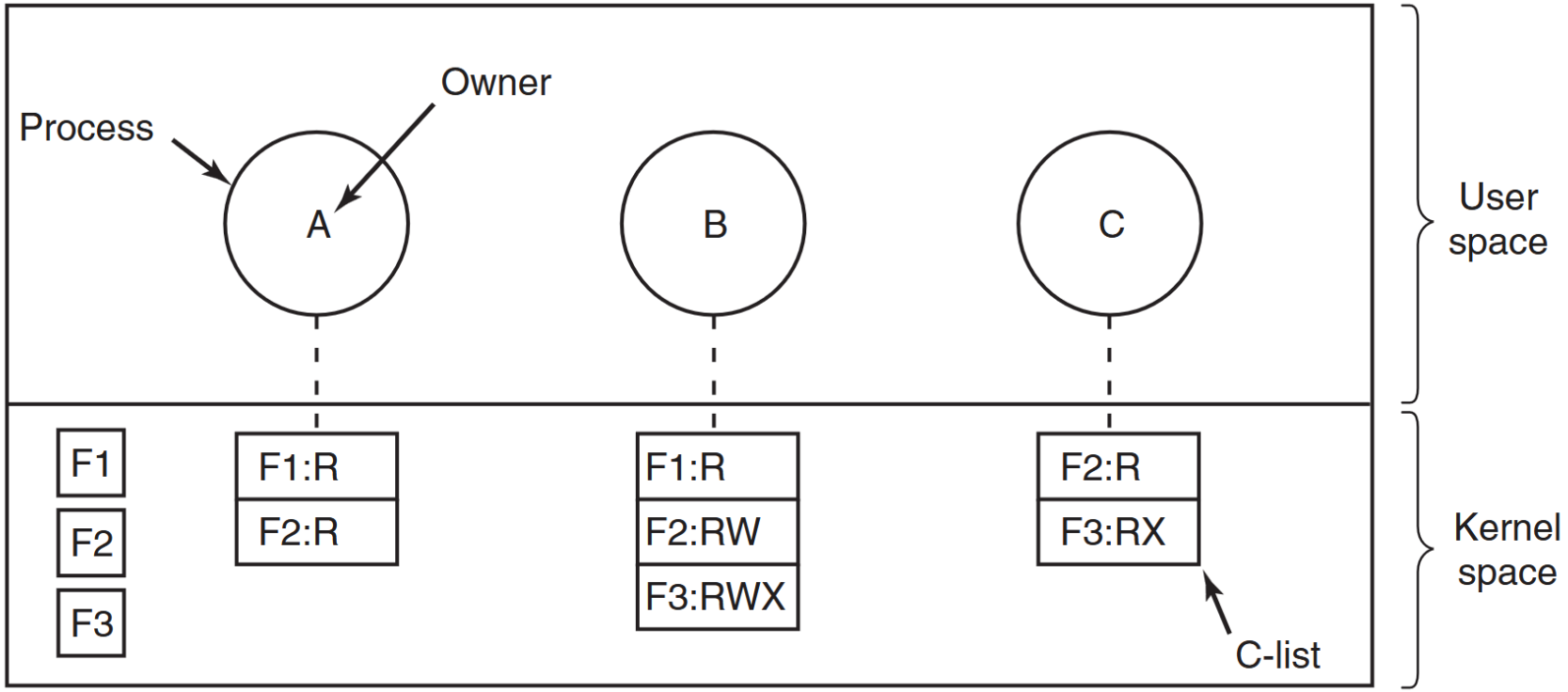
Figure 9-6. Use of access control lists to manage file access.

A **role** is a (UID, GID) that a user can take on that changes their access

Example: Running a process as sysadmin (sudo)

Protection Domains: Capabilities

Idea: Each process stores the resources it is allowed to use



Example: A mobile app that requires your permission to use the camera

Exercise: Represent the contents of this directory as a protection matrix

Note: asw is a member of two groups: users and devel; gmw is a member only of users. Treat each of the two users and two groups as a domain, so that the matrix has four rows (one per domain) and four columns (one per file).

```
– rw– r– – r– –  2 gmw users 908 May 26 16:45 PPP– Notes
– rwx r– x r– x   1 asw devel 432 May 13 12:35 prog1
– rw– rw– – – – 1 asw users 50094 May 30 17:51 project.t
– rw– r– – – – – 1 asw devel 13124 May 31 14:30 splash.gif
```

Protection Matrix

Draw the contents of the protection matrix using an ACL

Draw the contents of the protection matrix using capabilities. Assume the users are running processes that access each file.

