



OS Security

CSE 120

Winter 2001

Overview

Security include *social, legal, and technical issues*.

It is often difficult to look at these issues separately.

What should you do if you were to find a new security hole that could be exploited?

... we concentrate here on technical issues.

The “Gold Standard”

Three functions are used in the technical support of security:

1. Authentication

Identifying a subject.

2. Authorization

Determining the actions a subject can effect on a subject or object.

3. Auditing

Keeping track of what has happened for offline analysis and forensics.

Common Security Problems I

System does not authenticate itself to user.

Example:

◆ Stealing passwords.

Common Security Problems II

Improper handling of passwords.

Examples:

- ◆ Transmitting unencrypted passwords.
- ◆ Putting password in executable.
- ◆ Poor password selection.

Common Security Problems III

Improper implementation.

Examples:

- ◆ Careful alignment of password field.
- ◆ Leaving PATH defined.
- ◆ Breaking into DOS/VS.

Common Security Problems IV

Parameter passing by reference.

Example:

◆ Executing shell scripts.

```
#!/bin/csh
```

Common Security Problems V

Trojan Horse.

Example:

◆ Ken Thompson's trap door into Unix.

<http://www.acm.org/classics/sep95/>

Common Security Problems VI

The Confinement Problem.

One might be willing to allow an untrusted program to examine sensitive data if one can ensure that it cannot divulge that information.

... such a program could use *storage* or *covert* channels to leak information.

Leakage via Storage Channel I

```
int s[2], r[2], v[2];
char *sn[2] = { "sent1", "sent2"; } // files used to denote bit set
char *rn[2] = { "rcvd1", "rcvd2"; } // files used to denote bit read
char *vn[2] = { "val1", "val2"; } // files used to record bit value

void Setup(void) {
    int i;
    for (i = 0; i < 2; i++) {
        s[i] = open(sn[i], O_RDWR);
        r[i] = open(rn[i], O_RDWR);
        v[i] = open(vn[i], O_RDWR);
    }
}

void Teardown(void) {
    int i;
    for (i = 0; i < 2; i++) {
        close(s[i]);
        close(r[i]);
        close(v[i]);
    }
}
```

Leakage via Storage Channel II

```
void Send(int data) {
    Setup();
    int bits, bit, t = 0;
    for (bits = 0; bits < 32; bits++) {
        bit = data & 1;           // extract next bit to send
        data = data / 2;
        if (bit) flock(v[t], LOCK_EX | LOCK_NB);           // set value
        while (flock(s[t], LOCK_EX | LOCK_NB) == -1) ;    // denote that set
        while (flock(r[t], LOCK_EX | LOCK_NB) == 0)        // wait until bit read
            flock(r[t], LOCK_UN);
        if (bit) flock(v[t], LOCK_UN);                     // cleanup for this bit
        flock(s[t], LOCK_UN);
        t = 1 - t;
    }
    Teardown();
}
```

Leakage via Storage Channel III

```
int Receive(void) {
    Setup();
    int bits, data = 0, t = 0, fac = 1;
    for (bits = 0; bits < 32; bits++) {
        while (flock(s[t], LOCK_EX | LOCK_NB) == 0)    // wait until bit set
            flock(s[t], LOCK_UN);
        if (bits > 0) flock(r[1-t], LOCK_UN); // cleanup from last bit read
        if (flock(v[t], LOCK_EX | LOCK_NB) == 0)    // read and record value
            flock(v[t], lock_UN);
        else data = data + fac;
        while (flock(r[t], LOCK_EX | LOCK_NB) == -1) ;    // denote bit read
        t = 1 - t;
        fac = fac * 2;
    }
    sleep(1);    // allow sender to run so that it notes bit was read
    flock(r[1-t], LOCK_UN);    // cleanup
    Teardown();
    return data;
}
```

Authorization

Restricting access of *subjects* to *objects*.

Issue of *sharing* versus *protection*.

