Security and Protection

The Security Problem

- System **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **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 Violation Categories

- **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
- **Privilege escalation**
  - Common attack type with access beyond what a user or resource is supposed to have

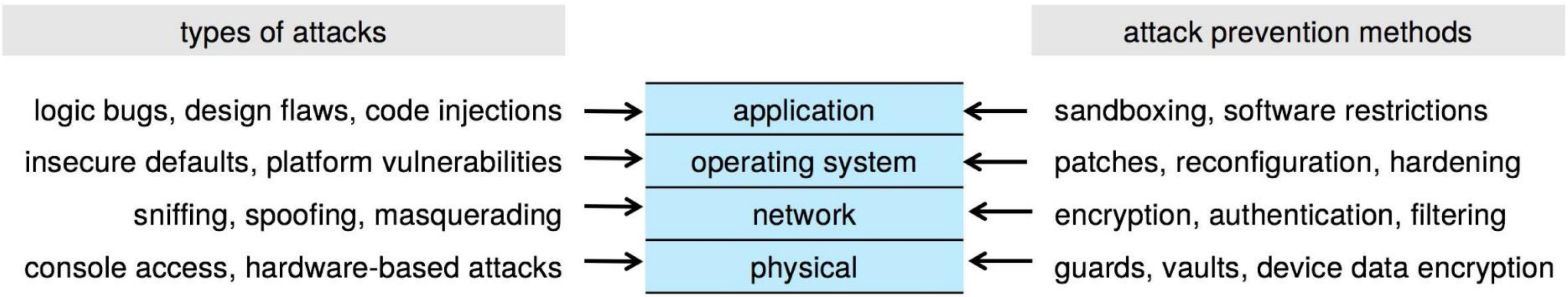
Security Measure Levels

- 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
  - **Application**
    - Benign or malicious apps can cause security problems
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- Humans a risk too via **phishing** and **social-engineering** attacks
- But can too much security be a problem?

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
  - **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?

Four-layered Model of Security



Program Threats (Cont.)

- **Malware -** Software designed to exploit, disable, or damage computer
- **Trojan Horse** – Program that acts in a clandestine manner
  - **Spyware** – Program frequently installed with legitimate software to display adds, capture user data
  - **Ransomware** – locks up data via encryption, demanding payment to unlock it
- Others include trap doors, logic boms
- All try to violate the Principle of Least Privilege

THE PRINCIPLE OF LEAST PRIVILEGE

"The principle of least privilege. Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. The purpose of this principle is to reduce the number of potential interactions among privileged programs to the minimum necessary to operate correctly, so that one may develop confidence that unintentional, unwanted, or improper uses of privilege do not occur."—Jerome H. Saltzer, describing a design principle of the Multics operating system in 1974: https://pdfs.semanticscholar.org/1c8d/06510ad449ad24fbdd164f8008cc730cab47.pdf.

- Goal frequently is to leave behind Remote Access Tool (RAT) for repeated access

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;
   }
}
```

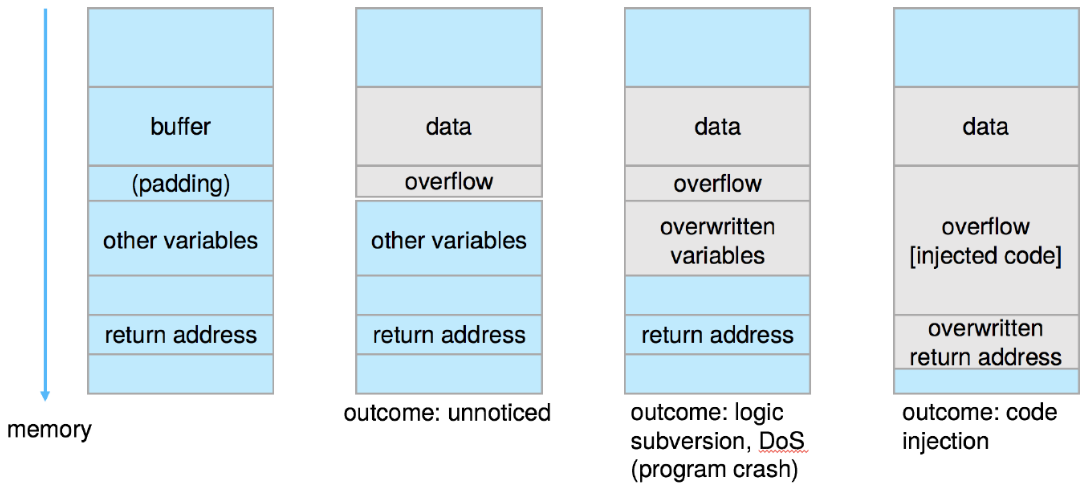
- **Code review** can help – programmers review each other's code, looking for logic flows, programming flaws

Code Injection

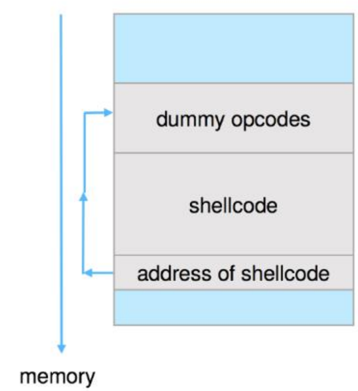
- **Code-injection attack** occurs when system code is not malicious but has bugs allowing executable code to be added or modified
  - Results from poor or insecure programming paradigms, commonly in low level languages like C or C++ which allow for direct memory access through pointers
  - Goal is a buffer overflow in which code is placed in a buffer and execution caused by the attack
  - Can be run by script kiddies – use tools written but exploit identifiers

Code Injection (Cont.)