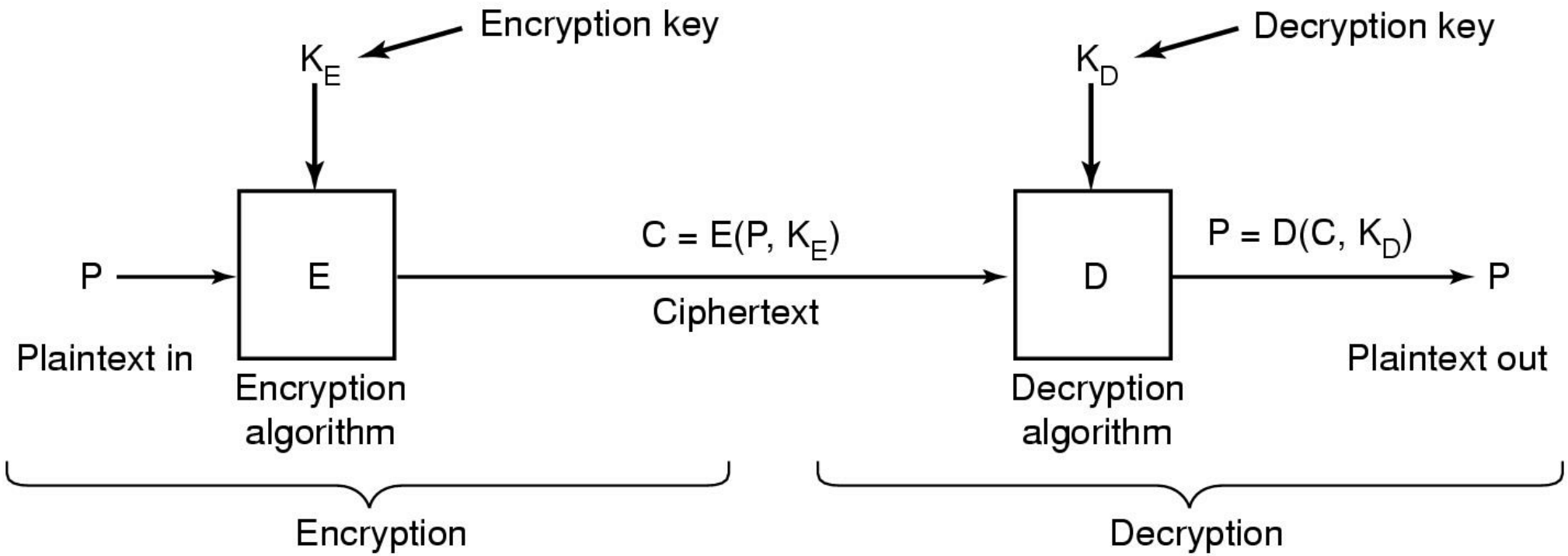
Cryptography: Basics

Goal: Hide information such that it can only be read with a key

Terminology

- *Authentication:* Verifying identity of sender and/or message integrity
- *Integrity:* Message tampering detection
- *Plaintext:* Original message
- *Ciphertext:* Encrypted message
- *Key*: Input for en- and decryption algorithm
- *Encryption:* Plaintext **+** Key $\rightarrow$ Ciphertext
- *Decryption:* Ciphertext **+** Key $\rightarrow$ Plaintext

Basic Set-up of Cryptography



Relationship between the plaintext and the ciphertext

Kerckhoff's Principle

Nonintuitively, the algorithms for encryption/decryption should be *public*

> **Security by obscurity** is the tactic of trying to keep information safe by keeping details hidden. It doesn't work. Why?
>
> Security should be enforced through the *keys* not algorithm

