R

بسم الله الرحمن الرحیم

سیستم عامل

جلسه بیست و پنجم – امنیت و حفاظت از سیستم عامل

File System Performance

- Optimizing File I/O:
  - *Memory-mapped files reduce system call overhead.*
  - *Buffer caches minimize costly disk reads, caching frequently accessed blocks.*
- Data Placement Matters:
  - *Careful organization (cylinder groups) and file system design (FFS) reduce seek times.*
  - *Log-structured file systems streamline writes by treating the disk as an append-only log.*
- Ensuring Reliability:
  - *Journaling, backups, and consistency checks maintain integrity and facilitate recovery.*

جلسه‌ی جدید

Overview

- Different aspects of security
- User authentication
- Protection mechanisms
- Attacks:
  - \- *trojan horses, spoofing, logic bombs, trap doors, buffer overflow attacks, viruses, worms, mobile code, sand boxing*
- Brief intro to cryptography tools
  - \- *one-way functions, public vs private key encryption, hash functions, and digital signatures*

Security Overview

- Security flavors
  - *Confidentiality - protecting secrets*
  - *Integrity - preventing data contents from being changed*
  - *Availability - ensuring continuous operation*

- Know thine enemy!
  - *User stupidity (bad default settings from companies)*
  - *Insider snooping*
  - *Outsider snooping*
  - *Attacks (viruses, worms, denial of service)*
  - *Bots*

Accidental Data Loss

Distinguishing security from reliability:

- Acts of God
  - *fires, floods, wars*
- Hardware or software errors
  - *CPU malfunction, bad disk, program bugs*
- Human errors
  - *data entry, wrong tape mounted*
  - – *you are probably the biggest threat you'll ever face!*

USER AUTHENTICATION

User Authentication

- Must be done before the user can use the system !
- Subsequent activities are associated with this user
  - Fork process
  - Execute program
  - Read file
  - Write file
  - Send message

- Authentication must identify:
  - *Something the user knows*
  - *Something the user has*
  - *Something the user is*

Authentication Using Passwords

User name: something the user knows
Password: something the user knows
How easy are they you guess (crack)?

```
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 (easier to crack)

(c) Login rejected after name and password typed (larger search space!)

Problems With Pre-Set Values

- Pre-set user account and default passwords are easy to guess