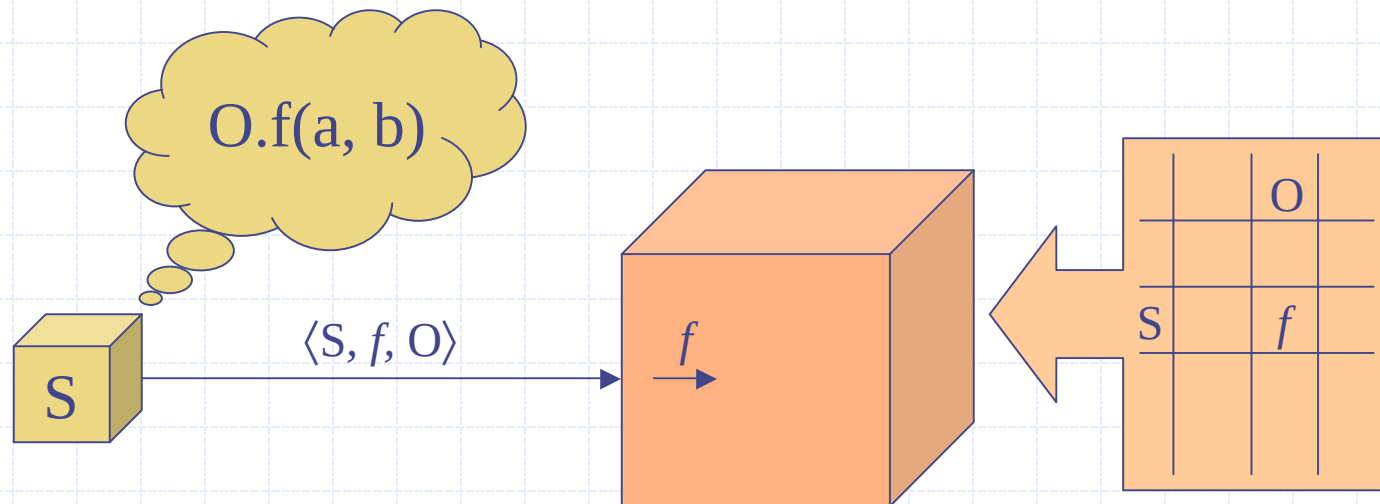
Levels of sharing:

1. Isolation.
2. Copies of files.
3. Originals of files.
4. Untrusted programs.
5. Between untrustworthy subsystems.

Access Matrix Model

	S1	S2	S3	O1	O2	O3
S1	<i>control</i>	<i>owner</i>	<i>owner</i>	<i>read*</i>	<i>owner</i> <i>read</i> <i>write</i>	
S2		<i>control</i>	<i>block</i>	<i>owner</i> <i>read</i> <i>write</i>		<i>read</i> <i>write</i> <i>delete</i>
S3		<i>unblock</i>	<i>control</i>			<i>owner</i> <i>read</i> <i>write</i>

Object Monitor



Protection System Commands

transfer (a/a^*) to S, X	a^* in $A[S_0, X]$	store (a/a^*) in $A[S, X]$
grant (a/a^*) to S, X	<i>owner</i> in $A[S_0, X]$	store (a/a^*) in $A[S, X]$
delete a from S, X	<i>control</i> in $A[S_0, S]$ or <i>owner</i> in $A[S_0, X]$	remove a from $A[S, X]$
read S, X	<i>control</i> in $A[S_0, S]$ or <i>owner</i> in $A[S_0, X]$	read $A[S, X]$
create object X	---	add column X and store <i>owner</i> in $A[S_0, X]$
destroy object X	<i>owner</i> in $A[S_0, X]$	delete column X from A
create subject S	---	add row S, create object S, and store <i>control</i> in $A[S_0, S]$
destroy subject S	<i>owner</i> in $A[S_0, S]$	delete row for S from A and delete object S

... from
Graham
and
Denning
1972

Access Control Lists

Store $A[*,X]$ with object X

When S attempts $X.f$, send $\langle S,f \rangle$ to X 's guard.

- Revocation is simple.
- Asking “who can do f on X ” is easy.
- Ensuring that only one subject has *control* and *ownership* is simple.
- Determining what S can do is hard.

