Overview

- **Intro to cryptography tools**
  - one-way functions, public vs private key encryption, hash functions, and digital signatures
- **Protection domains and protection mechanisms**
- **User authentication**
- **Internal attacks**
  - Trojan horses, spoofing, logic bombs, trap doors, buffer overflow attacks
- **External attacks**
  - Viruses, worms, mobile code, sand boxing, interpretation
Security overview

- **Security flavors**
  - Confidentiality - Ability to protect secrets
  - Integrity - Ability to protect the data contents
  - Availability - Ability to continue to operate

- **Know thy enemy!**
  - User stupidity (bad default settings from companies)
  - Insider snooping
  - Outsider snooping
  - Blatant attacks (viruses and worms)
  - Bots!
Accidental data loss

- **Acts of God**
  - fires, floods, wars
- **Hardware or software errors**
  - CPU malfunction, bad disk, program bugs
- **Human errors**
  - data entry, wrong tape mounted
  - “you” are probably the biggest threat you’ll ever face
Introduction to Cryptography Tools
Basics of Cryptography

Relationship between the plaintext and the ciphertext

Encryption

Decryption

Encryption key

Decryption key

P → E → C = E(P, K_E) → D → P

Plaintext in

Ciphertext

Plaintext out
Cryptography: confidentiality and integrity

Sender

Plaintext, P

Encryption method

Encryption key, $E_K$

Ciphertext $C = E_K(P)$

Decryption method

Decryption key, $D_K$

Plaintext

Receiver

Passive intruder only listens to C

Active intruder can alter messages

Active intruder can insert messages
Secret-key cryptography

- **Example: mono-alphabetic substitution**
  
  
  Cyphertext:    Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

- **Given the encryption key (Q W E R T Y U I O P A S D F G H J K L Z X C V B N M),**
  
  - easy to find decryption key using statistical properties of natural language (common letters and digrams)
  - ... despite size of search space of 26! possible keys

- **Function should be more complex and search space very large.**
Symmetric cryptography: DES

- DES operates on 64-bit blocks of data
  - initial permutation
  - 16 rounds of transformations each using a different encryption key
Per-round key generation in DES

- Each key derived from a 56-bit master by mangling function based on splitting, rotating, bit extraction and combination.
Symmetric (secret) key cryptography

- Fast for encryption and decryption
- Difficult to break analytically
- Subject to brute force attacks
  - as computers get faster must increase the number of rounds and length of keys

- Main problem
  - how to distribute the keys in the first place?
Public-key cryptography

- Use different keys for encryption and decryption
- Knowing the encryption key doesn’t help you decrypt
  - the encryption key can be made public
  - encryption key is given to sender
  - decryption key is held privately by the receiver

- But how does it work?
Public-key cryptography

- **Asymmetric (one-way) functions**
  - given function $f$ it is easy to evaluate $y = f(x)$
  - but given $y$ its computationally infeasible to find $x$

- **Trivial example of an asymmetric function**
  - encryption: $y = x^2$
  - decryption: $x = \sqrt{y}$

- **Challenge**
  - finding a function with strong security properties but efficient encryption and decryption
Public-key cryptography: RSA

- RSA (Rivest, Shamir, Adleman)
  - encryption involves multiplying large prime numbers
  - cracking involves finding prime factors of a large number

- Steps to generate encryption key (e) and decryption key (d)
  - Choose two very large prime numbers, p and q
  - Compute \( n = p \times q \) and \( z = (p - 1) \times (q - 1) \)
  - Choose a number d that is relatively prime to z
  - Compute the number e such that \( e \times d = 1 \mod z \)
Public-key cryptography: RSA

- Messages split into fixed length blocks of bits
  - interpreted as numbers with value $0 \leq m_i < n$

- Encryption
  $$c_i = m_i^e \pmod{n}$$
  - requires that you have $n$ and encryption key $e$

- Decryption
  $$m_i = c_i^d \pmod{n}$$
  - requires that you have $n$ and decryption key $d$
RSA vs DES

- RSA is more secure than DES
- RSA requires 100-1000 times more computation than DES to encrypt and decrypt
- RSA can be used to exchange private DES keys
- DES can be used for message contents
Secure hash functions

- **Hash functions** $h = H(m)$ are one way functions
  - can’t find input $m$ from output $h$
  - easy to compute $h$ from $m$

- **Weak collision resistance**
  - given $m$ and $h = H(m)$ difficult to find different input $m'$ such that $H(m) = H(m')$