```
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
```

Storing Passwords

- The system must store passwords in order to perform authentication
- How can passwords be protected?
  - *Rely on file protection*
    - store them in protected files
    - compare typed password with stored password
  - *Rely on encryption*
    - store them encrypted
    - use one way function (cryptographic hash)
    - can store encrypted passwords in readable files

Password Management In Unix

- Password file - /etc/passwd
  - *It's a world readable file!*

- /etc/passwd entries
  - *User name*
  - *Password (encrypted)*
  - *User id*
  - *Group id*
  - *Home directory*
  - *Shell*
  - *Real name*
  - …

Dictionary Attacks

- If encrypted passwords are stored in world readable files and you see an encrypted password is the same as yours
  - *The password is also the same as your password!*

- If the encryption method is well known, attackers can:
  - *Encrypt an entire dictionary*
  - *Compare encrypted dictionary words with encrypted passwords until they find a match*

Salting Passwords

- The salt is a number combined with the password prior to encryption
- The salt changes when the password changes
- The salt is stored with the password
- Different user’s with the same password see different encrypted values in /etc/passwd
- Dictionary attack requires time-consuming re-encoding of entire dictionary for every salt value

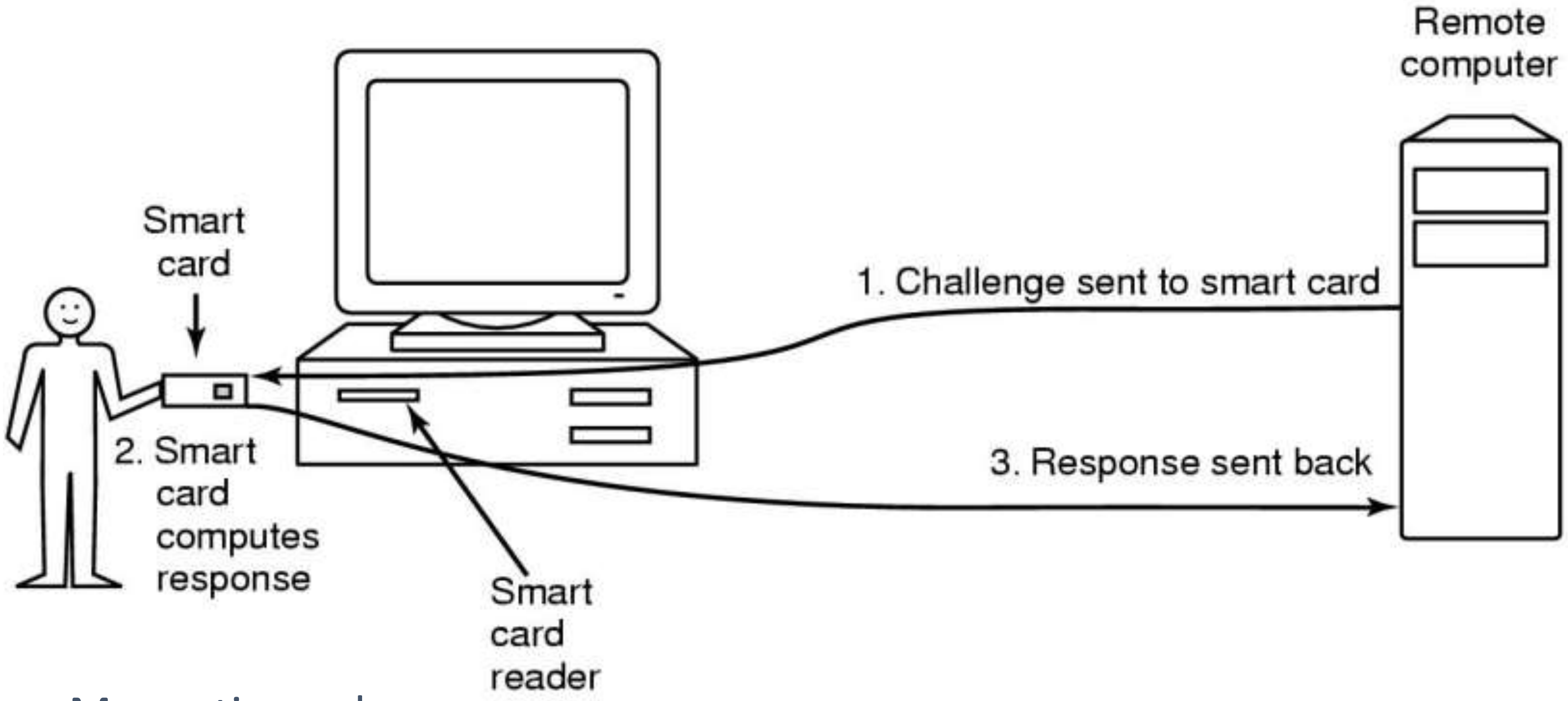
Attacking Passwords

- Guessing at the login prompt
  - *Time consuming*
  - *Only catches poorly chosen passwords*
  - *If the search space if large enough, manual guessing doesn't work*
- Automated guessing
  - *Requires dictionary to identify relevant portion of large search space*
  - *Only catches users whose password is a dictionary word, or a simple derivative of a dictionary word*
  - *But a random combination of characters in a long string is hard to remember!*
    - - If users store it somewhere it can be seen by others

General Counter Measures

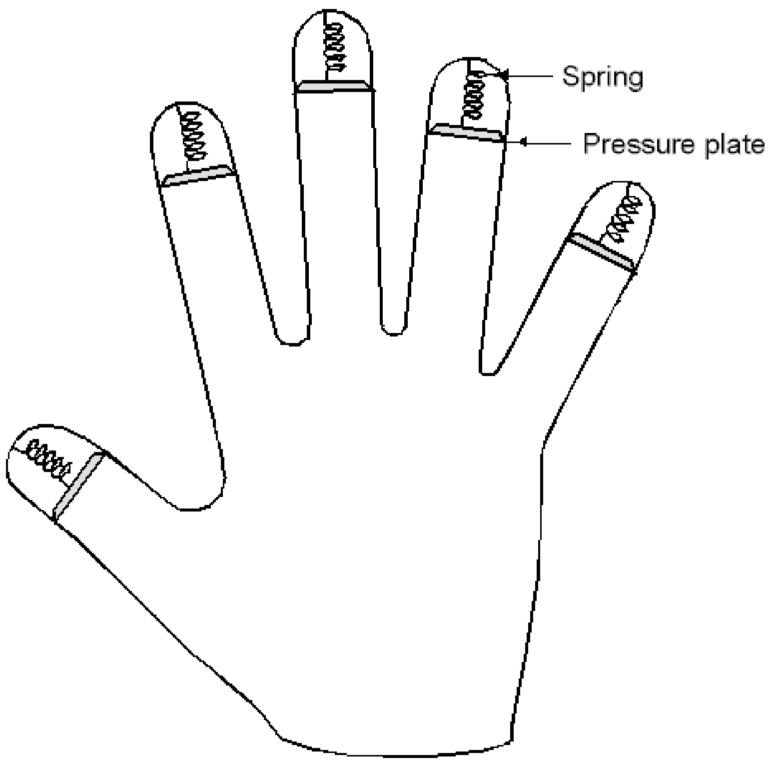
- Better passwords
  - \- No dictionary words, special characters, longer
- Don’t give up information
  - \- Login prompts or any other time
- One time passwords