Secret-key cryptography

Idea: Both the sender and receiver use a key to encrypt/decrypt

Property: Given the encryption key, it is easy to find the decryption key

Approach also called symmetric-key cryptography

Example: monoalphabetic substitution

Plain Text	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Cipher Text	Q W E R T Y U I O P A S D F GH J K L Z X C V B N M

Decode QHHAT

Secret-key cryptography

Advantages: Computationally simple/fast

Disadvantages:

- Can be easy to decode with natural languages, where letters appear with well-known frequencies
  - But can work well with a good key
- Both the sender/receiver need to same key

Public-key cryptography

Idea: Remove requirement that both sender/receiver has the key

Property: Given the encryption key, it is nearly impossible to guess the decryption key

How it works:

- Generate two keys: one public, one private
- Only give out the public key.

Example: RSA (Rivest-Shamir-Adleman)

Example: RSA

Based on functions involving large prime numbers p and q

- p and q should be secret
- Compute n = p * q
- Compute $\phi(n) = (p-1) * (q-1)$
- Find an e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$
- Find a d such that (d*e) is congruent to 1 mod $\phi(n)$

Key generation:

- Public key is (n, e) // easy
- Private key is (n, d) // hard

Encryption: ciphertext = pow(plaintext, e) mod n
Decryption: plaintext = pow(ciphertext, d) mod n

One-way functions

Idea: Use a function f such that y = f(x) is easy to compute but x = $f^{-1}(y)$ is hard to compute

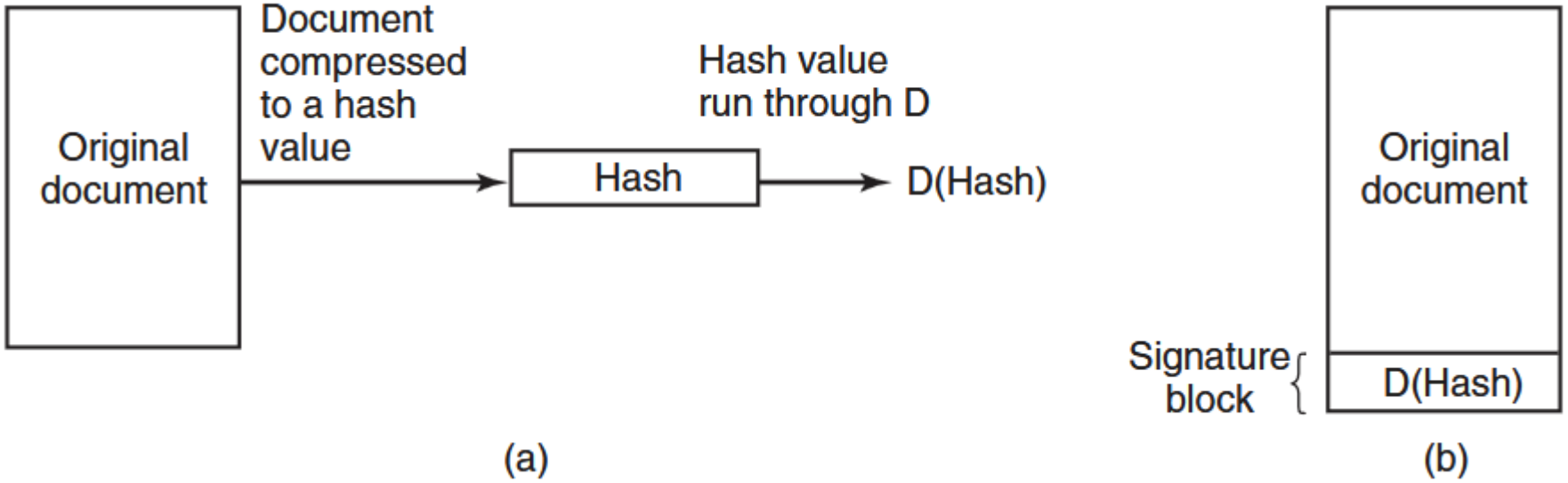
Example: Cryptographic hash functions such as SHA (Secure Hash Algorithm) can convert any document into a short byte sequence