- Outcomes from code injection



- Frequently use trampoline to code execution to exploit buffer overflow:



Great Programming Required?

- For the first step of determining the bug, and second step of writing exploit code, yes
- **Script kiddies** can run pre-written exploit code to attack a given system
- Attack code can get a shell with the processes' owner's permissions
  - Or open a network port, delete files, download a program, etc.
- Depending on bug, attack can be executed across a network using allowed connections, bypassing firewalls
- Buffer overflow can be disabled by disabling stack execution or adding bit to page table to indicate "non-executable" state
  - Available in SPARC and x86
  - But still have security exploits

Program Threats (Cont.)

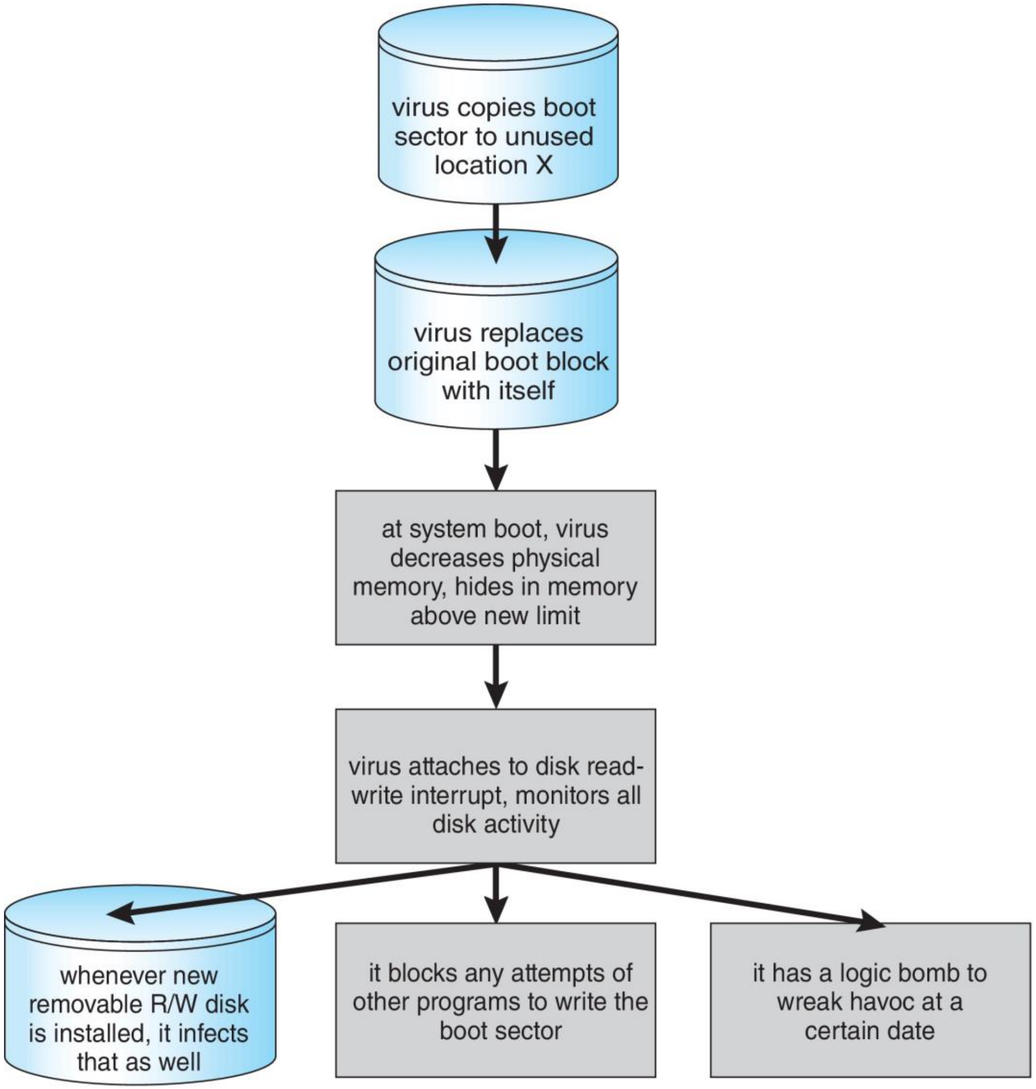
- **Viruses**
  - Code fragment embedded in legitimate program
  - Self-replicating, designed to infect other computers
  - Very specific to CPU architecture, operating system, applications
  - Usually borne via email or as a macro
  - Visual Basic Macro to reformat hard drive

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

Program Threats (Cont.)

