# System Protection

**Reading: Chapter14 of Textbook**

How to control the access of programs, processes, or users to the resources defined by a computer system?

How to ensure that each object is accessed correctly and only by those processes that are allowed to do so?

How to distinguish between authorized and unauthorized usage?

*The solution to all these issues is to maintain the system protection.*

# Protection Principles

A key, guiding principle for protection is the – *principle of least privilege* - Programs, users and systems should be given just enough *privileges* to perform their tasks

- Limits damage if entity has a bug, gets abused (misused)
- Can be static (during life of system, during life of process) or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**

Must consider "grain" aspect

Rough-grained privilege management easier, simpler, but least privilege now done in large chunks

For example, traditional Unix processes either have abilities of the associated user, or of root

Fine-grained management more complex, more overhead, but more protective

File ACL lists, RBAC (role-based-access-control)

# Protection Domain

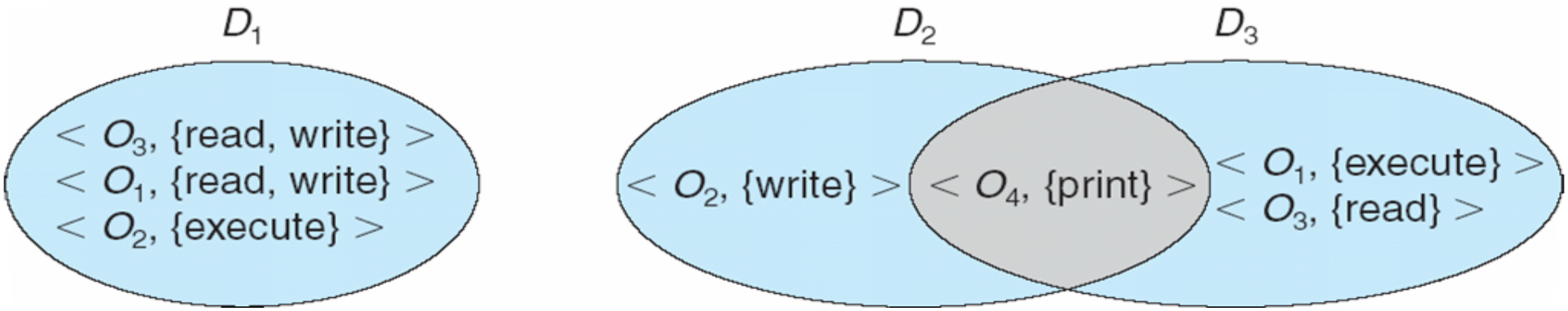
*Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations*

Protection Domain specifies the resources that the process may access. Each domain defines a set of objects and the types of operations that may be invoked on each object. The ability to execute an operation on an object is an *access right*

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

A process can move from protection domain to protection domain so, at any point, it has exactly the abilities it needs for the current job (the principle of least privilege)




System with three protection domain

# Protection Domain Example: UNIX

In UNIX, a domain is associated with the user, Domain = user-id, switching domain => changing user-id

Domain switch accomplished via file system

- Each file has associated with a domain bit (setuid bit)
- When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset. When bit is *off* the user-id does not change.
- e.g: User A (user-id =A) starts executing file owned by B with setuid = on, then user-id = B, if setuid = off then user-id =A

Domain switch accomplished via passwords

- `su` command temporarily switches to another user's domain when other domain's password provided

Domain switching via commands

- `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

# Access Matrix

- View protection as a matrix (*access matrix – a conceptual representation*)
- Rows represent domains and Columns represent objects
- Each entry, *Access(i, j),* is set of operations that process executing in $Domain_i$ can invoke on $Object_j$

| domain \ object | $F_1$ | $F_2$ | $F_3$ | printer |
|---|---|---|---|---|
| $D_1$ | read | | read | |
| $D_2$ | | | | print |
| $D_3$ | | read | execute | |
| $D_4$ | read<br>write | | read<br>write | |

# Use of Access Matrix

If a process in Domain $D_i$ tries to do "op" on object $O_j$, then "op" must be in the access matrix

User who creates object can define access column for that object

Example: 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 | | | |

# Unix/Linux Matrix

System comprises many domains:–

Each user

Each group

Kernel/System

| | *file1* | *file 2* | *file 3* | *device* | *domain* |
|---|---|---|---|---|---|
| *User/Domain 1* | *r* | *rx* | *rwx* | – | *enter* |
| *User/Domain 2* | *r* | *x* | *rx* | *rwx* | – |
| *User/Domain 3* | *rw* | – | – | – | – |
| ... | | | | | |

Columns are *access control lists* (ACLs)

Associated with each object

Rows are *capabilities*

Associated with each user or each domain

# Implementation Issues of Access Matrix

At run-time…

- What does the OS know about the user?
- What does the OS know about the resources?

What is the cost of checking and enforcing?

- Access to the data
- Cost of searching for a match

Impractical to implement full Access Matrix

- Size
- Access controls disjoint from both objects and domains

# Implementation of Access Matrix

Generally, a sparse matrix, i.e. most of the entries will be empty.
There are several methods of implementation of matrix

Option 1 – Global table – simplest method

Store ordered triples < *domain, object, rights-set* > in table. A requested operation M on object $O_j$ within domain $D_i$ -> search global table for < $D_i$, $O_j$, $R_k$ > with M ∈ $R_k$, if triple is found, the operation is allowed to continue.
But table could be large -> won't fit in main memory
Difficult to group objects (consider an object that all domains can read)

Option 2 – Access lists for objects

Each column implemented as an access list for one object. Resulting per-object list consists of ordered pairs < *domain, rights-set* > defining all domains with non-empty set of access rights for the object
Easily extended to contain default set -> If operation M ∈ default set, also allow access
For efficiency, first check default set and check for access list.