- SHA-1 converts a document to 20 bytes
- SHA-256 converts a document to 32 bytes
- SHA-512 converts a document to 64 bytes

Digital Signatures

Goal: Sign emails, documents in a way that can't be repudiated later

How it works: Compute SHA for a document and encode it using a private key. On receipt, compute the SHA again and compare it to E(D(Hash))



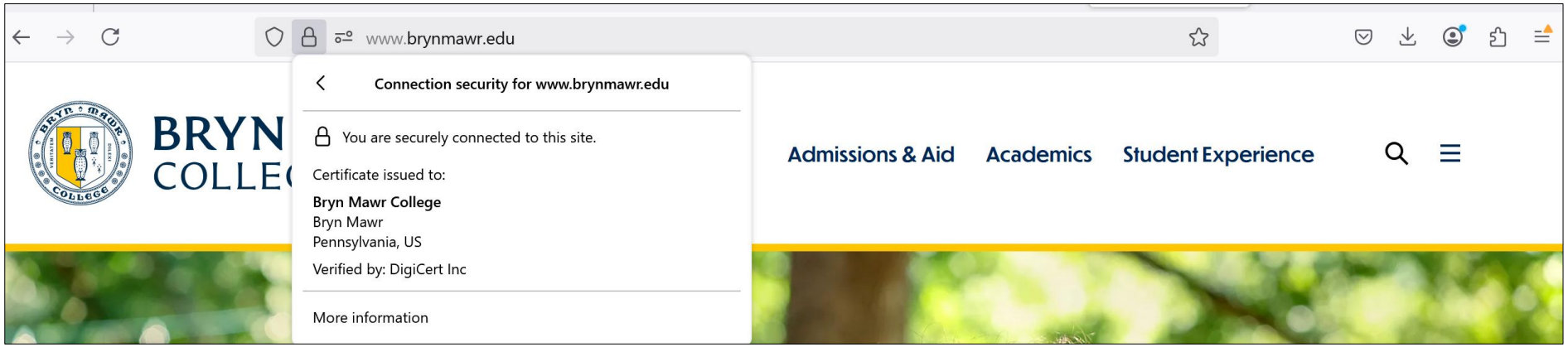
Certificate Authority (CA)

Problem: How can we be sure our public keys are valid?

Approach: Use a trusted third party to keep public keys

A **certificate** contains a name, and public key

Example: HTTPS certificates are stored using trusted authorities that verify the identity of the website (not necessarily the owner)



User Authentication

Authentication is the process of determining which user is making a request

Basic Principles. Authentication must identify:

1. Something the user knows (e.g. password)
2. Something the user has (e.g. ID card)
3. Something user is (e.g. retina scan)

Humans are the weakest link: phishing attacks, easy passwords, data leaks, ...

Passwords

The most commonly used way of authentication

Vulnerabilities

- Stealing passwords
- Poorly chosen passwords that are easy to guess
- Attacks that search through password directories

If you were to guess passwords, how would you go about doing that?

Survey of passwords by Morris&Thompson: could guess 86% of all passwords (1979)

- 15 single ASCII letters
- 72 two ASCII letters
- 464 three ASCII letters
- Words from dictionary, names of people/streets ….

Password Attacks

War dialers / password guessing

Once entrance to a system is gained:

- password file
- packet sniffer
- rsh/rlogin into other machines with known usr/passwd combo

Social Engineering

Unix: /etc/passwd

Passwords stored in a file system are vulnerable to automated attacks

- At first Unix was implemented with a password file holding the actual passwords of users, but with only root permissions.

