Chapter 14: Protection

Chapter 14: Protection

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Access Control

Goals of Protection

- **Multiprogramming** environment
- Operating system consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- **Problem**: damage to integrity, reliability of interfaces
- **to resources** - ensure that each object is accessed correctly and only by those processes that are allowed to do so

Goals of Protection

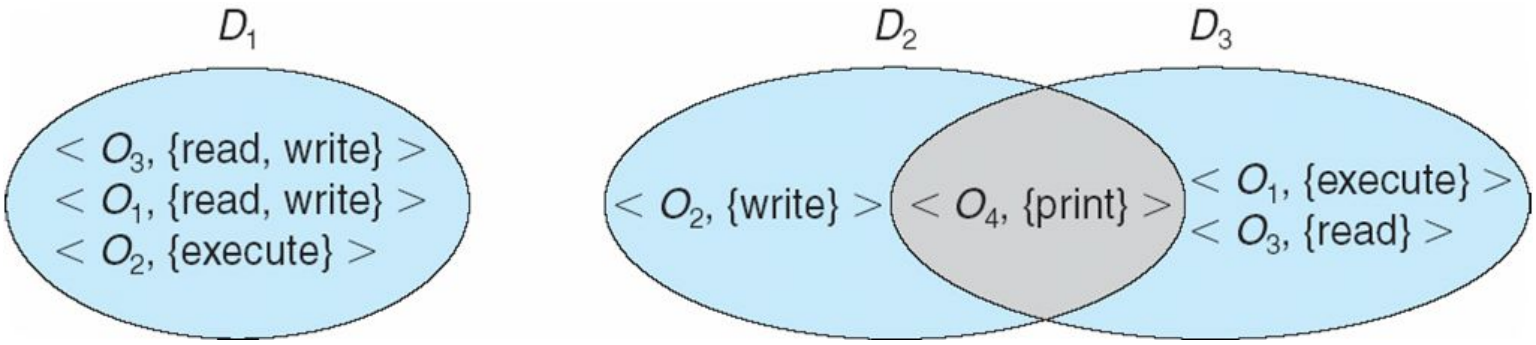
- How to protect?
  - Planning
  - Formulating access control policies
  - Mechanism to apply policy

Principles of Protection

- Guiding principle –
  - **Simplify design decisions**
    - Reduce complexity
  - **principle of least privilege**
    - Give only **just enough** privileges to perform their tasks using role/ACL.
  - **Need to know Principle**
    - **Uses fine grained access control**
      - Grant / Revoke privilege when required.
        - **Does minimum damage**
        - OS follows this principle.

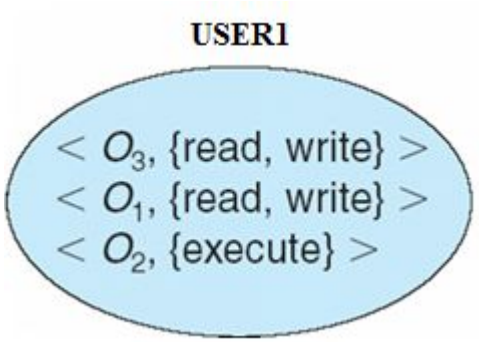
Domain Structure

- **Access-right** = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object.
- Domain = set of access-rights



- Process operates within a protection domain.
- **Association between process and domain can be static or dynamic.**

- Static
  - Set of resources will be fixed.
- Dynamic
  - Set of resources can be changed
  - Process can switch domain



- Domain ca be visualized as :
- **User as domain**
  - Objects defined by user id
  - **Domain switch occurs when user log out**
- **Process as domain**
  - Objects defined by pid
  - **Domain switching occurs when process communicate to other process using IPC.**
- **Procedure as domain**
  - Objects are local variables
  - **Domain switch occurs via procedure call**

Access Matrix

- View protection as a matrix (*access matrix*)

- **Is a mechanism to apply policies of protection**
- Rows represent domains
- Columns represent objects
- ***Access(i, j)*** is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$ (access right)
- We must know which process executes in which domain.
- Content of access matrix is decided by user while which process will be in which domain is decided by OS

Use of Access Matrix (Cont)

- Access matrix design separates mechanism from policy
  - Policy
    - User dictates policy
    - Who can access what object and in what mode
  - Mechanism
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced

Access Matrix illustration

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

4 domains and 4 objects (3 files and one printer).☐

When a process executes in D1, it can read files F1 and F3.☐

A process executing in D4 has the same privileges as it does in D1, it can also write onto files F1 and F3. ☐

Printer can be accessed by a process executing in D2.

Figure A

Process executing in domain D4 can execute read/write operation on object F1

Access Matrix

- Users decide the contents of the access-matrix entries.
- Access matrix provides mechanism for **defining and implementing strict control for both static and dynamic association between processes and domains**

- Controls changing content of access-matrix entries

- Dynamic: Controls switching between domains.
  - This can be done by including domain as an object in access matrix and switch access right.

Use of Access Matrix

- For static association:
- **For dynamic association: defining domain as object and switch access right.**

Access Matrix (static association)

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Process executing in domain D4
can execute read/write operation on object F1

Domain Switching:
Access Matrix with domains as objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure B

Process working in domain D1 can be switched to domain D2.

