

CSE 451: Operating Systems Autumn 2013

Module 27 Authentication / Authorization / Security

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

© 2013 Gribble, Lazowska, Levy, Zahorjan

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

© 2013 Gribble, Lazowska, Levy, Zahorjan

2

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

© 2013 Gribble, Lazowska, Levy, Zahorjan

3

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

© 2013 Gribble, Lazowska, Levy, Zahorjan

4

Local login



("Local" => this connection is assumed secure)

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

© 2013 Gribble, Lazowska, Levy, Zahorjan

5

Shared secret



The shared secret is typically a password, but it could be something else:

- Retina scan
- A key

© 2013 Gribble, Lazowska, Levy, Zahorjan

6

Simple enough ...

- This seems pretty trivial
- Like pretty much all aspects of security, there are 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!

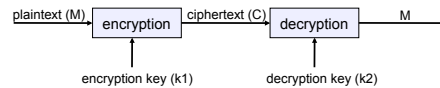
Two choices

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**: $E_{k_1}(M) = C$
- **Decryption:** takes ciphertext and a key and recovers plaintext: $D_{k_2}(C) = M$
- **Symmetric algorithms** (aka secret-key aka shared secret algorithms):
 - $k_1 = k_2$ (or can get k_2 from k_1)
- **Asymmetric, or public-key, algorithms**
 - decryption key (k_2) cannot be calculated from encryption key (k_1)
 - encryption key can be made public!
 - encryption key = "public key", decryption key = "private key"
- **Computational requirements:**
 - Deducing M from $E_{k_1}(M)$ is "really hard"
 - Computing $E_{k_1}(M)$ and $D_{k_2}(C)$ is efficient

Lousy idea

- Encrypt using some key k_1

$C[\text{password}] = [\text{password}]_{k_1}$

```
Bob: 14: S6Uu0cYDVdTak
David: 15: J2Z14ndBL6X.M
Mary: 16: VW2bqvTalBJKg
```

- Every password is encrypted using the key k_1
- "No problem if someone steals the file"
- Unless ... someone steals the key too!
 - (The system programmer wrote it on a PostIt note, and it's her boyfriend's name anyhow)

Unix password file (/etc/passwd)


- Encrypt passwords with passwords

$C[\text{allison}] = [\text{allison}]_{\text{allison}}$

```
Bob: 14: S6Uu0cYDVdTak
David: 15: J2Z14ndBL6X.M
Mary: 16: VW2bqvTalBJKg
```

- David's password, "allison," is encrypted using itself as the key and stored in that form.
- The password supplied by a 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

Autumn 2013
152,000,000 unsalted passwords
+
password hints in cleartext!



nakedsecurity
Award-winning news, opinion, advice and research from SANS
malware mac facebook android vulnerability data loss privacy more...

Lightbeam shines a light on which... NSA whistleblower Edward Snowden

Anatomy of a password disaster - Adobe's giant-sized cryptographic blunder

The dump looks like this:

```

4464 |--| xxx@yahoo.com | g286PmE836cDB5Cq | 00== | try: oem7y123 |--
4465 |--| xxx@jcom.home.ne.jp | EHLGOM-1h23urM0Bm== | 7777 |--
4466 |--| xxx@bnet1.com | xw2B8ELg8TmVQGVnwm== | quiero a... |--
4467 |--| xxx@yahoo.com | leTcmPEPj | oxG6CathBm== | |--
4468 | *username | xxx@adobe.com | 5C5R1rueRfE | toxG6CathBm== | |--
4469 |--| xxx@yahoo.com | AL51077204 | ruyg |--
4470 |--| xxx@bnet1.com | xG2G5G86 | toxG6CathBm== | |--
4471 |--| xxx@yahoo.com | x3a7f8fUR | toxG6CathBm== | myspace |--
4472 |--| xxx@bnet1.com | xby191Bm | toxG6CathBm== | regular |--

```

By inspection, the fields are as follows:

```

4464 | User ID | xxx@yahoo.com | g286PmE836cDB5Cq | Password hint | try: oem7y123
4465 | xxx@jcom.home.ne.jp | EHLGOM-1h23urM0Bm== | 7777
4466 | xxx@bnet1.com | xw2B8ELg8TmVQGVnwm== | quiero a...
4467 | *username | xxx@adobe.com | 5C5R1rueRfE | toxG6CathBm==
4468 | *username | xxx@yahoo.com | leTcmPEPj | oxG6CathBm==
4469 | *username | xxx@yahoo.com | AL51077204 | ruyg
4470 | xxx@bnet1.com | xG2G5G86 | toxG6CathBm==
4471 | xxx@yahoo.com | x3a7f8fUR | toxG6CathBm== | myspace
4472 | xxx@bnet1.com | xby191Bm | toxG6CathBm== | regular

```

© 2013 Gribble, Lazowska, Levy, Zahorjan 19

Adobe's FAQ as of 12/4/2013

Frequently Asked Questions

- What information exactly did the attacker gain access to?

Our investigation currently indicates that the attackers accessed Adobe customer IDs and encrypted passwords on our systems. We also believe the attackers removed from our systems certain information relating to 2.9 million Adobe customers, including customer names, encrypted credit or debit card numbers, expiration dates, and other information relating to customer orders. At this time, we do not believe the attackers removed decrypted credit or debit card numbers from our systems.

We are also investigating the illegal access to source code of numerous Adobe products. Based on our findings to date, we are not aware of any specific increased risk to customers as a result of this incident.

