NATIONAL ECONOMICS UNIVERSITY

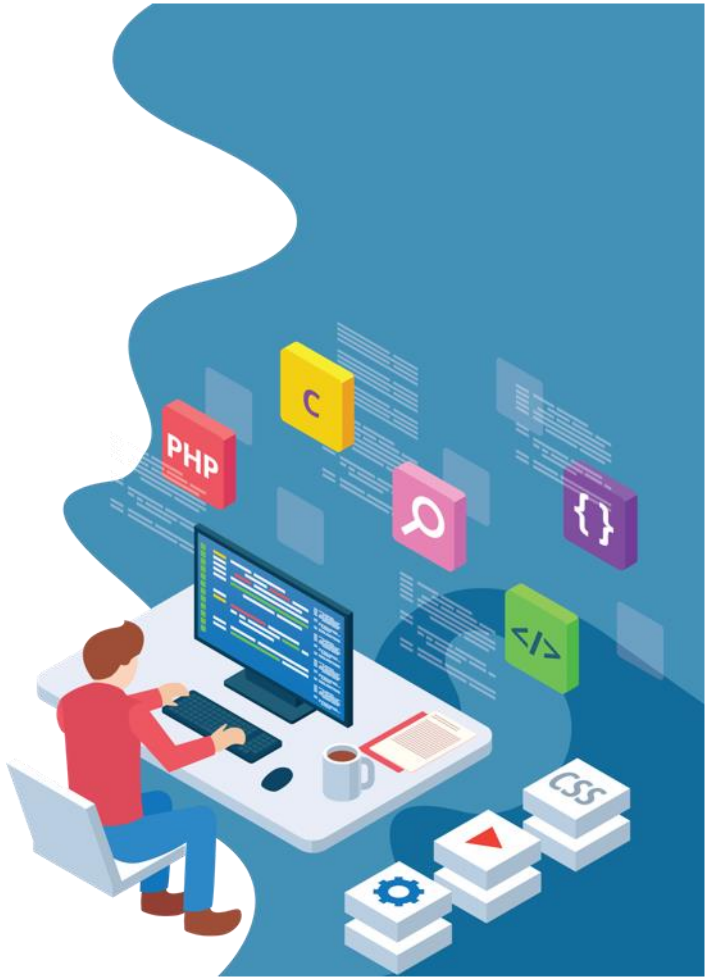
SCHOOL OF INFORMATION TECHNOLOGY AND DIGITAL ECONOMICS

CHAPTER 7

PROTECTION & SECURITY

OUTLINE

- Protection
  - Goals of Protection
  - Principles of Protection
  - Domain of Protection
  - Access Matrix
  - Implementation of Access Matrix
- Security
  - The Security Problem
  - Program Threats
  - System and Network Threats
  - Cryptography as a Security Tool
  - User Authentication
  - Implementing Security Defenses

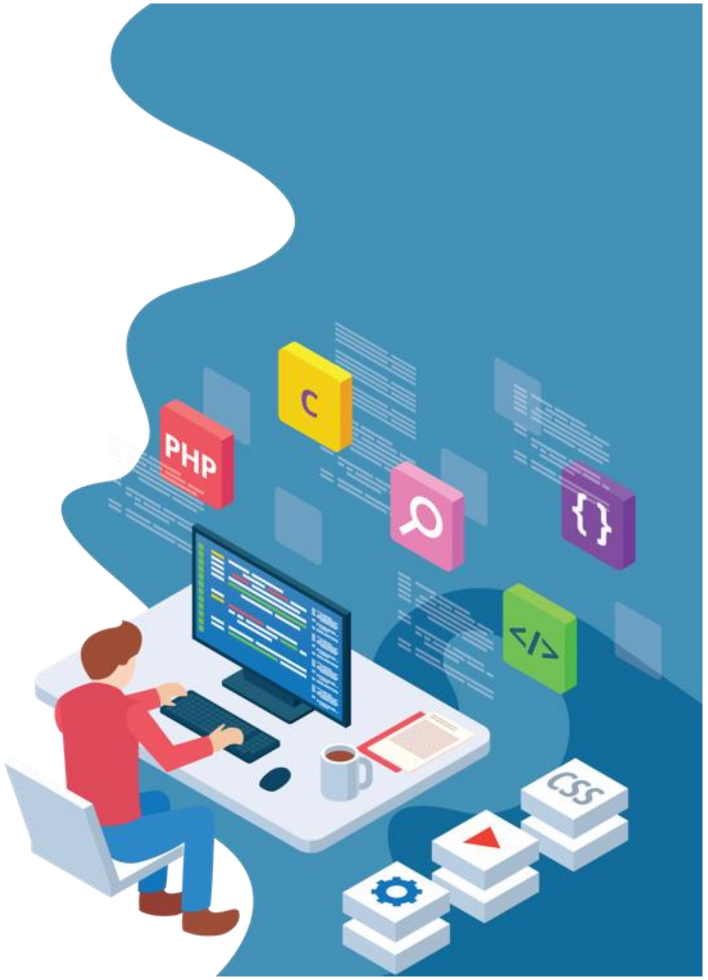


OBJECTIVES

- Discuss security threats and attacks
- Explain the fundamentals of encryption, authentication
- Examine the uses of cryptography in computing
- Describe the various countermeasures to security attacks
- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access

OUTLINE

- Protection
  - ➡ Goals & Principles of Protection
  - Domain of Protection
  - Access Matrix
  - Implementation of Access Matrix
- Security



GOALS OF PROTECTION

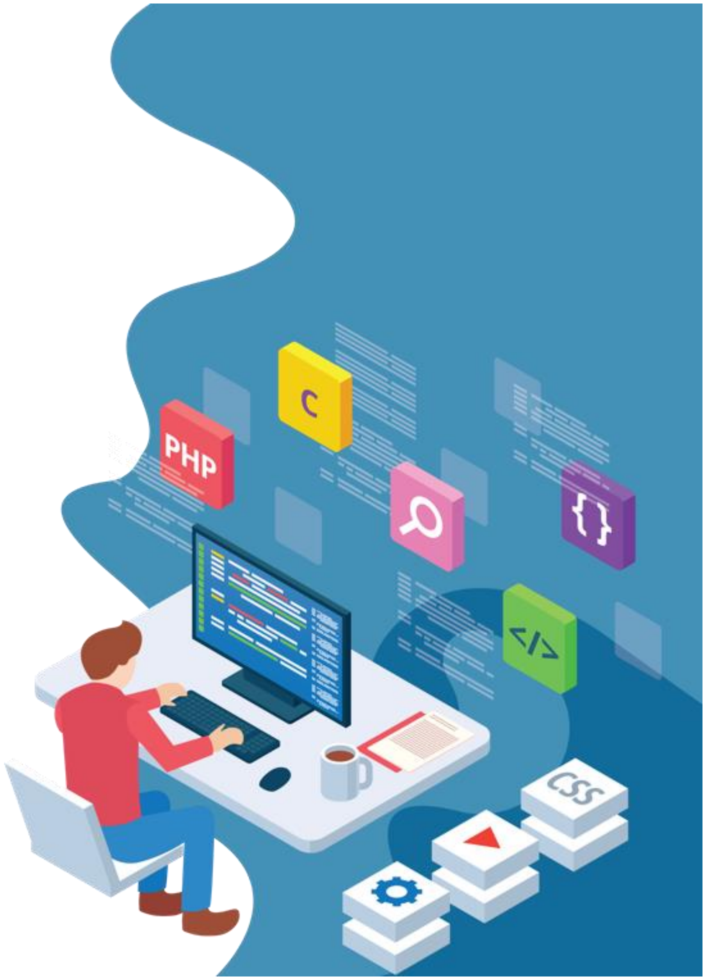
- The most obvious is the need to **prevent the mischievous, intentional violation of an access restriction** by a user.
- The need to ensure that each program component active in a system **uses system resources only in ways consistent with stated policies**
- Protection can **improve reliability** by detecting latent errors at the interfaces between component subsystems.
- Provide a mechanism for the enforcement of the **policies governing resource** use.

PRINCIPLES OF PROTECTION

- Guiding principle – **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
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
  - “Need to know” a similar concept regarding access to data

OUTLINE

- Protection
  - Goals & Principles of Protection
  - ➡ Domain of Protection
  - Access Matrix
  - Implementation of Access Matrix
- Security



DOMAIN OF PROTECTION

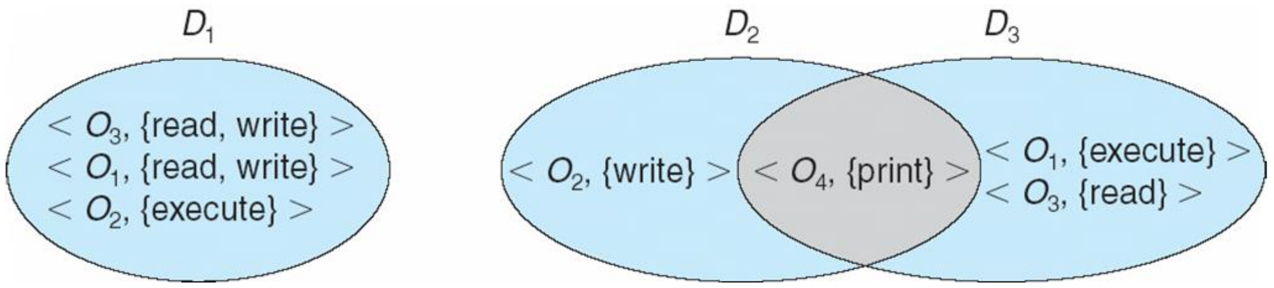
- A computer system is a collection of
  - Processes:
  - Objects:
    - Hardware objects (such as the CPU, memory segments, printers, disks, and tape drives)
    - Software objects (such as files, programs, and semaphores).
  - Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations.
  - A process should be allowed to access only those resources for which it has authorization.
- To facilitate the scheme just described, a process operates within a **protection domain**,

DOMAIN OF PROTECTION

- A process operates within a protection domain, which 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.**
- A domain is a collection of access rights, each of which is an ordered pair **<object-name, rights-set>.**
- For example, if domain D has the access right <file F, {read,write}>, then a process executing in domain D can both read and write file F. It cannot, however, perform any other operation on that object.

