

Security and Protection

DM510 Operating Systems

Lars Rohwedder



Windows



macOS



iOS

Disclaimer

These slides contain (modified) content and media from the official Operating System Concepts slides:
<https://www.os-book.com/OS10/slide-dir/index.html>

Today's lecture

- > Chapters 16+17 of course book

Security Overview

Security violations

Typical goals of attackers:

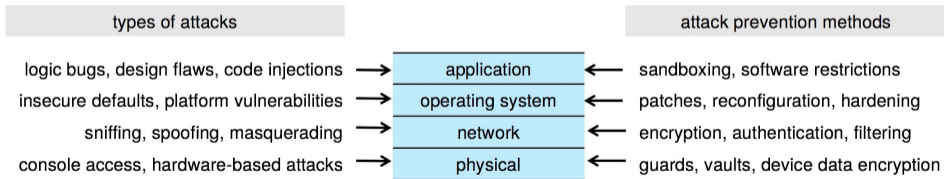
- > **Breach of confidentiality:** Unauthorized reading of data
- > **Breach of integrity:** Unauthorized modification of data
- > **Breach of availability:** Unauthorized destruction of data
- > **Theft of service:** Unauthorized use of resources
- > **Denial of service (DOS):** Prevention of legitimate use

Security measures

- > Absolute security impossible \rightsquigarrow make cost to perpetrator sufficiently high
- > Secure all four layers: security as weak as weakest link of the chain

Four layers of security

- > **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



Application Layer

Malicious programs

Typical goals of malicious programs:

- > **Spyware:** display ads, sniff sensitive data (passwords, etc.)
- > **Ransomware:** encrypt data to get ransom
- > Gain control of system for distributed DOS attacks, bitcoin mining ...

Malicious programs

Typical goals of malicious programs:

- > **Spyware:** display ads, sniff sensitive data (passwords, etc.)
- > **Ransomware:** encrypt data to get ransom
- > Gain control of system for distributed DOS attacks, bitcoin mining ...

Types of malicious programs:

- > **Trojan horse:** malicious code is part of seemingly useful program
- > **Virus/worm:** malicious code self-replicates by attaching itself to other programs, sometimes across networks

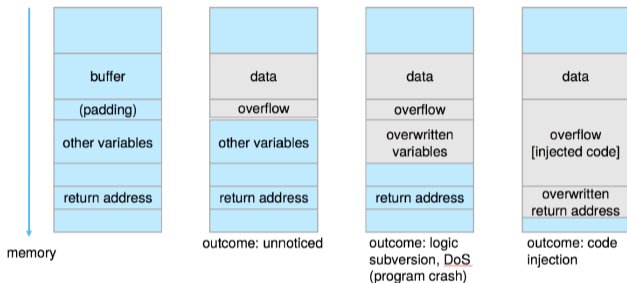


Code injections

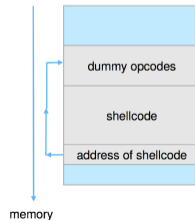
Program is not malicious, but attackers may still be able to make it execute their code

Buffer overflow

- > Program does not check buffer limits and writes beyond it
- > Can lead to input being written over other data or even return addresses



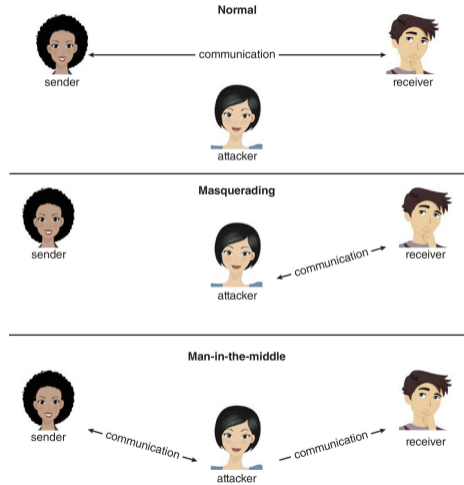
- > Attackers may redirect return address to memory segment they control and execute arbitrary code
- > **Prevention:** better code practices, safer languages or checks inserted by compiler, bits in page table for non-executable data



Network Layer

Types of network attacks

- > We generally cannot trust network channels
- > Attacker may listen to communication
- > **Masquerading**: attacker pretends to be someone else
- > **Man-in-the-middle**: attacker pretends to be opposite endpoint to sender and receiver and forwards (modified) communication between them
- > Main tool for security: **Cryptography**



Cryptography

Goal: authenticate (confirm correct identity) or ensure plaintext is received only by correct destination

Given

- > **Message set M :** all possible plaintexts, for example, blocks of 1024 bytes
- > **Cypher set C :** all possible encrypted messages, for example, blocks of 1024 bytes
- > **Keys K :** data used to encrypt/decrypt

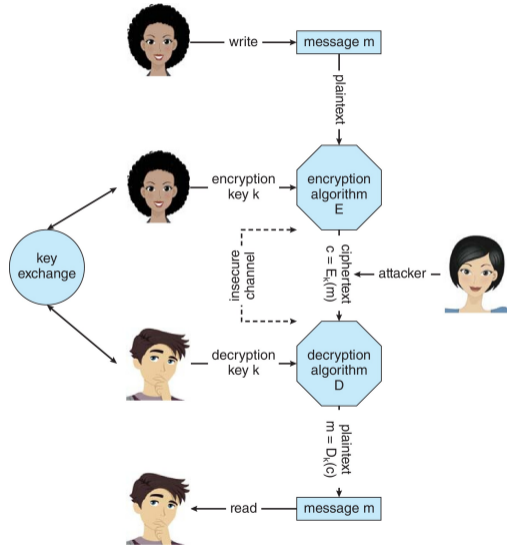
Basic functions

- > **Encryption:** for each key $k \in K$, we obtain encryption function $E_k : M \rightarrow C$
- > **Decryption:** for each key $k \in K$, we obtain decryption function $D_k : C \rightarrow M$

Crucial assumption: $E_k(m)$ and $D_k(c)$ can only be computed if k known

Symmetric encryption

- > Same key is used for encryption and decryption:
 $D_k(E_k(m)) = m$
- > Sender encrypts and transmits cypher, receiver decrypts
- > Only cipher is transmitted, so attacker cannot recover message
- > **Applications:** hard drive encryption, end-to-end encryption across networks
- > But how to safely perform key exchange?