- **Virus dropper** inserts virus onto the system
- Many categories of viruses, literally many thousands of viruses
  - File / parasitic
  - Boot / memory
  - Macro
  - Source code
  - Polymorphic to avoid having a **virus signature**
  - Encrypted
  - Stealth
  - Tunneling
  - Multipartite
  - Armored

A Boot-sector Computer Virus



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?
  - **Monoculture** considered harmful

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

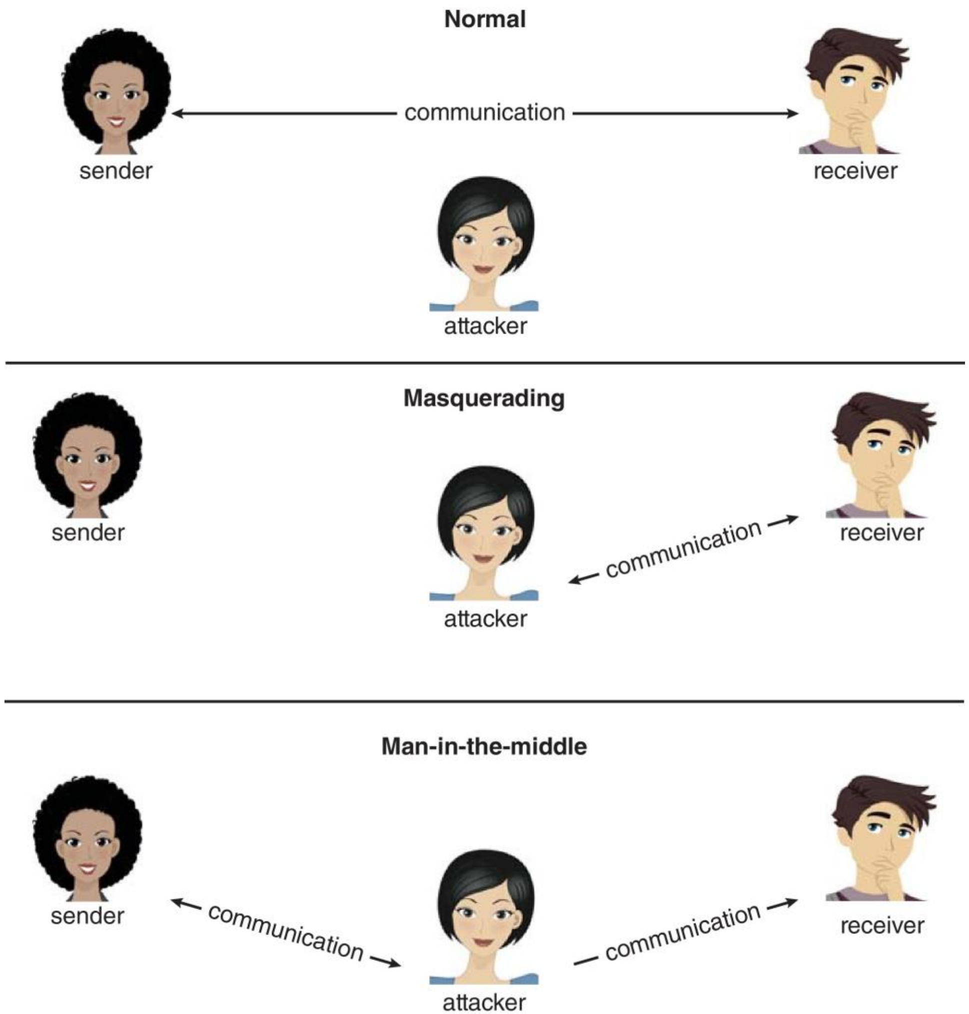
System and Network Threats (Cont.)