DOMAIN STRUCTURE

- Access-right = <*object-name*, *rights-set*>
- Domain = set of access-rights
- Domains may share access rights



- Domains may share access rights.
  - A process executing in either D2 or D3 can print O4
  - A process must be executing in D1 to read and write O1. Also, only process in D3 may execute O1

DOMAIN STRUCTURE

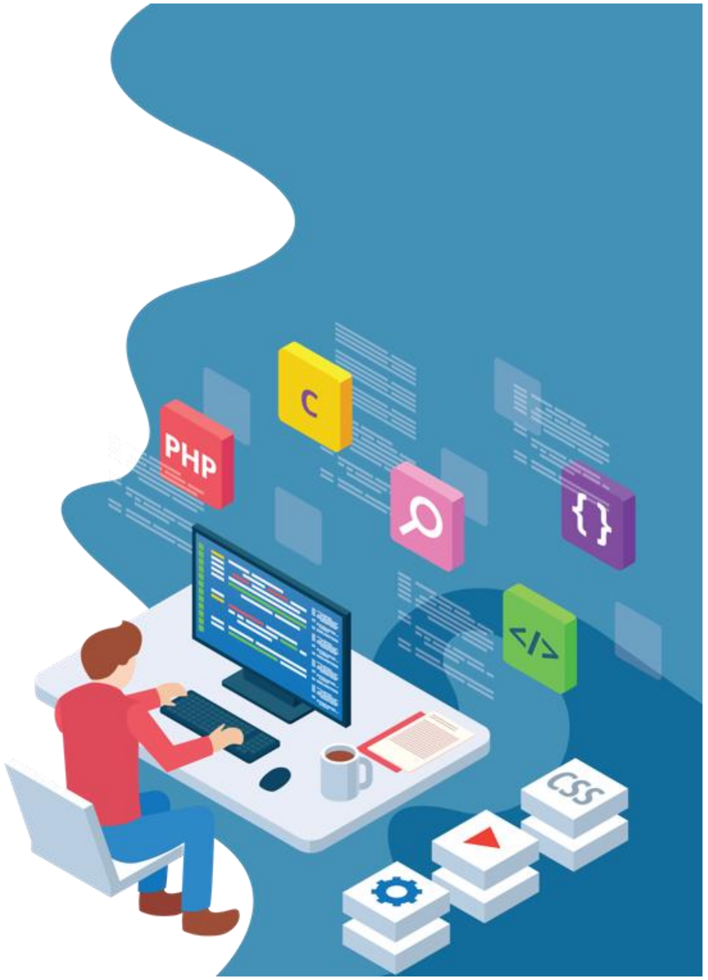
- A domain can be realized in a variety of ways:
  - Each user may be a domain. Set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed—generally when one user logs out and another user logs in.
  - Each process may be a domain. In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.
  - Each procedure may be a domain. In this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

DOMAIN IMPLEMENTATION (UNIX)

- System consists of 2 domains:
  - User
  - Supervisor
- UNIX
  - Domain = user-ID
  - Domain switching corresponds to user ID switching
  - Domain switching is accomplished through file system as follows:
    - Each file has associated with it a domain bit (setuid bit) and an owner ID.
    - When a user A starts executing a file owned by user B and the setuid bit is off, the user ID of the process is set to A
    - When setuid = on, then user-id is set to owner of the file being executed: B. When execution completes user-id is reset.

OUTLINE

- Protection
  - Goals & Principles of Protection
  - Domain of Protection
  - ➡ Access Matrix
  - Implementation of Access Matrix
- Security



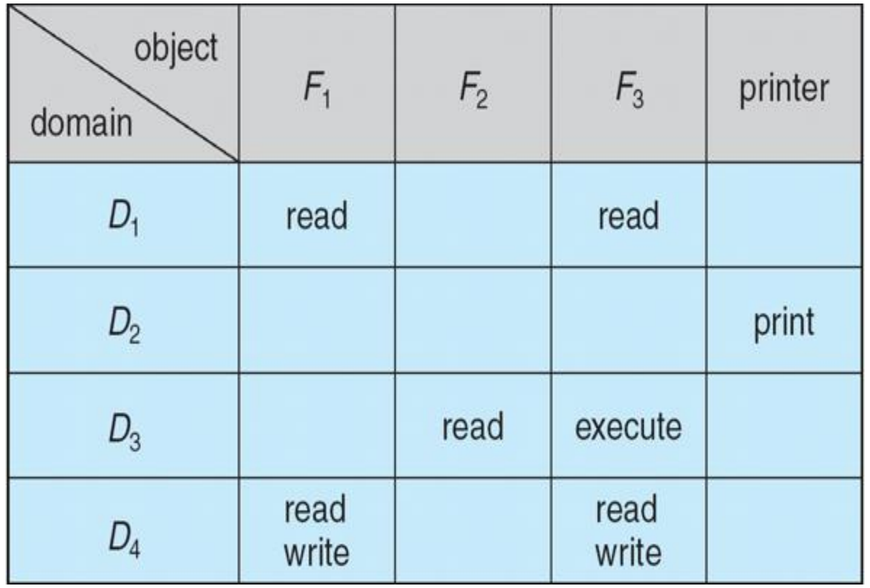
ACCESS MATRIX

- View protection as a matrix (*access matrix*)
  - Rows represent domains
  - Columns represent objects
  - Each entry in the matrix consists of a set of access rights.
  - *Access(i, j)* is the set of operations that a process executing in Domaini can invoke on Objectj
- Summary: access list keeps track of which object belongs to which domain
- Note: Most domains have no access at all to most objects, so storing a very large, mostly empty, matrix is a waste of disk space
  - ACL
  - Capability List

ACCESS MATRIX (CONT.)

Access Matrix illustration:

- 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.



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	

ACCESS MATRIX (CONT.)

- Users decide the contents of the access-matrix entries.
  - When a user creates a new object Oj, the column Oj is added to the access matrix with the appropriate initialization entries.
  - The user may decide to enter some rights in some entries in column j and other rights in other 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.
  - Controls switching between domains

ACCESS MATRIX (CONT.)

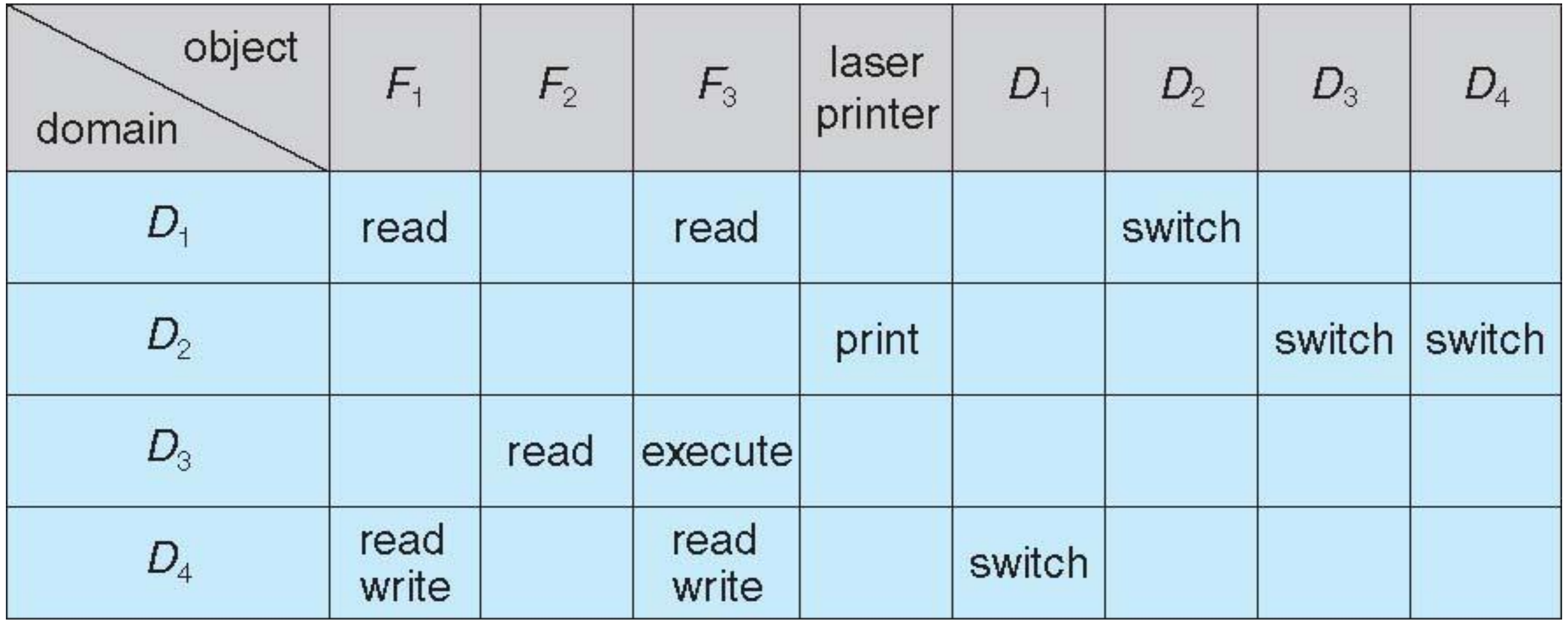
- When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).
  - We can control domain switching by including domains among objects of the access matrix.
- When we change the content of the access matrix, we are performing an operation on an object which is the access matrix.
  - We can control these changes by including the access matrix itself as an object.
- Processes should be able to **switch** from one domain to another.
  - Domain Switching from Di to Dj is allowed to occur if access right **switch** ϵ access(i,j).