  - \- Satellite driven security cards
- Limited-time passwords
  - \- Annoying but effective
- Challenge-response pairs
  - \- Ask questions
- Physical authentication combined with passwords
  - \- Perhaps combined with challenge response too

Physical Authentication



- Magnetic cards
  - \- *magnetic stripe cards*
  - \- *chip cards: stored value cards, smart cards*

Biometric Authentication

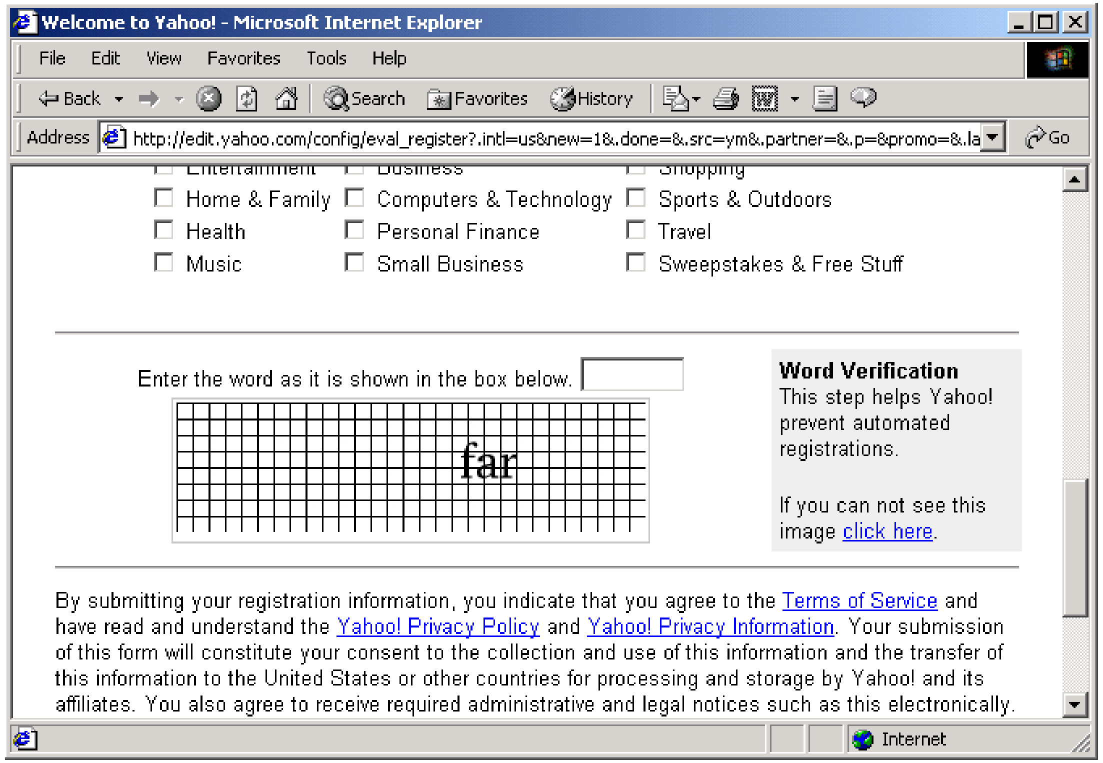


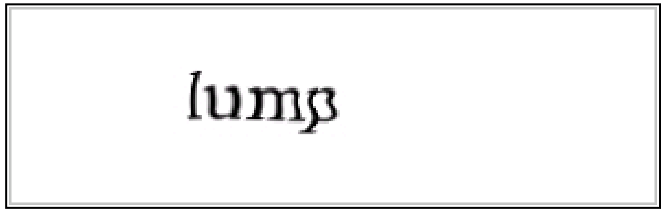
A device for measuring finger length

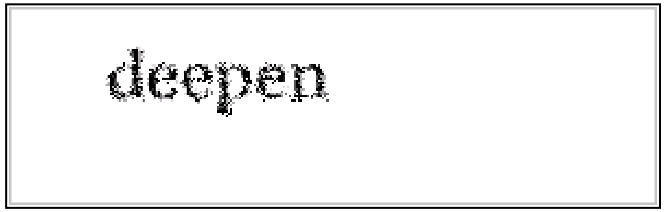
More Counter Measures

- Limiting times when someone can log in
- Automatic callback at a pre-specified number
- Limited number or frequency of login tries
- Keep a database of all logins
- Honey pot
  - *leave simple login name/password as a trap*
  - *security personnel notified when attacker bites*

Is The User Human?







PROTECTION DOMAINS

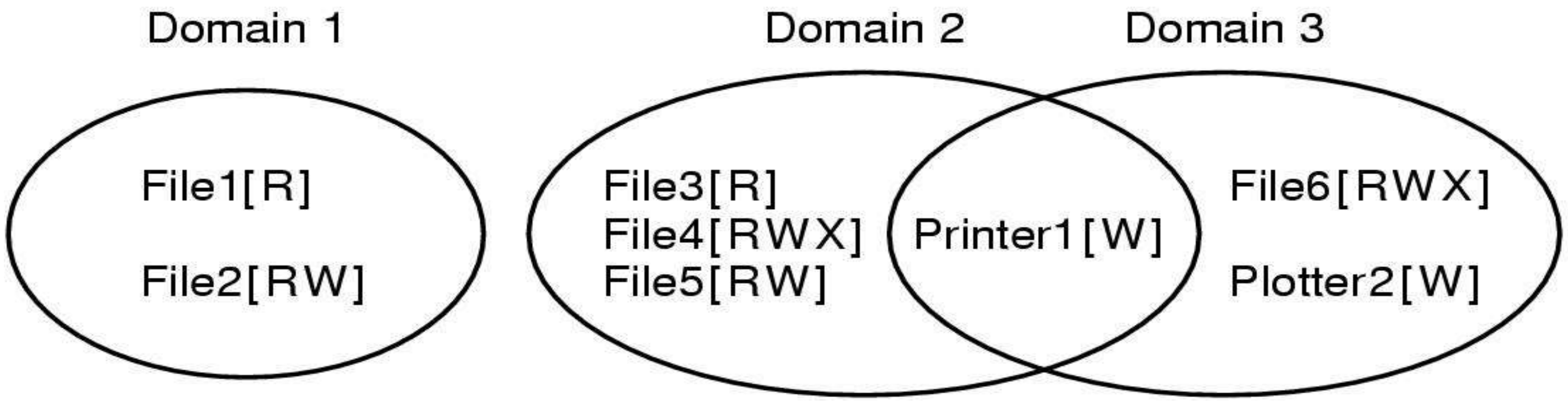
Protection Domains

- We have successfully authenticated the user, now what?
  - *For each process created we can keep track of who it belongs to*
    - \- All its activities are on behalf of this user