Access Matrix with Copy Rights

- Allowing controlled change to the contents of the access-matrix entries requires **3 additional operations**:
  - **Copy, Owner, and Control„**
- Copy
  - The ability to copy an access right from one domain (row) to another is denoted by an **asterisk (*) appended to the access right. (E.g read*)**
  - Copy right allows the copying of the access right only **within the column (that is, for the object) for which the right is defined.**

□

Access Matrix with Copy Rights

- need to update access matrix
- Special access rights: **(can change column entries)**
  - ***copy:***
    - ***Simple copy***
    - ***transfer***
    - ***Limited Copy:***

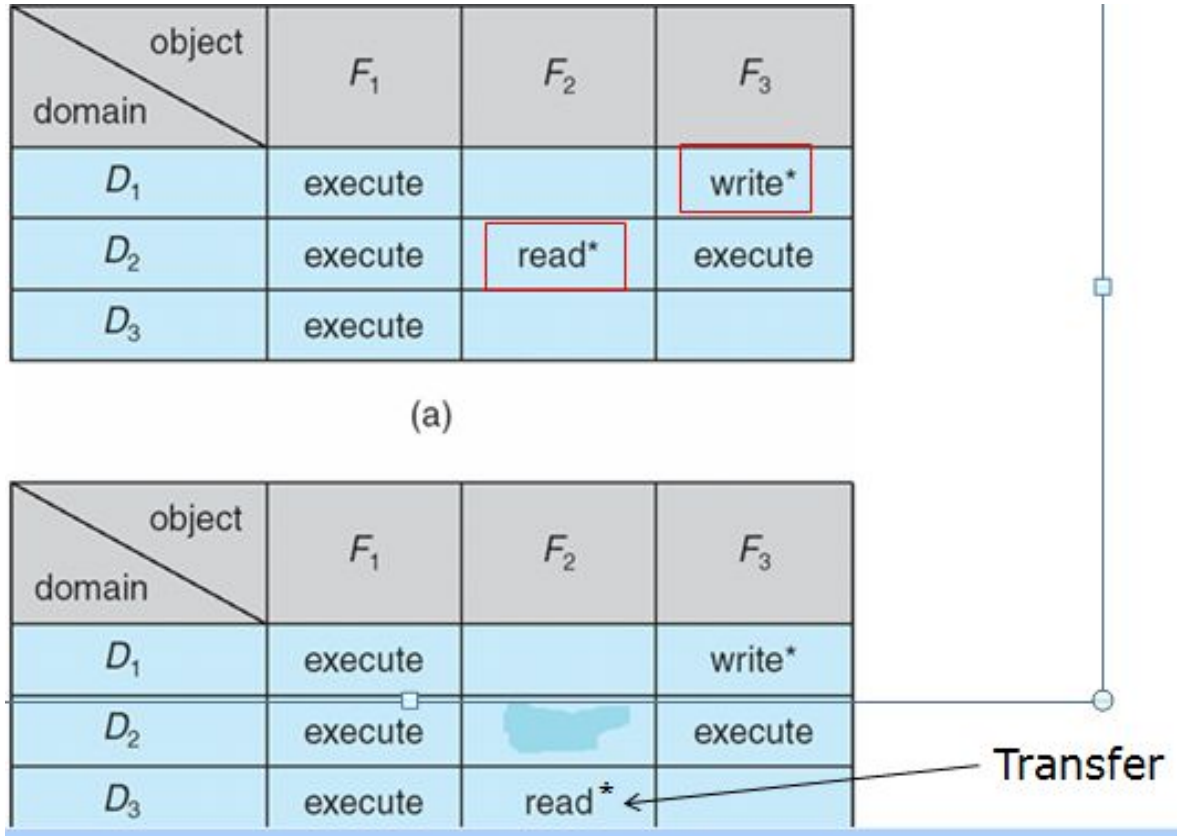
Access Matrix with Copy Rights

- ***Simple copy :*** *copy access right from Di to Dj.*
- When the right Read* is copied from access(i,j) to access(k,j), the Read* is created.So, a process executing in Dk can further copy the right Read*.

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	**read***	

Access Matrix with Copy Rights

- ***transfer*** – A right is copied from access(i,j) to access(k,j); it is then removed from access(i,j).



object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute		execute
D_3	execute	read*	

Access Matrix with *Copy* Rights

Propogation of copy right may be limited

- Limited Copy
- When the right Read* is copied from access(i,j) to access(k,j), only the Read (not Read*) is created. So, a process executing in Dkcannot further copy the right Read.

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

Limited Copy

(b)

Access Matrix With Owner Rights

- We need a mechanism to allow **addition of new rights and removal of some rights.**
- The owner right controls these operations.
- If access(i,j) includes the owner right, then a process execution in Di can add and remove any right in any entry in column j.
- „D1 is the owner of F1, and can add and delete any valid right in column F1.„
- D2 is the owner of F2 and F3, and can add and delete any valid right within these 2 columns.

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Control Access Right

- The copy and owner rights allow a process to change the entries in a column.
- So, a mechanism is needed to **change the entries in a row.**
- The control right is applicable only to domain objects (rows).
- If access(i,j) includes the control right, then a process executing in Di can remove any access right from row j.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Modified Access Matrix

We include control right in access(D2, D4).□
Then, a process executing in D2 (row) could modify D4 (row).

Implementation of Access Matrix

- 1. Global Table: whenever operation M is to be performed on Oj within Di, global table is searched for triple <Di,Oj,Rk> where M belongs to Rk.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Implementation of Access Matrix

- **2. Each column = Access-control list for one object**
  Defines who can perform what operation.