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
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by "* ")*
    - *control –* D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

DOMAIN SWITCHING: ACCESS MATRIX WITH DOMAINS AS OBJECTS

- A process executing in D4 (row) can switch to D1.
- A process executing in D2 (row) can switch to D3 or D4.
- A process executing in D1 (row) can switch to D2.



object / domain	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			

ACCESS MATRIX WITH *COPY* RIGHTS

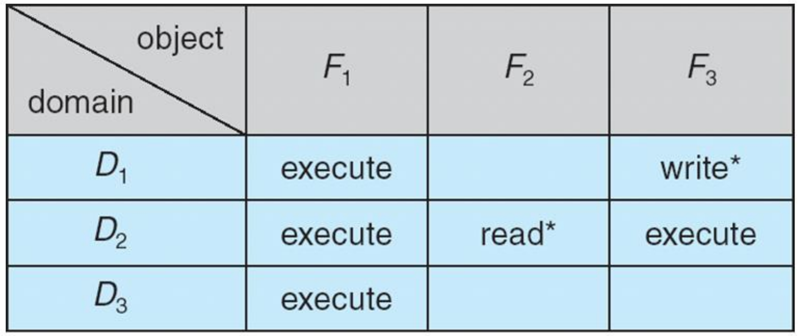
- Allowing controlled change to the contents of the access-matrix entries requires 3 additional operations:
  - Copy, Owner, and Control.
- The ability to copy an access right from one domain (row) to another is denoted by an asterisk (*) appended to the access right.
- **Copy** right allows the copying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

ACCESS MATRIX WITH *COPY* RIGHTS (CONT.)

- A process executing in D2 can copy the read operation into any entry associated with F2.
- The ***copy*** scheme has 3 variants:

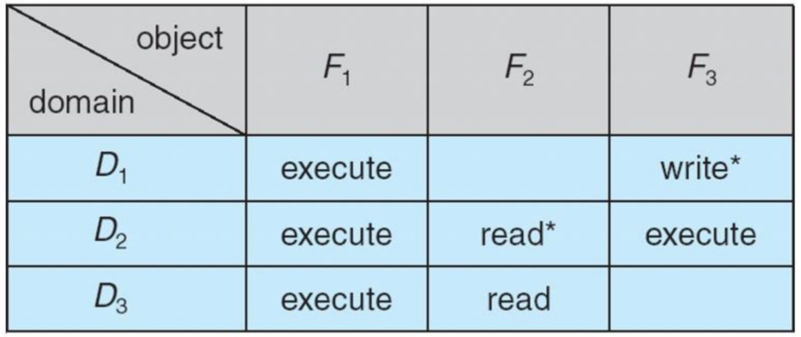
 1. A right is copied from access(i,j) to access(k,j) is not limited: This action is called ***copy***.

- 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		

(a)



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

(b)

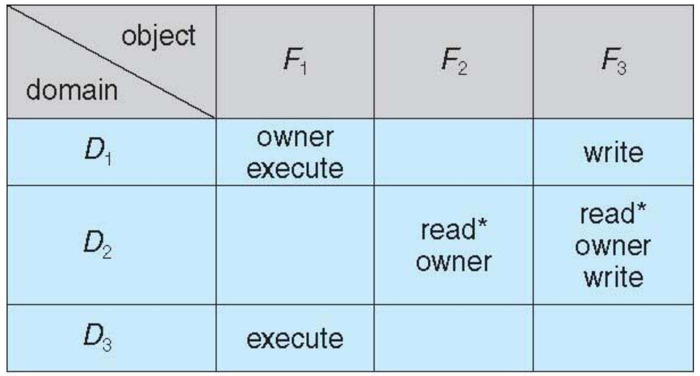
ACCESS MATRIX WITH *COPY* RIGHTS (CONT.)

2. Propagation of the copy right may be limited: This action is called ***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 Dk cannot further copy the right Read.

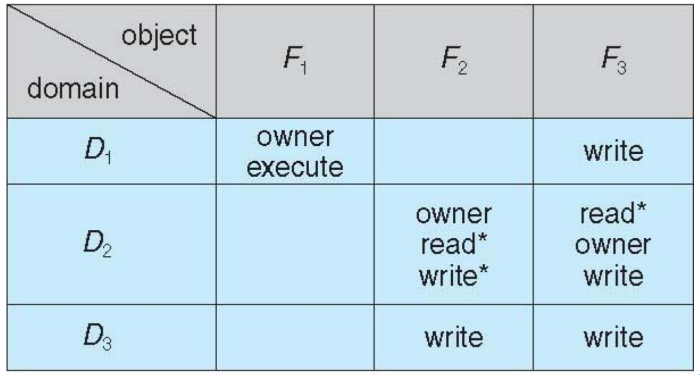
3. A right is copied from access(i,j) to access(k,j); it is then removed from access(i,j).

- This action is called a ***transfer*** of a right, rather than a copy (denoted by *R+*)



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

(a)

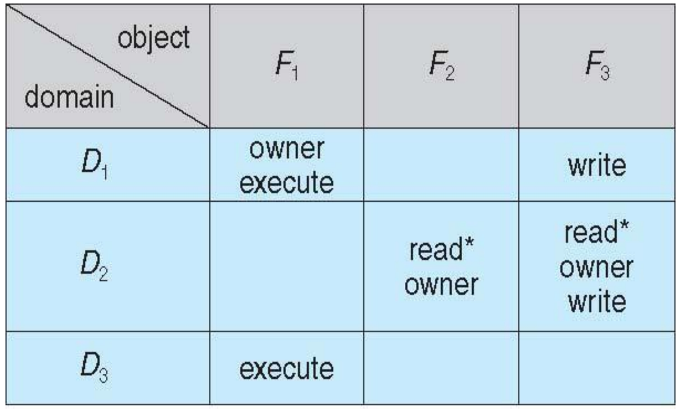


domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

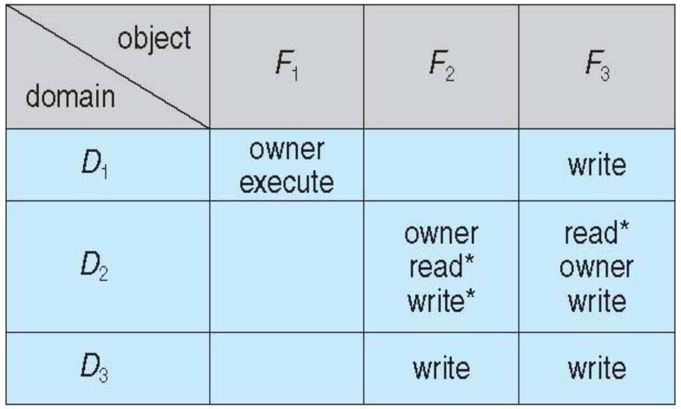
ACCESS MATRIX WITH *OWNER* RIGHTS

- 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

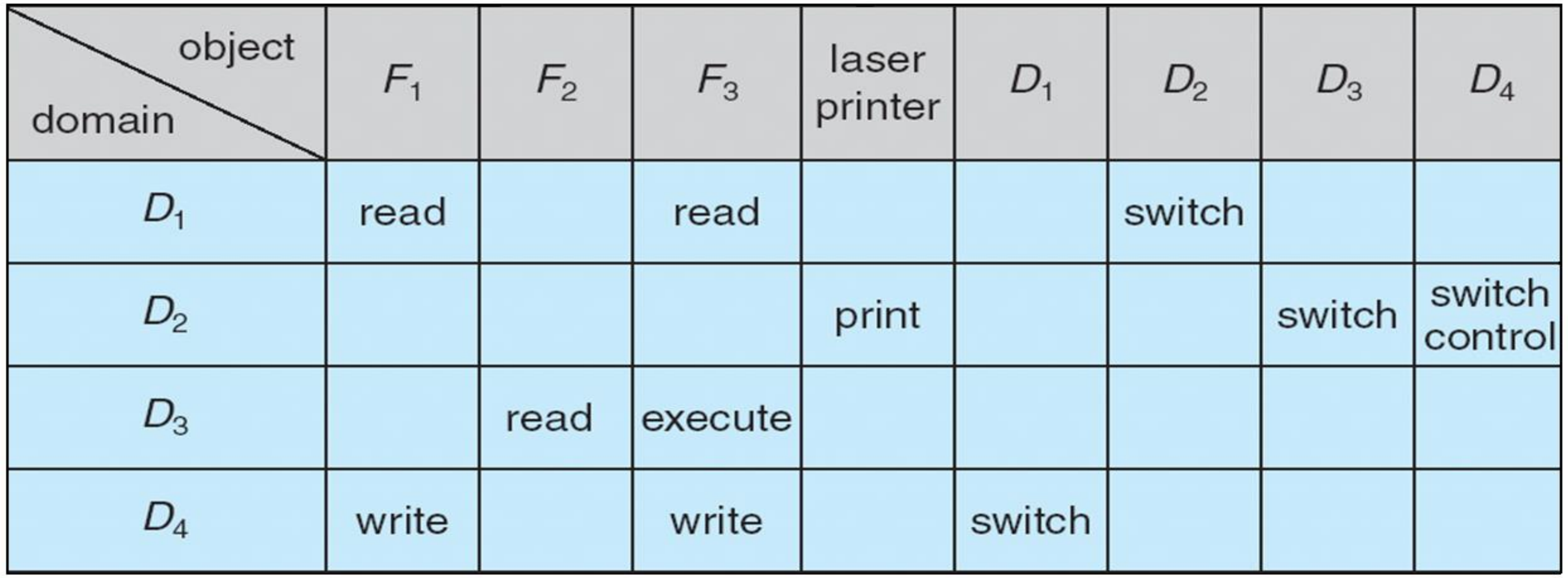
(b)

ACCESS MATRIX WITH *CONTROL* RIGHTS

- The ***copy*** and ***owner*** rights allow a process to change the entries in a column.
  - Limit the propagation of access rights
  - However, they do not give us the appropriate tools for preventing the propagation (or disclosure) of information.
- 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.

ACCESS MATRIX WITH *CONTROL* RIGHTS

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



object / domain	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			

ACCESS MATRIX EXERCISE

Object / Domain	F1	F2	F3	Printer
D1	Read		Read	
D2		write		Print
D3		Read	execute	
D4	Read Write		Read write	

1. Can the process at domain D1 read F2?
2. Will the process at domain D4 get F1 write?
3. Can a process in domain D3 use the printer?
4. Does the process at domain D2 get F3 handle?

