COS 318: Operating Systems

Security

Jaswinder Pal Singh
Computer Science Department
Princeton University

(http://www.cs.princeton.edu/courses/cos318/)

Security

- The security environment
- Basics of cryptography
- User authentication
- Attacks in a non-networked world
- Attacks in a networked world

Security Goals and Threats

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service
Exclusion of Outsiders	System Takeover (e.g. by viruses)

- Operating systems have goals
  - Confidentiality, Integrity, Availability, Exclusion of outsiders
- Someone attempts to subvert the goals
  - Fun or accomplishment
  - Commercial gain

What kinds of intruders are there?

- Casual prying by nontechnical users
  - Curiosity
- Snooping by insiders
  - Often motivated by curiosity or money
- Determined attempt to make trouble, or personal gain
  - May not be an insider
- Commercial or military espionage

Accidents cause problems, too…

- Fires, Earthquakes, Floods
- Hardware or software error
  - CPU malfunction
  - Disk crash
  - Program bugs
- Human errors
  - Data entry
  - Wrong tape mounted
  - rm *

How to Protect?

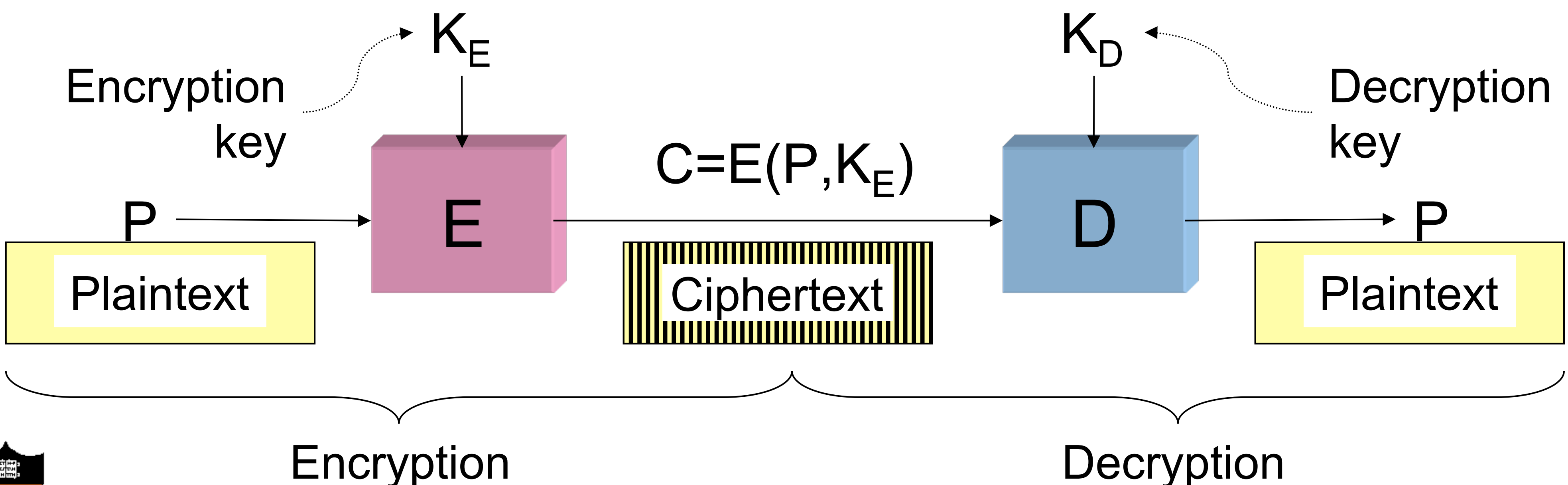
- Hardware?
  - Parity and error correction
  - Physical access
  - Hardware assistance for memory isolation/protection
  - Timers
  - …
- OS?
  - Process isolation, scheduling, encryption, privileges, passwords
- Communication protocols?