This had many vulnerabilities

- Copies were made by privileged users
- Copies were made by bugs: classic example posted password file on daily message file

Improvements to First Approach

Hashing and encryption: use password to create a key, then hash based on the DES algorithm for encryption

- Data Encryption Standard (DES) was an early symmetric-key block cipher
- Speed OK for legitimate users
- Takes longer to do automatic search

Password files contains these encrypted entries

Intruder cannot figure out the passwords just by gaining access to password file, but can keep guessing passwords, apply hash/encryption and compare the results to entries in password file

Process is made easier when password files are leaked from major companies (Equifax, LinkedIn, etc)

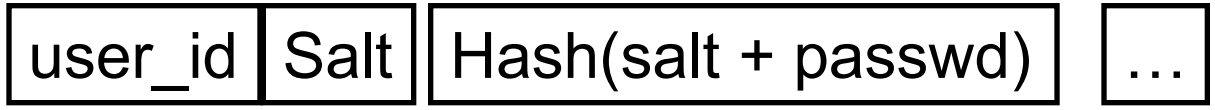
Add Salt

"Salt" the passwords by adding random bits.

- Makes dictionary attacks more expensive.
- Decreases the likelihood that two identical passwords will appear as identical entries in the password file.

12 bit salt results in 4,096 versions of each password.

/etc/passwd entry:



How does this help?

Passwords

Nearly everyone's data has been leaked at some point

https://haveibeenpwned.com

Example: Donald Trump's LinkedIn password was exposed in a 2012 data breach. (So was mine!) Three Dutch hackers tried his exposed password, *yourefired*, on his twitter account @realDonaldTrump and it worked...

Use a **password manager** to help you avoid re-using passwords.

- Bitwarden (online), 1password (online), KeePassXC (local storage only)
- Only you have the password to your pw mmg, so even if the company is hacked, your data cannot be retrieved.

Password Best Practices

Avoid biometric unlocks!!

- Anyone can aim a phone at your face or force your finger to touch the screen

Use a password not a pin

Passphrases are better than passwords

- Ex: Kitty tuckson Termite arugula
- Longer and easier to remember

End-to-end encryption

Idea: Only the sender/receiver can decode the message

Be careful: Most communication is not end-to-end encrypted: Google/Microsoft tools, emails, text, zoom

Signal: End-to-end encryption for text messages

Proton.me: Private alternative to Google mail, calendar, etc

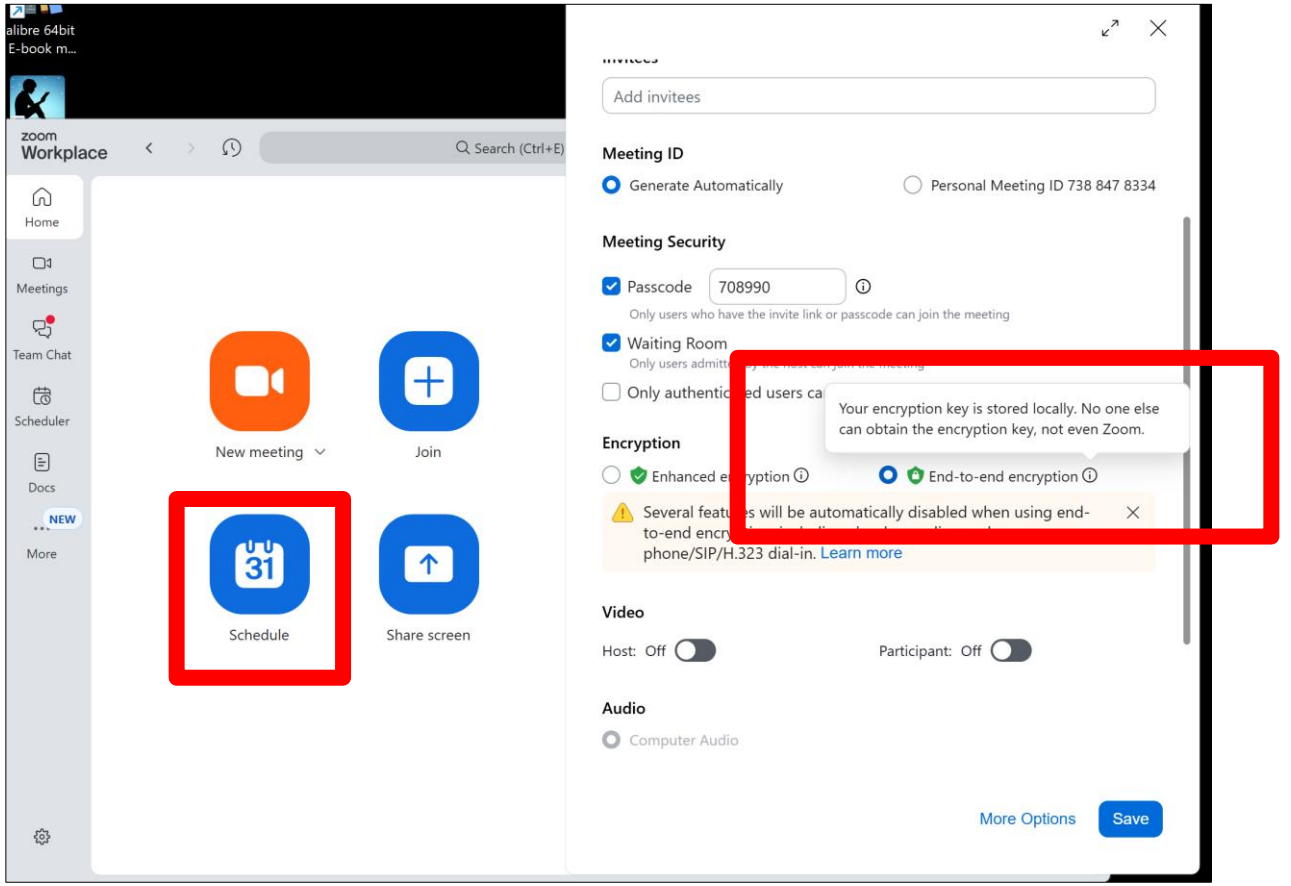
Zoom meetings that contain sensitive information should be E2EE and password protected