  - *How can we track all of its accesses to resources?*
    - \- Files, memory, devices …

Real vs Effective User Ids

- We may need mechanisms for temporarily allowing access to privileged resources in a controlled way
  - *Give user a temporary "effective user id" for the execution of a specific program*
  - *Similar concept to system calls that allow the OS to perform privileged operations on behalf of a user*
  - *A program (executable file) may have setuid root privilege associated with it*
  - *When executed by a user, that user's effective id is temporarily raised to root privilege*

Protection Domain Model



- Every process executes in some protection domain determined by its creator who is authenticated at login time

- OS mechanisms for switching protection domains
  - *System calls*
  - *Set UID capability on executable file*
  - *Re-authenticating user (su)*

Domains as Objects in The Matrix

Domain	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			

Object (column heading over File1–Domain3)

Operations may include switching to other domains

Protection Domains

- A protection matrix is just an abstract representation for allowable operations
  - *We need protection "mechanisms" to enforce the rules defined by a set of protection domains*

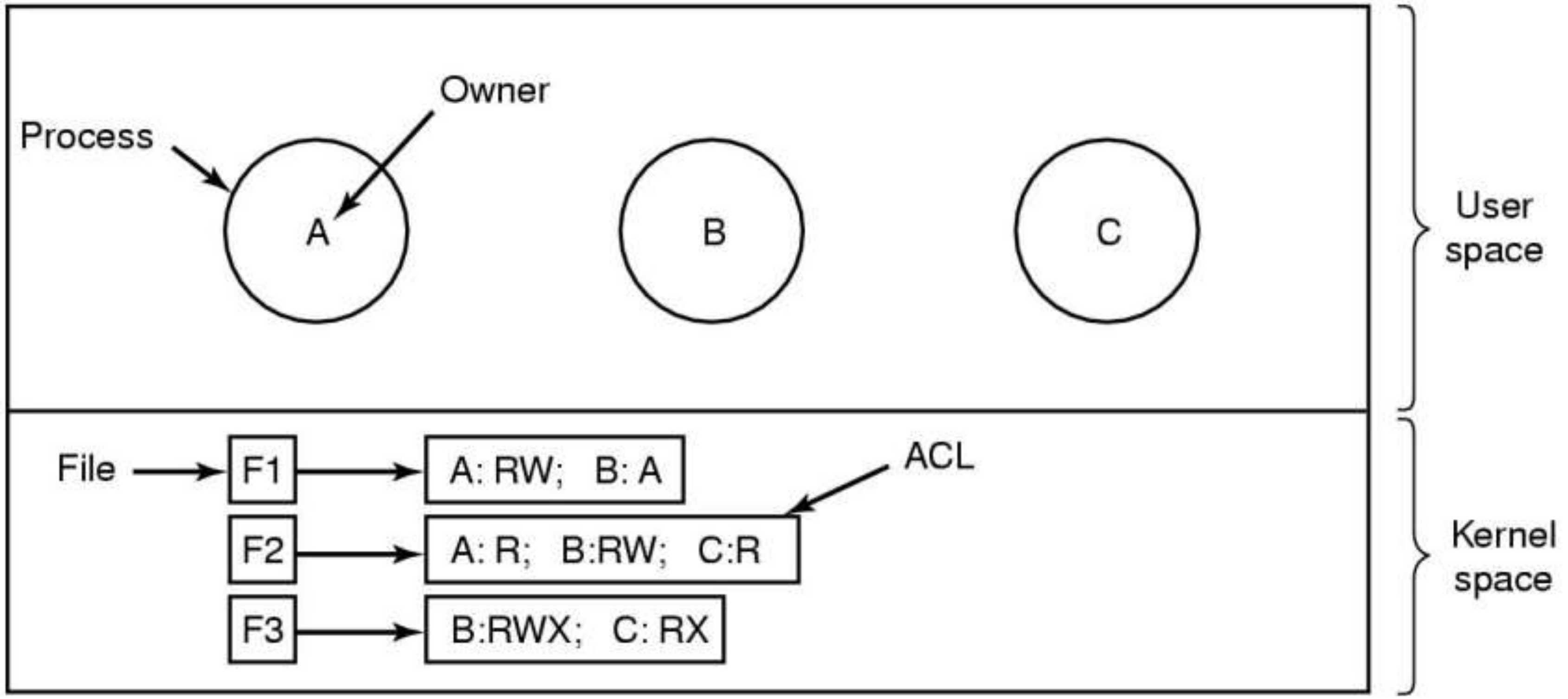
PROTECTION MECHANISMS

Access Control Lists (ACLs)

	Object										
Domain	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			

- Domain matrix is typically large and sparse
  - *inefficient to store the whole thing*
  - *store occupied columns only, with the resource? - ACLs*
  - *store occupied rows only, with the domain? - Capabilities*

Access Control Lists



Example:

User's ID stored in PCB

Access permissions stored in inodes

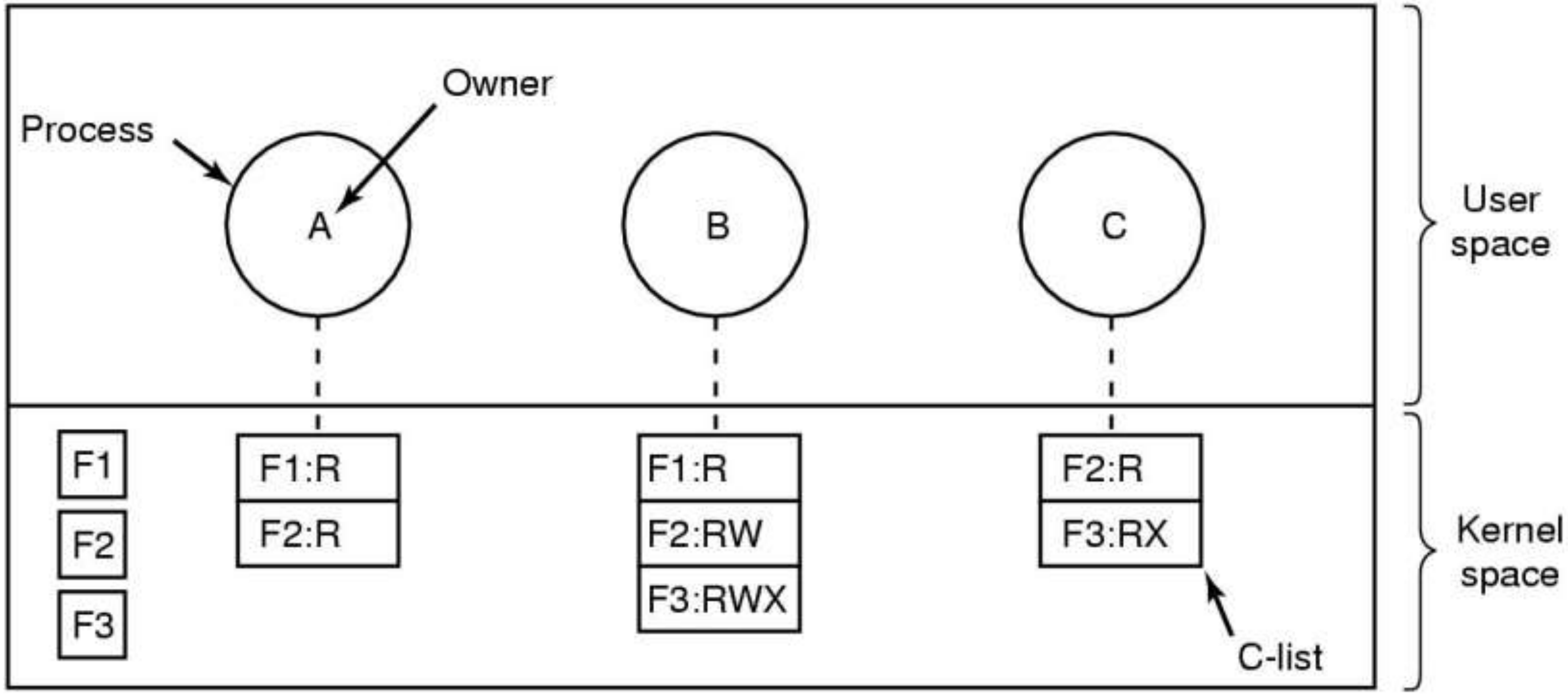
Implementing ACLs

- Problem
  - *ACLs require an entry per domain (user, role)*

- Storing on deviations from the default
  - *Default = no access*
    - High overhead for widely accessible resources
  - *Default = open access*
    - High overhead for private resources

- Uniform space requirements are desirable
  - *Unix Owner, Group, Others, RWX approach*

Process Capabilities