- **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
- Port scanning
  - Automated tool to look for network ports accepting connections
  - Used for good and evil

Standard Security Attacks



Cryptography as a Security Tool

- Broadest security tool available
  - Internal to a given computer, source and destination of messages can be known and protected
    - OS creates, manages, protects process IDs, communication ports
  - Source and destination of messages on network cannot be trusted without cryptography
    - Local network – IP address?
      - Consider unauthorized host added
    - WAN / Internet – how to establish authenticity
      - Not via IP address

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

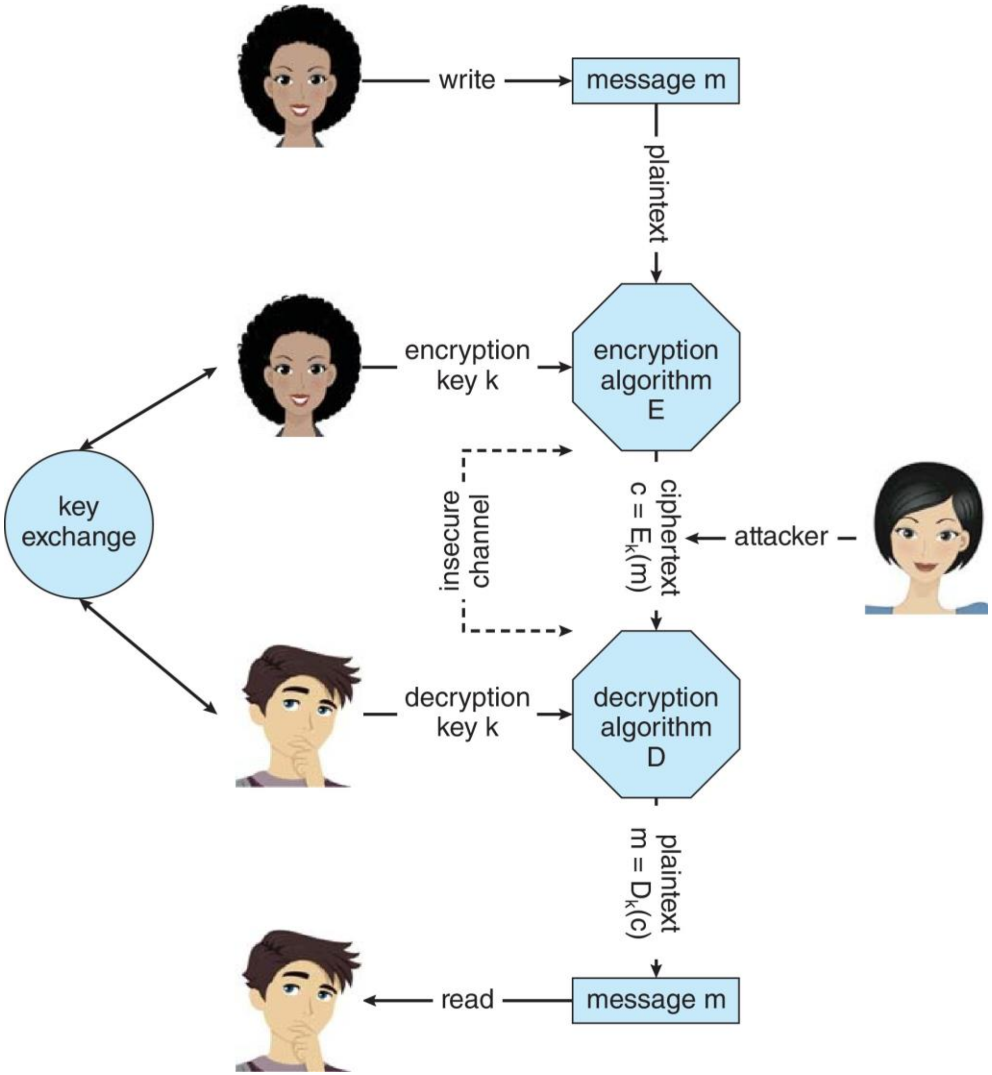
Encryption (Cont.)