Cryptography

- Goal: keep information from those who aren't supposed to see it
  - Do this by "scrambling" the data
- Use a well-known algorithm to scramble data
  - Algorithm has two inputs: data & key
  - Algorithms are publicly known
  - Key is known only to "authorized" users
- Cracking good codes is ***very*** difficult. But possible

Cryptography basics

- Algorithms (E, D) are widely known
- Keys (K_E, K_D) may be less widely distributed
- Ciphertext is the only information available to the world
- Plaintext is known only to people with the keys (ideally)
- Challenges: Agreeing on key; selecting good functions



Secret-key encryption

- Also called symmetric-key encryption
- Simple example: Monoalphabetic substitution
  - Each letter replaced by different letter
- Easy to break!
- Given the encryption key, easy to generate the decryption key
- Alternatively, use different (but similar) algorithms for encryption and decryption

Modern encryption algorithms

- Data Encryption Standard (DES)
  - Uses 56-bit keys
  - Same key is used to encrypt & decrypt
  - Keys used to be difficult to guess
    - Modern computers can try millions of keys per second with special hardware
    - For $250K, EFF built a machine that broke DES quickly
- More recent algorithms (AES, Blowfish) use 128 bit keys
  - Adding one bit makes it twice as hard to guess
  - Must try 2^{127} keys, on average, to find the right one
  - At 10^{15} keys per second, this would require over 10^{21} seconds, or 1000 billion years!
  - Modern encryption isn't usually broken by brute force

Unbreakable codes

- There ***is*** such a thing as an unbreakable code
  - Use a truly random key, as long as the message to be encoded
  - XOR the message with the key a bit at a time
- Code is unbreakable because
  - Key could be anything
  - Without knowing key, message could be anything with the correct number of bits in it
- Difficulty: distributing key is as hard as distributing msg
- Difficulty: generating truly random bits
  - May use physical processes: radioactive decay, leaky diode, etc.
    - Lava lamp (!) [http://www.sciencenews.org/20010505/mathtrek.asp]

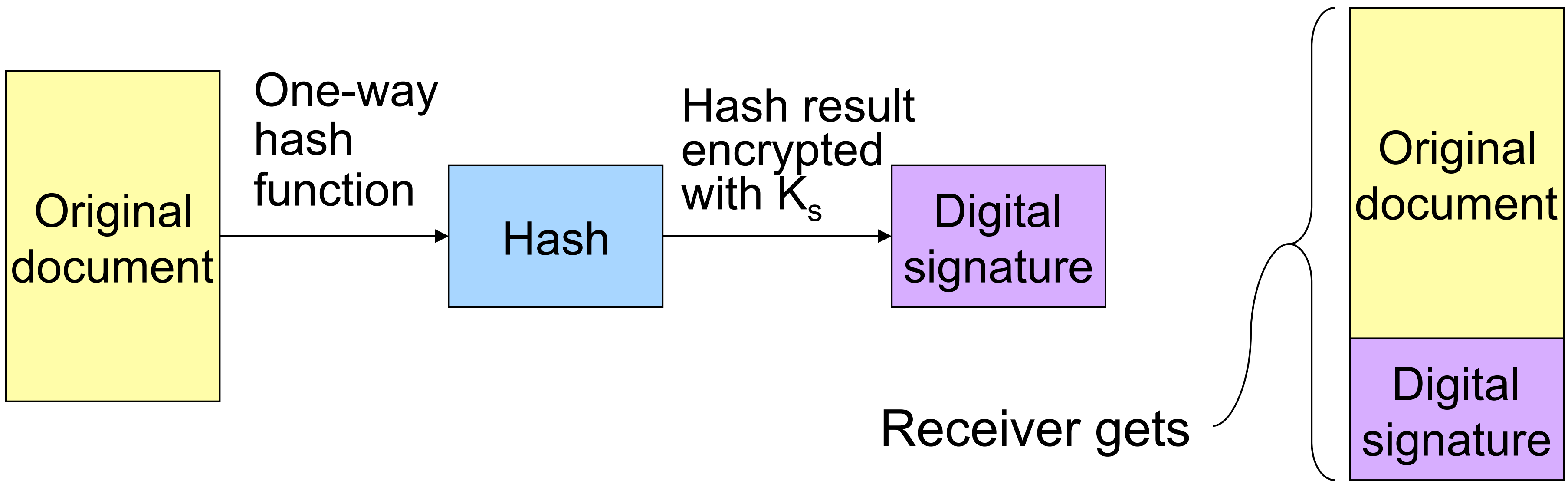
Public-key cryptography

- Instead of using a single shared secret, keys come in pairs
  - One key of each pair distributed widely (*public key*), K_p
  - One key of each pair kept secret (*private/secret key*), K_s
  - Two keys are inverses of one another, but not identical
  - Encryption & decryption are the same algorithm, so $E(K_p,E(K_s,M) = E(K_s,E(K_p,M) = M$
  - Usually, public key for encryption, private for decryption
- Most popular method involves primes and exponentiation
  - Difficult to crack unless large numbers can be factored
    - Multiplying numbers is easy, factoring is hard
  - Issue: Very slow for large messages