ACL for F1

Domain 1=Read
Domain 4=Read,write

ACL for F2

Domain 3=Read

ACL for F3

Domain 1 = Read
Domain 3 = Execute
Domain 4 = Read,write

ACL for Printer

Domain D2=Print

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

- **Each Row = Capability List (like a key)**
  For each domain, what operations allowed on what objects.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Capability List for D1

F1-read
F3-read

Capability List for D2

Printer-print

Capability List for D3

F2-read
F3-execute

Capability List for D4

F1-read,write
F3-read,write

Implementation of Access Matrix

- **Lock-Key mechanism**
  - Each **object** has unique bit pattern called **lock**
  - Each **domain** has unique bit pattern called **key**
  - Process executing in a domain can access object if it has key that matches one of the lock of an object

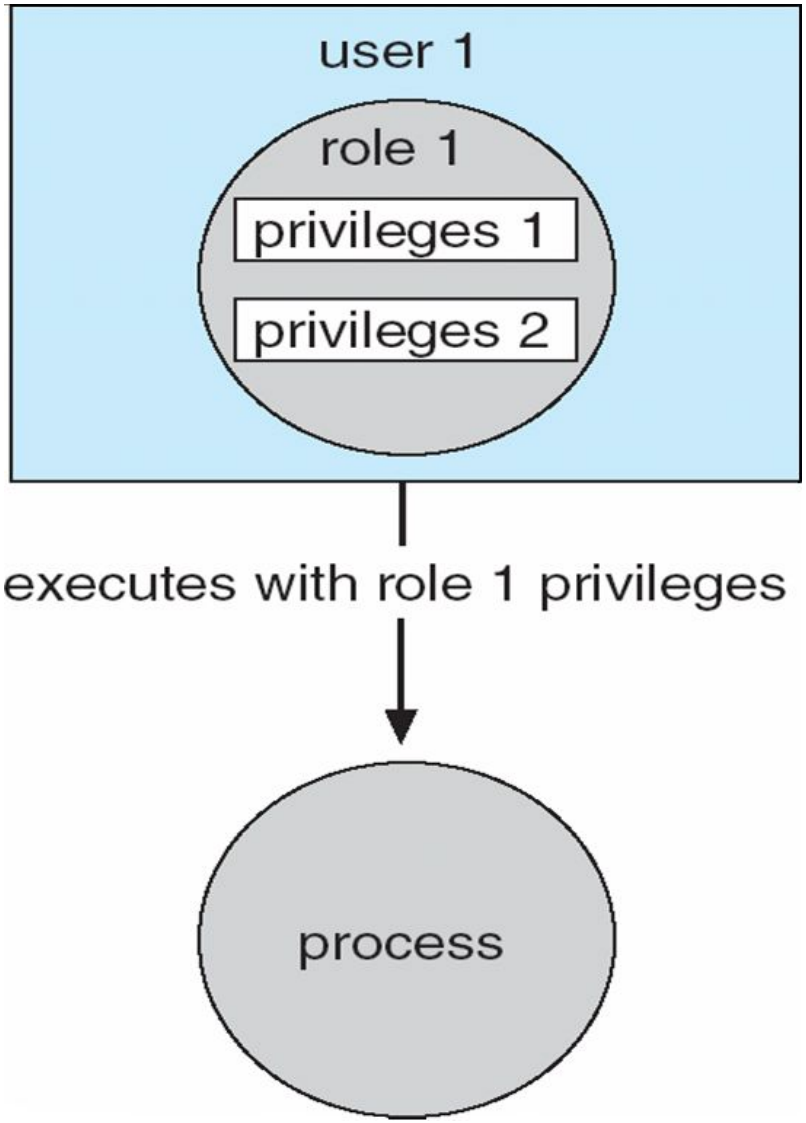
Access Control

- Privilege can be assigned to owner, group, or other users for a file or directory.
- Role based access control.

Access Control (RBAC in Solaris 10)

- Protection can be applied to non-file resources
- Solaris 10 provides role-based access control (**RBAC**) to implement least privilege
  - Privilege is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned roles granting access to privileges and programs

Role-based Access Control in Solaris 10



Revocation of Privileges & issues

- Removing privileges from the process:
- Types:
- **Immediate / delayed**
  - If delayed when to be revoked?
- **Selective / general**
  - Revoked from selected / from all?
- **Total / partial**
  - Revoked all / some privileges?
- **Permanent / temporary**
  - Revocation permanent / temporary ?

Revocation for ACL / capability

- **Access control list**: Revocation or deletion of privilege is easy
- But difficult in **capability list**
  - Problem of **reacquisition**
  - Some systems use
    - **back pointers** (pointers to capability is checked while revocation)
    - **Indirect method**: pointer to capability via global table
    - **Key:** Assigning master key to object and generating key while creating capability . If capability key matched with master key access is allowed.

Chapter 15: Security

Chapter 15: Security

- The Security Problem
- Program Threats
- System and Network Threats
- Cryptography as a Security Tool
- User Authentication
- Implementing Security Defenses
- Firewalling to Protect Systems and Networks
- Computer-Security Classifications
- An Example: Windows XP

Objectives

- To discuss security threats and attacks
- To explain the fundamentals of encryption, authentication, and hashing
- To examine the uses of cryptography in computing
- To describe the various countermeasures to security attacks

The Security Problem

- Security must consider **external environment** of the system, and protect the system resources
- **Threat** is potential **security violation**
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