- **Strong collision resistance**
  - given $H$ it is difficult to find any two different input values $m$ and $m'$ such that $H(m) = H(m')$

- They typically generate a short fixed length output string from arbitrary length input string
Example secure hash functions

- **MD5** - *(Message Digest)*
  - produces a 16 byte result
- **SHA** - *(Secure Hash Algorithm)*
  - produces a 20 byte result
Secure hash functions: MD5

- The structure of MD5
  - produces a 128-bit digest from a set of 512-bit blocks
  - k block digests require k phases of processing each with four rounds of processing to produce one message digest
Per phase processing in MD5

- Each phase involves for rounds of processing

\[
\begin{align*}
F(x, y, z) &= (x \text{ AND } y) \text{ OR } ((\text{NOT } x) \text{ AND } z) \\
G(x, y, z) &= (x \text{ AND } z) \text{ OR } (y \text{ AND } (\text{NOT } z)) \\
H(x, y, z) &= x \text{ XOR } y \text{ XOR } z \\
I(x, y, z) &= y \text{ XOR } (x \text{ OR } (\text{NOT } z))
\end{align*}
\]
Per round processing in MD5

- The 16 iterations during the first round in a phase of MD5 using function F

<table>
<thead>
<tr>
<th>Iterations 1-8</th>
<th>Iterations 9-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \leftarrow (p + F(q,r,s) + b_0 + C_1) \leftarrow 7 )</td>
<td>( p \leftarrow (p + F(q,r,s) + b_8 + C_9) \leftarrow 7 )</td>
</tr>
<tr>
<td>( s \leftarrow (s + F(p,q,r) + b_1 + C_2) \leftarrow 12 )</td>
<td>( s \leftarrow (s + F(p,q,r) + b_9 + C_{10}) \leftarrow 12 )</td>
</tr>
<tr>
<td>( r \leftarrow (r + F(s,p,q) + b_2 + C_3) \leftarrow 17 )</td>
<td>( r \leftarrow (r + F(s,p,q) + b_{10} + C_{11}) \leftarrow 17 )</td>
</tr>
<tr>
<td>( q \leftarrow (q + F(r,s,p) + b_3 + C_4) \leftarrow 22 )</td>
<td>( q \leftarrow (q + F(r,s,p) + b_{11} + C_{12}) \leftarrow 22 )</td>
</tr>
<tr>
<td>( p \leftarrow (p + F(q,r,s) + b_4 + C_5) \leftarrow 7 )</td>
<td>( p \leftarrow (p + F(q,r,s) + b_{12} + C_{13}) \leftarrow 7 )</td>
</tr>
<tr>
<td>( s \leftarrow (s + F(p,q,r) + b_5 + C_6) \leftarrow 12 )</td>
<td>( s \leftarrow (s + F(p,q,r) + b_{13} + C_{14}) \leftarrow 12 )</td>
</tr>
<tr>
<td>( r \leftarrow (r + F(s,p,q) + b_6 + C_7) \leftarrow 17 )</td>
<td>( r \leftarrow (r + F(s,p,q) + b_{14} + C_{15}) \leftarrow 17 )</td>
</tr>
<tr>
<td>( q \leftarrow (q + F(r,s,p) + b_7 + C_8) \leftarrow 22 )</td>
<td>( q \leftarrow (q + F(r,s,p) + b_{15} + C_{16}) \leftarrow 22 )</td>
</tr>
</tbody>
</table>
What can you use a hash function for?

- To verify the integrity of data
  - if the data has changed the hash will change (weak and strong collision resistance properties)

- To “sign” or “certify” data or software
Digital signatures

- Computing a signature block
- What the receiver gets
Digital signatures using a message digest

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{A, B}$</td>
<td>Secret key shared by A and B</td>
</tr>
<tr>
<td>$K_A^+$</td>
<td>Public key of A</td>
</tr>
<tr>
<td>$K_A^-$</td>
<td>Private key of A</td>
</tr>
</tbody>
</table>
Digital signatures with public-key cryptography

Notation | Description
---|---
\( K_{A,B} \) | Secret key shared by A and B
\( K_A^+ \) | Public key of A
\( K_A^- \) | Private key of A
Protection Domains
Protection domains

- Every process executes in some protection domain
  - determined by its creator, authenticated at login time

- OS mechanisms for switching protection domains
  - system calls
  - set UID capability on executable file
  - re-authenticating user
# A protection matrix

