CSE 451: Operating Systems
Spring 2013

Module 27
Authentication / Authorization / Security

Ed Lazowska
lazowska@cs.washington.edu
Allen Center 570

Terminology I: the entities

- Principals – who is acting?
  - User / Process Creator
  - Code Author
- Objects – what is that principal acting on?
  - File
  - Network connection
- Rights – what actions might you take?
  - Read
  - Write
- Familiar UNIX file system example:
  - owner / group / world
  - read / write / execute

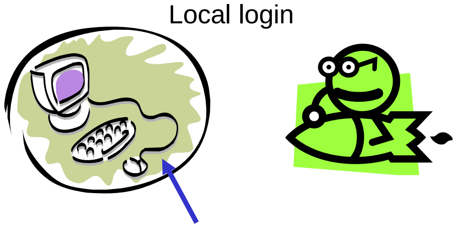
Terminology II: the activities

- Authentication – who are you?
  - identifying principals (users / programs)
- Authorization – what are you allowed to do?
  - determining what access users and programs have to specific objects
- 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 briefly at a distributed system

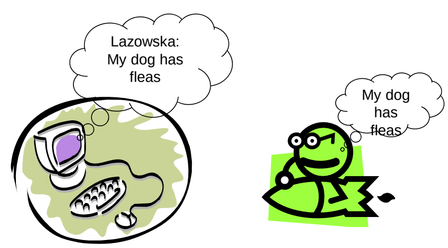
Local login



("Local" ⇒ this connection is assumed secure)

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

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

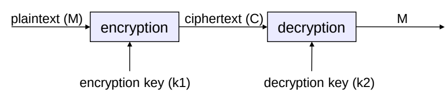
- **Any** flaw in the system that compromises the password file compromises all accounts!

Two choices

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

Unix (1974): store encrypted forms of the passwords

An 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)
- Asymmetric, or 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/password)

- Encrypt passwords with passwords

$K=[allison]_{allison}$

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"
- Also no need to secure a key

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} 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, frequently changed 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 / key loggers / …), there may be other clever attacks that we haven't thought of
- Attack Model: when reasoning about the security of a mechanism, we 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
- False fronts have been used repeatedly to steal bank ATM passwords!

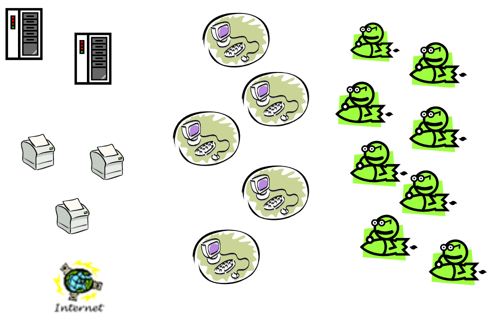
Example 2: Page faults as a signal

- 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?

So imagine how complicated life is in the distributed world!



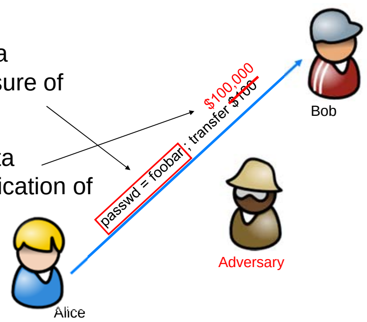
Issues

- How do I know that I'm talking to the server I intend (vs. a "man in the middle")?
- How does the server know it's talking to me?
- How do we ensure that others can't eavesdrop on our conversation?
- How do we ensure that others can't manipulate our conversation?
- How do we avoid replay attacks?