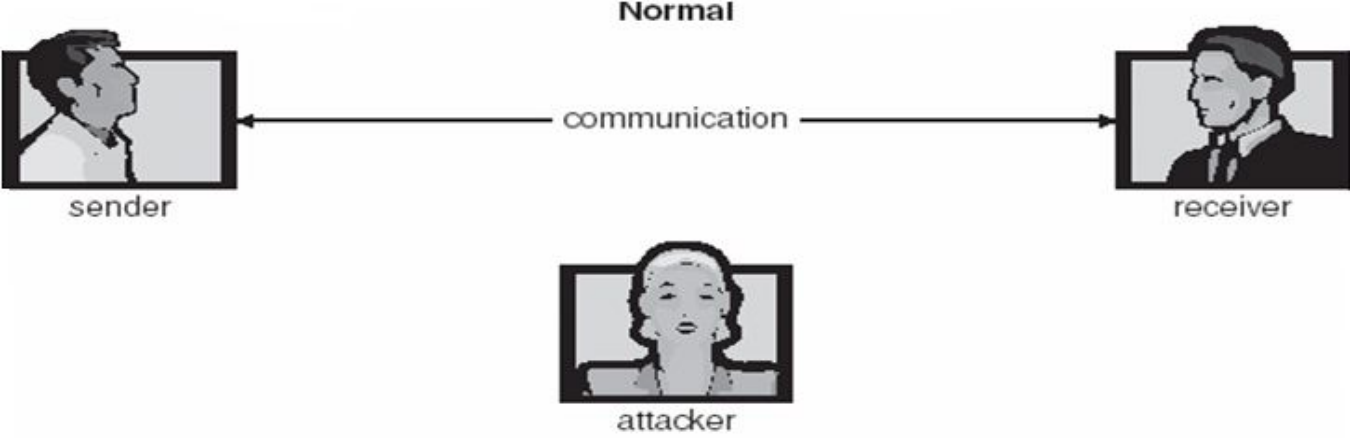
Security Violations: Threat

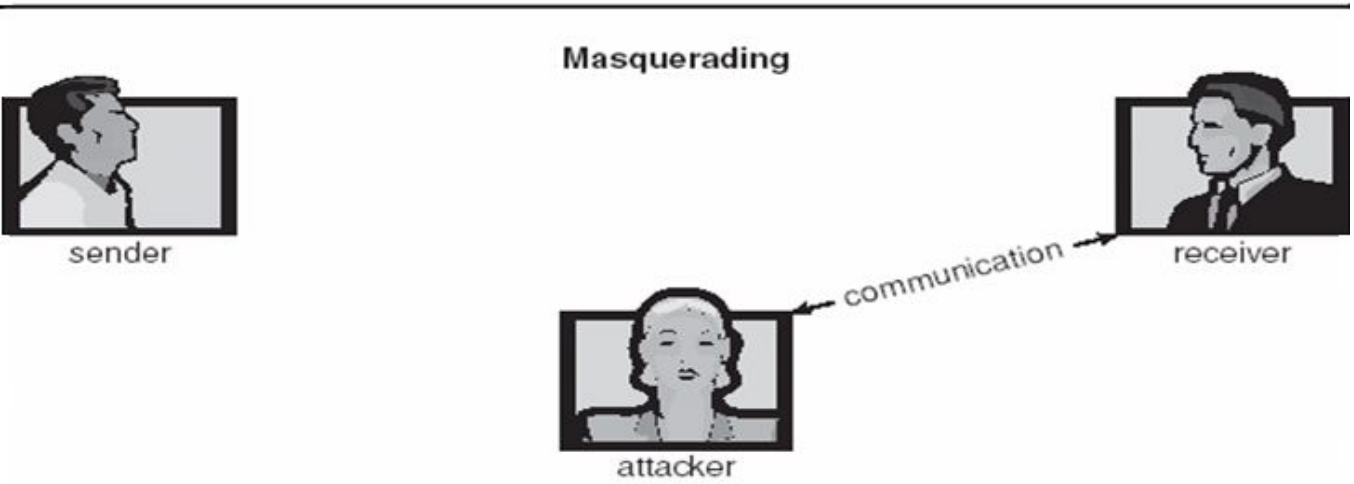
- **Categories**
  - **Breach of confidentiality: unauthorized access of data**
  - **Breach of integrity: unauthorized modification of data**
  - **Breach of availability: unauthorized destruction of data**
  - **Theft of service: unauthorized use of system**
  - **Denial of service (DOS): prevent legitimate use of service.**

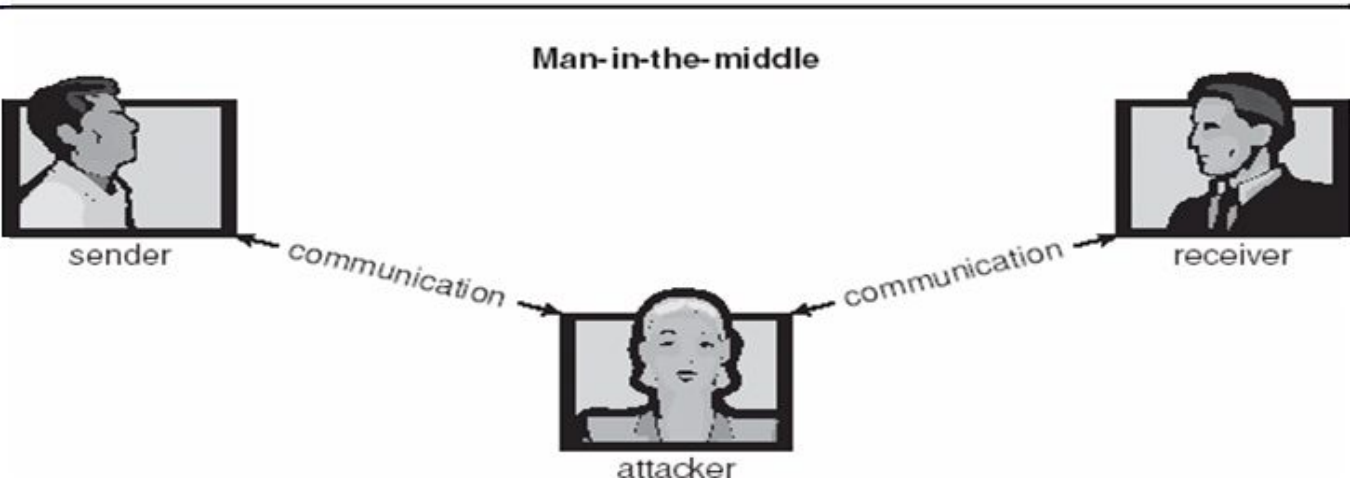
Security Violations: Threat

- **Methods to break security:**
  - **Masquerading (breach authentication)**
    - **One of the person hide identification**
    - **Obtain privileges to which they are not entitled.**
  - **Replay attack**
    - **Message modification(transfer of money,authentication)**
  - **Man-in-the-middle attack**
  - **Session hijacking**