# Implementation of Access Matrix

## Option 3 – Capability list for domains

- Instead of object-based, list is domain based. *Capability list* for domain is list of objects together with operations allows on them
- Object represented by its name or address, called a *capability*
- Execute operation M on object $O_j$, process requests operation and specifies capability as parameter
  - Possession of capability means access is allowed
- Capability list associated with domain but never directly accessible by domain
  - Rather, protected object, maintained by OS and accessed indirectly
  - Like a "secure pointer"
  - Idea can be extended up to applications

## Option 4 – Lock-key

- Compromise between access lists and capability lists
- Each object has list of unique bit patterns, called *locks*
- Each domain as list of unique bit patterns called *keys*
- Process in a domain can only access object if domain has key that matches one of the locks

# Comparison of Implementations

Many trade-offs to consider

- Global table is simple, but can be large
- Access lists correspond to needs of users
  - Determining set of access rights for domain non-localized so difficult
  - Every access to an object must be checked
  - Many objects and access rights -> slow
- Capability lists useful for localizing information for a given process
  - But revocation capabilities can be inefficient
- Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

*Most systems use combination of access lists and capabilities*

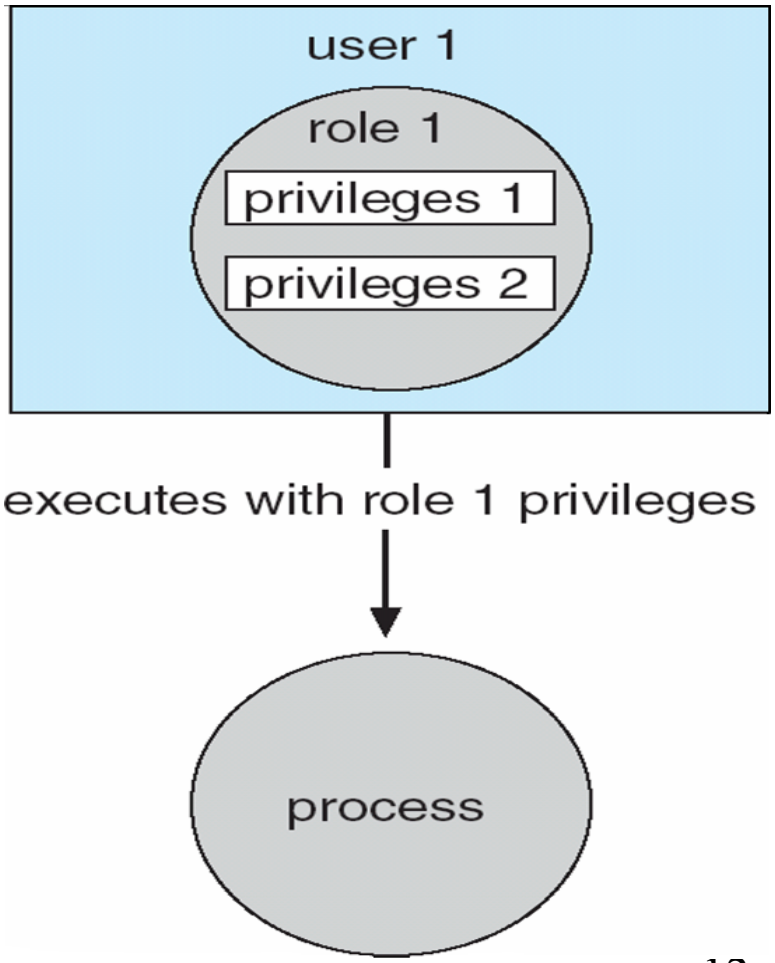
- First access to an object -> access list searched
  - If allowed, capability created and attached to process
  - Additional accesses need not be checked
  - After last access, capability destroyed
  - Consider file system with ACLs per file

# Access Control

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
Enable role via password to gain its privileges




Similar to access matrix

# Revocation of Access Rights

Various questions to remove the access right of a domain to an object
- Immediate vs. delayed?
- Selective vs. general?
- Partial vs. total?
- Temporary vs. permanent

**Access List** – Delete access rights from access list
- Simple – search access list and remove entry
- Revocation is immediate, general or selective, total or partial, permanent or temporary

**Capability List** – Scheme required to locate capability in the system before capability can be revoked
- Reacquisition – periodic delete, with require and denial if revoked
- Back-pointers – set of pointers from each object to all capabilities of that object (Multics)
- Indirection – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
- Keys – unique bits associated with capability, generated when capability created
  - Master key associated with object, key matches master key for access
  - Revocation – create new master key
  - Policy decision of who can create and modify keys – object owner or others?