Communication security goals

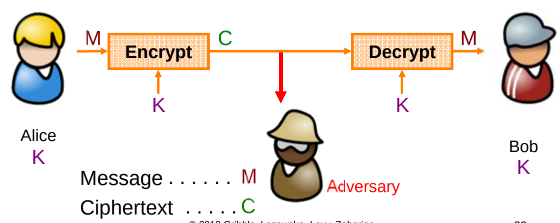
Privacy of data
Prevent exposure of information

Integrity of data
Prevent modification of information

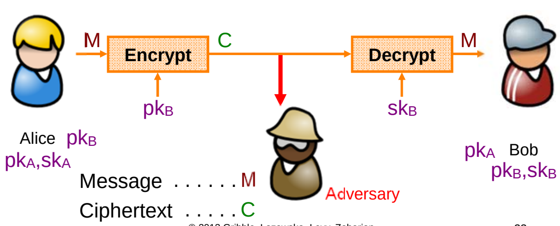


Encryption schemes: A tool for protecting privacy

Symmetric setting: Both communicating parties have access to a shared random string K, called the key

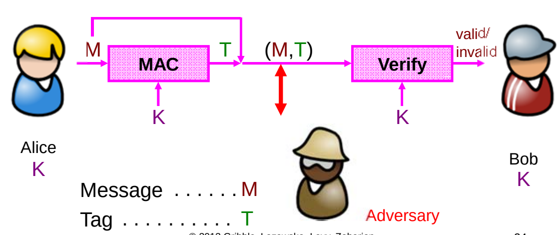


Asymmetric setting: Each party creates a public key pk and a secret key sk

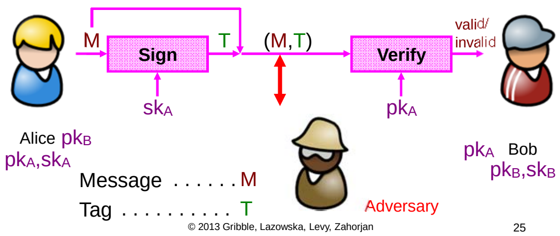


Message authentication schemes (aka Message Authentication Codes, MACs): A tool for protecting integrity

Symmetric setting

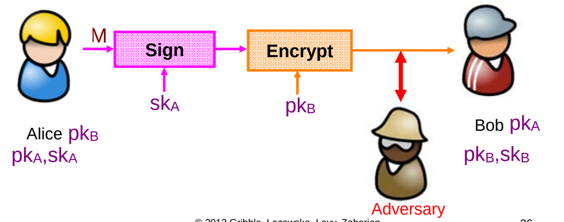


Asymmetric setting: Digital signature, protects integrity and authenticity



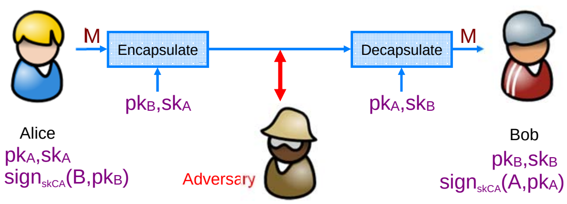
To achieve privacy *and* integrity – to send you a message that only you can read, and that you know was created by me

Asymmetric setting



How can I be sure that I'm using *your* public key, vs. an imposter's?

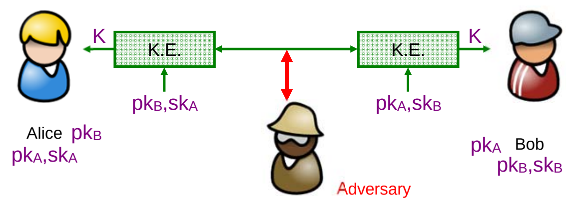
- Each party creates a public key pk and a secret key sk
- Public keys are registered with a trusted third party – a certificate authority (CA)
- I get your public key from a CA, signed by that CA



Key exchange

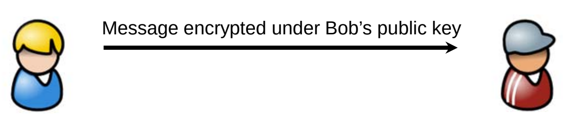
Key exchange protocols: A tool for establishing a shared symmetric key

(Why? Public key systems are relatively slow!)



One-way Communications

PGP is a good example



But life is never this simple!