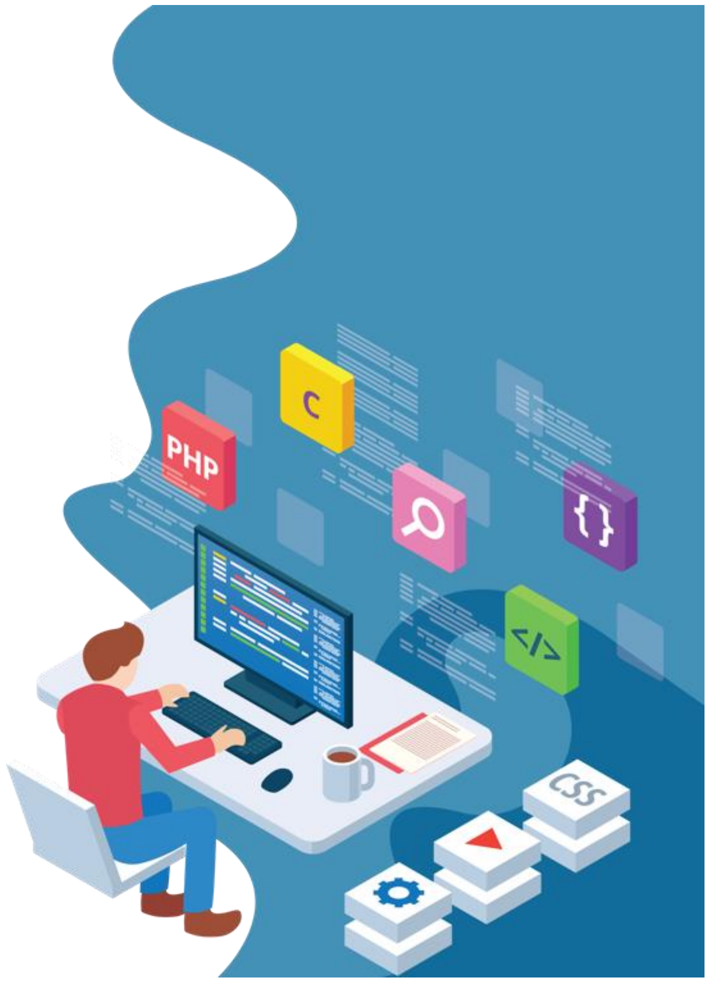
ACCESS MATRIX EXERCISE (CONT.)

Object / Domain	F1	F2	F3	Printer	D1	D2	D3	D4
D1	Read		Read			switch		
D2				Print			switch	switch
D3		Read	Exe					
D4	Read write		Read Write		switch			

- Does the process at domain D1 have a chance to read F2?
- Does the process in domain D4 have a chance to be write to F1?
- Does the process in domain D3 have a chance to use the printer?
- Does the process in domain D2 have a chance to be processed F3

OUTLINE

- Protection
  - Goals & Principles of Protection
  - Domain of Protection
  - Access Matrix
  - Implementation of Access Matrix
- Security

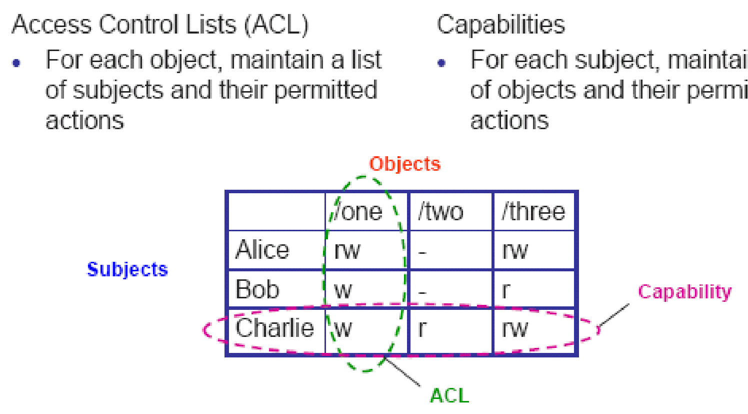


IMPLEMENTATION OF ACCESS MATRIX

- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples **`<domain, object, rights-set>`** in table
  - A requested operation M on object O_j within domain D_i -> search table for < D_i, O_j, R_k >
    - with $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)

IMPLEMENTATION OF ACCESS MATRIX (CONT.)

- 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 M ∈ default set, also allow access
- Examples:
  - A, B, C is users
  - root, users are groups
    - file1: (A, *, rwx)
    - file2: (B, users, rx), (C, root, rwx)



Access Control Lists (ACL)
- For each object, maintain a list of subjects and their permitted actions

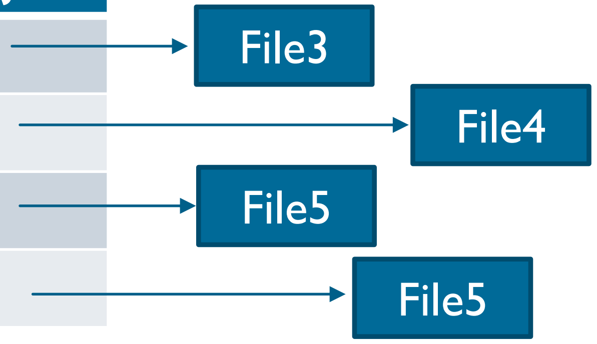
Capabilities
- For each subject, maintain a list of objects and their permitted actions

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

IMPLEMENTATION OF ACCESS MATRIX (CONT.)

- 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

Types	Right	Object
File	R--	
File	Rwx	
File	Rw-	
Printer	-w-	



IMPLEMENTATION OF ACCESS MATRIX (CONT.)

- 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

COMPARISON OF IMPLEMENTATIONS (CONT.)

- 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

SECURITY

OUTLINE

- Protection
- Security
  - ➡ The Security Problem
  - Program Threats
  - System and Network Threats
  - Cryptography as a Security Tool
  - User Authentication
  - Implementing Security Defenses

THE SECURITY PROBLEM

- System **secure** if resources used and accessed as intended under all circumstances
- **Total security** cannot be achieved
- Nonetheless, we must have **mechanisms** to make security breaches a rare occurrence, rather than the norm
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation, such as the discovery of a vulnerability
- **Attack** is attempt to breach security
- Attack can be **accidental** or **malicious**
- Easier to protect against **accidental** than **malicious** misuse

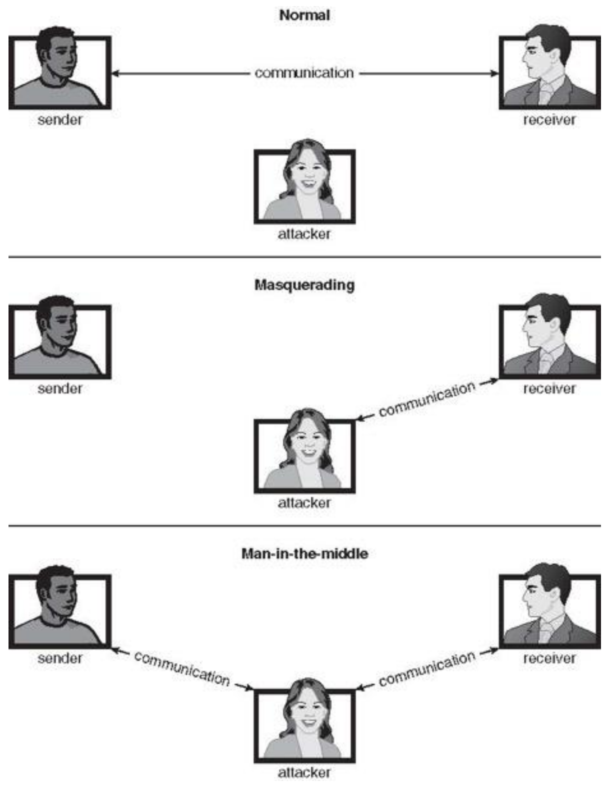
FORMS OF ACCIDENTAL AND MALICIOUS SECURITY VIOLATIONS

- **Breach of confidentiality**
  - Unauthorized reading of data
- **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**
  - As is or with **message modification**
- **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

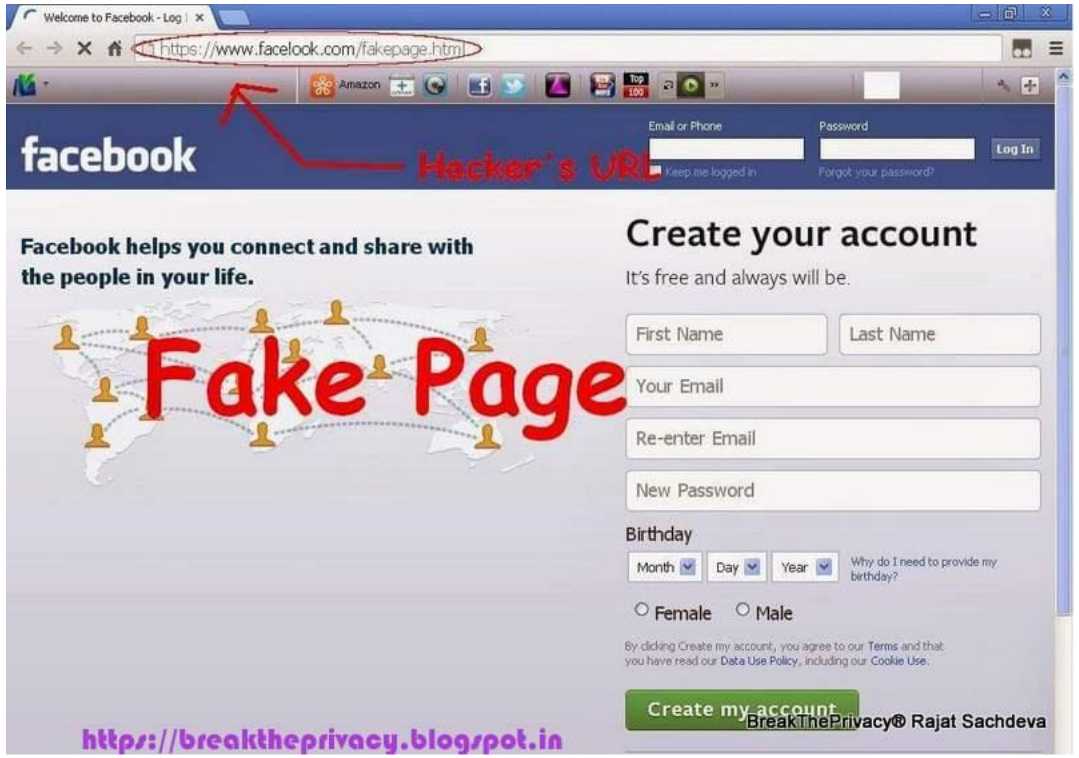
STANDARD SECURITY ATTACKS



SECURITY MEASURE LEVELS

- 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**
    - Data centers, servers, connected terminals
  - **Human**
    - Avoid **social engineering**: **phishing**, **dumpster diving, shoulder surfing**
  - **Operating System**
    - The system must protect itself from accidental or purposeful security breaches
  - **Network**
    - Intercepted communications, interruption, DoS
- Security is as weak as the weakest link in the chain

