Protection

What we have discussed so far:
- Dual-Mode Execution
- Process Isolation
- Memory Protection

Protection can be considered the enforcement of rules governing resource use.

Authentication

Passwords
- Shared secret
- Knowing the password is the proof of authenticity
  - Authenticated user \rightarrow Permissions

- Shared secret \rightarrow computer must have a copy to validate against
  - Plain text? Bad idea!
  - Encryption

SmartCard
- Contains microprocessor, an encrypted certificate

Biometrics
- Test for a biometric feature
Access Control

- Subjects
  - User, Group
- Objects
  - Files, Servers, ...
- Grant permissions for Subject to perform Action on Object

Access Control Matrix

<table>
<thead>
<tr>
<th></th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U2</td>
<td></td>
<td></td>
<td>R W</td>
</tr>
<tr>
<td>U3</td>
<td></td>
<td></td>
<td>OWN</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: Unix

- Subjects in Unix: Users, Groups
  - Identified by numeric ids.
  - Processes

- Login process
  - Runs as root
- Start process for user
  - Identity (uid, gid) is set by login
- Every process started by user has a pair of uid, gid
  - Effective uid, gid
  - Real uid, gid

Access Control Lists

- Grouping the permissions per object

Example: Unix

- Objects in Unix are Files
  - Regular files
  - Devices
  - Sockets
  - FIFOs, ...

- Access control through mode bits
  - Read, write, execute
  - Owner, user, world

- Use chown, chmod to change this
Capabilities

- Idea: capability implies access privileges
- Showing a capability grants permissions

Physical Access

- Console access
- CD-ROM or USB Port
  - Re-boot machine from image
  - Get access as root
- Ethernet port
  - PXE-boot
- Firewire
  - DMA attack on a life system
  - “Cold Boot”
  - DRAM can store data for several seconds or several minutes if cooled

Trojan Horse

- Malware hiding in possibly legitimately looking program
- Misuses the user privileges
  - E.g., backdoor
  - E.g., spyware
  - Fake login prompt

Phishing
12/2/2015

Vulnerabilities

```c
#include <stdio.h>
#include <string.h>
#define BUF_SIZE 10
int main(int argc, char * argv[]) {
    char buffer[BUF_SIZE];
    strcpy(buffer, argv[1]);
    printf("done\n");
    return 0;
}
```

Stack/Buffer Overflow

- Overwrite the return address
  - Jump to malicious code
  - Steal privileges from exploited program
- Overwrite a function pointer or exception handler
- ...

What can be done to protect from this kind of attacks?

Canaries

- Historic example:
  - Canaries in coal mines
  - Used for detecting dangerous carbon monoxide concentration
- Canary value
  - Place a marker value on the stack
  - If somebody tries to manipulate the stack through buffer overflow canary value will show

Code Injection

- SELECT email, uid, passwd FROM users WHERE email = 'Semail';
- What would happen if somebody entered:
  - x';UPDATE users SET email = 'attacker@hacker.net' WHERE email = 'bob@users.com'
  - SELECT email, uid, passwd FROM users WHERE email = 'x';
    UPDATE users SET email = 'attacker@hacker.net' WHERE email = 'bob@users.com';
**Virus**
- Self-replicating malicious code
- Typically embedded into a legitimate program
- Trojan horse as virus dropper
- Tries to take over the system in order to hide
  - E.g., manipulates the system calls like “read”
  - E.g., hijacks boot sector
- Sometimes self-modifying code

**Port Scanning and Fingerprinting**
- Port scanning
  - Checking a host for open ports
  - Open port → potential security vulnerability
- Fingerprinting
  - Some properties in TCP are left to implementation
  - Observe TCP/IP stack properties like flags, initial TTL, Window size, ... to conclude the OS and version
  - Check against known vulnerabilities

**Masquerading**

**Man-in-the-Middle**

intended communication
Cryptography

- We have the message in cleartext
  - Use an encryption algorithm (cypher) to produce the cyphertext
  - \( C = \text{encrypt}(M) \)

- Having the cyphertext and knowing the cypher we can restore the message
  - \( M = \text{decrypt}(C) \)

One-Way Functions

- Authentication problem (e.g., McCathy’s Puzzle)
  - How do you verify that somebody knows a secret?

- \( y = f(x) \) is easy to compute
  - \( x = f^{-1}(y) \) is very hard to compute

- Verifier gets \( y \), can check if you know \( x \) but cannot compute \( x \).

- Example:
  - Factorization: \( p \cdot q = N \)
    - Middle squares: square the number \( x \) of length \( l \), take the middle \( l \) digits of the result (which has length \( 2l \)!

One-Time Pad

- Use a sequence of random bits
  - \( \text{ES439RULES5} \)

- Add (with modulus) to plaintext
  - \( \text{ES439RULES5} + \text{WSFOI167} \)

- Resulting cyphertext “perfectly secure”
  - Information-theoretically secure

- Drawbacks?

Symmetric Keys

- Both sender and receiver share the same private key
  - Symmetric

- Key distribution must be secret

- Any (other) drawbacks?
Key Distribution

- **Diffie-Hellman**
- Idea: public key $Y$ and publicly shared key $X$

Alice: $k = Y_B^{X_A} \mod p$

Bob: $k' = Y_A^{X_B} \mod p$

$K = (Y_B^{X_A} \mod p)^{X_A} \mod p$

$= Y_B^{X_B} \mod p$

$= Y_B^{X_B} \mod p$

$= (Y^X_A \mod p)^X_B \mod p = K'$


Asymmetric Keys

- Public key (encryption key)
- Private key (decryption key)

Private key cannot be derived from public key

Public key can therefore be widely distributed

Example: RSA

- Large prime numbers $p, q$
- Compute $n=p*q$ (modulus)
- Randomly choose $K_e$ (public key) such that $K_e$ and $(p-1) \times (q-1)$ are “relatively prime”
- $(K_e \times K_d) \mod ((p-1) \times (q-1)) = 1$
- Extended Euclidean algorithm
- $K_d = K_e^{-1} \mod (p-1) \times (q-1)$

Certificates

- Link public key to identity
- Certificate signed by a trusted authority

SSL

- Intended communication

SSL, TLS

SSL, TLS
Countermeasures

- Secure Communication
  - Encryption
  - Certificates
- Virus, Malware, Worms, …
  - Antivirus, etc.
- Network / host intrusion
  - ???

Intrusion Detection

- Ideally detect attackers before they break into a system
- Worst case detect them when they entered a system

- Network Intrusion Detection
  - Packet inspection
  - Try to detect “abnormal behavior”

- Host Intrusion Detection
  - E.g., catch unsuccessful login events
  - E.g., catch modification of system files

Honeypot

- Dedicated machine to attract and detect intruders
- Isolated, intruders cannot cause any harm

Firewall

- Filter packets
- Set of rules to match against packets
- Decide whether to forward or drop packet