One-way functions

- Function such that
  - Given formula for f(x), easy to evaluate y = f(x)
  - Given y, computationally infeasible to find x such that y = f(x)
- Often, operate similarly to encryption algorithms
  - Produce fixed-length rather than variable output
  - Similar to XOR-ing blocks of ciphertext together
- Common algorithms include
  - MD5: 128-bit result
  - SHA-1: 160-bit result

Digital signatures



- Digital signature computed by
  - Applying one-way hash function to original document
  - Encrypting result with sender's *private* key
- Receiver can verify by
  - Applying one-way hash function to received document
  - Decrypting signature using sender's public key
  - Comparing the two results: equality means document unmodified

User authentication

- Problem: how does the computer know who you are?
- Solution: Use *authentication* to identify:
  - Something the user knows
  - Something the user has
  - Something the user is
- This must be done before user can use the system
- Important: from the computer's point of view…
  - Anyone who can duplicate your ID *is* you
  - Fooling a computer isn't all that hard…

Authentication using passwords

```
Login: elm
Password: foobar

Welcome to Linux!
```

```
Login: jimp
User not found!

Login:
```

```
Login: elm
Password: barfle
Invalid password!

Login:
```

- Successful login lets the user in
- If things don't go so well…
  - Login rejected after name entered
  - Login rejected after name and incorrect password entered
- Don't notify the user of incorrect user name until *after* the password is entered!
  - Early notification can make it easier to guess valid user names