PHISHING


https://www.facelook.com/fakepage.html

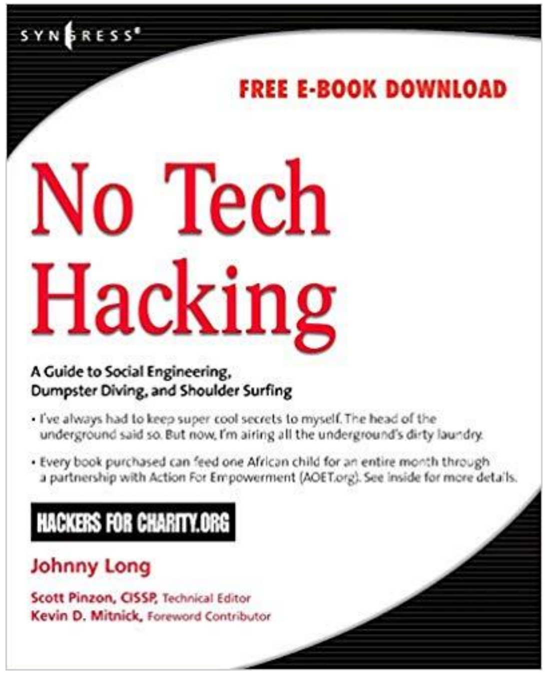
Facebook helps you connect and share with the people in your life.

Create your account

It's free and always will be.

By clicking Create my account, you agree to our Terms and that you have read our Data Use Policy, including our Cookie Use.

NO TECH HACKING



SYNGRESS®

FREE E-BOOK DOWNLOAD

No Tech Hacking

A Guide to Social Engineering, Dumpster Diving, and Shoulder Surfing

- I've always had to keep super cool secrets to myself. The head of the underground said so. But now, I'm airing all the underground's dirty laundry.
- Every book purchased can feed one African child for an entire month through a partnership with Action For Empowerment (AOET.org). See inside for more details.

HACKERS FOR CHARITY.ORG

Johnny Long

Scott Pinzon, CISSP, Technical Editor
Kevin D. Mitnick, Foreword Contributor

OUTLINE

- Protection
- Security
  - The Security Problem
  - Program Threats
  - System and Network Threats
  - Cryptography as a Security Tool
  - User Authentication
  - Implementing Security Defenses

PROGRAM THREATS

- Many variations, many names
- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - **Login Program, Spyware**, **pop-up browser windows**, **covert channels**
  - Up to 80% of spam delivered by spyware-infected systems
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?

PROGRAM THREATS (CONT.)

- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
- **Stack** and **Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

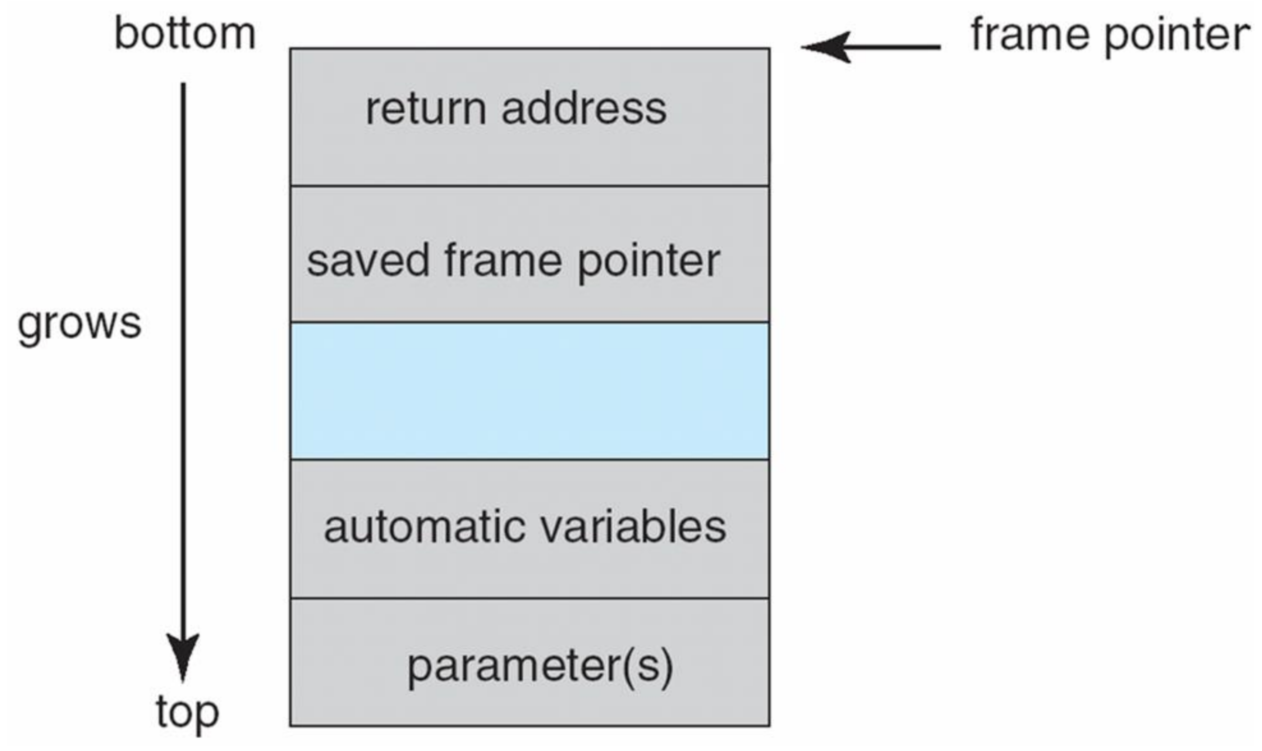
C PROGRAM WITH BUFFER-OVERFLOW CONDITION

```
#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;
   }
}
```

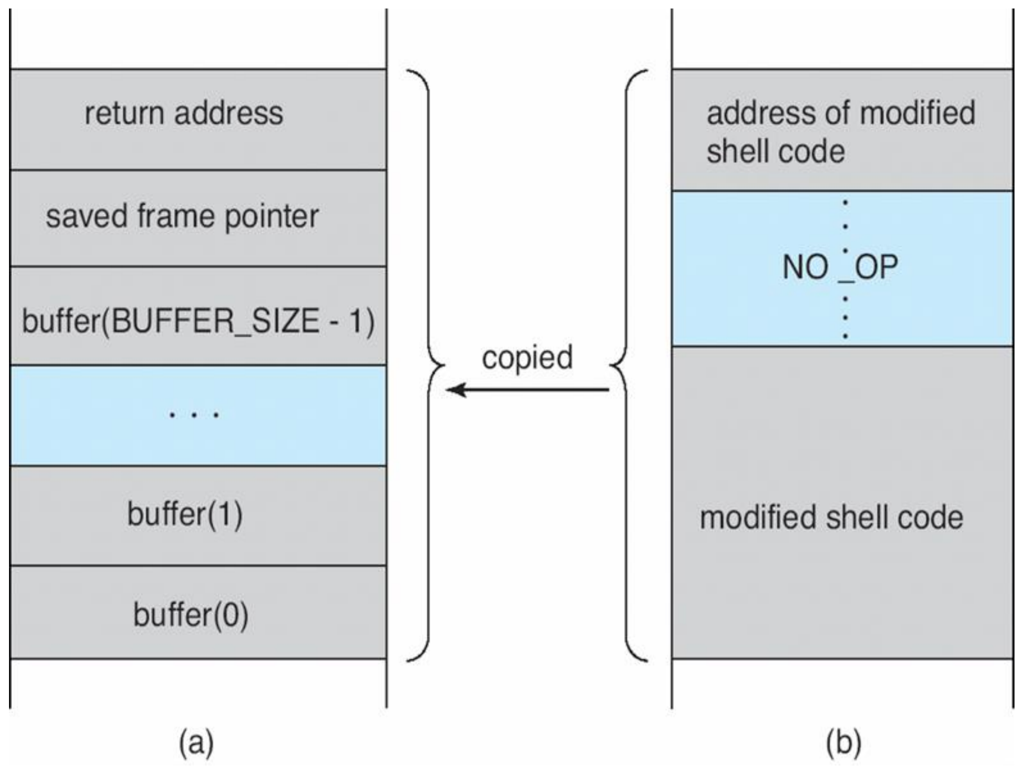
```
#include <stdio.h>
   } #include <stdio.h>
   #include <string.h>

   int main(void){
   char str1[4];
   char str2[30] = “insert long string";

   //Copy str2 into str1
   strcpy(str1, str2);
   printf("%s\n", str1);
   }
}
```

⇒ strcpy_c(target, rsize_t, source);

LAYOUT OF TYPICAL STACK FRAME



HYPOTHETICAL STACK FRAME



Before attack

After attack

PROGRAM THREATS (CONT.)