- 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
  - Algorithm used 3 times using 2 or 3 keys
  - For example $c = E_{k3}(D_{k2}(E_{k1}(m)))$
- 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

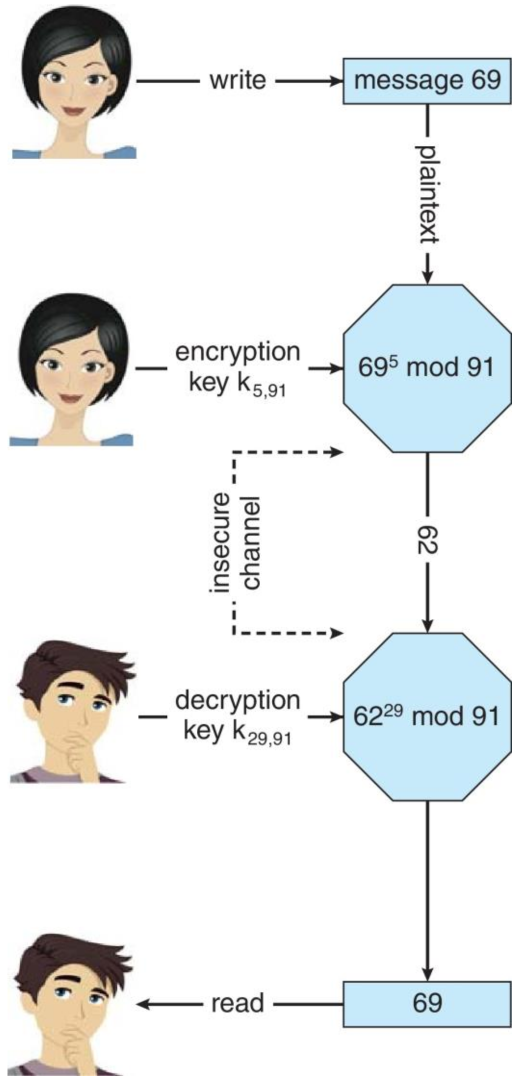
Asymmetric Encryption (Cont.)

- 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 $E_{ke,N}(m) = m^{k_e} \bmod N$, where k_e satisfies $k_e k_d \bmod (p-1)(q-1) = 1$
  - The decryption algorithm is then $D_{kd,N}(c) = c^{k_d} \bmod N$

Asymmetric Encryption Example

- For example. make p = 7and q = 13
- We then calculate $N = 7*13 = 91$ and $(p-1)(q-1) = 72$
- We next select k_e relatively prime to 72 and< 72, yielding 5
- Finally, we calculate k_d such that $k_e k_d \bmod 72 = 1$, 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

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 $V : K \rightarrow (M \times A \rightarrow \{\text{true, false}\})$. 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 of data **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 - MAC

- Symmetric encryption used in **message-authentication code** (**MAC**) authentication algorithm
- Cryptographic checksum generated from message using secret key
  - Can securely authenticate short values
- If used to authenticate *H(m)* for an *H* that is collision resistant, then obtain a way to securely authenticate long message by hashing them first
- Note that *k* is needed to compute both S_k and V_k, so anyone able to compute one can compute the other

Authentication – Digital Signature

- Based on asymmetric keys and digital signature algorithm
- Authenticators produced are **digital signatures**
- 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 product of two large, randomly chosen prime numbers p and q
  - Verification algorithm is $V_{kv}(m, a) \quad (a^{k_v} \bmod N = H(m))$
    - 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**

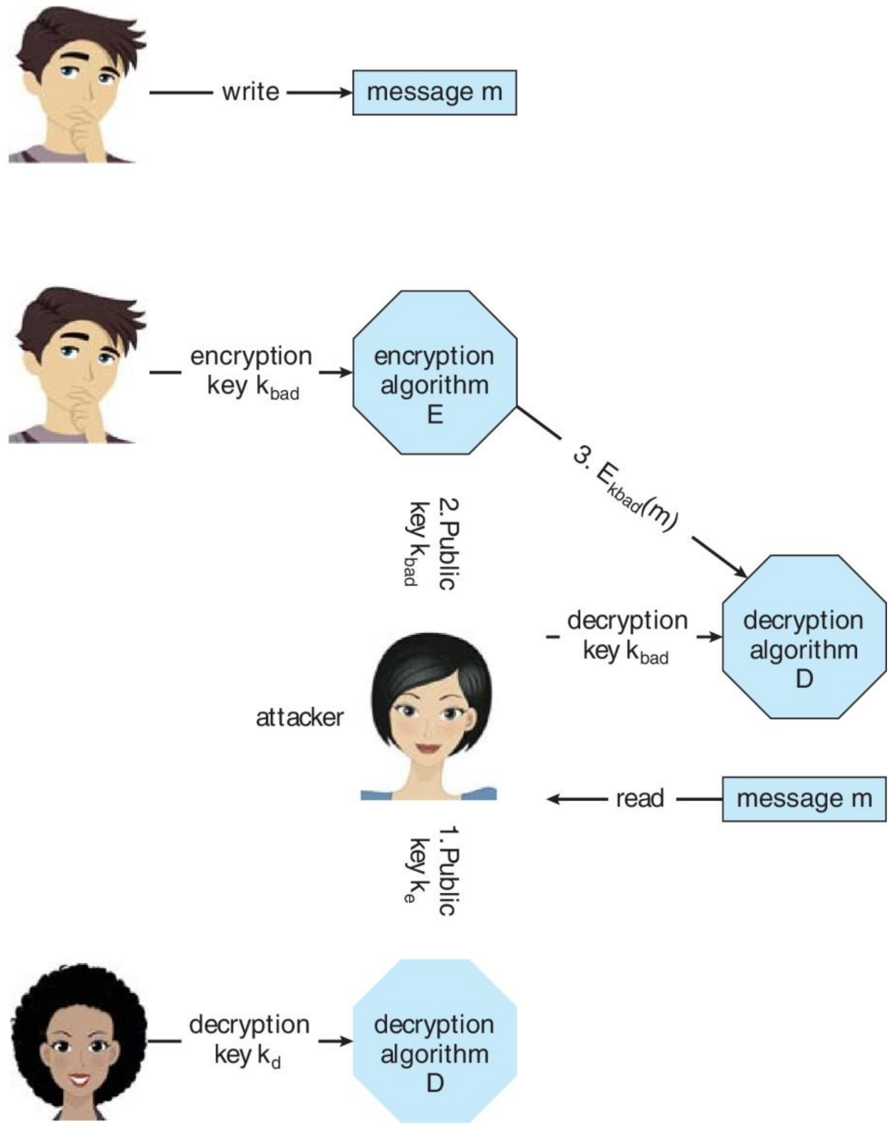
Key Distribution

- Delivery of symmetric key is huge challenge
  - Sometimes done **out-of-band**
- Asymmetric keys can proliferate – stored on **key ring**
  - Even asymmetric key distribution needs care – man-in-the-middle attack

Digital Certificates

- Proof of who or what owns a public key
- Public key digitally signed a trusted party
- Trusted party receives proof of identification from entity and certifies that public key belongs to entity
- **Certificate authority** are trusted party – their public keys included with web browser distributions
  - They vouch for other authorities via digitally signing their keys, and so on

Man-in-the-middle Attack on Asymmetric Cryptography



Implementation of Cryptography

- Can be done at various **layers** of ISO Reference Model
  - SSL at the Transport layer
  - Network layer is typically **IPSec**
    - **IKE** for key exchange
    - Basis of **Virtual Private Networks** (**VPNs**)
- Why not just at lowest level?