Sample breakin (from LBL)

```
LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

Lesson: change all the default system passwords!

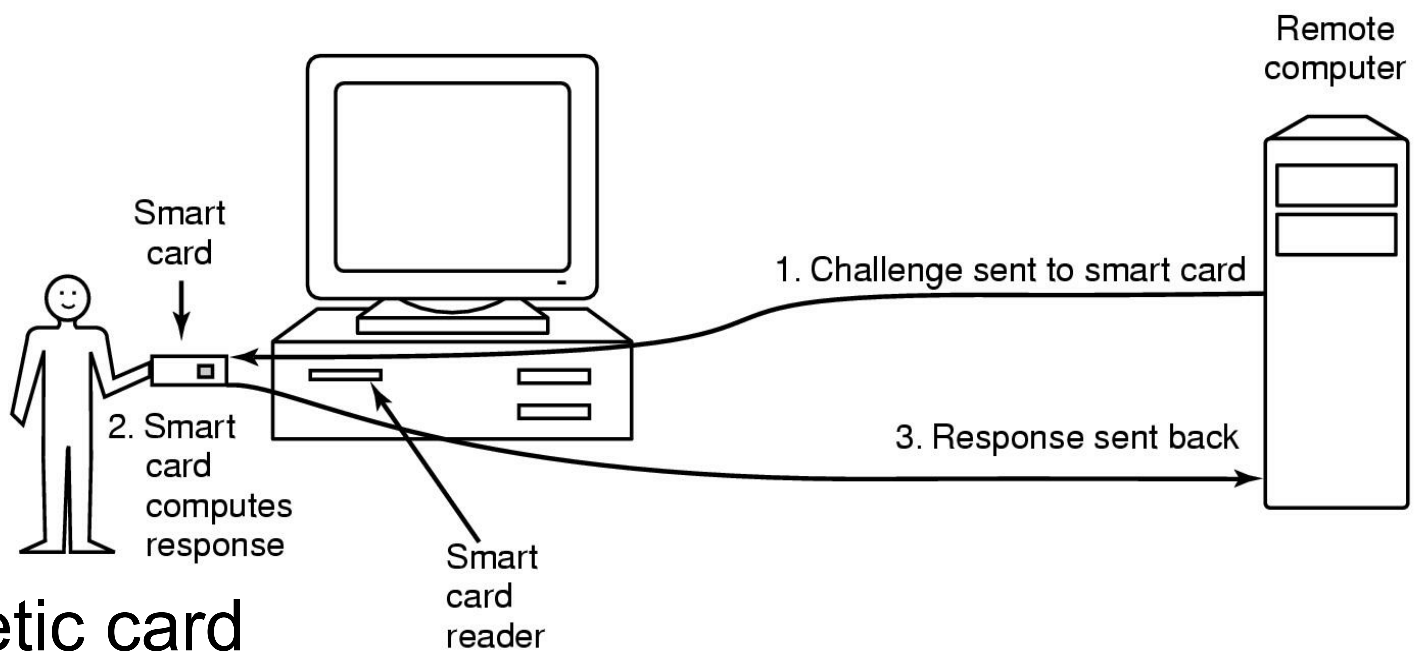
Dealing with passwords

- Passwords should be memorable
  - Users shouldn’t need to write them down
  - Users should be able to recall them easily
- Passwords shouldn’t be stored “in the clear”
  - Password file is often readable by all system user!
  - Password must be checked against entry in this file
- Solution: use hashing to hide “real” password
  - One-way function converts password to meaningless string of digits (Unix password hash, MD5, SHA-1)
  - Difficult to find another password that hashes to the same random-looking string
  - Knowing the hashed value and hash function gives no clue to the original password

Salting the passwords

- Hashing is not enough
  - Hackers can get a copy of the password file
  - Run through dictionary words and names for possible passwords
    - Hash each name
    - Look for a match in the file
- Solution: use a “salt”
  - Random characters added to the password before hashing
  - Salt characters stored “in the clear”
  - Increases the number of possible hash values for a given password
    - Actual password is “pass”
    - Salt = “aa” => hash “passaa”
    - Salt = “bb” => hash “passbb”
  - Result: cracker has to try many more combinations

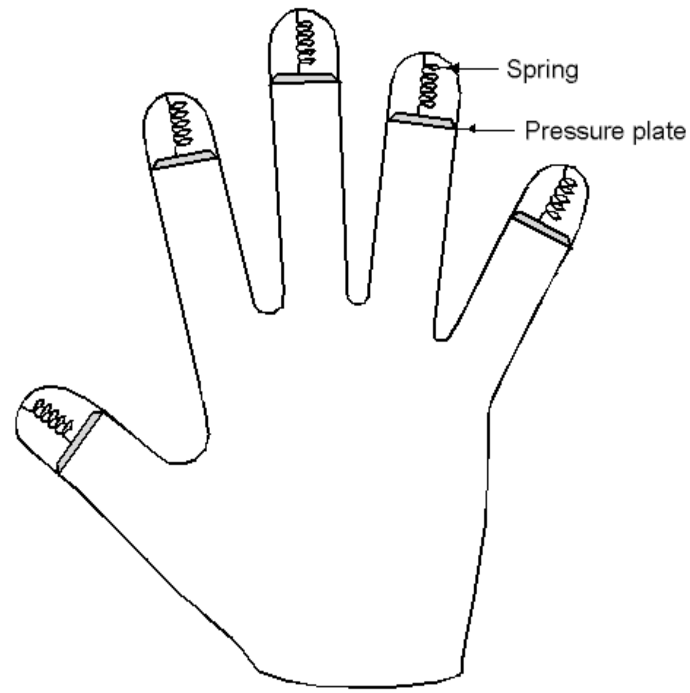
Authentication using a physical object



- Magnetic card
  - Stores a password encoded in the magnetic strip
  - Allows for longer, harder to memorize passwords
- Smart card
  - Card has secret encoded on it, but not externally readable
  - Remote computer issues challenge to the smart card
  - Smart card computes the response and proves it knows the secret

Authentication using biometrics

- Use basic body properties to prove identity
- Examples include
  - Fingerprints
  - Voice
  - Hand size, finger length
  - Retina patterns
  - Iris patterns
  - Facial features
  - Image analysis, gait analysis
- Potential problems
  - Duplicating the measurement
  - Stealing it from its original owner?



Countermeasures

- Limiting times when someone can log in
- Automatic callback at number prespecified
  - Can be hard to use unless there's a modem involved
- Limited number of login tries
- A database of all logins
- Simple login name/password as a trap
  - Security personnel notified when attacker bites

Attacks on computer systems

- Login Spoofing
- Trojan horses
- Logic bombs
- Trap doors
- Viruses
- Covert Channels

Login spoofing

Real login screen

Phony login screen

- No difference between real & phony login screens
- Intruder sets up phony login, walks away
- User logs into phony screen
  - Phony screen records user name, password
  - Phony screen prints “login incorrect” and starts real screen
  - User retypes password, thinking there was an error
- Solution: don’t allow certain characters to be “caught”

Trojan horses

- Free program made available to unsuspecting user
  - Actually contains code to do harm
  - May do something useful as well…
- Altered version of utility program on victim's computer
  - Trick user into running that program

Logic bombs

- Programmer writes (complex) program
  - Wants to ensure that he's treated well
  - Embeds logic "flaws" that are triggered if certain things aren't done
    - Enters a password daily (weekly, or whatever)
    - Adds a bit of code to fix things up
    - Provides a certain set of inputs
- If conditions aren't met
  - Program simply stops working
  - Program may even do damage
    - Overwriting data
    - Failing to process new data (and not notifying anyone)
- Programmer can blackmail employer
- Needless to say, this is highly unethical!

Trap doors