One-way Communications

(*Informal* example; ignoring, e.g., signatures)

1. Alice gets Bob's public key; Alice *verifies* Bob's public key (e.g., via CA)
2. Alice generates random symmetric keys K1 and K2
3. Alice encrypts the message M with the key K1; call result C ← **privacy**
4. Alice authenticates (MACs) C with key K2; call the result T ← **integrity**
5. Alice encrypts K1 and K2 with Bob's public key; call the result D

6. Send D, C, T

(Assume Bob's private key is encrypted on Bob's disk.)

7. Bob takes his password to derive key K3
8. Bob decrypts his private key with key K3
9. Bob uses private key to decrypt K1 and K2
10. Bob uses K2 to verify MAC tag T
11. Bob uses K1 to decrypt C

Why use K1 to encrypt M, rather than Bob's public key?

Interactive Communications

Let's talk securely; here are the algorithms I understand →

← I choose these algorithms; start key exchange

Continue key exchange →

⋮

↔ Communicate using exchanged key

Again, life is never this simple!

Interactive Communications

(*Informal* example; details omitted)

1. Alice and Bob exchange public keys and certificates
2. Alice and Bob use CA's public keys to verify certificates and each other's public keys
3. Alice and Bob take their passwords and derive symmetric keys
4. Alice and Bob use those symmetric keys to decrypt and recover their asymmetric private keys.
5. Alice and Bob use their asymmetric private keys and a *key exchange* algorithm to derive a shared symmetric key
6. Alice and Bob use shared symmetric key to encrypt and authenticate messages

(Will need to rekey regularly; may need to avoid replay attacks, ...)

(Replay attacks: thwart using counters or timestamps …)

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.

Spyware

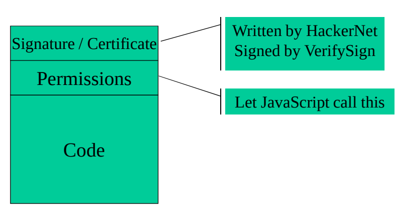
- Software that is installed that collects information and reports it to third party
  - key logger, adware, browser hijacker, …
- Installed one of two ways
  - piggybacked on software you choose to download
  - "drive-by" download
    - your web browser has vulnerabilities
    - web server can exploit by sending you bad web content
- Estimates
  - majority (50-90%) of Internet-connected PCs have it
  - 1 in 20 executables on the Web have it
  - about 0.5% of Web pages attack you with drive-by-downloads

Additional modern security problems

- Confinement
  - How do I run code that I don't trust?
    - e.g., RealPlayer, Flash
  - How do I restrict the data it can communicate?
  - What if trusted code has bugs?
    - e.g., Internet Explorer
- Solutions
  - Restricted contexts – let the user divide their identity
  - ActiveX – make code writer identify self
  - Java – use a virtual machine that intercepts all calls
  - Binary rewriting – modify the program to force it to be safe

ActiveX

- All code comes with a public-key signature
- Code indicates what privileges it needs
- Web browser verifies certificate
- Once verified, code is completely trusted



Java / C#

- All problems are solved by a layer of indirection
  - All code runs on a virtual machine
  - Virtual machine tracks security permissions
  - Allows fancier access control models - allows stack walking
- Interposition using language VM doesn't work for other languages
- Virtual machines can be used with all languages
  - Run virtual machine for hardware
  - Inspect stack to determine *subject* for access checks

Binary rewriting

- Goal: enforce code safety by *embedding* checks in the code
- Solution:
  - Compute a mask of accessible addresses
  - Replace system calls with calls to special code

Original Code:

```
lw   $a0, 14($s4)
jal  ($s5)
move $a0, $v0
jal $printf
```

Rewritten Code:

```
and $t6,$s4,0x001fff0
lw   $a0, 14($t6)
and  $t6,$s5, 0x001fff0
jal  ($t6)
move $a0, $v0
jal $sfi_printf
```