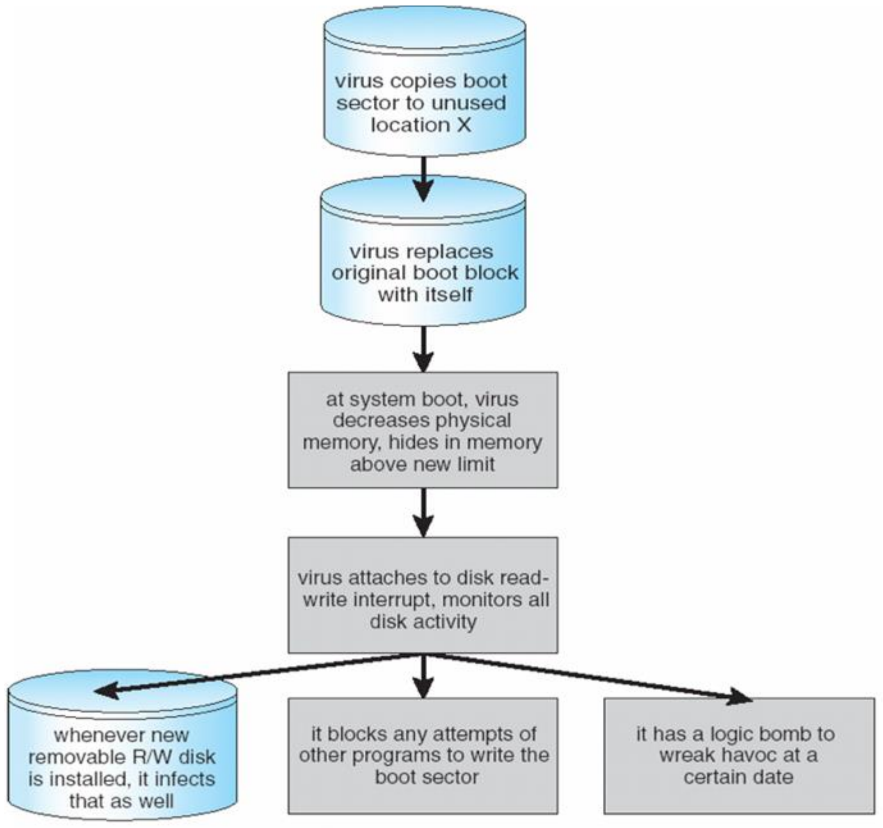
- **Viruses**
  - Code fragment embedded in legitimate program
  - Self-replicating, designed to infect other computers
  - Very specific to CPU architecture, operating system, applications
  - Usually borne via email or as a macro
  - Visual Basic Macro to reformat hard drive

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

PROGRAM THREATS (CONT.)

- Many categories of viruses, literally many thousands of viruses
  - File / parasitic
  - Boot / memory
  - Macro
  - Source code
  - Polymorphic
  - Encrypted
  - Stealth
  - Tunneling
  - Multipartite
  - Armored

A BOOT-SECTOR COMPUTER VIRUS


virus copies boot sector to unused location X
virus replaces original boot block with itself
at system boot, virus decreases physical memory, hides in memory above new limit
virus attaches to disk read-write interrupt, monitors all disk activity
whenever new removable R/W disk is installed, it infects that as well
it blocks any attempts of other programs to write the boot sector
it has a logic bomb to wreak havoc at a certain date

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
- Why is Windows the target for most attacks?
  - Most common
  - Everyone is an administrator
    - Licensing required?

OUTLINE

- Protection
- Security
  - The Security Problem
  - Program Threats
  - ➡ System and Network Threats
  - Cryptography as a Security Tool
  - User Authentication
  - Implementing Security Defenses

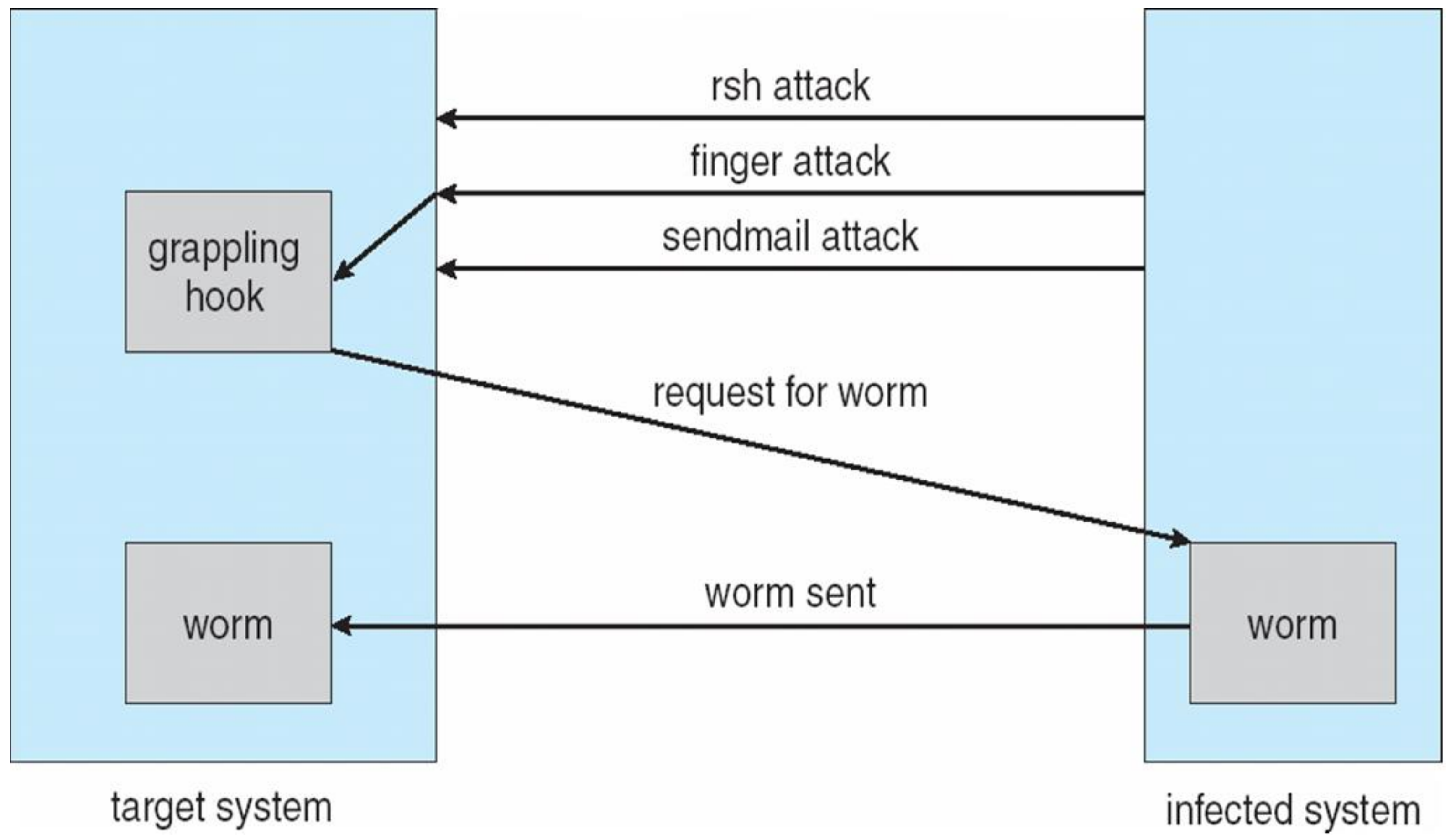
SYSTEM AND NETWORK THREATS

- Some systems “open” rather than **secure by default**
  - Reduce **attack surface**
  - But harder to use, more knowledge needed to administer
- Network threats harder to detect, prevent
  - Protection systems weaker
  - More difficult to have a shared secret on which to base access
  - No physical limits once system attached to internet
    - Or on network with system attached to internet
  - Even determining location of connecting system difficult
    - IP address is only knowledge

SYSTEM AND NETWORK THREATS (CONT.)

- **Worms** – use **spawn** mechanism; standalone program
- Internet worm
  - Exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs
  - Exploited trust-relationship mechanism used by *rsh* to access friendly systems without use of password
  - **Grappling hook** program uploaded main worm program
    - 99 lines of C code
  - Hooked system then uploaded main code, tried to attack connected systems
  - Also tried to break into other users accounts on local system via password guessing
  - If target system already infected, abort, except for every 7th time

THE MORRIS INTERNET WORM



SYSTEM AND NETWORK THREATS (CONT.)

- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
  - Detection of answering service protocol
  - Detection of 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
  - Frequently launched from **zombie systems**
    - To decrease trace-ability

SYSTEM AND NETWORK THREATS (CONT.)