```
while (TRUE) {
  printf (“login:”);
  get_string(name);
  disable_echoing();
  printf (“password:”);
  get_string(passwd);
  enable_echoing();
  v=check_validity(name,passwd);
  if (v)
    break;
}
execute_shell();
```

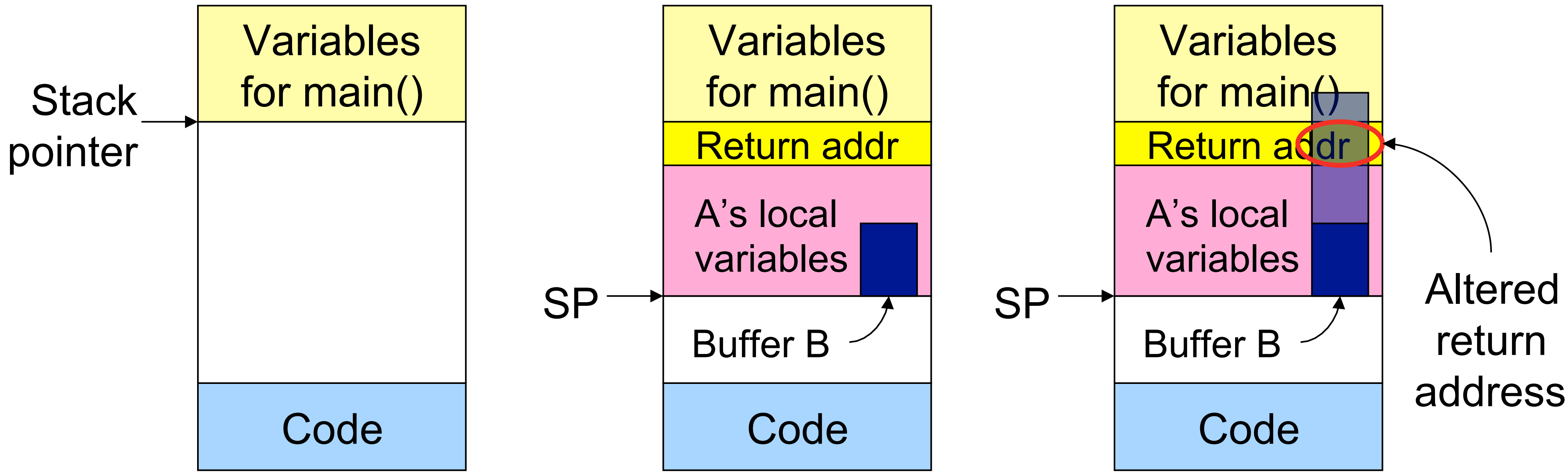
Normal code