<table>
<thead>
<tr>
<th>Domain</th>
<th>File1</th>
<th>File2</th>
<th>File3</th>
<th>File4</th>
<th>File5</th>
<th>File6</th>
<th>Printer1</th>
<th>Plotter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Read</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Read</td>
<td>Read</td>
<td>Write</td>
<td>Read</td>
<td>Write</td>
<td></td>
<td></td>
<td>Write</td>
</tr>
<tr>
<td>3</td>
<td>Read</td>
<td>Write</td>
<td>Execute</td>
<td>Write</td>
<td>Write</td>
<td>Write</td>
<td></td>
<td>Write</td>
</tr>
</tbody>
</table>

OGI SCHOOL OF SCIENCE & ENGINEERING  
OREGON HEALTH & SCIENCE UNIVERSITY
## Protection matrix with domains as objects

<table>
<thead>
<tr>
<th>Domain</th>
<th>File1</th>
<th>File2</th>
<th>File3</th>
<th>File4</th>
<th>File5</th>
<th>File6</th>
<th>Printer1</th>
<th>Plotter2</th>
<th>Domain1</th>
<th>Domain2</th>
<th>Domain3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enter</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read</td>
<td>Read</td>
<td>Write</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Execute</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write</td>
</tr>
</tbody>
</table>
Protection Mechanisms
**Access control lists (ACLs)**

<table>
<thead>
<tr>
<th>Domain</th>
<th>File1</th>
<th>File2</th>
<th>File3</th>
<th>File4</th>
<th>File5</th>
<th>File6</th>
<th>Printer1</th>
<th>Plotter2</th>
<th>Domain1</th>
<th>Domain2</th>
<th>Domain3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enter</td>
</tr>
<tr>
<td>2</td>
<td>Read</td>
<td></td>
<td>Read</td>
<td>Read</td>
<td></td>
<td>Write</td>
<td></td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Read</td>
<td>Read</td>
<td>Write</td>
<td>Write</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Domain matrix is typically large and sparse**
  - inefficient to store the whole thing
  - store occupied columns only, with the resource? - ACLs
  - store occupied rows only, with the domain? - Capabilities
Access control lists for file access

- **User space**
  - Process
  - Owner
  - File
- **Kernel space**
  - File entries:
    - F1: A: RW; B: A
    - F2: A: R; B:RW; C:R
    - F3: B:RWX; C: RX
Access Control Lists (2)

<table>
<thead>
<tr>
<th>File</th>
<th>Access control list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>tana, sysadm: RW</td>
</tr>
<tr>
<td>Pigeon_data</td>
<td>bill, pigfan: RW; tana, pigfan: RW; ...</td>
</tr>
</tbody>
</table>

- Two access control lists with user names and roles (groups)
## Capabilities

### Domain matrix is typically large and sparse
- inefficient to store the whole thing
- store occupied columns only, with the resource? - ACLs
- store occupied rows only, with the domain? - Capabilities
Capabilities associated with processes

- Each process has a capability list
Cryptographically-protected capabilities

- Cryptographically-protected capability can be held in user space

<table>
<thead>
<tr>
<th>Server</th>
<th>Object</th>
<th>Rights</th>
<th>f(Objects, Rights, Check)</th>
</tr>
</thead>
</table>

- **Generic Rights**
  - Copy capability
  - Copy object
  - Remove capability
  - Destroy object
User Authentication
User authentication

- Basic Principles. Authentication must identify:
  - Something the user knows
  - Something the user has
  - Something the user is

- This is done before user can use the system!
Authentication using passwords

(a) A successful login
(b) Login rejected after name entered (easier to crack)
(c) Login rejected after name and password typed

LOGIN: ken
PASSWORD: FooBar
SUCCESSFUL LOGIN

LOGIN: carol
INVALID LOGIN NAME
LOGIN:

LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:

