

CSE 120

Summer, 2005

Day 9
Security
System Design

Instructor: Neil Rhodes

Security

Goal

- Data confidentiality

Threat

- Exposure of data

Goal

- Data integrity

Threat

- Tampering with data

Goal

- System availability

Threat

- Denial of service

2

Cryptography

Definitions:

- Plaintext: message being encoded
- Ciphertext: encoded plaintext
- Key: parameter to crypto algorithms
 - $C = E(P, K_E)$
 - $P = D(C, K_D)$
- Symmetric key: $K_E = K_D$
- Public-key: $K_E \neq K_D$
 - K_E can be public
 - K_D is secret (private)
 - Relies on one-way functions (multiplying 2 big numbers much easier than factoring 2 big numbers)
- Digital Signatures
 - Compute one-way hash, H
 - Append $C = E(H, K_{\text{private}})$ to signature block
 - Verify by computing $H' = D(C, K_{\text{public}}) = \text{Hash}(\text{document})$
- Public Key distribution an issue
 - Certification authorities
 - Web of trust

3

Authentication

Methods:

- Something the user knows
 - password
- Something the user has
 - Smart card
 - Token
 - Key
 - Magstripe card
- Something the user is (biometrics)
 - Voice biometrics
 - Signature biometrics
 - Retinal scan
 - Fingerprints

4

Passwords

Notoriously poorly chosen

- Dictionary attacks

Sent in the clear

- Solution: challenge-response

5

Trojan Horses

Innocent-appearing code that hides greeks

- Example: [<http://cm.bell-labs.com/who/ken/trust.html>](http://cm.bell-labs.com/who/ken/trust.html)
- Trap door in login program (secret password that allows you in).
- Trojan horse in compiler that inserts special code when compiling login.c
- Trojan horse in compiler that inserts special code when compiling the compiler

6

Login Spoofing

Write a program that looks like the login screen

When user enters name/password:

- Store, quit, and begin real user login

User retypes name/password

- Thinks all is well

Windows solution

- Ctrl-Alt-Del

7

Buffer Overflow (strcpy considered harmful)

Code like:

```
▪ char ReadFromUser(char *s)
{
    char buffer[500];

    strcpy(buffer, s);
    // processing here
}
```

If ReadFromUser gets string larger than 500, it overflows stack

- If overflows return PC, can change where PC returns to

8

Viruses

Reproduces itself by infecting programs or documents

- Not standalone

9

Worm

Self-replicating computer program

- Standalone: unlike virus
- Uses network to spread
- Most famous: the Internet Worm 1988, by Robert Tappan Morris
- Named after similar software described in *The Shockwave Rider* by John Brunner

10

Protection

Policy

- Whose data is to be protected from whom

Mechanism

- How the system enforces the policy
- Usually create a general mechanism. Administrator sets up the policy

Domain

- Set of objects and rights associated with those objects

11

Protection Matrix

Access Control Lists: Columns of the protection matrix

- Each object keeps list of domains and associated permissions
- uid 1: RW, uid 2: RWX, uid 4: R
 - any not specified are disallowed
- uid 5: none, uid *: RW:

	File 1	File 2	File 3	File 1	Domain 2
Domain 1	Read	Read/Write			Enter
Domain 2		Read	Read/Write/ Execute	Write	
Domain 3				Write	

Capabilities: Rows of the protection matrix

- Each process contains a *capability list*: list of objects that may be accessed, along with which operations
 - Stored in kernel
 - Or, stored in tagged memory
 - Tag on memory word specifies a capability: can only be modified by kernel
 - Or, stored with crypto signature

12

Unix Protection

Domain is (user ID, group ID)

Setuid program changes domain to (userID, groupID) of the program itself

13

Computer System Design

From *Hints for Computer System Design*, by Butler Lampson

	Functionality	Speed	Fault-tolerance
Completeness	Separate normal and worst case	Shed load End-to-end Safety first	End-to-end
Interface	Do one thing well Don't generalize/Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide & Conquer	Cache answers Use hints Use brute force Compute in background	Make actions atomic Use hints

14

Do one thing at a time, and do it well

If your interface tries to do too much

- Its implementation will be large, slow, and complicated

An interface is a contract to deliver a certain amount of service

- Clients depend on the contract
- Clients depend on incurring a reasonable cost for using the service
- If there are six levels of abstraction, and each costs 50% more than is reasonable
 - Final service will be a factor of 10 above reasonable

Offering too much

- Too slow
- Too complicated
 - Tenex
 - Unassigned virtual page reported to user program
 - System calls executed on behalf of user program also report same
 - Password-checking done by reference from user program
- Example:
 - Alto OS: 900 lines of code for file system, 500 for paging and fast
 - Successor with virtual pages mapped to files: 11,000 lines and slow

15

Corollaries

Make it fast

- Much better to have fast basic operations than slow powerful ones
 - Fast, powerful even better
 - Don't make user who needs only basic pay for slow

Don't hide power

- Abstractions should conceal *undesirable* properties, not desirable ones

Use procedure arguments

- Elements of a set satisfying a property: pass in filter function

Leave it to the client

- Solve one problem and leave the rest to the client. Yacc, for example

16

Continuity

Keep basic interfaces stable

Keep a place to stand (if you do need to change interfaces)

- Compatibility package
- Debugger stub (read word, write word, stop, go)

17

Implementation

Plan to throw one away

- You will anyhow

Keep secrets

- The interface defines the things that cannot change. Leave flexibility for the things that can

Divide and conquer

- When resources are limited, do as much as possible, leaving the rest for another iteration
- Example: printing

18

Handling all the cases

The normal case must be fast

The worst case must make *some* progress

- Keep a reserve to make progress
 - Example: Banker's algorithm

19

Speed

Cache answers

Use hints

- Hint may be wrong, but saves time if it is right
- Example:
 - symbolic loop
 - Ethernet carrier not signaled

When in doubt, use brute force

- Extreme programming tenet: DTSTTCPW: do the simplest thing that could possibly work

Compute in background

- Pre-fetch
- Pre-clean

20

Speed

Shed load to control demand

- If thrashing: swap jobs to disk
- Interactive shared system: big red button: improve service, or throw me off!

21

Fault-tolerance

End-to-end

- For reliability: application-level must check data
- Any other checking is for performance
 - ECC memory
 - CRCs for packets

Log updates

- Append-only
- Example: file system

Make actions atomic

- Example: FSExchangeFiles in Mac OS

22

Don't Repeat Yourself

When one piece of knowledge in the domain changes, one piece of the system changes

- From *The Pragmatic Programmer* by Andrew Hunt and David Thomas

23

Reading List

The Pragmatic Programmer by Andrew Hunt & David Thoms

Code Complete by Steve McConnell

Refactoring: Improving the Design of Existing Code by Martin Fowler

Writing Solid Code by Steve Maguire

Programming Pearls by Jon Bentley

Practice of Programming by Brian Kernighan & Rob Pike

Extreme Programming Explained by Kent Beck

24