End-to-end encryption (E2EE) ensures that no one other than the participants of your meeting can access your conversation.

“Enhanced encryption” stores the key on a Zoom server. Thus, Zoom could choose to share meetings with third parties.

E2EE security is **not the default.** It must be set when you create the meeting.

For example, *WebinarTV* scrapes the internet for Zoom calls, records them, and turns them in AI-generated podcasts – without the knowledge of the people in the zoom calls*.



** Reporting from 404Media, https://www.404media.co/this-company-is-secretly-turning-your-zoom-calls-into-ai-podcasts*

Be careful: Security can be undermined

Example: Signal notifications can still be read when notifications are enabled

Example: Meta-data on photos can reveal your location. Meta-data on Word documents can reveal your identity.

Example: Financial records can reveal the owner of an account

404 MEDIA

FBI Extracts Suspect's Deleted Signal Messages Saved in iPhone Notification Database

By Joseph Cox • 9 Apr 2026

View in browser

While on the run from officials in Belize in 2012, John McAfee was located in Guatemala due to a photo of him posted by *Vice*

Signal best practices

Don’t show notifications on your screen

Set an alias. Don’t use an identifiable photo

Enable disappearing messages by default

Hide your phone number

Enable call relay

Disk and file encryption

Disk encryption allows you to protect your data from people who have physical access to your phone, laptop, or USB

Disk encryption doesn’t protect against network attacks

Don’t leave your laptop open in public places

Example: Ross Ulbricht (Silk Road) had his encrypted laptop confiscated in 2013 when he left it briefly unattended at the San Francisco Public Library. Two undercover FBI agents distracted him by pretending to be lovers in a fight. Ulbricht was charged with money laundering, hacking, drug trafficking, and other crimes

Encryption should be an option on both your laptops and phones (Bitlocker, Veracrypt, Filevault). Xubuntu (Linux) allows you to encrypt your disk when you install it.

7-zip allows you to create password-protected ZIP files

Operating System Security

Insider Attacks

- Executed by programmers/employess within an organization
- Leverages insider knowledge and access

Malware

- Malicious software installed without a users' consent/knowledge

Insider Attacks

A **logic bomb** is a piece of code designed to release information, lock information, or delete files UNLESS a password is given at regular intervals

- Idea: Form of blackmail against getting fired

A **back door** is a piece of code that circumvents a security measure

- Examples: A hidden code that allows law enforcement to unlock your phone

Login spoofing refers to a fake login screen that imitates the real login for the purposes of stealing usernames and passwords (related to a phishing attack).