(c)
Problems with pre-set values

```
LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

- How a cracker broke into LBL
  - a U.S. Dept. of Energy research lab
Authentication using passwords and salt

<table>
<thead>
<tr>
<th>Name</th>
<th>Salt</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bobbie</td>
<td>4238</td>
<td>e(Dog,4238)</td>
</tr>
<tr>
<td>Tony</td>
<td>2918</td>
<td>e(6%%TaeFF,2918)</td>
</tr>
<tr>
<td>Laura</td>
<td>6902</td>
<td>e(Shakespeare,6902)</td>
</tr>
<tr>
<td>Mark</td>
<td>1694</td>
<td>e(XaB@Bwcz,1694)</td>
</tr>
<tr>
<td>Deborah</td>
<td>1092</td>
<td>e(LordByron,1092)</td>
</tr>
</tbody>
</table>

- The use of salt to defeat precomputation of encrypted passwords
  - salt changes each time password changes
  - increases the size of the search space
Authentication using a physical object

- Magnetic cards
  - magnetic stripe cards
  - chip cards: stored value cards, smart cards
Authentication using biometrics

A device for measuring finger length.
Attacks on the authentication process

- **Authentication** - making sure the user is the user

- **Attacks include**
  - Placement of passwords in the clear
    - Written on desk, included in a network packet etc...
  - Network packet sniffers
    - Listen to the network and record login sessions
  - Snooping
    - observing key strokes
  - Automated bots
    - Try a password every minute (don't get greedy)
Counter-measures to combat attackers

- Limiting times when someone can log in
- Automatic callback at number prespecified
- Limited number of login tries
- Keep a database of all logins
- Honey pot
  - leave simple login name/password as a trap
  - security personnel notified when attacker bites
More counter-measures

- Better passwords
  - No dictionary words, special characters, longer
- Don’t give up information
  - Login prompts or any other time
- One time passwords
  - Satellite driven security cards
- Limited-time passwords
  - Annoying but effective
- Challenge-response pairs
  - Ask questions
- Physical authentication combined with passwords
Verifying the user is a person
Internal Attacks
Login spoofing

(a) Correct login screen
(b) Phony login screen
Which would you rather log into?
Trojan horses

- Free program made available to unsuspecting user
  - Actually contains code to do harm

- Place altered version of utility program on victim's computer
  - trick user into running that program
  - example, ls attack

- Trick the user into executing something they shouldn't
Logic bombs

- Revenge driven attack
- Company programmer writes program
  - potential to do harm
  - OK as long as he/she enters password daily
  - if programmer fired, no password and bomb “explodes”
Trap doors

while (TRUE) {
    printf("login: ");
    get_string(name);
    disable_echoing();
    printf("password: ");
    get_string(password);
    enable_echoing();
    v = check_validity(name, password);
    if (v) break;
}
execute_shell(name);

(a) Normal code.

(b) Code with a trapdoor inserted
Buffer overflow attacks

- (a) Situation when main program is running
- (b) After program A called
- (c) Buffer overflow shown in gray
Buffer overflow attacks

- The basic idea
  - exploit lack of bounds checking to overwrite return address and to insert new return address and code at that address
  - exploit lack of separation between stack and code (ability to execute both)
  - allows user (attacker) code to be placed in a set UID root process and hence executed in a more privileged protection domain
Other generic security attacks

- Request memory, disk space, tapes and just read
- Try illegal system calls
- Start a login and hit DEL, RUBOUT, or BREAK
- Try modifying complex OS structures
- Try to do specified DO NOTs
- Convince a system programmer to add a trap door
- Beg someone with access to help a poor user who forgot their password
Famous security flaws

- The TENEX password problem
  - requires $128n$ tries instead of $128^n$
Design principles for security

- System design should be public
- Default should be no access
- Check for current authority
- Give each process least privilege possible
- Protection mechanism should be
  - simple
  - uniform
  - in lowest layers of system
- Scheme should be psychologically acceptable

And ... keep it simple!
External Attacks
External threats and viruses

- **External threat**
  - code transmitted to target machine
  - code executed there, doing damage
  - may utilize an internal attack to gain more privilege (ie. Buffer overflow)

- **Goals of virus writer**
  - quickly spreading virus
  - difficult to detect
  - hard to get rid of

- **Virus = program that can reproduce itself**
  - attach its code to another program
Virus damage scenarios

- Blackmail
- Denial of service as long as virus runs
- Permanently damage hardware
- Target a competitor's computer
  - do harm
  - espionage
- Intra-corporate dirty tricks
  - sabotage another corporate officer's files
How viruses work

- Virus written in assembly language
- Inserted into another program
  - use tool called a “dropper”
- Virus dormant until program executed
  - then infects other programs
  - eventually executes its “payload”
## Searching for executable files to infect

**Recursive procedure that finds executable files on a UNIX system**

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>

struct stat sbuf;

search(char *dir_name)
{
    DIR *dirp;
    struct dirent *dp;
    dirp = opendir(dir_name);
    if (dirp == NULL) return;
    while (TRUE) {
        dp = readdir(dirp);
        if (dp == NULL) {
            chdir ("..");
            break;
        }
        if (dp->d_name[0] == '.') continue; /* skip the . and .. directories */
        lstat(dp->d_name, &sbuf);
        if (S_ISLNK(sbuf.st_mode)) continue; /* skip symbolic links */
        if (chdir(dp->d_name) == 0) {
            search("..");
            if (access(dp->d_name, X_OK) == 0) /* if executable, infect it */
                infect(dp->d_name);
        } else {
            if (access(dp->d_name, X_OK) == 0) /* if executable, infect it */
                infect(dp->d_name);
        }
    }
    closedir(dirp); /* dir processed; close and return */
}
```