Standard Security Attacks







Security Measure Levels

- Security must occur at four levels to be effective:
  - **Physical : alarm system, lock & key**
  - **Human**
    - Avoid **social engineering**
      - **Phishing**: access to mail web page
      - **Dumpster diving**: gathering info from trash, notes
  - **Operating System**
    - Password
    - Antivirus
    - Password to requested service, stack overflow
    - Require physical, human security.
    - Need of protection
  - **Network: leased line / shared / wifi**
- Security is as weak as the **weakest link** in the chain

Threat Types

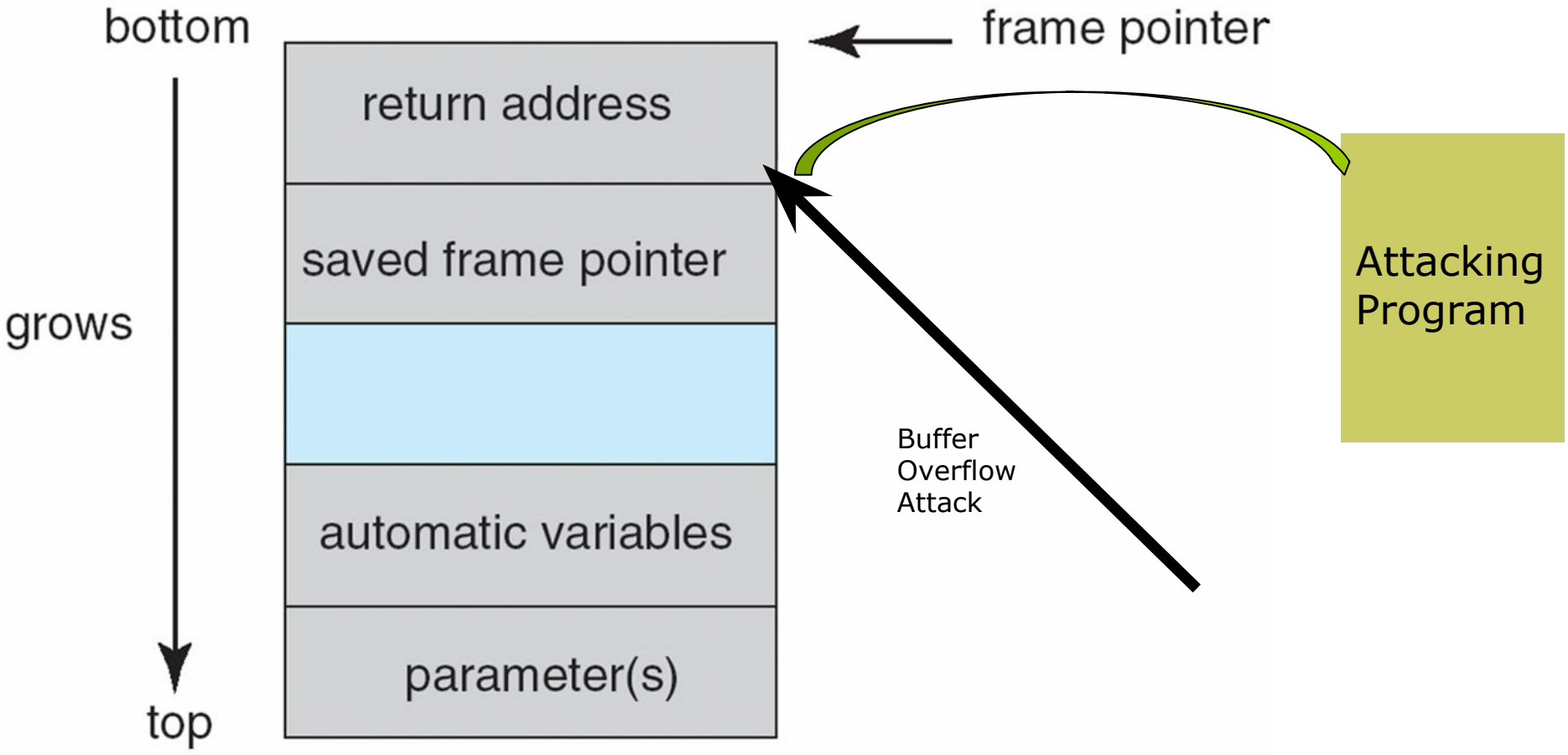
- Program Threats
  - Trojan horse
  - Trap door
  - Logic bomb
  - Stack and buffer overflow
  - viruses
- System and Network threats
  - Worms
  - Port scanning
  - Denial of service attack

Program Threats

- **Trojan Horse**
  - Code segment that **misuses user environment**
  - **programs written by users are executed by other users, thus misusing access privileges.**
  - **Spyware in free s/w, pop-up browser windows, login emulator**
- **Trap Door**
  - Program designer intentionally include a code in users system that only he will be capable of using.
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
- **Logic Bomb**
  - Program that creates security hole under certain circumstances

- **Stack** and **Buffer Overflow**
  - **Poor programming. E.g. program without data validation.**
  - **Attacker sends more data than program was expecting.**
  - Exploits a bug in a program (overflow either the stack or memory buffers)

Layout of Typical Stack Frame



C Program with Buffer-overflow Condition

```
#include <stdio.h>
#define BUFFER SIZE 256
int main(int argc, char *argv[])
{
   char buffer[BUFFER SIZE];
   if (argc < 2)
    return -1;
   else {
    strcpy(buffer,argv[1]);
    return 0;
   }
}
```

No validation.