- **Denial of Service**
  - Overload the targeted computer preventing it from doing any useful work
  - **Distributed denial-of-service** (**DDOS**) come from multiple sites at once
  - Consider the start of the IP-connection handshake (SYN)
    - How many started-connections can the OS handle?
  - Consider traffic to a web site
    - How can you tell the difference between being a target and being really popular?
  - Accidental – CS students writing bad `fork()` code
  - Purposeful – extortion, punishment

SOBIG.F WORM

- More modern example
- 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 innocuous subject lines, looking like it came from someone known
- Attachment was executable program that created **`WINPPR23.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
```

OUTLINE

- Protection
- Security
  - The Security Problem
  - Program Threats
  - System and Network Threats
  - ➡ Cryptography as a Security Tool
  - User Authentication
  - Implementing Security Defenses

CRYPTOGRAPHY

- Means to constrain potential senders (*sources*) and / or receivers (*destinations*) of *messages*
  - Based on secrets (**keys**)
  - Enables
    - Confirmation of source
    - Receipt only by certain destination
    - Trust relationship between sender and receiver

ENCRYPTION

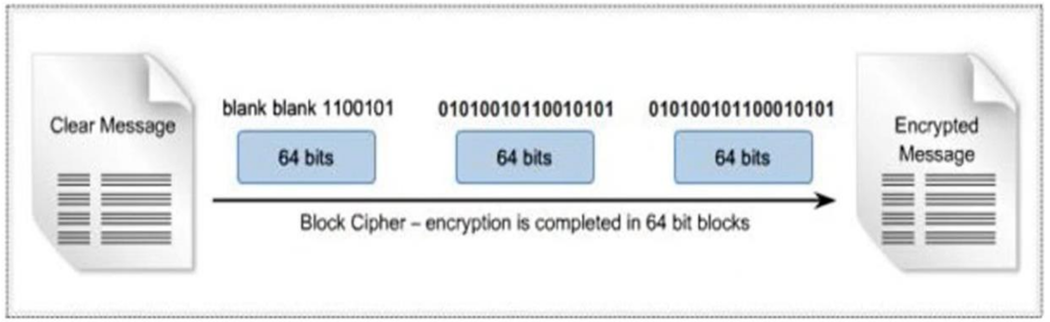
- Constrains the set of possible receivers of a message
- **Encryption** algorithm consists of
  - Set K of keys
  - Set M of Messages
  - Set C of ciphertexts (encrypted messages)
  - A function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, E_k is a function for generating ciphertexts from messages
    - Both E and E_k for any k should be efficiently computable functions
  - A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, D_k is a function for generating messages from ciphertexts
    - Both D and D_k for any k should be efficiently computable functions

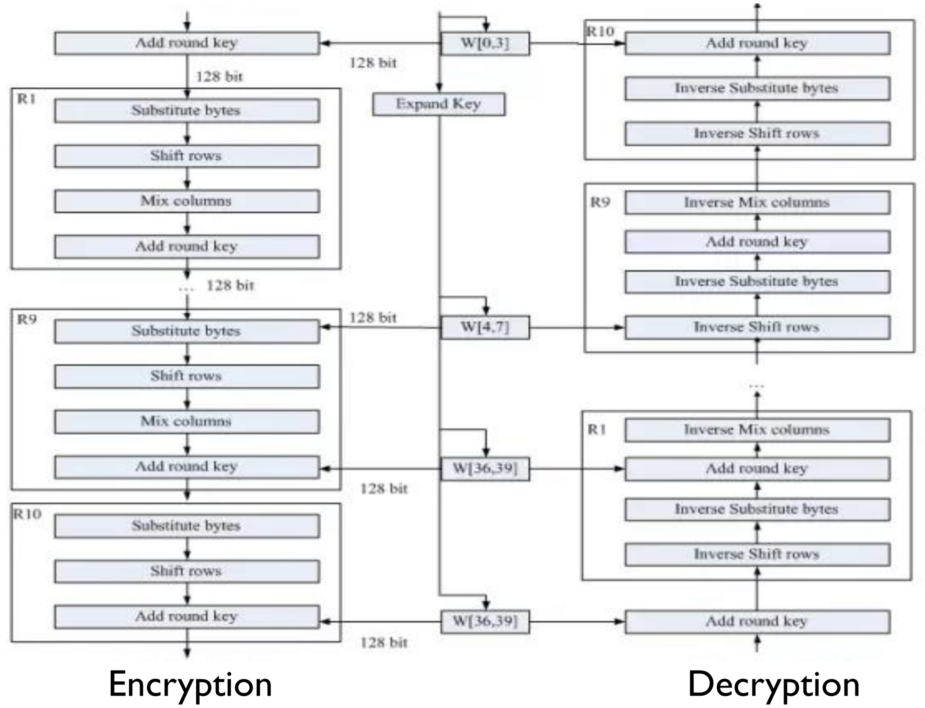
ENCRYPTION (CONT.)

- An encryption algorithm must provide this essential property: Given a ciphertext $c \in C$, a computer can compute m such that $E_k(m) = c$ only if it possesses k
  - Thus, a computer holding k can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding k cannot decrypt ciphertexts
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive k from the ciphertexts

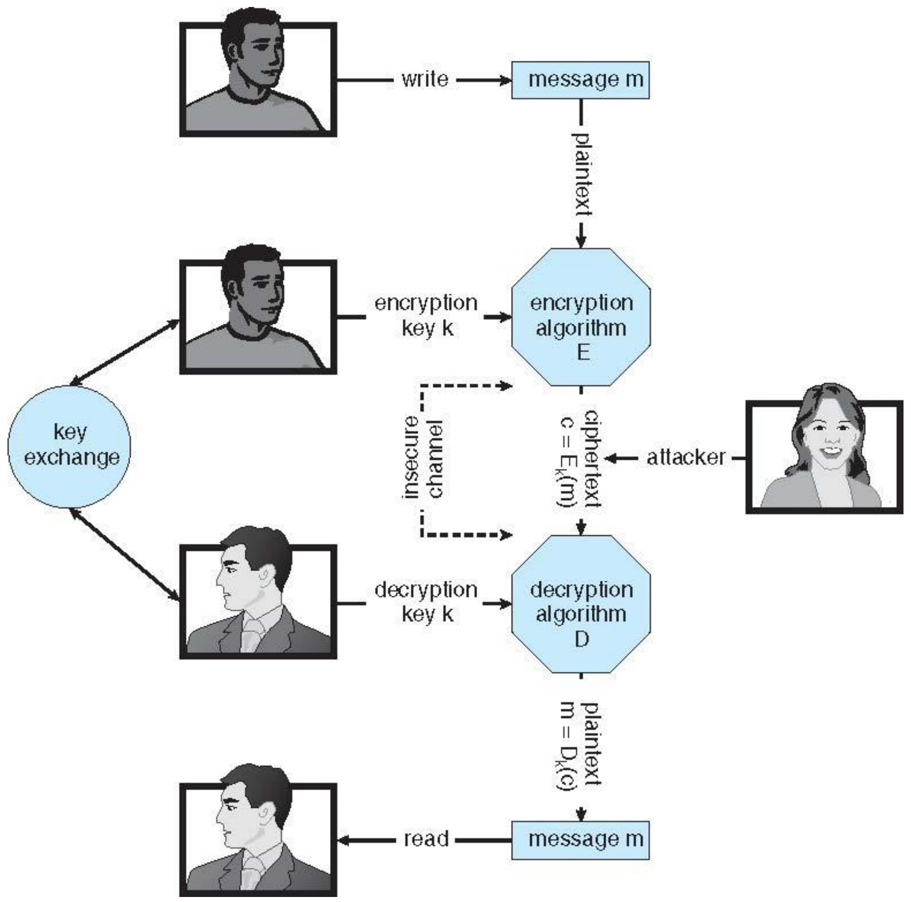
SYMMETRIC ENCRYPTION

- Same key used to encrypt and decrypt
  - Therefore *k* must be kept secret
- DES was most commonly used symmetric block-encryption algorithm (created by US Govt)
  - Encrypts a block of data at a time
  - Keys too short so now considered insecure
- Triple-DES considered more secure $c = E_{k3}(D_{k2}(E_{k1}(m)))$
  - Algorithm used 3 times using 2 or 3 keys
  - For example
- 2001 NIST adopted new block cipher - Advanced Encryption Standard (**AES**)
  - Keys of 128, 192, or 256 bits, works on 128 bit blocks
- RC4 is most common symmetric stream cipher, but known to have vulnerabilities
  - Encrypts/decrypts a stream of bytes (i.e., wireless transmission)
  - Key is a input to pseudo-random-bit generator
    - Generates an infinite **keystream**





SECURE COMMUNICATION OVER INSECURE MEDIUM



ASYMMETRIC ENCRYPTION

- **Public-key encryption** based on each user having two keys:
  - **public key** – published key used to encrypt data
  - **private key** – key known only to individual user used to decrypt data
- Must be an encryption scheme that can be made public without making it easy to figure out the decryption scheme
  - Most common is **RSA** block cipher
  - Efficient algorithm for testing whether or not a number is prime
  - No efficient algorithm is know for finding the prime factors of a number

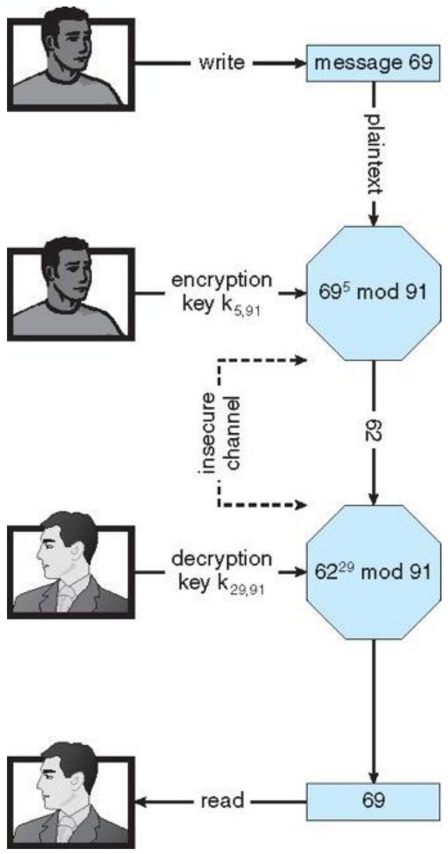
ASYMMETRIC ENCRYPTION (CONT.)

- Formally, it is computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, and so k_e need not be kept secret and can be widely disseminated
  - k_e is the **public key**
  - k_d is the **private key**
  - N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 512 bits each)
  - Encryption algorithm is $E_{ke,N}(m)$ = 69 ^{5} mod 91=>62, where k_e satisfies $k_e k_d$ mod $(p-1)(q-1) = 1$
  - The decryption algorithm is then $D_{kd,N}(c) = c^{k_d}$ mod N=62 ^{29} $mod(N)$=69

ASYMMETRIC ENCRYPTION EXAMPLE

- For example. make $p = 7$and $q = 13$
- We then calculate $N = 7*13 = 91$ and phi (n)=$(p-1)(q-1) = 72$
- We next select k_e relatively prime to 72 and< 72, yielding 5
- Finally, we calculate k_d such that $k_e k_d$ mod 72 = 1, yielding 29
- We how have our keys
  - Public key, $k_{e,N} = 5, 91$
  - Private key, $k_{d,N} = 29, 91$
- Encrypting the message 69 with the public key results in the cyphertext 62
- Cyphertext can be decoded with the private key