Virus could infect them all
How viruses hide

- An executable program
- Virus at the front (program shifted, size increased)
- Virus at the end (size increased)
- With a virus spread over free space within program
  * less easy to spot, size may not increase
Viruses that capture interrupt vectors

- After virus has captured interrupt, trap vectors
- After OS has retaken printer interrupt vector
- After virus has noticed loss of printer interrupt vector and recaptured it
How viruses spread

- Virus placed where likely to be copied or executed
- When it arrives at a new machine
  - infects programs on hard drive, floppy
  - may try to spread over LAN
- Attach to innocent looking email
  - when it runs, use mailing list to replicate further
Antivirus and anti-antivirus techniques

(a) A program
(b) Infected program
(c) Compressed infected program
(d) Encrypted virus
(e) Compressed virus with encrypted compression code
### Anti-antivirus techniques

- **Examples of a polymorphic virus**
  - All of these examples do the same thing

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
</tr>
<tr>
<td>ADD B,R1</td>
<td>NOP</td>
<td>ADD #0,R1</td>
<td>OR R1,R1</td>
<td>TST R1</td>
</tr>
<tr>
<td>ADD C,R1</td>
<td>ADD B,R1</td>
<td>ADD B,R1</td>
<td>ADD B,R1</td>
<td>ADD C,R1</td>
</tr>
<tr>
<td>SUB #4,R1</td>
<td>NOP</td>
<td>OR R1,R1</td>
<td>MOV R1,R5</td>
<td>MOV R1,R5</td>
</tr>
<tr>
<td>MOV R1,X</td>
<td>ADD C,R1</td>
<td>ADD C,R1</td>
<td>ADD C,R1</td>
<td>ADD B,R1</td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td>SHL #0,R1</td>
<td>SHL R1,0</td>
<td>CMP R2,R5</td>
</tr>
<tr>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
</tr>
<tr>
<td>NOP</td>
<td>JMP .+1</td>
<td>JMP .+1</td>
<td>ADD R5,R5</td>
<td>MOV R5,Y</td>
</tr>
<tr>
<td>MOV R1,X</td>
<td>MOV R1,X</td>
<td>MOV R1,X</td>
<td>MOV R5,Y</td>
<td>MOV R5,Y</td>
</tr>
</tbody>
</table>
Antivirus software

- **Integrity checkers**
  - use checksums on executable files
  - hide checksums to prevent tampering?
  - encrypt checksums and keep key private

- **Behavioral checkers**
  - catch system calls and check for suspicious activity
  - what does “normal” activity look like?
Virus avoidance and recovery

- **Virus avoidance**
  - good OS
  - install only shrink-wrapped software
  - use antivirus software
  - do not click on attachments to email
  - frequent backups

- **Recovery from virus attack**
  - halt computer, reboot from safe disk, run antivirus
The Internet worm

- Robert Morris constructed the first Internet worm
  - Consisted of two programs
    - bootstrap to upload worm and the worm itself
  - Worm first hid its existence then replicated itself on new machines
  - Focused on three flaws in UNIX
    - rsh – exploit local trusted machines
    - fingerd – buffer overflow attack
    - sendmail – debug problem
- It was too aggressive and he was caught
Availability and denial of service attacks

- Denial of service (DoS) attacks
  - Examples of known attacks
    - Breaking end systems
      - Ping of death – large ping packets
      - Teardrop – overlapping IP segments
    - SYN floods
    - UDP floods
    - Window bombs (in browsers)

- Usually prevented by some sort of firewall but not always effective
Security Approaches for Mobile Code
(a) Memory divided into 1-MB sandboxes
   - each applet has two sandboxed for code and data
   - some static checking of addresses