# Capability-Based Systems

A survey of two capability-based protection systems

Hydra - Fixed set of access rights known to and interpreted by the system

i.e. read, write, or execute each memory segment

User can declare other *auxiliary rights* and register those with protection system

Accessing process must hold capability and know name of operation

*Rights amplification* allowed by trustworthy procedures for a specific type

Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights

Operations on objects defined procedurally – procedures are objects accessed indirectly by capabilities

Solves the *problem of mutually suspicious subsystems*

Includes library of prewritten security routines

Cambridge CAP System - Simpler but powerful

*Data capability* - provides standard read, write, execute of individual storage segments associated with object – implemented in microcode

*Software capability* -interpretation left to the subsystem, through its protected procedures

Only has access to its own subsystem

Programmers must learn principles and techniques of protection

# Language-Based Protection

Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources

Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable

Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

# Protection in Java 2

Protection is handled by the Java Virtual Machine (JVM)

A class is assigned a protection domain when it is loaded by the JVM

The protection domain indicates what operations the class can (and cannot) perform

If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library

Stack Inspection

| protection domain: | untrusted applet | URL loader | networking |
|---|---|---|---|
| socket permission: | none | *.lucent.com:80, connect | any |
| class: | gui:<br>...<br>get(url);<br>open(addr);<br>... | get(URL u):<br>...<br>doPrivileged {<br>open('proxy.lucent.com:80');<br>}<br><request u from proxy><br>... | open(Addr a):<br>...<br>checkPermission (a, connect);<br>connect (a);<br>... |

# Home Works

**HW #12**

Q. No. 14.1, 14.2, 14.3, 14.4, 14.7, 14.10,14.11 of book

# System Security

**Reading: Chapter 15**

What are the security threats and attacks?

What are the security and authentication tools used in OS and how they are implemented?

What are the mechanism to guard or detect attacks

# Security Issues

Defense of the system against internal and external attacks.
Security violations (or misuse) of the system can be *intentional (malicious)* or *accidental*

The terms *Intruder* and *cracker* used for those attempting to breach security and a *threat* is the potential for a security violation, such as the discovery of a vulnerability, whereas an *attack* is the attempt to break security.

Forms of security violation:

- Breach of confidentiality - Unauthorized reading of data (or theft of information)
- Breach of integrity - Unauthorized modification of data
- Breach of availability - Unauthorized destruction of data
- Theft of service - Unauthorized use of resources
- Denial of service (DOS) - Prevention of legitimate use

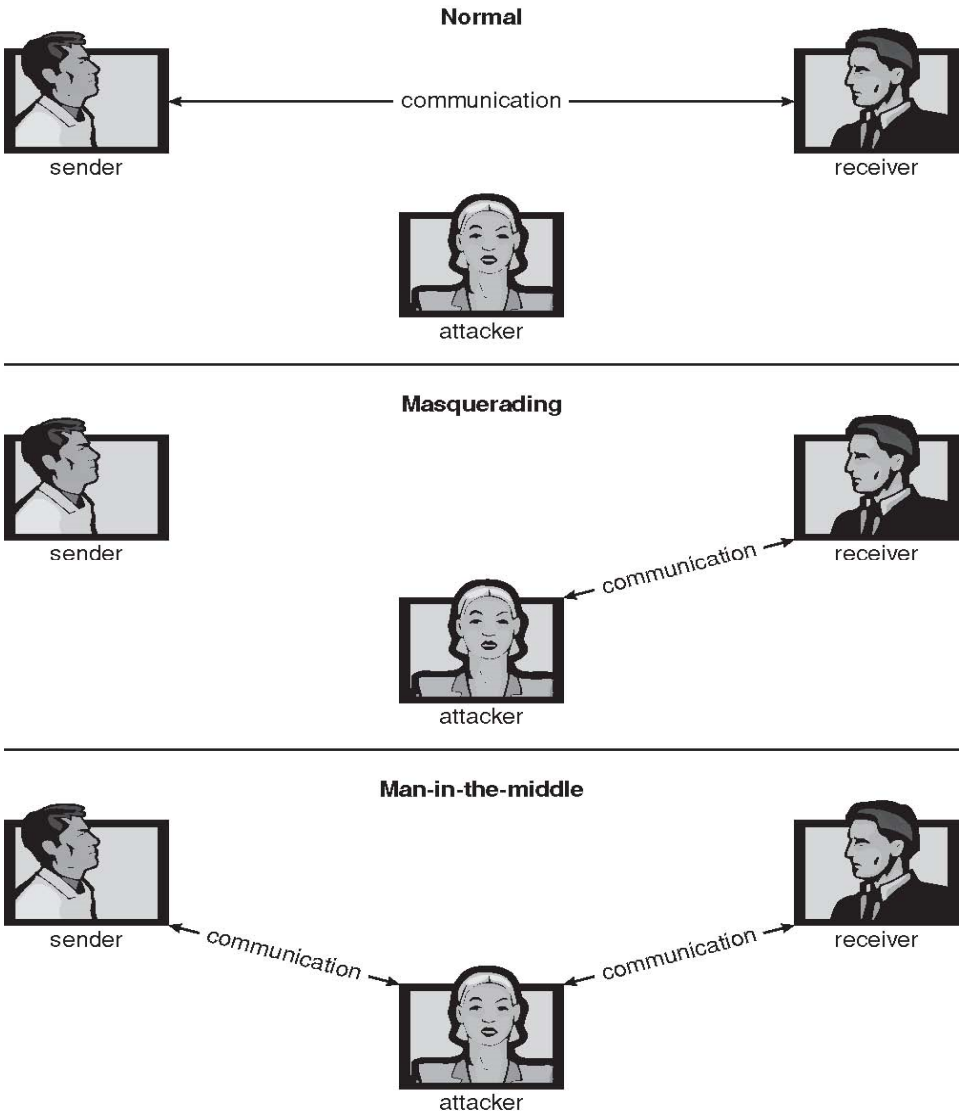
# Security Violation Methods

*Masquerading* (breach authentication) - Pretending to be an authorized user to escalate privileges

*Replay attack* – malicious or fraudulent repetition of valid data transmission, e.g request of transfer of money

*Man-in-the-middle attack* - Intruder sits in data flow, masquerading as sender to receiver and vice versa

*Session hijacking* - Intercept an already-established session to bypass authentication



# Protection Measures

Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders .

Security must occur at four levels to be effective:

*Physical* – The site containing the computer system must be physically secured against the suspicious entry. E.g. protection of Data centers, servers, connected terminals

*Human* – Proper user authorization, Avoid social engineering, phishing, dumpster diving. E.g a lagimate looking e-mail, web page, finding phone books, finding hints of passwords, etc.

*Operating System* – Protect from accidental or purposeful security breaches, protection mechanisms, debugging, updating etc. (finding possible breaches is endless)

*Network* – Most data today travels over the shared lines like internet, wireless, lease lines etc.

*Security is as weak as the weakest link in the chain. But can too much security be a problem?*

# Program Threats

Many variations, many names

**Trojan Horse**

A piece of code that misuses its environment. The program seems innocent enough, however when executed, unexpected behavior occurs.

Exploits mechanisms for allowing programs written by users to be executed by other users

Some variations: *Spyware* – freeware or shareware or comes with commercial software to download ads to display with user system, create *pop-up browser windows* – capture information and return to central site, called *covert channels*

Up to 80% of spam delivered by spyware-infected systems

Occurs by two mistakes – user get more privilege than necessary, poor OS decision on default values

**Trap Door**

The designer of system leave a hole that is used by designer for access.

Inserting a method of breaching security in a system. For instance, some secret set of inputs to a program might provide special privileges

E.g in rounding error bank system and transfer rounding petty amount to some account!

How to detect them? – difficult, need to analyze whole code!

# Program Threats

**Logic Bomb -**Program that create a security hole when predefined set of parameter were met

**Stack** and **Buffer Overflow**

Exploits a bug in a program, such as checking input field, (overflow either the stack or memory buffers) by attacker to gain unauthorized access

Attacker write the program for the following:

Send more data than program expecting causes the overflow in input buffer.

Overwrite return address with address of malicious code, when routine returns from call, returns to hacked address

Write a simple set of malicious code for the next space in the stack.

Example : 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;
    }
}
```

What happens if the parameter provided on the command line is longer than BUFFER_SIZE?

strcpy() copying from argv[1], until it encounter /0 or until the program crashes

Solution is better programming replacing “strcpy(buffer, argv[1]” by “strncpy(buffer, argv[1], sizeof(buffer) -1)”

Stack over flow can be exploited by attacker in many ways.

# Program Threats

## Viruses

Fragment of code embedded in a legitimate program. Mainly effects personal PC systems. These are often downloaded via e-mail or as active components in web pages, spared via sharing resources
Very specific to CPU architecture, operating system, applications

Commonly virus transmitted via MS Office file sharing such as MS Word Documents, those contains macros – that execute automatically.
Example: Visual Basic Macros that a virus – format the hard drive of a Window computer when the file containing this is open