Modified Shell Code

Attacker take advantage of extra permissions given to others and group users.

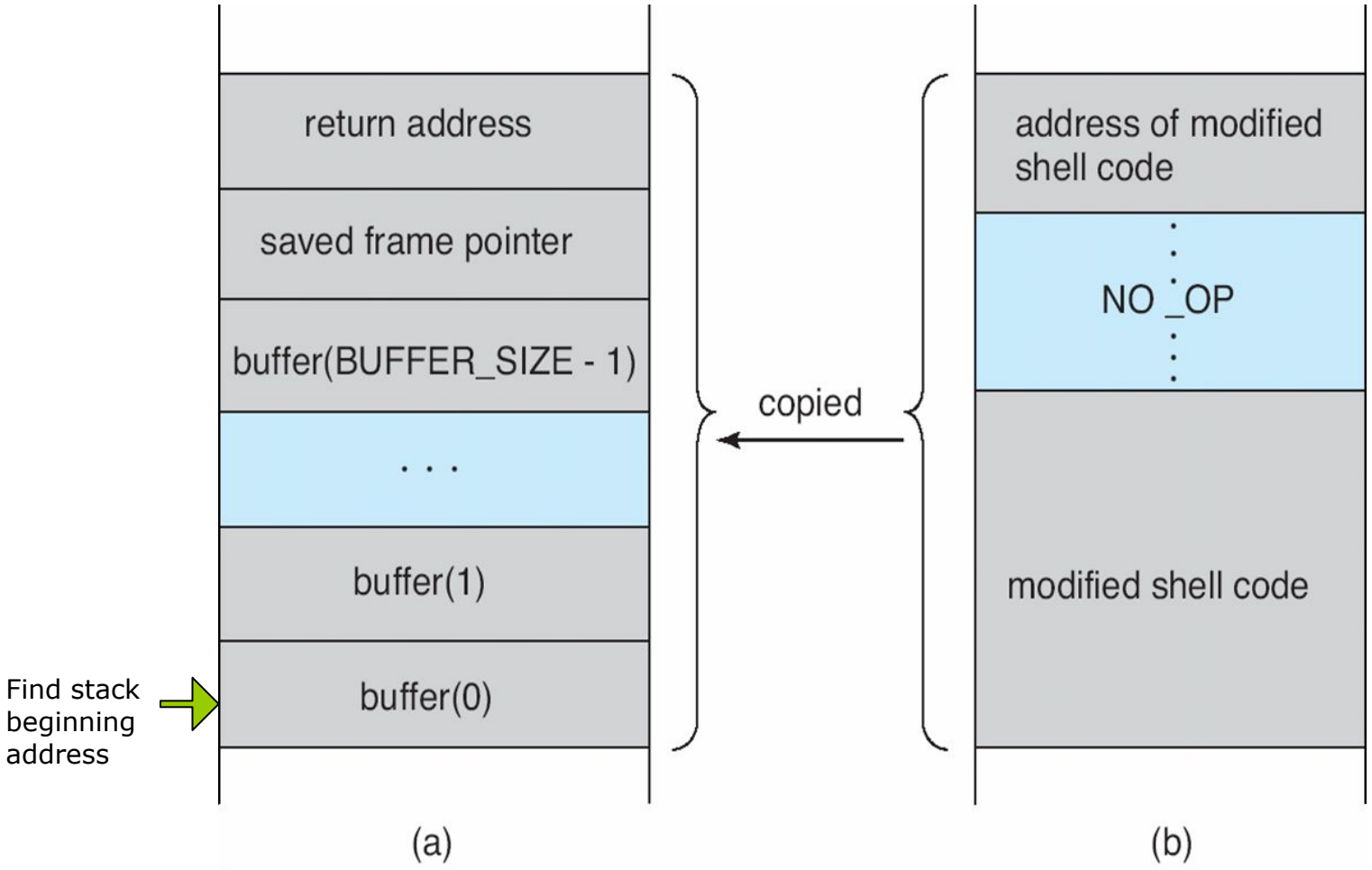
Find out loopholes in a program, such as not using data validations

```
#include <stdio.h>
// creation of new shell process
int main(int argc, char *argv[])
{
   execvp(''\bin\sh'',''\bin \sh'', NULL);
   return 0;
}
```

New shell created by attacker

- Create a new shell
- Reduce the size of code so that it can be accommodated in a stack.
- identify stack frame pointer, using debugger.
- Change return address of the code.
- Compile and create binary file
- This changed file is then given as i/p to process.

Hypothetical Stack Frame



Main() called

Before attack

After attack

Program Threats (Cont.)

- Viruses
  - Code fragment **embedded in legitimate program**
  - **Written in VB**
  - Very **specific to CPU architecture, operating system, applications**
  - Usually borne **via email or as a macro**