```
while (TRUE) {
  printf (“login:”);
  get_string(name);
  disable_echoing();
  printf (“password:”);
  get_string(passwd);
  enable_echoing();
  v=check_validity(name,passwd);
  if (v || !strcmp(name, “jps”))
    break;
}
execute_shell();
```

Code with trapdoor

Trap door: user’s access privileges coded into program

Buffer overflow



- Big source of bugs in operating systems
  - Most common in user-level programs that help the OS do something
  - May appear in "trusted" daemons
- Exploited by modifying the stack to
  - Return to a different address than that intended
  - Include code that does something malicious
- Accomplished by writing past end of a buffer on stack

Covert channels

- Circumvent security model by using more subtle ways of passing information
- Can’t directly send data against system’s wishes
- Send data using “side effects”
  - Allocating resources
  - Using the CPU
  - Locking a file
  - Making small changes in legal data exchange
- *Very* difficult to plug leaks in covert channels!

Covert channel using file locking

- Exchange information using file locking
- Assume n+1 files accessible to both A and B
- A sends information by
  - Locking files 0..n-1 according to an n-bit quantity to be conveyed to B
  - Locking file n to indicate that information is available
- B gets information by
  - Reading the lock state of files 0..n+1
  - Unlocking file n to show that the information was received
- May not even need access to the files (on some systems) to detect lock status!

Steganography

- Hide information in other data
- Picture on right has text of 5 Shakespeare plays
  - Encrypted, inserted into low order bits of color values

Zebras

Hamlet, Macbeth, Julius Caesar
Merchant of Venice, King Lear

Social Engineering

- Convince a system programmer to add a trap door
- Beg admin's secretary (or other people) to help a poor user who forgot password
- Pretend you're tech support and ask random users for their help in debugging a problem

Design principles for security

- System design should be public
- Default should be no access
- Check for current authority
- Give each process least privilege possible