```
Sub AutoOpen()
Dim oFS
  Set oFS = CreateObject(''Scripting.FileSystemObject'')
  vs = Shell(''c:command.com /k format  c:'',vbHide)
End Sub
```

# Program Threats

**Viruses (contd…)**

Virus dropper inserts virus onto the system

There are thousands of viruses they mainly categorized into:

- File / parasitic – appending itself to a file, change the start of program & jumps to this.
- Boot / memory – infect the boot sector, executed every time when OS is booted
- Macro – Triggered when program executing macro, mostly written in Visual Basic
- Source code – looks for source code and modify it to include virus
- Polymorphic – changes each time it is installed to avoid detection by antivirus software, changes do not effect the virus functionality.
- Encrypted – with decryption code, to avoid detection. Frist decrypt and execute
- Stealth – modify the part of system that could be used for detection
- Tunneling – bypass the antivirus detection by installing itself in the interrupt handler chain
- Multipartite - affect the multiple part of the chain, difficult to detect and remove
- Armored – Coded hidden attributes to make hard to antivirus researcher to unravel and understand

# The Threat Continues

Attacks still common, still occurring

Attacks moved over time from science experiments to tools of organized crime

- Targeting specific companies
- Creating botnets to use as tool for spam and DDOS delivery
- *Keystroke logger* to grab passwords, credit card numbers entered by keyboard

Why is Windows the target for most attacks?

Most common

- Everyone is an administrator
  - Licensing required?
- Monoculture considered harmful

# System and Network Threats

System and network threats involves the abuse of services and network connections

**Worms** – use *spawn* mechanism; standalone program

The worm spawns copies of itself, using up system resources and perhaps locking out all other system

**Port scanning**

Automated attempt to connect to a range of ports on one or a range of IP addresses.

Detection of answering service protocol, OS and version running on system

`nmap` scans all ports in a given IP range for a response

`nessus` has a database of protocols and bugs (and exploits) to apply against a system

**Denial of Service**

Overload the targeted computer preventing it from doing any useful work, for e.g downloaded java applet that proceeds to use all available CPU time or infinitely pop-up windows

Distributed denial-of-service (**DDOS**) come from multiple sites at once

# Stuxnet

**Stuxnet** is a computer worm discovered in June 2010. It initially spreads via Microsoft Windows, and targets Siemens industrial software and equipment.

Different variants of Stuxnet targeted five Iranian organizations, with the probable target widely suspected to be the uranium enrichment infrastructure in Iran.

It is initially spread using infected removable drives such as USB flash drives, and then uses other exploits and techniques to infect and update other computers inside private networks that are not directly connected to the Internet.

The malware has both user-mode and kernel-mode rootkit capability under Windows, and its device drivers have been digitally signed with the private keys of two certificates that were stolen from two separate companies. The driver signing helped it install kernel mode rootkit drivers successfully and therefore remain undetected for a relatively long period of time.