- Each process has a capability for every resource it can access
  - *Kept with other process meta data*
  - *Checked by the kernel on every access*

ATTACKS

Login Spoofing

(a) Correct login screen

(b) Phony login screen

Which do you prefer?


Debian GNU/Linux (xdmserver)
login:
Password:


WELCOME TO THE OGI DEPT. OF CSE
MAINTAINED BY THE COMPUTING FACILITIES GROUP
LOGIN:


Log On to Windows
Microsoft
Windows 2000
Professional
Built on NT Technology
User name:
Password:
Log on to:
Log on using dial-up connection
OK
Cancel
Shutdown...
Options <<

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*
  - *example, ls attack*
- Trick the user into executing something they shouldn’t

Logic Bombs

- Revenge driven attack
- Company programmer writes program
  - *Program includes potential to do harm*
  - *But its OK as long as he/she enters a password daily*
  - *If programmer is fired, no password and bomb "explodes"*

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)

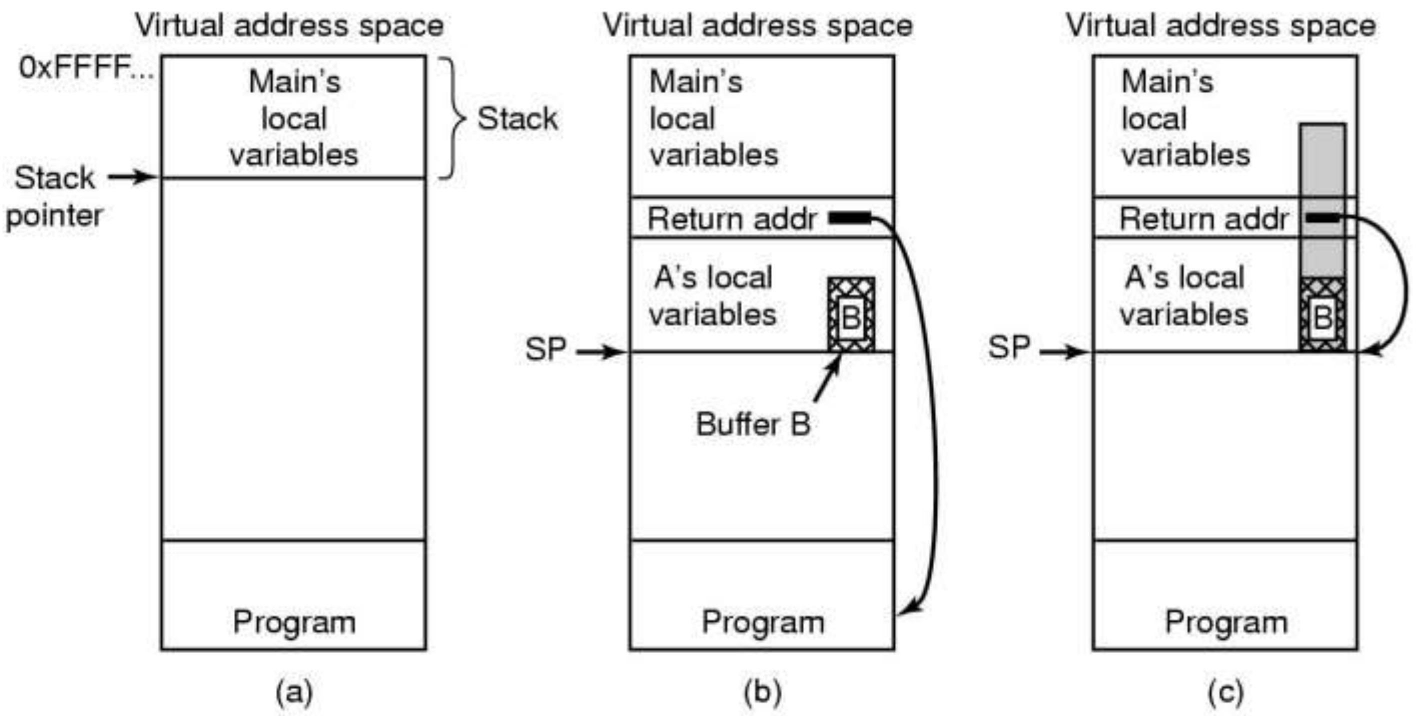
```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name);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