Malware

Goal: Spread widely across machines using the internet. Once infected,

- IP of the machine is reported back
- Backdoor is installed so that the computer can controlled remotely
  - Machines taken over by malware are called **zombies**
  - A collection of machines working together under malware is called a **botnet**

Types: Ransomware attacks, keyloggers, spyware, rootkits

Methods for spread: Drive-by-download, Trojan Horses, Viruses, Worms

Malware: Trojan Horses

Idea: Embed the malware in a useful program

Like a QR code generator, movie player, system command, etc

Examples

- User installs an application from the web that writes the malware (as well as doing to desired task!)
- User runs an existing utility that has been replaced with an imposter (Demo: Exploiting PATH)

Greek warriors during the Trojan War hid inside a horse to enter the city of Troy and win the war.

Malware: Viruses

A **virus** is a program that can reproduce itself by attaching its code to another program.

Types:

- A **companion virus** runs when another (usually legitimate) program runs
- An **overwrite virus** replaces an executable with different code
- A **parasitic virus** attaches itself to an existing program but allows it to still run normally.
- A **memory virus** sits in RAM and might replace a signal or trap handler with its own code
- A **boot sector virus** hides in the master boot record. It moves the real boot sector to another boot section so the machine can still appear to boot manually.
- A **device driver virus** hides inside a driver.
- A **macro virus** (windows) hides in executable code embedded in a document or email.