(b) Code inserted for runtime checking of dynamic target addresses
Applets can be interpreted by a Web browser.
Code signing

Software vendor

- Applet
- Signature

Signature generation

\[ H = \text{hash(Applet)} \]
\[ \text{Signature} = \text{encrypt}(H) \]

User

- Applet
- Signature

Signature verification

\[ H_1 = \text{hash(Applet)} \]
\[ H_2 = \text{decrypt(Signature)} \]

Accept Applet if \( H_1 = H_2 \)

Internet

How code signing works
Type safe languages

- **A type safe language**
  - compiler rejects attempts to misuse variables

- **Checks include ...**
  - Attempts to forge pointers
  - Violation of access restrictions on private class members
  - Misuse of variables by type
  - Generation of stack over/underflows
  - Illegal conversion of variables to another type
Java security

<table>
<thead>
<tr>
<th>URL</th>
<th>Signer</th>
<th>Object</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.taxprep.com">www.taxprep.com</a></td>
<td>TaxPrep</td>
<td>/usr/susan/1040.xls</td>
<td>Read</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td>/usr/tmp/*</td>
<td>Read, Write</td>
</tr>
<tr>
<td><a href="http://www.microsoft.com">www.microsoft.com</a></td>
<td>Microsoft</td>
<td>/usr/susan/Office/-</td>
<td>Read, Write, Delete</td>
</tr>
</tbody>
</table>

Examples of specified protection with JDK 1.2
Covert Channels
Covert channels

(a) Client, server and collaborator processes

(b) Encapsulated server can still leak to collaborator via covert channels
Locking as a covert channel

A covert channel using file locking
Covert channels

- Pictures appear the same
- Picture on right has text of 5 Shakespeare plays
  - encrypted, inserted into low order bits of color values

Zebras

Hamlet, Macbeth, Julius Caesar
Merchant of Venice, King Lear
Spare Slides
Trusted Systems and Formal Models
Trusted Systems
Trusted Computing Base

User process

All system calls go through the reference monitor for security checking

Reference monitor
Trusted computing base
Operating system kernel

User space
Kernel space

A reference monitor
Formal Models of Secure Systems

(a) An authorized state
(b) An unauthorized state

<table>
<thead>
<tr>
<th></th>
<th>Compiler</th>
<th>Mailbox 7</th>
<th>Secret</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eric</td>
<td>Read Execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Henry</td>
<td>Read Execute</td>
<td>Read Write</td>
<td></td>
</tr>
<tr>
<td>Robert</td>
<td>Read Execute</td>
<td>Read Write</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Compiler</th>
<th>Mailbox 7</th>
<th>Secret</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eric</td>
<td>Read Execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Henry</td>
<td>Read Execute</td>
<td>Read Write</td>
<td></td>
</tr>
<tr>
<td>Robert</td>
<td>Read Execute</td>
<td>Read</td>
<td>Read Write</td>
</tr>
</tbody>
</table>

OGI SCHOOL OF SCIENCE & ENGINEERING
OREGON HEALTH & SCIENCE UNIVERSITY
Multilevel Security (1)

The Bell-La Padula multilevel security model
Multilevel Security (2)

The Biba Model

- Principles to guarantee integrity of data
  - Simple integrity principle
    - process can write only objects at its security level or lower
  - The integrity * property
    - process can read only objects at its security level or higher
Orange Book Security (1)

<table>
<thead>
<tr>
<th>Criterion</th>
<th>D</th>
<th>C1</th>
<th>C2</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security policy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discretionary access control</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object reuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Labels</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label integrity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exportation of labeled information</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Labeling human readable output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mandatory access control</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subject sensitivity labels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device labels</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accountability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identification and authentication</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trusted path</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Symbol X means new requirements
- Symbol -> requirements from next lower category apply here also
## Orange Book Security (2)

<table>
<thead>
<tr>
<th>Assurance</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System architecture</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>System integrity</td>
<td>X</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
</tr>
<tr>
<td>Security testing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Design specification and verification</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Covert channel analysis</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Trusted facility management</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration management</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>Trusted recovery</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>→</td>
</tr>
<tr>
<td>Trusted distribution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Documentation</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Security features user’s guide</td>
<td>X</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
</tr>
<tr>
<td>Trusted facility manual</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Test documentation</td>
<td>X</td>
<td>→</td>
<td>→</td>
<td>X</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>Design documentation</td>
<td>X</td>
<td>→</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>