(b)

(a) Normal login prompt code.

(b) Login prompt code with a trapdoor inserted

Buffer Overflow Attacks



- (a) Situation when main program is running
- (b) After procedure *A* called
  - Buffer B waiting for input
- (c) Buffer overflow shown in gray
  - Buffer B overflowed after input of wrong type

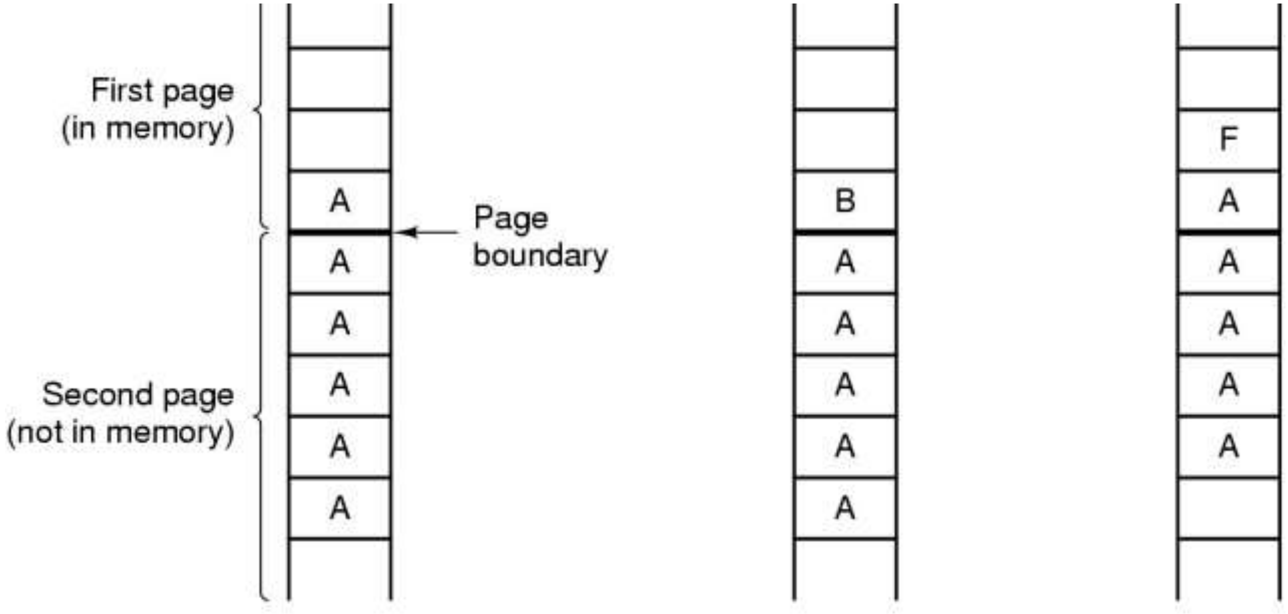
Buffer Overflow Attacks

- The basic idea
  - *exploit lack of bounds checking to overwrite return address and to insert new return address and code at that address*
  - *exploit lack of separation between stack and code (ability to execute both)*
  - *allows user (attacker) code to be placed in a set UID root process and hence executed in a more privileged protection domain !*
  - *If setuid root programs have this vulnerability (many do!).*

Other Generic Security Attacks

- Request memory, disk space, tapes and just read it
  - *Secrecy attack based on omission of zero filling on free*
- Try to do the specified DO NOTs
  - *Try illegal operations in the hope of errors in rarely executed error paths*
    - i.e, start a login and hit DEL, RUBOUT, or BREAK
- Convince a system programmer to add a trap door
- Beg someone with access to help a poor user who forgot their password

Subtle Security Flaws



- The TENEX password problem
  - *Place password across page boundary, ensure second page not in memory, and register user-level page fault handler*
  - *OS checks password one char at a time*
    - If first char incorrect, no page fault occurs
    - requires 128n tries instead of 128^n

Design Principles For Security

- System design should be public
  - *Security through obscurity doesn't work!*
- Default should be no access
- Check for "current" authority
  - *Allows access to be revoked*
- Give each process the least privilege possible
- Protection mechanism should be
  - *simple*
  - *uniform*
  - *in lowest layers of system*
- Scheme should be psychologically acceptable

External Attacks

Viruses & Worms

- External threat
  - *code transmitted to target machine*
  - *code executed there, doing damage*
  - *may utilize an internal attack to gain more privilege (ie. Buffer overflow)*

- Malware = program that can reproduce itself
  - *Virus: requires human action to propagate*
    - Typically attaches its code to another program
  - *Worm: propagates by itself*
    - Typically a stand-alone program

- Goals of malware writer
  - *quickly spreading virus/worm*
  - *difficult to detect*
  - *hard to get rid of*

Virus Damage Scenarios

- Blackmail
- Denial of service as long as malware runs
- Damage data/software/hardware
- Target a competitor's computer
  - *do harm*
  - *espionage*
- Intra-corporate dirty tricks
  - *sabotage another corporate officer's files*

How Viruses Work

- Virus written in assembly language
- Inserted into a program using a tool called a *dropper*
- Virus dormant until program executed
  - *then infects other programs*
  - *eventually executes its payload*

How Viruses Spread

- Virus is placed where its likely to be copied or executed

- When it arrives at a new machine
  - *infects programs on hard drive or portable storage*
  - *may try to spread over LAN*

- Attach to innocent looking email
  - - *when it runs, use mailing list to replicate further*

Denial of Service Attacks

- Denial of service (DoS) attacks
  - *May not be able to break into a system, but if you keep it busy enough you can tie up all its resources and prevent others from using it*
- Distributed denial of service (DDOS) attacks
  - *Involve large numbers of machines (botnet)*
- Examples of known attacks
  - Ping of death – large ping packets cause system crash
  - SYN floods – tie up buffer in establishment of TCP flows
  - UDP floods
- Some attacks are sometimes prevented by a firewall