Unix ACLs I

- ◆ Users can be members of *groups*.
e.g. CSFaculty, Systems, GrizulloLab.
- ◆ Object (file) have an *owner* and a *group*.
- ◆ File has three permissions: one for *owner*, one for *group*, and one for *other*.

A subject is governed by *group* if that subject is a member of the file's group.

Unix ACLs II

◆ Permissions are *read*, *write*, and *execute*.

For directories, *execute* means *access files* and *read* means *enumerate*.

◆ File has a *setuid* bit.

If set when file is executed, then is run with permissions of *owner*.

◆ `setuid` system call.

- Example: copying a file from one user's domain to another.

Capabilities I

Store $A[*,X]$ with subject S

When S attempts $X.f$, send *unforgable capability* to X 's guard. If valid capability then access is allowed.

- Revocation is difficult.
- Asking “who can do f on X ” is hard.
- Ensuring that only one subject has *control* and *ownership* is not simple.
- Determining what S can do is straightforward.

Capabilities II

Capabilities can be a good choice for a distributed system.

- Open architecture makes knowing the set of subjects hard.
- Encryption can be used to protect capabilities.

... textbook has good discussion of the use and implementation in *Amoeba*.

Capabilities III

Capability revocation can be provided in a limited manner.

- Use *indirection* (lock/key): Subjects have capability to an indirect object (the *lock*) that contains the capability to the object itself (the *key*).
- Use *UIDs*: capability contains a UID generated by object. Changing this UID invalidates extant capabilities.

... second provides more flexibility than the first.

Securing Physical Channels

In a distributed system one needs to protect against:

- passive wiretapping
- modification of messages
- introduction of spurious messages
- replaying of prior messages
- disruption of communication of selected messages

... the only practical technique for doing so is *encryption*.

Cryptography I

An encryption function $E(M, K_E)$ encrypts a plaintext message M with encryption key K_E into ciphertext.

The corresponding decryption function is $D(C, K_D)$ for ciphertext C and decryption key K_D . If C was not encrypted by K_E then D returns detectable garbage.

... so, $D(E(M, K_E), K_D) = M$

Cryptography II

◆ Private key: $K_D = K_E$

- knowledge of K can be used to authenticate.

◆ Public key: $K_D \neq K_E$

- K_D is *private* and K_E is *public*.
- Will also assume that $\mathbf{D}(E(M, K_D), K_E) = M$

Cryptography III

The hard problem is finding K_D . The quality of an encryption algorithm can be defined as how intense an attack it can withstand:

1. ciphertext only: attacker knows C and language of M .
2. known plaintext: attacker also knows substantial number of C/M pairs.
3. **chosen plaintext**: attacker can also generate C for any M .
4. chosen ciphertext: attacker can also generate M for any

Cryptography IV

Basic techniques for private key ciphers are:

Transposition: break text into blocks of *words*.

Define and use function $f(i)$: the input word number for output word number i .

Ex: word is a character, period = 3, $f = ((1, 3), (2, 1), (3, 2))$. secret $\rightarrow$ csetre

Substitution: replace each word of plaintext with a word of ciphertext.

Ex: *Caesar cipher*: $rot_i(w) = (w + i) \bmod \text{wordsize}$

For $i = 7$, secret $\rightarrow$ zljyla

Cryptography V

Neither of the previous are very strong.
Increase strength with *cipher feedback*:

$$C_1 = E(W_1, K)$$

$$C_2 = E(W_2, K) \otimes C_1$$

$$\dots C_i = E(W_{i-1}, K) \otimes C_{i-1}$$

Can also use *product cipher*:

$$C_i = E_2(E_1(W_i, K_1), K_2)$$

... DES (NBS 1977) is a product cipher with feedback.

Cryptography VI

Public key encryption is based on trapdoor functions: finding a function F that is easy to compute but hard to invert.

RSA (Rivest, Shamir, Adleman 1978) is based on the difficulty of factoring the product of two large prime numbers.