Once installed on Windows Stuxnet infects files belonging to Siemens' control software[3] and subverts a communication library. Doing so intercepts communications between software running under Windows and the target Siemens devices. The malware can install itself on PLC devices unnoticed.
Stuxnet malware periodically modifies a control frequency to and thus affects the operation of the connected centrifuge motors by changing their rotational speed.
This causes the centrifuges to be destroyed.

Siemens Simatic S7-300 PLC CPU with three I/O modules attached

# Sobig.F Worm

Disguised as a photo uploaded to adult newsgroup via account created with stolen credit card
Targeted Windows systems
Had own SMTP engine to mail itself as attachment to everyone in infect system's address book
Disguised with harmless subject lines, looking like it came from someone known

Attachment was executable program that created WINPPR32.EXE in default Windows system directory
Plus the Windows Registry
[HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"TrayX" = %windir%\winppr32.exe /sinc
[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"TrayX" = %windir%\winppr32.exe /sinc

# Cryptography as a Security Tool

How to trust source and destination of messages on network? And how to grant access to OS without trusting the user? – one solution is cryptography

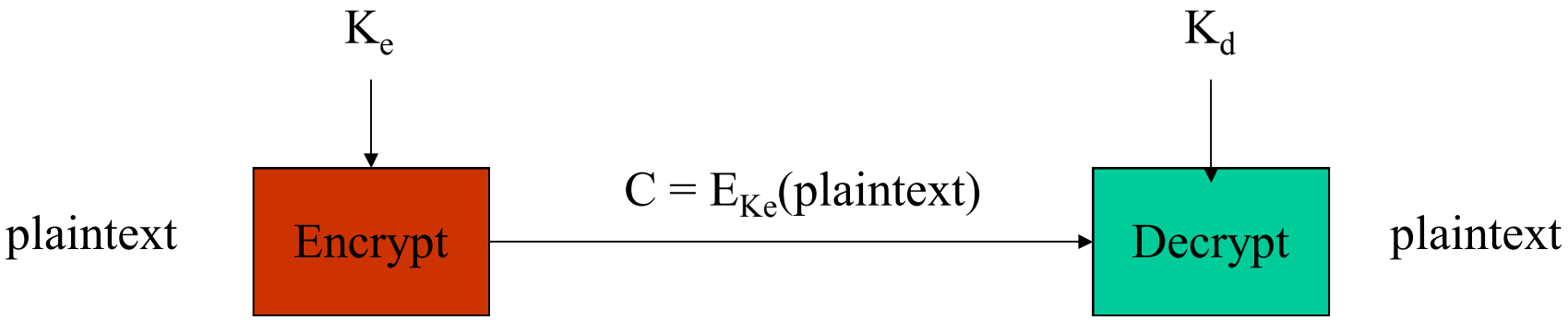
Information can be encoded using a *key* when it is written (or transferred) – *encryption* **C = E( M, Ke )**

E = Encyphering Algorithm
M = Message - plain text
Ke = Encryption key
C = Cyphered text

It is then decoded using a key when it is read (or received) -- *decryption*

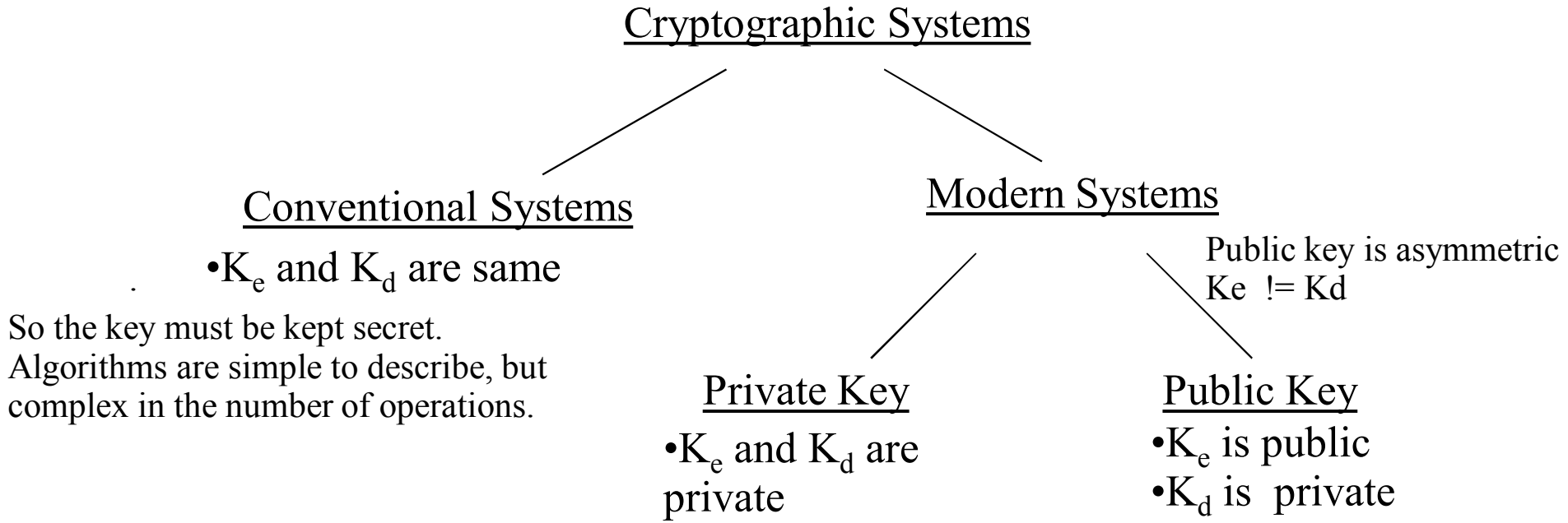
**M = D( C, Kd )**

D = Decyphering Algorithm
Kd = Decryption key



# Cryptographic Systems

**Cryptosystems are either Conventional or Modern**




**Security** against attack is either:

**Unconditionally secure** - Ke can't be determined regardless of available computational power.

**Computationally secure:** - calculation of Kd is computationally unfeasible

The only known unconditionally secure system in common use!

Involves a random key that has the same length as the plain text to be encrypted.

The key is used once and then discarded. The key is exclusively OR'd with the message to produce the cypher.

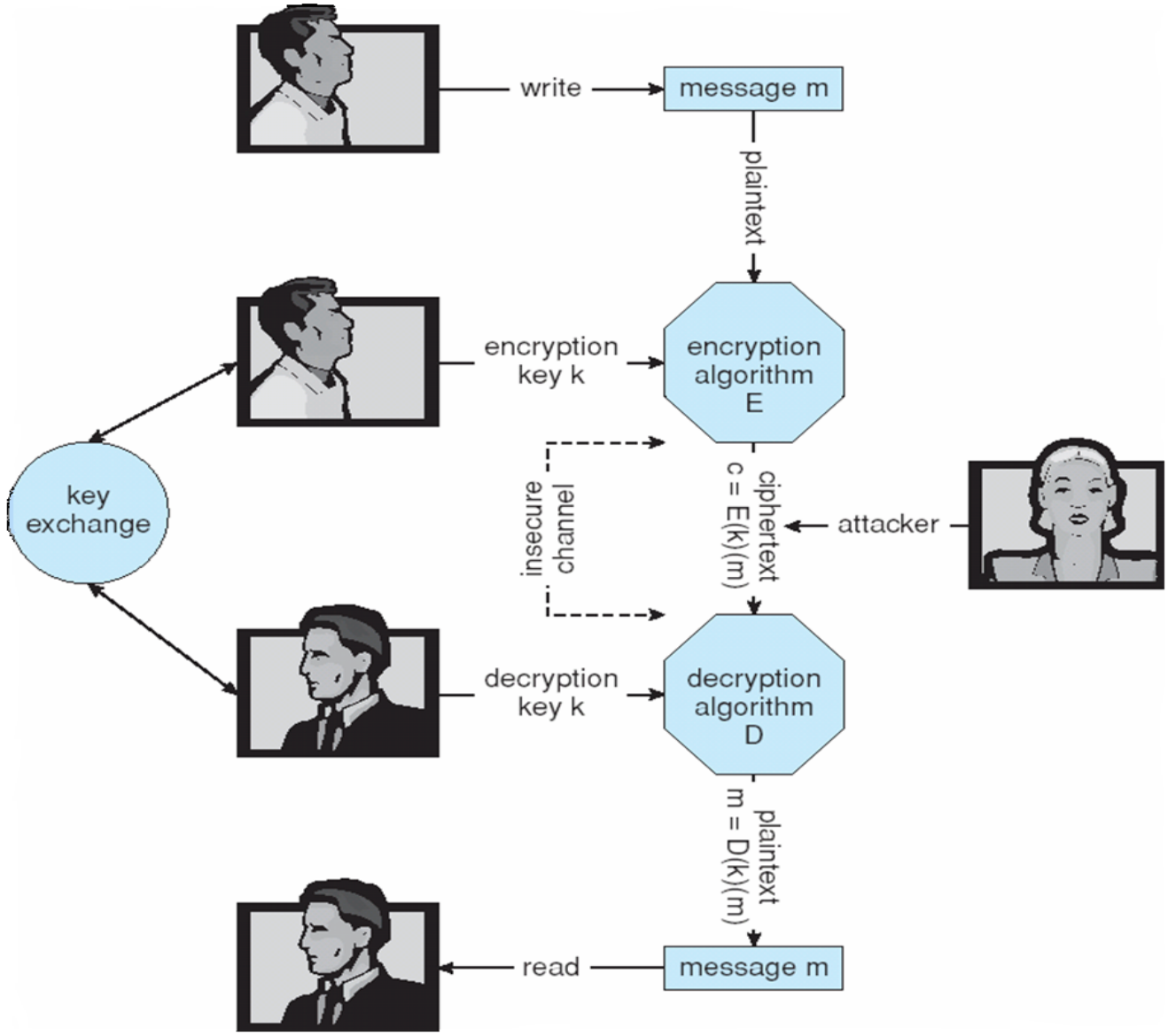
Given the key and the cypher, the receiver uses the same method to reproduce the message.

# Cryptographic Systems

A function E : K → (M→C). That is, for each k, E(k) is a function for generating ciphertexts from messages
A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k$, $D(k)$ is a function for generating messages from ciphertexts
An encryption algorithm must provide this essential property: Given a ciphertext c ∈ C, a computer can compute m such that E(k)(m) = c only if it possesses D(k)

Thus, a computer holding $D(k)$ can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding $D(k)$ cannot decrypt ciphertexts

Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive $D(k)$ from the ciphertexts



# Data Encryption Standard (DES)

The official National Institute of Standards and Technology (NIST), (formerly the National Bureau of Standards) encryption for use by Federal agencies.

The source of security is the non-linear many-to-one function applied to a block of data (64 bits chunk). This function uses transposition and substitution. The algorithm is public, but the key (56 bits) is secret.

Computational power today can crack a 56 bit code.

In common use today is Triple DES in which 3 different keys are used, making the effective key length 168 bits