- Protection mechanism should be
  - Simple
  - Uniform
  - In the lowest layers of system
- Scheme should be psychologically acceptable
- Keep it simple!

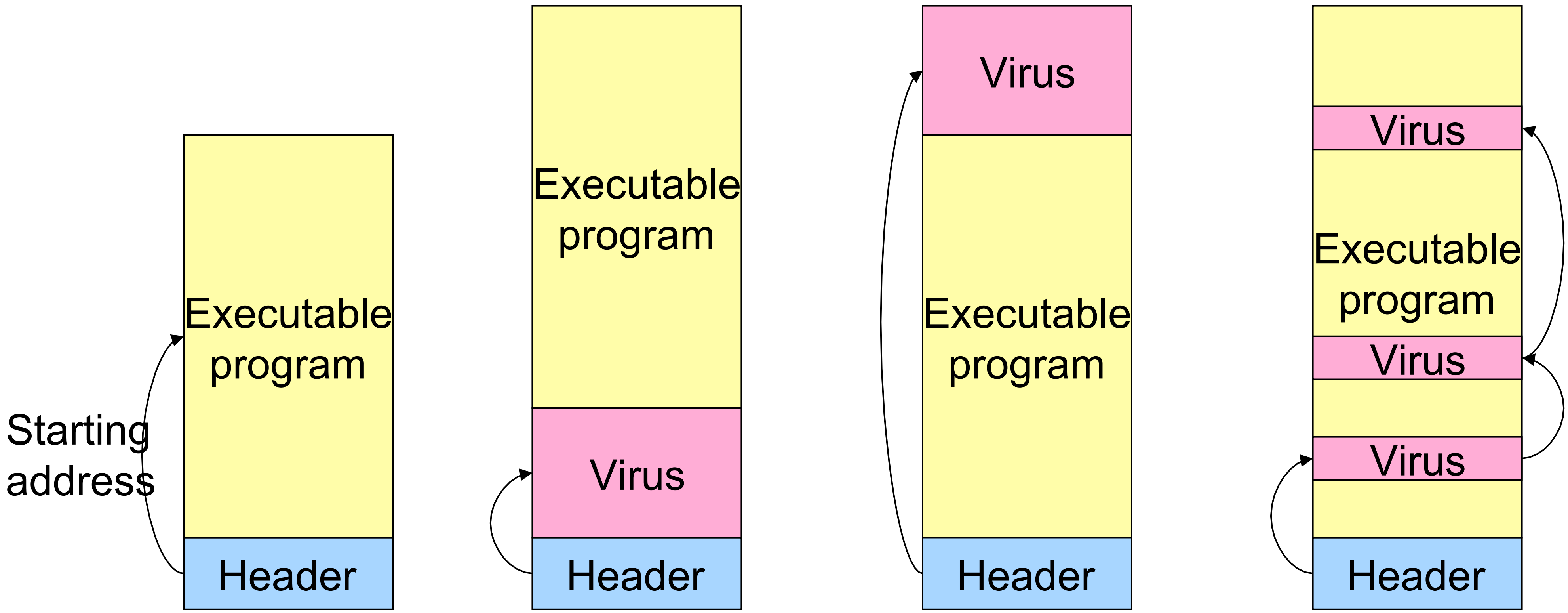
Security in a networked world

- External threat
  - Code transmitted to target machine
  - Code executed there, doing damage
- Goals of virus writer
  - Quickly spreading virus
  - Difficult to detect
  - Hard to get rid of
  - Optional: does something malicious
- Virus: embeds itself into other (legitimate) code to reproduce and do its job
  - Attach its code to another program
  - Additionally, may do harm

How viruses work

- Virus language
  - Assembly language: infects programs
  - "Macro" language: infects email and other documents
    - Runs when email reader / browser opens message
    - Program "runs" virus (as attachment) automatically
- Inserted into another program
  - Use tool called a "dropper"
  - May also infect system code (boot block, etc.)
- Virus dormant until program executed
  - Then infects other programs
  - Eventually executes its "payload"

Where viruses live in the program



Uninfected program

Virus at start of program

Virus at end of program

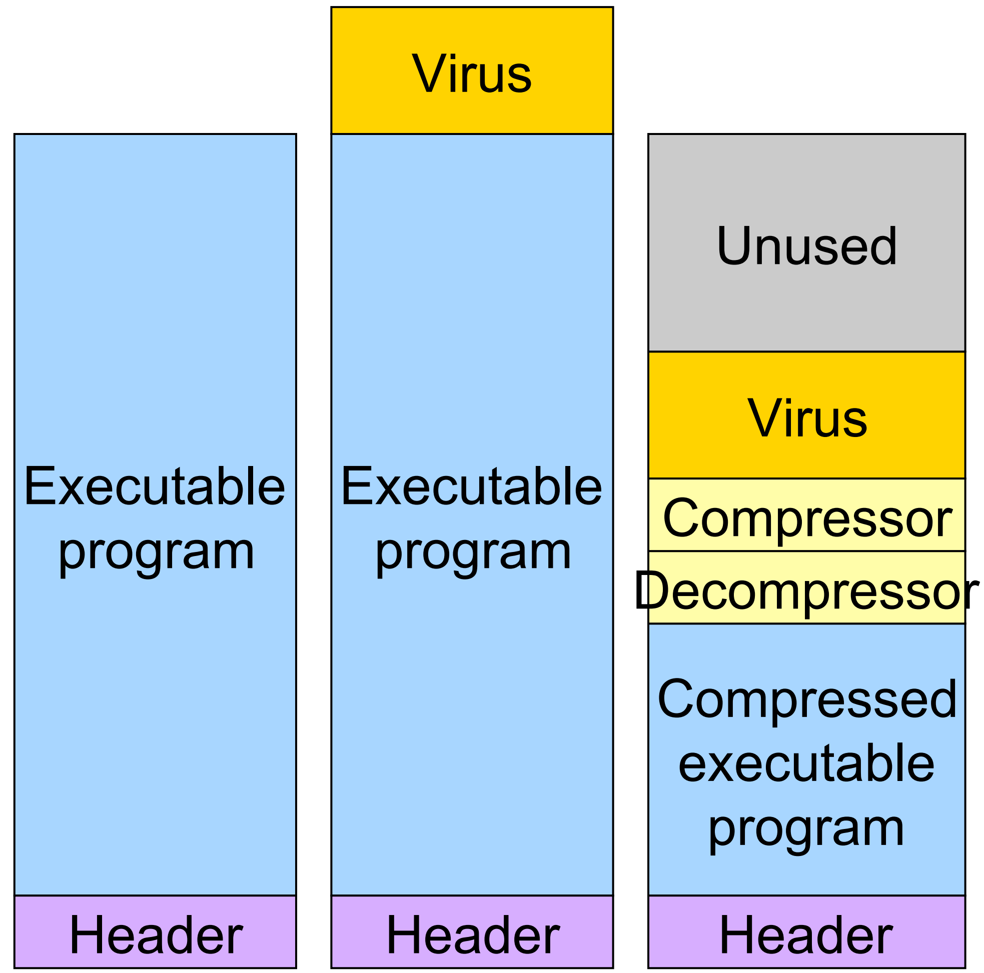
Virus in program's free spaces

How do viruses spread?

- Virus placed where likely to be copied
  - Popular download site
  - Photo site
- When copied
  - Infects programs on hard drive, floppy
  - May try to spread over LAN or WAN
- Attach to innocent looking email
  - When it runs, use mailing list to replicate
  - May mutate slightly so recipients don't get suspicious

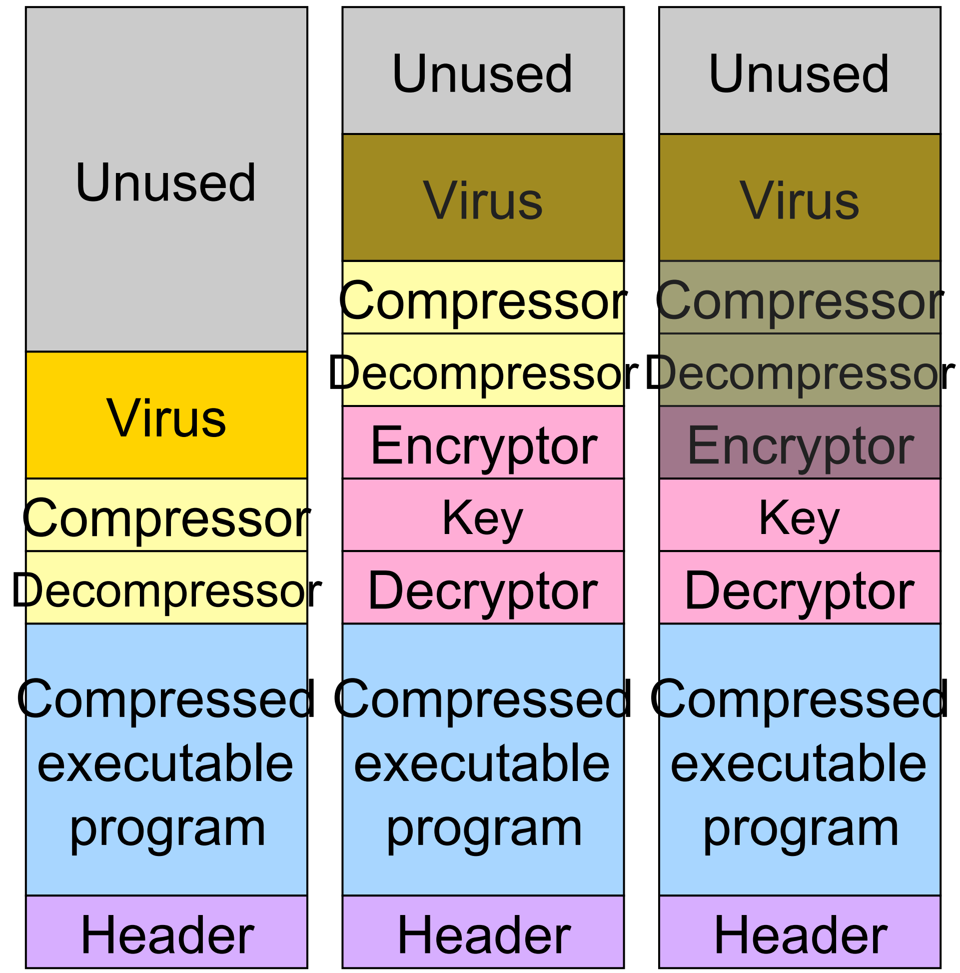
Hiding a virus in a file

- Start with an uninfected program
- Add the virus to the end of the program
  - Problem: file size changes
  - Solution: compression
- Compressed infected program
  - Decompressor: for running executable
  - Compressor: for compressing newly infected binaries
  - Lots of free space (if needed)
- Problem (for virus writer): virus easy to recognize



Using encryption to hide a virus

- Hide virus by encrypting it
  - Vary the key in each file
  - Virus "code" varies in each infected file