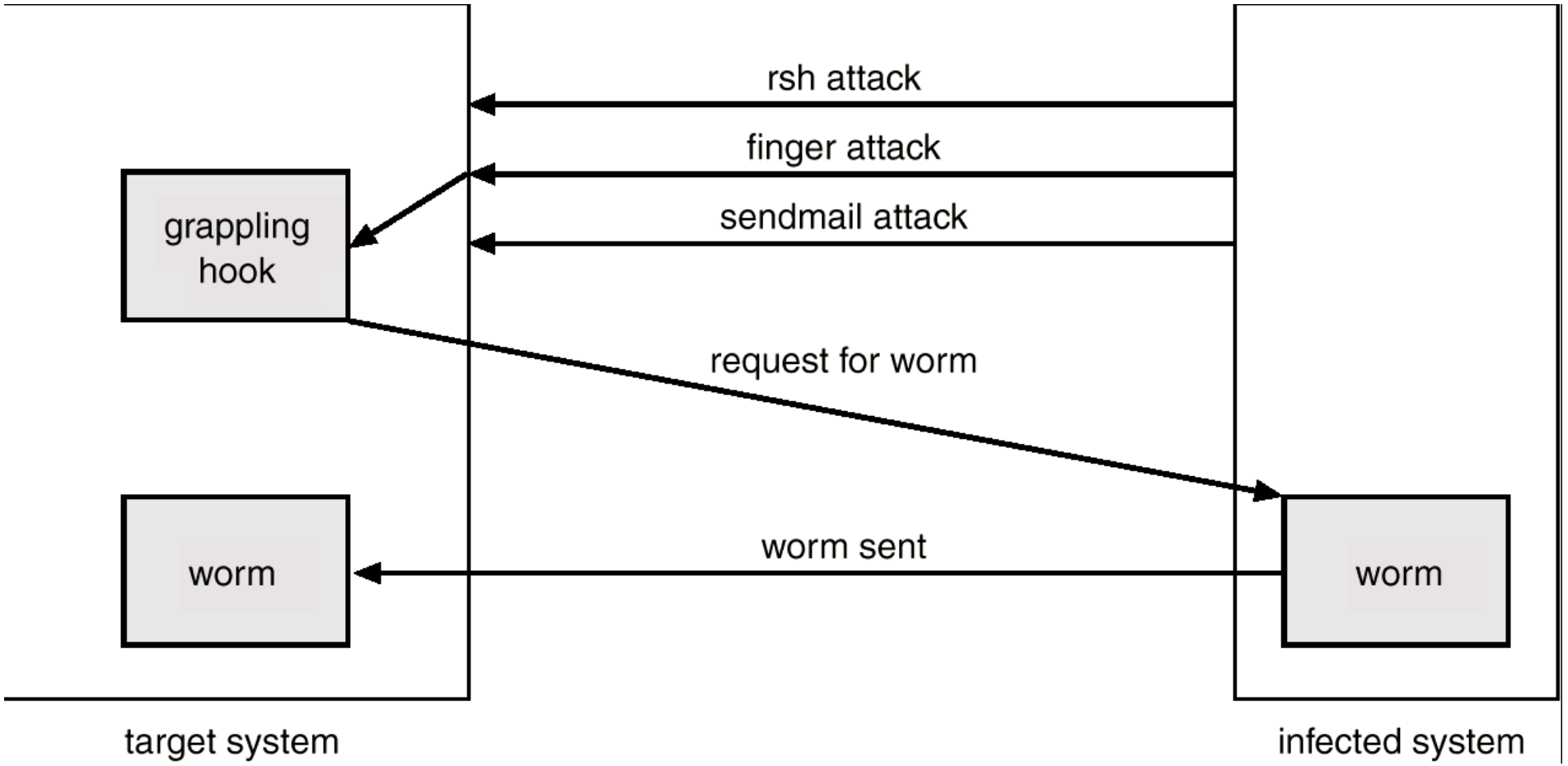
Malware: Worms

A **worm** is a self-replicating program

Example: Morris Worm (1988)

- Two programs: bootstrap and worm
  - The bootstrap was compiled and executed on the remote machine. It would connect to a source machine and download the worm.
  - The worm looked through the computer's routing tables to find other IP addresses to try to spread itself to. It would try to break passwords and use open remote shell connections to install itself.
- If a worm were already on a machine, the worm would run an additional copy of itself 1 in 7 times: This filled machines with worm such that they could run, eventually bringing attention to Morris who was tried and convicted in federal court (10K fine, 400 hrs or community service, 3 yrs probation)

The Morris Internet Worm



Malware: Spyware

Spyware is software that surreptitiously runs in the background, doing other work

Examples:

- marketing telemetry (what you click on, length of time looking at something, etc)
- surveillance (employee software that monitors progress, websites visited, etc)
- windows update (kidding?)

A **drive-by-download** is when malware is installed by clicking on a bad website.

Modern browsers have protections against this

What might spyware do?

- Change browser homepage, bookmarks, toolbar (**browser hijacking**)
- Change the default media player or browser on your computer
- Add new icons to your desktop
- Place ads in the standard Windows dialog boxed
- Generate popup advertisements

Malware: Rootkits

A **rootkit** is a program that attempts to conceal its existence.

Property: The malware is stored such that it cannot be found by typical anti-virus scanning

Types:

- A **firmware rootkit** might hide itself in the BIOS, UEFI
- A **hypervisor rootkit** runs the entire OS as a virtual machine, under its control
- A **kernel rootkit** hides in a device driver or loadable kernel module
- A **library rootkit** hides in a system library, like libc
- An **application rootkit** hides in a large application that creates files, such as an image viewer that creates thumbnail images.

Defenses: Protective software/hardware

A **firewall** funnels all network messages through a central place where it can be checked against rules for what types of network traffic is allowed versus not

Example: Web traffic is usually allows but remote ssh connections are not

A **virus scanner**, such as an antivirus program, scans the disk

- check for known virus code
- check for code integrity (using a checksum to see if a known exe has been modified)
- check behaviors, e.g. is the performance of a known program suspiciously slow or using more memory?
- can severely slow down the performance of your machine

Defenses: Application

Use **code signing** based on digital signatures, to ensure that an executable has not been tampered with (Idea: Run only unmodified software from reliable vendors)

Have a trusted process **jail** untrusted software. Monitor the process. Ask the kernel (or user with admin privileges) whether the application should be allowed to run.

Model-based Intrusion Detection: Graph system calls and detect those that don't fit the pattern

Encapsulate applications (e.g. **sandbox** them and when with a **reference monitor**) with as few resources and privileges as possible. Any damage the application might cause is contained.

Run code through an interpreter (ex. JVM). This allows all commands to checked before executing.