Advanced Encryption Standard (AES) replace DES, use the key length of 128, 192 & 256 bits and works in 128 bit block.

# Public Key Cryptosystems

The general principle is this:

1. Any **RECEIVER A** uses an algorithm to calculate an encryption key **KEa** and a decryption key **KDa**.
2. Then the receiver PUBLICIZES **KEa** to anyone who cares to hear. But the receiver keeps secret the decryption key **KDa**.
3. **User B** sends a message to **A** by first encrypting that message using the publicized key for that receiver **A**, **KEa**.
4. Since only **A** knows how to decrypt the message, it's secure.

# Public Key Cryptosystems

To be effective, a system must satisfy the following rules:

a) Given plaintext and ciphertext, the problem of determining the keys is computationally complex.

b) It is easy to generate matched pairs of keys Ke, Kd that satisfy the property

**D( E( M, Ke ), Kd ) = M.**

This implies some sort of trapdoor, such that Ke and Kd can be calculated from first principles, but one can't be derived from the other.

The encryption and decryption functions E and D are efficient and easy to use.

c) Given Ke , the problem of determining Kd is computationally complex.

e) For almost all messages it must be computationally unfeasible to find ciphertext key pairs that will produce the message.

f) Decryption is the inverse of encryption.

**E( D( M, Kd ), Ke ) = D( E( M, Ke ), Kd )**

# Public Key Cryptosystems -Example

1. Two large prime numbers p and q are selected using some efficient test for primality. These numbers are secret:

   Let p = 3, q = 11

2. The product n = p * q is computed.

   n = 3 * 11 = 33.

3. The number Kd > max( p, q ) is picked at random from the of integers that are relatively prime to and less than L(n) = ( p - 1 ) ( q - 1).

   L(n) = ( p - 1 ) ( q - 1 ) = 20.
   Choose Kd > 11 and prime to 20.
   Choose Kd = 13.

4. The integer Ke , 0 < Ke < L(n) is computed from L(n) and Kd such that Ke*Kd (mod L(n))=1.

   0 < Ke < 20
   Ke = 17. (since 17 * 13 = 221( mod 20 ) ) = 1

Separate the text to be encoded into chunks with values 0 - ( n - 1 ).

In our example, we'll use < space = 0, A = 1, B = 2, C = 3, D = 4, E = 5 >.
Then "BA<sp>D<sp><sp>BE<sp>E " --> "21 04 00 25 05"

| | | | | | |
|---|---|---|---|---|---|
| 21 ^ 17 ( mod 33 ) | = | 21. | 21 ^ 13 ( mod 33 ) | = | 21. |
| 04 ^ 17 ( mod 33 ) | = | 16. | 16 ^ 13 ( mod 33 ) | = | 04. |
| 00 ^ 17 ( mod 33 ) | = | 00. | 00 ^ 13 ( mod 33 ) | = | 00. |
| 25 ^ 17 ( mod 33 ) | = | 31. | 31 ^ 13 ( mod 33 ) | = | 25. |
| 05 ^ 17 ( mod 33 ) | = | 14. | 14 ^ 13 ( mod 33 ) | = | 05. |

This whole operation works because, though n and Ke are known (public key) , p and q are not public. Thus Kd is hard to guess.

[Note: recently a 100 digit number was successfully factored into two prime

# Authentication and Digital Signature

## Sender Authentication:

In a public key system, how does the receiver know who sent a message (since the receiver's encryption key is public)?

Suppose **A** sends message **M** to **B**: (RSA Digital Signature algorithm – similar to RSA encryption algorithm bur key use is reversed)

**A** DECRYPTS **M** using **A's** **Kd(A )** .

**a)** **A** attaches its identification to the message.

**b)** **A** ENCRYPTS the entire message using **B's** encryption, **Ke(B)**

**C = E ( ( A, D( M, Kd(A) ) ), Ke(B) )**

**d)** **B** decrypts using its private key **Kd(A)** to produce the pair **A, D( M, Kd(A) ).**

e) Since the proclaimed sender is **A**, **B** knows to use the public encryption key **Ke(A)**.

