CSE 451: Operating Systems
Spring 2006

Module 19
Security: Authentication

John Zahorjan
zahorjan@cs.washington.edu
Allen Center 534

Basic Concepts

- Principals – who is acting
  - User / Process Creator
  - Code Author
- Objects – what is that principal acting on?
  - File
  - Network connection
- Rights
  - Read
  - Write

Policy: The Access Matrix Concept

	Alice	Bob	Carl
/etc	Read	Read	Read Write
/homes	Read Write	Read Write	Read Write
/usr	None	None	Read

- This is a picture of a concept
- There are multiple implemenation alternatives
  - Policy / mechanism distinction
- We'll get back to this later

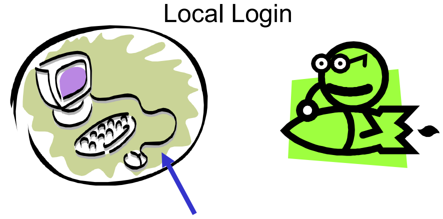
Some Fundamental Concepts

- Authentication (who are you)
  - identifying principals (users / programs)
- Authorization (what are you allowed to do)
  - determining what access users and programs have to things
- Auditing (what happened)
  - record what users and programs are doing for later analysis / prosecution

Authentication

- How does the provider of a secure service know who it's talking with?
  - Example: login
- We'll start with the local case (the keyboard is attached to the machine you want to login to)
- Then we'll look at a distributed system

Local Login



("Local" ⇒ this connection is assumed secure)

How does the OS know that I'm 'zahorjan'?

Shared Secret

The shared secret is typically a password, but it could be something else:
- Retina scan
- A key

Simple Enough

- This seems pretty trivial
- Like pretty much all aspects of security, there are perhaps unexpected complications
- As an introduction to this, let's look at briefly at the history of password use

Storing passwords

- CTSS (1962): password file {user name, user identifier, password}

Bob, 14, "12.14.52"
David, 15, "allison"
Mary, 16, "!ofotc2n"

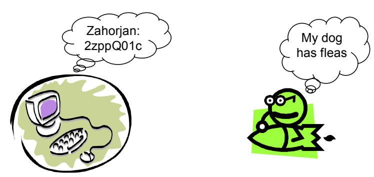
If a bad guy gets hold of the password file, you're in deep trouble

– **Any** flaw in the system that compromises the password file compromises all accounts!

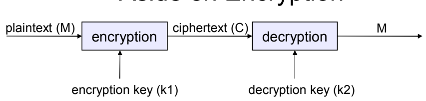
Two Choices

1. Make sure there are no flaws in the system
2. Render knowledge of the password file useless

Unix (1974): store encrypted forms of the passwords



Aside on Encryption



- Encryption: takes a key and plaintext and creates ciphertext: $E_{k1}(M) = C$
- Decryption: takes ciphertext and a key and recovers plaintext: $D_{k2}(C) = M$
- Symmetric algorithms (aka secret-key aka shared secret algorithms):
  – k1 = k2 (or can get k2 from k1)
- Public-Key Algorithms
  – decryption key (k2) cannot be calculated from encryption key (k1)
  – encryption key can be made public!
    - encryption key = "public key", decryption key = "private key"
- Computational requirements:
  – Deducing M from $E_k(M)$ is "really hard"
  – Computing $E_k(M)$ and $D_k(C)$ is efficient

Unix Password File (/etc/passwd)

- Encrypt passwords with passwords

K=[alison]$_{allison}$

Bob: 14: S6Uu0cYDVdTAk
David: 15: J2ZI4ndBL6X.M
Mary: 16: VW2bqvTalBJKg

- David's password, "allison," is encrypted using itself as the key and stored in that form.
- Password supplied by user is encrypted with itself as key, and result compared to stored result.
- "No problem if someone steals the file"

The Dictionary Attack

- Encrypt many (all) possible password strings offline, and store results in a dictionary
  - I may not be able to invert any particular password, but the odds are very high I can invert one or more
- 26 letters used, 7 letters long
  - 8 billion passwords (33 bits)
  - Generating 100,000/second requires 22 hours
- But most people's passwords are not random sequences of letters!
  - girlfriend's/boyfriend's/spouse's/dog's name/words in the dictionary
- Dictionary attacks have traditionally been incredibly easy

Making it harder

- Using symbols and numbers and longer passwords
  - 95 characters, 14 characters long
  - $10^{27 \text{ passwords} =}$ 91 bits
  - Checking 100,000/second breaks in 10^{14} years
- Require frequent changing of passwords
  - guards against loaning it out, writing it down, etc.

Do longer passwords work?