  - Sometimes need more knowledge than available at low levels
    - i.e., User authentication
    - i.e., e-mail delivery

OSI model

7. Application Layer

NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · SMPP · SMTP · SNMP · Telnet · Netconf · (more)

6. Presentation Layer

MIME · XDR · TLS · SSL

5. Session Layer

Named Pipes · NetBIOS · SAP · L2TP · PPTP · SPDY

4. Transport Layer

TCP · UDP · SCTP · DCCP · SPX

3. Network Layer

IP (IPv4, IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk

2. Data Link Layer

ATM · SDLC · HDLC · ARP · CSLIP · SLIP · GFP · PLIP · IEEE 802.3 · Frame Relay · ITU-T G.hn DLL · PPP · X.25 · Network Switch · DHCP

1. Physical Layer

EIA/TIA-232 · EIA/TIA-449 · ITU-T V-Series · I.430 · I.431 · POTS · PDH · SONET/SDH · PON · OTN · DSL · IEEE 802.3 · IEEE 802.11 · IEEE 802.15 · IEEE 802.16 · IEEE 1394 · ITU-T G.hn PHY · USB · Bluetooth · Hubs

This box: **view** · talk · edit

OSI Model

	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
		5. Session	Interhost communication
	Segments	4. Transport	End-to-end connections and reliability, flow control
Media layers	Packet/Datagram	3. Network	Path determination and logical addressing
	Frame	2. Data Link	Physical addressing
	Bit	1. Physical	Media, signal and binary transmission

Source: http://en.wikipedia.org/wiki/OSI_model

Encryption Example - TLS

- Insertion of cryptography at one layer of the ISO network model (the transport layer)
- SSL – Secure Socket Layer (also called TLS)
- 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)
- The server is verified with a **certificate** assuring client is talking to correct server
- Asymmetric cryptography used to establish a secure **session key** (symmetric encryption) for bulk of communication during session
- Communication between each computer then uses symmetric key cryptography
- More details in textbook

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

Passwords (Cont.)

STRONG AND EASY TO REMEMBER PASSWORDS

It is extremely important to use strong (hard to guess and hard to shoulder surf) passwords on critical systems like bank accounts. It is also important to not use the same password on lots of systems, as one less important, easily hacked system could reveal the password you use on more important systems. A good technique is to generate your password by using the first letter of each word of an easily remembered phrase using both upper and lower characters with a number or punctuation mark thrown in for good measure. For example, the phrase "My girlfriend's name is Katherine" might yield the password "Mgn.isK!". The password is hard to crack but easy for the user to remember. A more secure system would allow more characters in its passwords. Indeed, a system might also allow passwords to include the space character, so that a user could create a **passphrase** which is easy to remember but difficult to break.