## Capture/Replay

In this case, a third party could capture / replay a message.

The solution is to use a rapidly changing value such as time or a sequence number as part of the message.

# Encryption Example - SSL

- Insertion of cryptography at one layer of the ISO network model (the transport layer)
- SSL – Secure Socket Layer
- Cryptographic protocol that limits two computers to only exchange messages with each other
  - Very complicated, with many variations
- Used between web servers and browsers for secure communication (credit card numbers)
- Designed by Netscape and evolved into industry standard TLS protocol
- The server is verified with a **certificate** assuring client is talking to correct server
- Asymmetric cryptography used to establish a secure **session key,** forgotten once a session is completed, (symmetric encryption) for bulk of communication during session
- Communication between each computer uses symmetric key cryptography

# User Authentication

How to authenticate users?

User authentication to identify user correctly, as protection systems depend on user ID. (protection system identify the program and processes currently executing)

User identity most often established through *passwords*, can be considered a special case of either keys or capabilities

Passwords must be kept secret

- Frequent change of passwords
- History to avoid repeats
- Use of "non-guessable" passwords
- Log all invalid access attempts (but not the passwords themselves)
- Unauthorized transfer

Passwords may also either be encrypted or allowed to be used only once

- Does encrypting passwords solve the exposure problem?
  - Might solve **sniffing**
  - Consider **shoulder surfing**
  - Consider Trojan horse keystroke logger
  - How are passwords stored at authenticating site?

# Passwords

Encrypt to avoid having to keep secret
- But keep secret anyway (i.e. Unix uses superuser-only readably file `/etc/shadow`)
- Use algorithm easy to compute but difficult to invert
- Only encrypted password stored, never decrypted
- Add "salt" to avoid the same password being encrypted to the same value

One-time passwords
- Use a function based on a seed to compute a password, both user and computer
- Hardware device / calculator / key fob to generate the password
- Changes very frequently

Biometrics
- Some physical attribute (fingerprint, hand scan)

Multi-factor authentication
- Need two or more factors for authentication
  - i.e. USB "dongle", biometric measure, and password

# 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

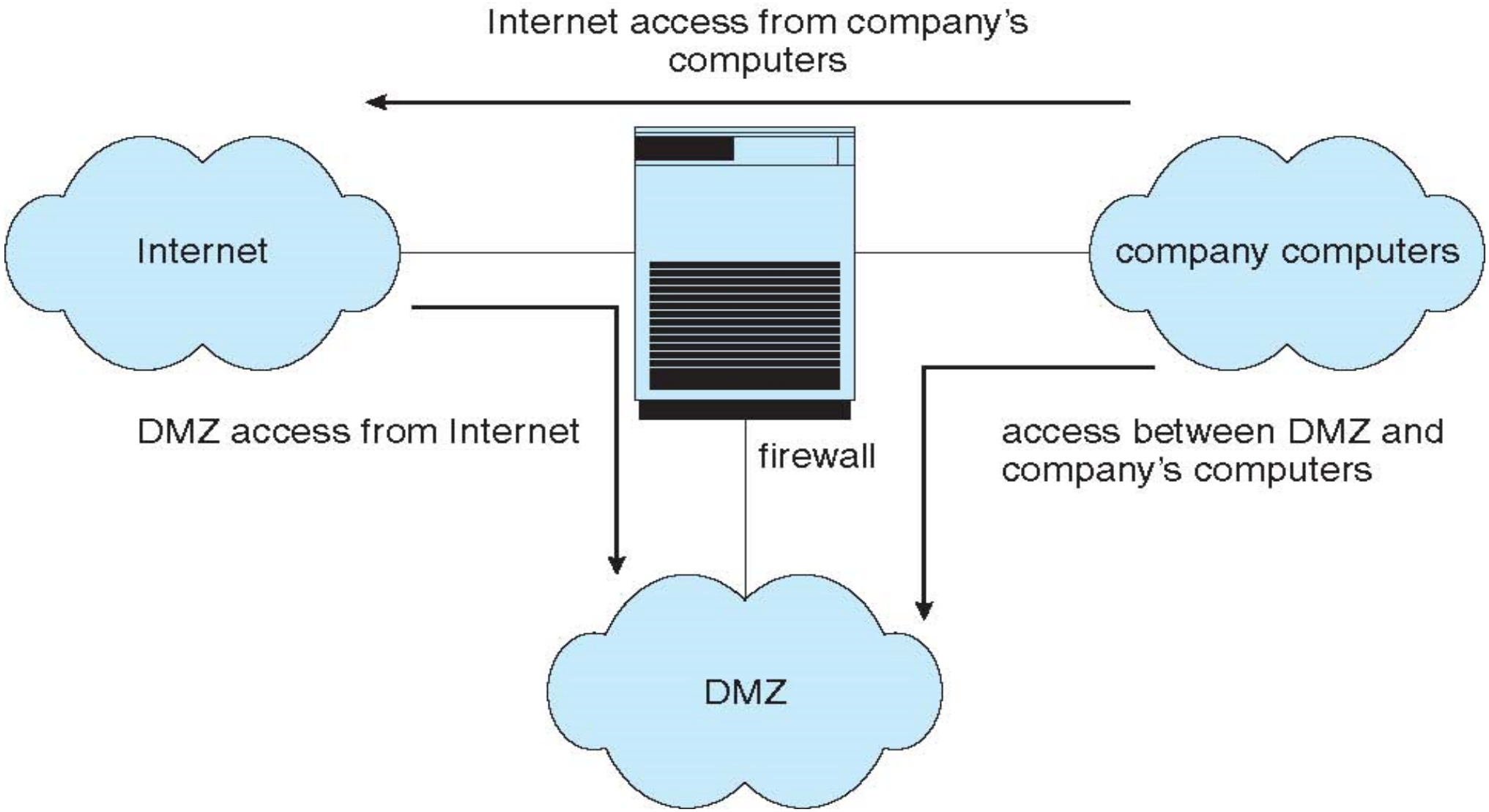
**Personal firewall** is software layer on given host

- 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)

# Network Security Through Domain Separation Via Firewall



# Computer Security Classifications

U.S. Department of Defense outlines four divisions of computer security: **A**, **B**, **C**, and **D**

**D** – Minimal security

**C** – Provides discretionary protection through auditing
Divided into **C1** and **C2**
**C1** identifies cooperating users with the same level of protection
**C2** allows user-level access control

**B** – All the properties of **C**, however each object may have unique sensitivity labels
Divided into **B1**, **B2**, and **B3**

**A** – Uses formal design and verification techniques to ensure security

# Home Works

## HW #13:

1. Question No 15.3, 15.4, 15.5, 15.7 & 15.10 from book
2. What is the minimum, average and maximum time it would take to crake a six-digit password if one password can be checked in 1 millisecond?