Asymmetric encryption

- > Pair of keys: **public key** and **private key**. One to encrypt, other to decrypt
- > Relies on advanced algorithms, for example RSA

Authentication

Alice wants to sign message m proving that she is origin

- > Alice has private key k_1
- > Public key k_2 is known by everyone
- > $D_{k_2}(E_{k_1}(m)) = m$
- > Only Alice is able to construct the cypher $c = E_{k_1}(m)$ and everyone can verify it by using the public key

Restricting access to message

Alice wants to send message to Bob that only Bob can read

- > Bob has private key k_1
- > Public key k_2 is known by everyone
- > $D_{k_1}(E_{k_2}(m)) = m$
- > Alice sends cypher $c = E_{k_2}(m)$ and only Bob can decrypt it

Hash functions

- > Function $H : M \rightarrow C$ (like encryption, but it is not meant to be decrypted)
- > Usually hashed values C are much smaller than M
- > Should be hard to invert: for $c \in C$ finding some $m \in M$ with $H(m) = c$

Signatures of hashes

- > Instead of encrypting entire message for authentication, encrypt only hash of message
- > Others can also compute hash and check it against decrypted cypher
- > More efficient (fewer bytes to encrypt/decrypt) and message can be transmitted in plaintext

Hashed passwords

- > Storing or transmitting plaintext passwords should be avoided
- > Safer: use hashed password
- > Even safer: use hash of password with added **salt**: add random “salt” bytes to password (stored with hash). Prevents attacker from trying through list of known hashes

Operating System Layer

Protection

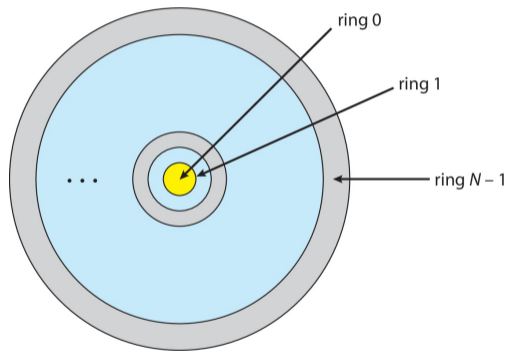
- > **Protection problem:** Ensure that each object (hardware or software) is accessed correctly and only by those processes that are allowed to do so
- > **Mechanism:** how something should be done
- > **Policy:** what should be done
- > Mechanism and policy should be separated for flexibility

Best practices

- > **Principle of least privilege:** give each process and user only enough privileges to complete its task
- > **Need-to-know principle:** only the information required for its task should be shared with process (similar to principle of least privilege)
- > **Compartmentalization:** protect each system component by permissions and access restrictions

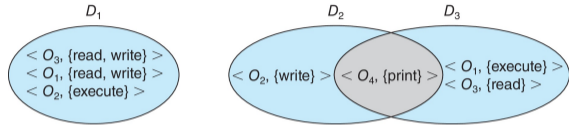
Protection rings

- > Hierarchical abstraction for different privilege levels
- > Process runs at specific ring level
- > Innermost ring corresponds to having all privileges
- > Outer rings have fewer privileges than inner rings
- > Often requires hardware support, for example, user mode and kernel mode



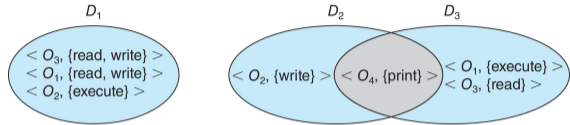
Protection domains

- > Processes operate with privileges of specific domain
- > Domain has rights over various operations for each object
- > In **dynamic** setting, processes can switch domains
- > More fine-grained than hierarchical protection rings



Protection domains

- > Processes operate with privileges of specific domain
- > Domain has rights over various operations for each object
- > In **dynamic** setting, processes can switch domains
- > More fine-grained than hierarchical protection rings



Internal representation of protection domains:

- > Arbitrary policies can be specified via an **access matrix**
- > Storing the entire matrix explicitly usually not efficient, since it is very sparse. Alternatives: capability-list for objects (list of domains and allowed operations) or capability-lists for domains

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			

File and device privileges in Unix

- > Each file belongs to a specific user (“owner”) and a specific group (“owning group”). Even on systems used by only one person, there is usually at least one regular user and one superuser (root, admin, etc.)
- > Different permissions to read/write/execute a file are given to owner, users of file’s group, and other users. Sometimes indicated by three tripels:

rw-rw-rw-

Forbidden actions, are replaced by '-'. For example, only owner can execute:

rw-rw--

We can also write decimal number based on the three bits. Previous example:

766

Explanation: $7 = 111 = \text{rwx}$ and $6 = 110 = \text{rw-}$

- > Permissions and ownership can be changed by **chmod**, **chown**, and **chgrp**