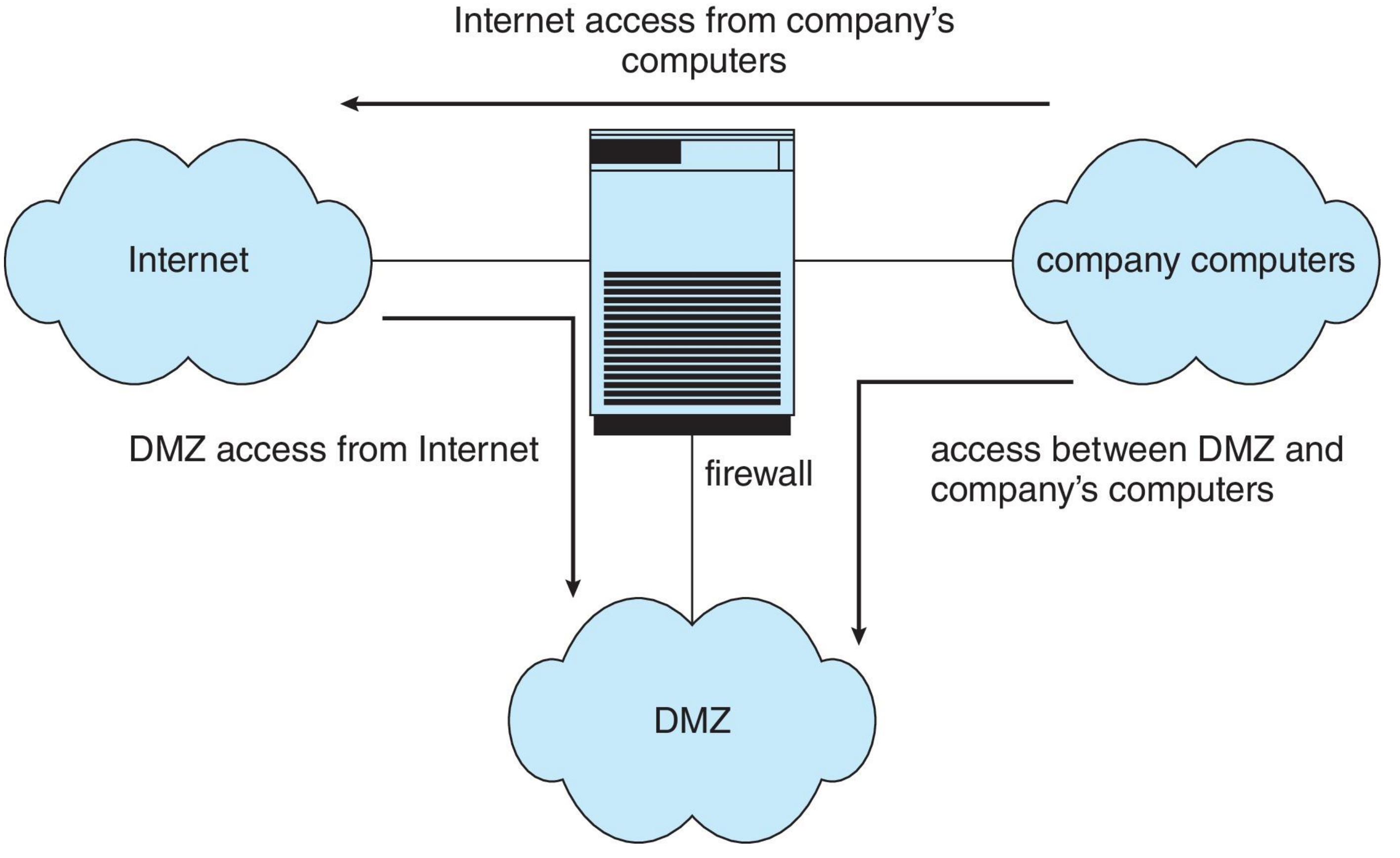
Implementing Security Defenses

- **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



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

Security Defenses Summarized

- By applying appropriate layers of defense, we can keep systems safe from all but the most persistent attackers. In summary, these layers may include the following:
  - Educate users about safe computing—don’t attach devices of unknown origin to the computer, don’t share passwords, use strong passwords, avoid falling for social engineering appeals, realize that an e-mail is not necessarily a private communication, and so on
  - Educate users about how to prevent phishing attacks—don’t click on email attachments or links from unknown (or even known) senders; authenticate (for example, via a phone call) that a request is legitimate
  - Use secure communication when possible
  - Physically protect computer hardware
  - Configure the operating system to minimize the attack surface; disable all unused services
  - Configure system daemons, privileges applications, and services to be as secure as possible

Security Defenses Summarized (Cont.)