  - Public key can be distributed in cleartext to anyone who wants to communicate with holder of public key

ENCRYPTION USING RSA ASYMMETRIC CRYPTOGRAPHY



CRYPTOGRAPHY (CONT.)

- Note symmetric cryptography based on transformations, asymmetric based on mathematical functions
  - Asymmetric much more compute intensive
  - Typically not used for bulk data encryption

OUTLINE

- Protection
- Security
  - The Security Problem
  - Program Threats
  - System and Network Threats
  - Cryptography as a Security Tool
  - Authentication
  - Implementing Security Defenses

AUTHENTICATION

- Constraining set of potential senders of a message
  - Complementary to encryption
  - Also can prove message unmodified
- Algorithm components
  - A set K of keys
  - A set M of messages
  - A set A of authenticators
  - A function $S : K \rightarrow (M \rightarrow A)$
    - That is, for each $k \in K$, S_k is a function for generating authenticators from messages
    - Both S and S_k for any k should be efficiently computable functions
  - A function $V : K \rightarrow (M \times A \rightarrow \{\text{true}, \text{false}\})$. That is, for each $k \in K$, V_k is a function for verifying authenticators on messages
    - Both V and V_k for any k should be efficiently computable functions

AUTHENTICATION (CONT.)

- For a message *m*, a computer can generate an authenticator $a \in A$ such that $V_k(m, a)$ = `true` only if it possesses *k*
- Thus, computer holding *k* can generate authenticators on messages so that any other computer possessing *k* can verify them
- Computer not holding *k* cannot generate authenticators on messages that can be verified using V_k
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive *k* from the authenticators
- Practically, if $V_k(m,a)$ = **`true`** then we know *m* has not been modified and that send of message has *k*
  - If we share *k* with only one entity, know where the message originated

AUTHENTICATION – HASH FUNCTIONS

- Basis of authentication
- Creates small, fixed-size block =>**message digest (hash value)** from *m*
- Hash Function *H* must be collision resistant on *m*
  - Must be infeasible to find an $m' \neq m$ such that $H(m) = H(m')$
- If $H(m) = H(m')$, then $m = m'$
  - The message has not been modified
- Common message-digest functions include **MD5**, which produces a 128-bit hash, and **SHA-1**, which outputs a 160-bit hash
- Not useful as authenticators
  - For example *H(m)* can be sent with a message
    - But if *H* is known someone could modify *m* to *m'* and recompute *H(m')* and modification not detected
    - So must authenticate *H(m)*

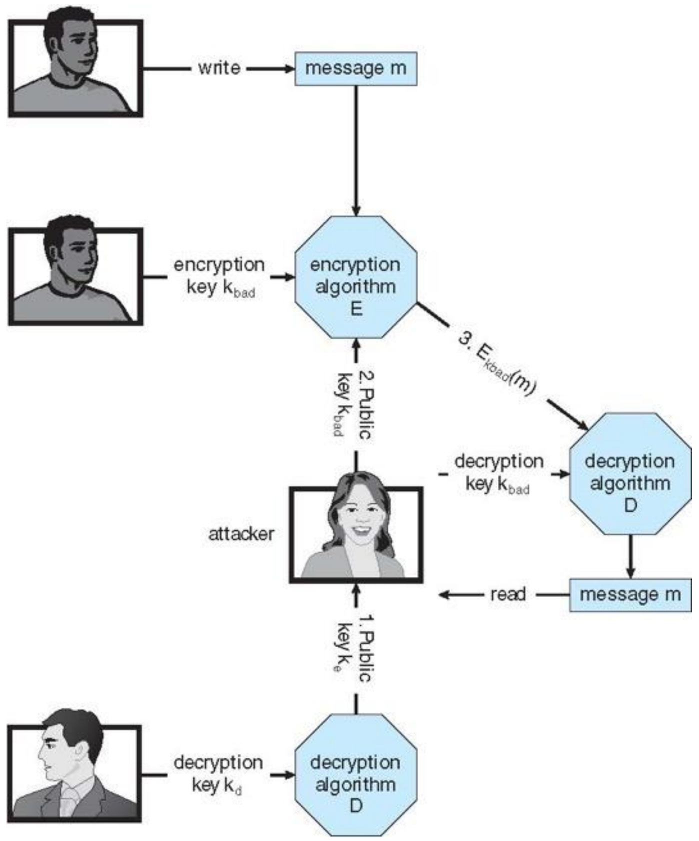
AUTHENTICATION – DIGITAL SIGNATURE

- Based on asymmetric keys and digital signature algorithm
- Authenticators produced are **digital signatures 5**
- Very useful – ***anyone*** can verify authenticity of a message
- In a digital-signature algorithm, computationally infeasible to derive k_s from k_v
  - V is a one-way function
  - Thus, k_v is the public key and k_s is the private key
- Consider the RSA digital-signature algorithm
  - Similar to the RSA encryption algorithm, but the key use is reversed
  - Digital signature of message $S_{ks}(m) = H(m)^{k_s} \bmod N$
  - The key k_s again is a pair (d, N), where N is the $\overset{?}{=}$product of two large, randomly chosen prime numbers p and q
  - Verification algorithm is $V_{kv}(m, a)$ $(a^{k_v} \bmod N = H(m))$
    - Where k_v satisfies $k_v k_s \bmod (p - 1)(q - 1) = 1$

AUTHENTICATION (CONT.)

- Why authentication if a subset of encryption?
  - Fewer computations (except for RSA digital signatures)
  - Authenticator usually shorter than message
  - Sometimes want authentication but not confidentiality
    - Signed patches et al
  - Can be basis for **non-repudiation**

MAN-IN-THE-MIDDLE ATTACK ON ASYMMETRIC CRYPTOGRAPHY



USER AUTHENTICATION

- Crucial to identify user correctly, as protection systems depend on user ID
- 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

OUTLINE

- Protection
- Security
  - The Security Problem
  - Program Threats
  - System and Network Threats
  - Cryptography as a Security Tool
  - User Authentication
  - Implementing Security Defenses

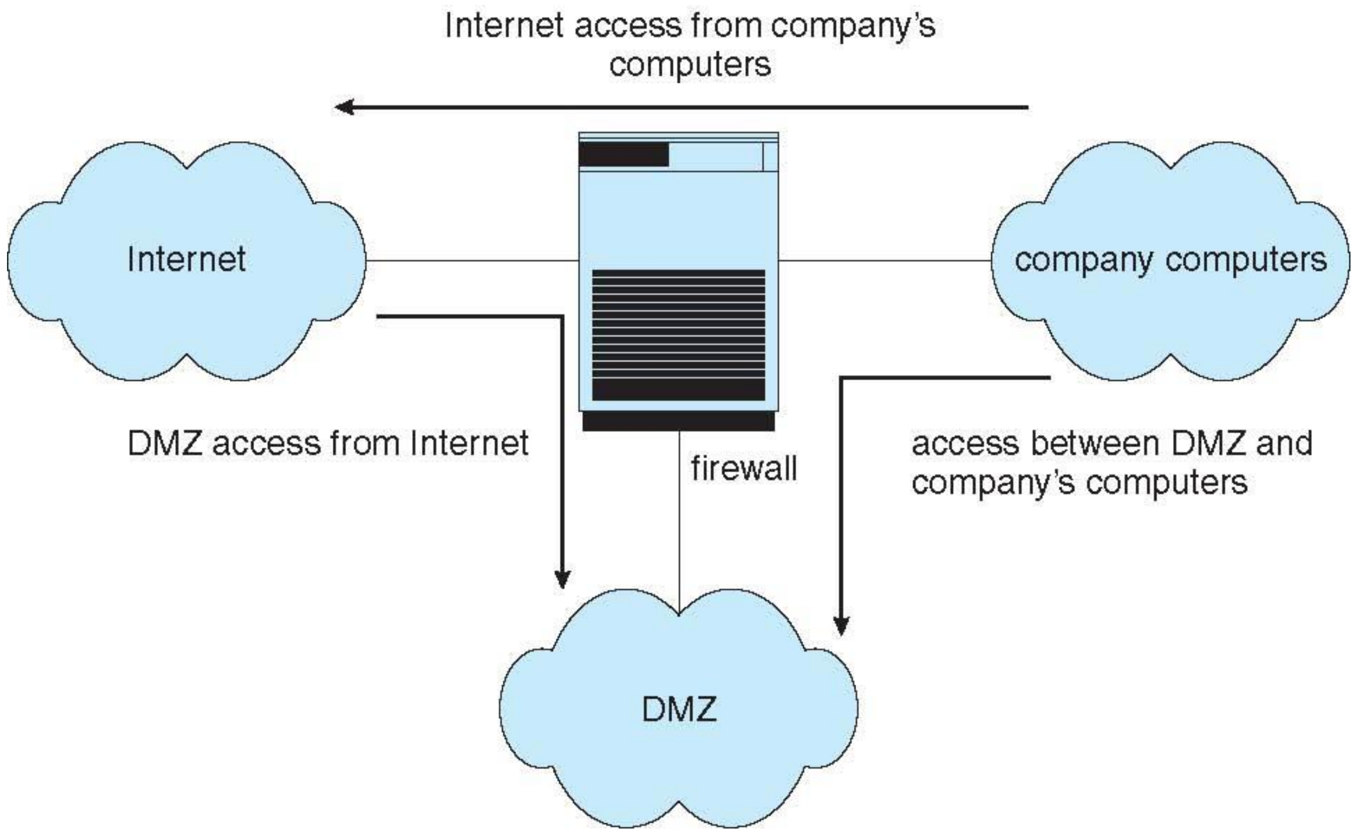
IMPLEMENTING SECURITY DEFENSES

- **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
  - Searching all programs or programs at execution for known virus patterns
  - Or run in **sandbox** so can't damage system
- Auditing, accounting, and logging of all or specific system or network activities
- Practice **safe computing** – avoid sources of infection, download from only "good" sites, etc

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


Internet access from company's computers
Internet
company computers
DMZ access from Internet
firewall
access between DMZ and company's computers
DMZ