- People can't remember 14-character strings of random characters
- People write down difficult passwords
- People give out passwords to strangers
- Passwords can show up on disk
- If you are forced to change your password periodically, you probably choose an even dumber one
  - "feb04" "mar04" "apr04"
- How do we handle this in CSE?

Countermeasure to the Dictionary Attack: Salt

- Unix (1979): salted passwords
  - The salt is just a random number from a large space

$K=[alison392]_{allison392}$

Bob: 14: T7Vs1dZEWeRcL: 45
David: 15: K3AJ5ocCM4ZM$: 392
Mary: 16: WX3crwUbmCKLf: 152

Encryption is computed after affixing a number to the password. Thwarts pre-computed dictionary attacks

Okay, are we done? Problem solved?

Attack Models

- Besides the problems already mentioned that obviously remain (people give out their passwords / write them down / keyloggers / …), there may be other clever attacks that we haven't thought of
- Attack Model: when reasoning about the security of a mechanism, we need typically need to carefully describe what kinds of attacks we're thinking of
  - helps us reason about what vulnerabilities still remain

Example 1: Login spoofers

- Login spoofers are a specialized class of Trojan horses
  - Attacker runs a program that presents a screen identical to the login screen and walks away from the machine
  - Victim types password and gets a message saying "password incorrect, try again"
- Can be circumvented by requiring an operation that unprivileged programs cannot perform
  - E.g., start login sequence with a key combination user programs cannot catch, CTRL+ALT+DEL on Windows

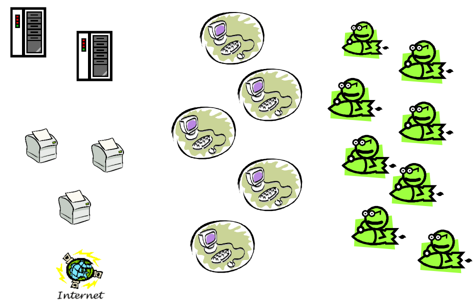
Example 2: Cool password attack

- VMS (early 80's) password checking flaw
  - password checking algorithm:

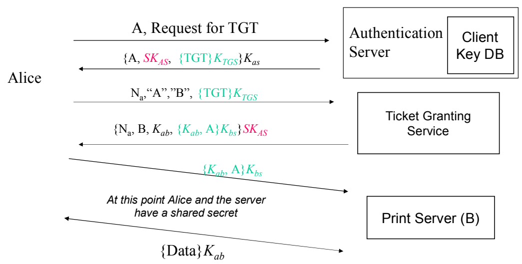
```
for (I=0; I<password.length( ); I++) {
    if password[I] == supplied_password[I]
        return false;
}
return true;
```

  - can you see the problem?
    - hint: think about virtual memory…
    - another hint: think about page faults…
    - final hint: who controls where in memory supplied_password lives?

Distributed Authentication (Single Domain)



Kerberos



Trust Relationships

- Both Alice and the server must trust the Kerberos servers ("trusted third party")
- This architecture is essentially what Microsoft passport is:

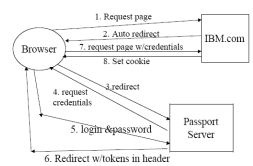


Figure 1. The Passport architecture.

CBS NEWS BACK PRINT

Microsoft's Passport Flaw Fixed

WASHINGTON, May 8, 2003

(AP) A computer researcher in Pakistan discovered how to breach Microsoft Corp.'s security procedures for its popular Internet Passport service, designed to protect customers visiting some retail Web sites, sending e-mails and in some cases making credit-card purchases.

Microsoft acknowledged the flaw affected all its 200 million Passport accounts but said it fixed the problem early Thursday, after details were published on the Internet. Product Manager Adam Sohn said the company was unaware of hackers actually hijacking anyone's Passport account, but several experts said they successfully tested the procedure overnight.

In theory, Microsoft could face a staggering fine by U.S. regulators of up to $2.2 trillion. Under a settlement with the Federal Trade Commission last year over lapsed Passport security, Microsoft pledged to take reasonable safeguards to protect personal consumer information during the next two decades or risk fines up to $11,000 per violation.

The FTC said it was investigating this latest lapse. The agency's assistant director for financial practices, Jessica Rich, said Thursday that each vulnerable account could constitute a separate violation - raising the maximum fine that could be assessed against Microsoft to $2.2 trillion.

"If we were to find that they didn't take reasonable safeguards to protect the information, that could be an order violation," Rich said.

The researcher, Muhammad Faisal Rauf Danka, determined that by typing a specific Web address that included the phrase "emailpwdreset," he could seize any person's Passport account and change the password associated with it.