- Use modern hardware and software, as they are likely to have up-to-date security features
- Keep systems and applications up to date and patched
- Only run applications from trusted sources (such as those that are code signed)
- Enable logging and auditing; review the logs periodically, or automate alerts
- Install and use antivirus software on systems susceptible to viruses, and keep the software up to date
- Use strong passwords and passphrases, and don't record them where they could be found
- Use intrusion detection, firewalling, and other network-based protection systems as appropriate
- For important facilities, use periodic vulnerability assessments and other testing methods to test security and response to incidents

Security Defenses Summarized (Cont.)

- Encrypt mass-storage devices, and consider encrypting important individual files as well
- Have a security policy for important systems and facilities, and keep it up to date

Example: Windows 10

- Security is based on **user accounts**
  - Each user has unique security ID
  - Login to ID creates **security access token**
    - Includes security ID for user, for user's groups, and special privileges
    - Every process gets copy of token
    - System checks token to determine if access allowed or denied
- Uses a **subject** model to ensure access security
  - A subject tracks and manages permissions for each program that a user runs
- Each object in Windows has a security attribute defined by a security descriptor
  - For example, a file has a **security descriptor** that indicates the access permissions for all users

Example: Windows 7 (Cont.)

- Win added mandatory integrity controls – assigns **integrity label** to each securable object and subject
  - Subject must have access requested in discretionary access-control list to gain access to object
- Security attributes described by security descriptor
  - Owner ID, group security ID, discretionary access-control list, system access-control list
- Objects are either **container objects** (containing other objects, for example a file system directory) or **noncontainer objects**
  - By default an object created in a container inherits permissions from the parent object
- Some Win 10 security challenges result from security settings being weak by default, the number of services included in a Win 10 system, and the number of applications typically installed on a Win 10 system

Goals of Protection

- A computer 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
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so

Principles of least privilege

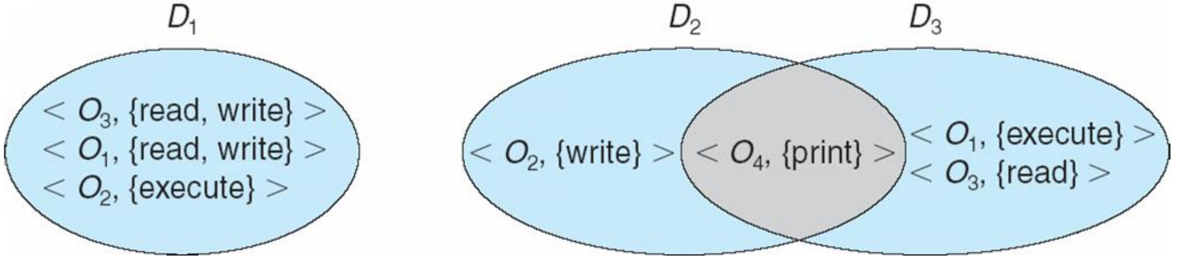
- Programs, users and systems should be given just enough **privileges** to perform their tasks
- Limits damage if entity has a bug, gets abused
- Can be static (during life of system, during life of process)
- Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
- "**Need to know**" a similar concept regarding access to data

Principles of Protection

- 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
    - Access Control List (ACL) lists,
    - Role Based Access Control (RBAC)
- Domain can be user, process, procedure

Domain Structure

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



Access Matrix

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

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	

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 the access column for that object
- Example

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	

Use of Access Matrix

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

Access Matrix with *Copy* Rights

- A process executing in Domain D_2 can copy the read access to file object F_2 to domain D_3

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 *owner* Rights

- A process executing in Domain D_2 can create the write access-right to file F_2 to domain D_2 and D_3

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 Domains as Objects

- A process executing in Domain D_2 can switch to domain D_3

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			

Mechanism and Policy

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

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 -> will not 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

Implementation: Option 2 (Cont.)

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

    Domain 1 = Read, Write
    Domain 2 = Read
    Domain 3 = Read

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

    Object F1 – Read

    Object F4 – Read, Write, Execute

    Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix (Cont.)

- Option 3 – Capability list for domains
  - Instead of list being 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

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

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			

Use of Access Matrix

- 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

References

- Operating Systems Concepts by Silberschatz, Galvin, and Gagne