  - Problem: lots of common code still in the clear
    - Compress / decompress
    - Encrypt / decrypt
- Even better: leave only decryptor and key in the clear
  - Less constant per virus
  - Use polymorphic code (more in a bit) to hide even this



How can viruses be foiled?

- Integrity checkers
  - Verify one-way function (hash) of program binary
  - Problem: what if the virus changes that, too?
- Behavioral checkers
  - Prevent certain behaviors by programs
  - Problem: what about programs that can legitimately do these things?
- Avoid viruses by
  - Having a good (secure) OS
  - Installing only shrink-wrapped software (just hope that the shrink-wrapped software isn't infected!)
  - Using antivirus software
  - Not opening email attachments
- Recovery from virus attack
  - Hope you made a recent backup
  - Recover by halting computer, rebooting from safe disk (CD-ROM?), using an antivirus program

Worms vs. viruses

- Viruses require other programs to run
- Worms are self-running (separate process)
- The 1988 Internet Worm
  - Consisted of two programs
    - Bootstrap to upload worm
    - The worm itself
  - Exploited bugs in sendmail and finger
  - Worm first hid its existence
  - Next replicated itself on new machines
  - Brought the Internet (1988 version) to a screeching halt

Mobile code

- Goal: run (untrusted) code on my machine
- Problem: how can untrusted code be prevented from damaging my resources?
- One solution: sandboxing
  - Memory divided into 1 MB sandboxes
  - Accesses may not cross sandbox boundaries
  - Sensitive system calls not in the sandbox
- Another solution: interpreted code
  - Run the interpreter rather than the untrusted code
  - Interpreter doesn’t allow unsafe operations
- Third solution: signed code
  - Use cryptographic techniques to sign code
  - Check to ensure that mobile code signed by reputable organization

Virus damage scenarios

- Blackmail
- Denial of service as long as virus runs
- Permanently damage hardware
- Target a competitor's computer
  - Do harm
  - Espionage
- Intra-corporate dirty tricks
  - Practical joke
  - Sabotage another corporate officer's files