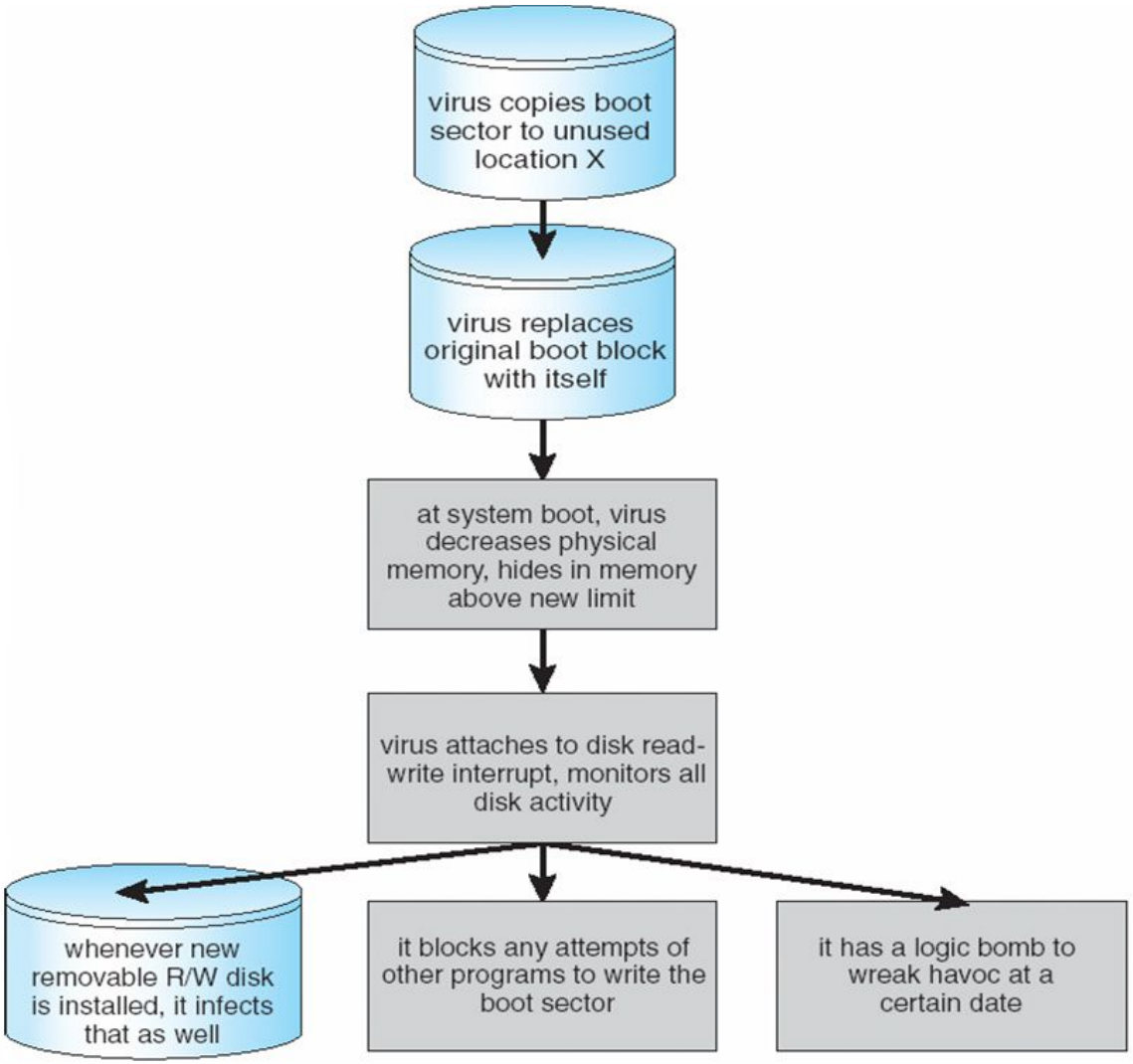
Program Threats (Cont.)

- **Virus dropper** inserts virus onto the system
- Many categories of viruses, literally many thousands of viruses
  - File
    - Parasitic virus
    - Appended to a program file
    - Change execution startup
    - After execution control returns to main program
  - Boot (memory virus)
    - Found in boot sector
    - Executed at every booting process
    - Attack on bootable devices

- Macro
  - Written in VB
  - Executed while executing macro
  - Found in word, excel files
- Source code
  - Search for source program
  - **Modifies the code**
- Polymorphic
  - Follow different steps while installation
  - So difficult to detect
- Encrypted
  - Found in encrypted form
  - Virus code itself contain decrypted code.

- Stealth
  - Attack on **parts of system that are used for virus detection**
  - E.g read system call
- Tunneling
  - Inject itself in interrupt-handlers chain
  - Thus **bypass by antivirus**
- Multipartite
  - Affect **multiple parts of system**
- Armored
  - Can be found in encrypted form
  - Virus dropper, related files are hidden
- Browser virus
  - Uses keystroke logger.

