Protection
Goals of Protection

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

- Guiding principle – principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks
Domain Structure

- Access-right = \( <\text{object-name}, \text{rights-set}> \)
  where \( \text{rights-set} \) is a subset of all valid operations that can be performed on the object.

- Domain = set of access-rights
Domain Implementation (UNIX)

- System consists of 2 domains:
  - User
  - Supervisor

- UNIX
  - Domain = user-id
  - Domain switch accomplished via file system
    - Each file has associated with it 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
## Access Matrix

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td></td>
<td>read</td>
<td></td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>print</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td></td>
<td>read</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td></td>
<td>read</td>
<td>write</td>
<td></td>
<td>read</td>
</tr>
</tbody>
</table>
Access Matrix

- Access matrix design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If 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
## Access Matrix With Domains as Objects

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>laser printer</th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>read</td>
<td>read</td>
<td></td>
<td></td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td>print</td>
<td></td>
<td>switch</td>
<td>switch</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>read write</td>
<td>read write</td>
<td></td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access Matrix with Copy Rights

(a)

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>execute</td>
<td></td>
<td>write*</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>execute</td>
<td></td>
<td>write*</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td>execute</td>
<td>read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Access Matrix With Owner Rights

<table>
<thead>
<tr>
<th>Domain</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>owner</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td>read*</td>
<td>read*</td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td>owner</td>
<td>owner write</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td></td>
<td>write</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Domain</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>owner</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td>owner read*</td>
<td>read*</td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td>write*</td>
<td>owner write</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td></td>
<td>write</td>
</tr>
</tbody>
</table>

(b)
## Modified Access Matrix

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>laser printer</th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>read</td>
<td></td>
<td>read</td>
<td></td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td></td>
<td>print</td>
<td></td>
<td>switch</td>
<td>switch</td>
<td>switch</td>
<td>control</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td>read</td>
<td></td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>write</td>
<td></td>
<td>write</td>
<td></td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation of Access Matrix

- 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 1 – Read
  Object 4 – Read, Write, Execute
  Object 5 – Read, Write, Delete, Copy
Revocation of Access Rights

- **Access List** – Delete access rights from access list
  - Simple, Immediate

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
Security
The Security Problem

- Security must consider external environment of the system, and protect the system resources

- **Intruders** attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security

- Attack can be accidental or malicious, but easier to protect against accidental than malicious misuse
Security Violations

Categories
- Breach of confidentiality
- Breach of integrity
- Breach of availability
- Theft of service
- Denial of service

Methods
- Masquerading (breach authentication)
- Replay attack
  - Message modification
- Man-in-the-middle attack
- Session hijacking
Security Measure Levels

- Security must occur at four levels to be effective:
  - Physical
  - Human
    - Avoid social engineering, phishing, dumpster diving
  - Operating System
  - Network
- Security is as weak as the weakest link in the chain
# Program Threats

- **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

- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler

- **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)
#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;
    }
}
See you next time

We will talk about how to play poker online
```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    execvp("/bin/sh", "/bin \sh", NULL);
    return 0;
}
```
Hypothetical Stack Frame

Before attack

After attack
Program Threats (Cont.)

- **Viruses**
  - Code fragment embedded in legitimate program
  - Very specific to CPU architecture, operating system, applications
  - Usually borne via email or as a macro
    - **Visual Basic Macro to reformat hard drive**
      ```vbs
      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
  - Boot
  - Macro
  - Source code
  - Polymorphic
  - Encrypted
  - Stealth
  - Tunneling
  - Multipartite
  - Armored
The Morris Internet Worm
Cryptography as a Security Tool

- Brodest security tool available
  - Source and destination of messages cannot be trusted without cryptography
  - Means to constrain potential senders (*sources*) and/or receivers (*destinations*) of *messages*
- Based on secrets (*keys*)
**Encryption**

- **Encryption** algorithm consists of
  - Set of \( K \) keys
  - Set of \( M \) Messages
  - Set of \( C \) 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

- 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 \( D(k) \).
  - 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