A Boot-sector Computer Virus



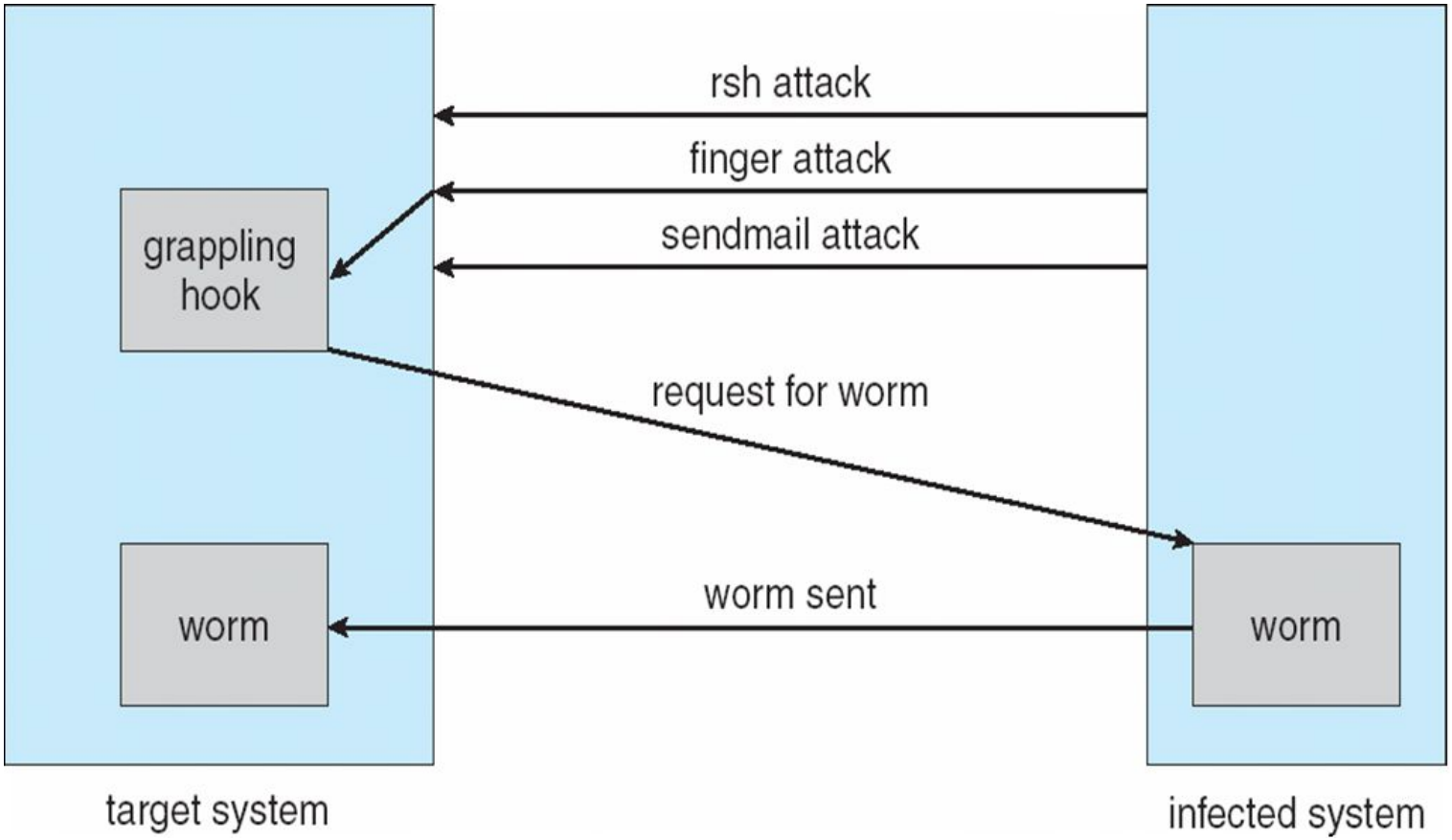
System and Network Threats

- **Worms** – standalone program
  - use **spawn (reproduction)** mechanism, use system resources and lock out all other processes to damage system;
  - Mainly attack on network thus shut down an entire network.
  - Robert Morris used this attack

- **Attack on remote access feature and bugs in *finger* and *sendmail* programs.**

- **It is made up of two programs**
  - **Grappling hook (also called bootstrap or vector)**
  - **Main program (*l*1.c)**

The Morris Internet Worm



- **rsh** : user can omit entering a password each time they access remote account by configuring some special files. Worm search these files for sites that allow remote execution without password.
- **Finger** : users can provide their personal information. Attacker uses buffer overflow attack on finger.
- **Sendmail** : worm discover user passwords.

- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses to identify holes in a system
- **Denial of Service**
  - Network based attack
  - Disrupting legitimate use of service.
  - Attack categories
    - Attack uses maximum resources so that no work can be done.
    - E.g web site click download java applet, uses all CPU time, or pop up window indefinitely.
    - Abuse of TCP/IP connection
    - Request is send to target machine
    - Connection completed
    - But no connection standard followed.
  - Distributed denial-of-service (DDOS) come from multiple sites at once

User Authentication

- Checking validity of user
- Can be done using three things
  - User's possession of something (key / card)
  - Users knowledge of something (id / password)
  - User's attribute (finger print / face / retina pattern / signature)

User Authentication : Passwords

- User identity most often established through *passwords*.
- *Password can be also given to each resource. E.f. file*
- *Different passords may be associated with different access rights. (read / write / read-write)*
- Password Vulnerabilities
  - Password guessing
    - User selected password
      - Personal information
      - Trying enumerations of characters (use long password)
  - Shoulder surfing
  - Monitoring network (n/w sniffing)
  - Password exposure
    - System selected password is hard to guess.
  - Sharing user id

- Passwords must be kept secret

Password

- Some systems accept only **strong password**
- Some systems forces to **change password regularly** / at the end of each session.
- Users can toggle password. **System records N passwords and do not allow user to use that password again.**

Encrypted Password (e.g Unix)

- User created simple password or system generated hard password can be theft easily.
- Password can be secured using encryption. e.g Unix.
- In Unix a function f(x) uses an encryption algorithm to encode password x. But identifying x from f(x) is extremely difficult.
- Random numbers are used to generate password.
- Even if password is same the “salt” generates different password.
- stored encrypted password file is under control of super-user only.

Implementing Security Defenses

- **Defense in depth** is most common security theory – multiple layers of security
- **Security policy describes what is being secured**
- Vulnerability assessment compares real state of system / network compared to security policy
- Intrusion detection endeavors to detect attempted or successful intrusions
  - **Signature-based** detection spots known bad patterns
  - **Anomaly detection** spots differences from normal behavior
    - Can detect **zero-day** attacks
  - **False-positives** and **false-negatives** a problem
- Virus protection
- Auditing, accounting, and logging of all or specific system or network activities

Firewalling to Protect Systems and Networks

- A network firewall is placed between trusted and untrusted hosts
  - The firewall limits network access between these two security domains
- Can be tunneled or spoofed
  - Tunneling allows disallowed protocol to travel within allowed protocol (i.e., telnet inside of HTTP)
  - Firewall rules typically based on host name or IP address which can be spoofed
- **Personal firewall** is software layer on given host
  - Can monitor / limit traffic to and from the host
- **Application proxy firewall** understands application protocol and can control them (i.e., SMTP)
- **System-call firewall** monitors all important system calls and apply rules to them (i.e., this program can execute that system call)