© 2013 Gribble, Lazowska, Levy, Zahorjan 20

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

© 2013 Gribble, Lazowska, Levy, Zahorjan 21

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!**

© 2013 Gribble, Lazowska, Levy, Zahorjan 22

Example 2: Page faults as a signal

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

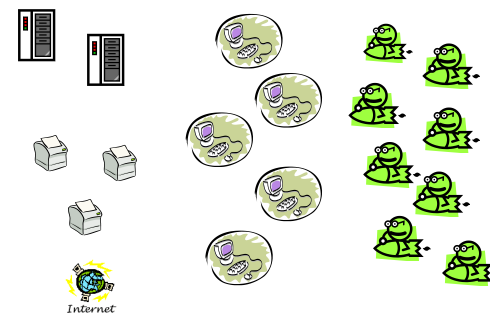

```

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?

© 2013 Gribble, Lazowska, Levy, Zahorjan 23

So imagine how complicated life is in the distributed world!



© 2013 Gribble, Lazowska, Levy, Zahorjan 24

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?

© 2013 Gribble, Lazowska, Levy, Zahorjan

25

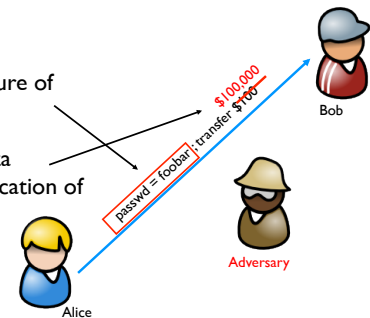
Communication security goals

Privacy of data

Prevent exposure of information

Integrity of data

Prevent modification of information

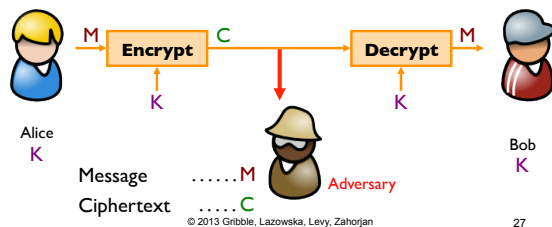


© 2013 Gribble, Lazowska, Levy, Zahorjan

26

Encryption schemes: A tool for protecting privacy

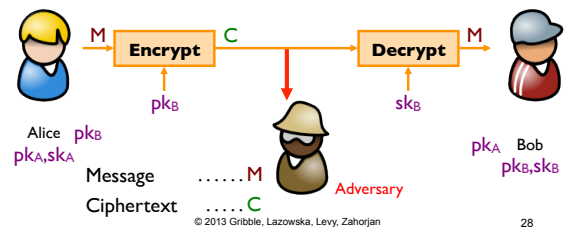
Symmetric setting: Both communicating parties have access to a shared random string K , called the **key**



© 2013 Gribble, Lazowska, Levy, Zahorjan

27

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

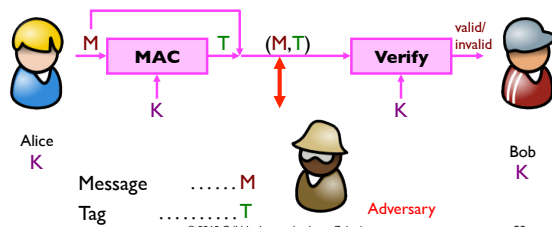


© 2013 Gribble, Lazowska, Levy, Zahorjan

28

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

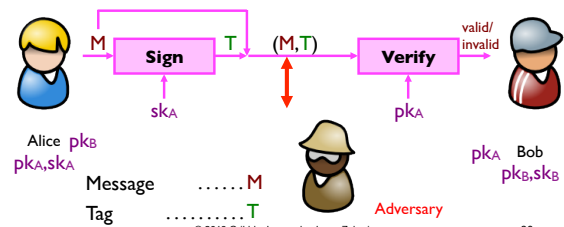
Symmetric setting



© 2013 Gribble, Lazowska, Levy, Zahorjan

29

Asymmetric setting: Digital signature, protects **integrity** and **authenticity**

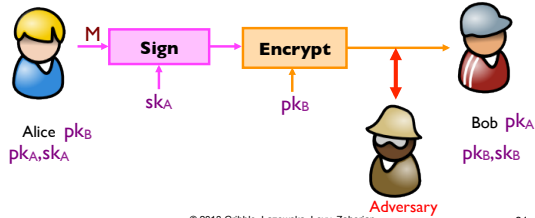


© 2013 Gribble, Lazowska, Levy, Zahorjan

30

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

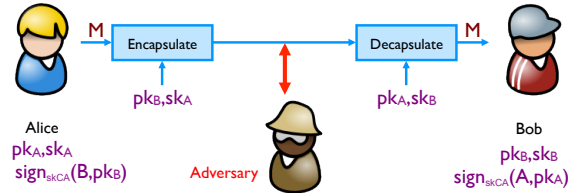


© 2013 Gribble, Lazowska, Levy, Zahorjan

31

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



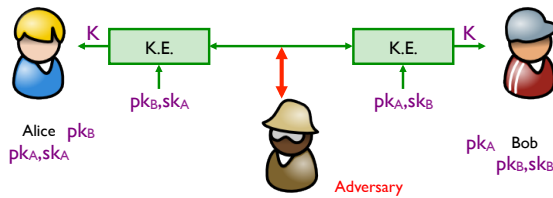
© 2013 Gribble, Lazowska, Levy, Zahorjan

32

Key exchange

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

(Why? Public key systems are relatively slow!)

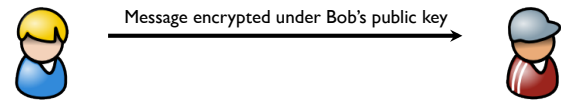


© 2013 Gribble, Lazowska, Levy, Zahorjan

33

One-way Communications

PGP is a good example



But life is never this simple!

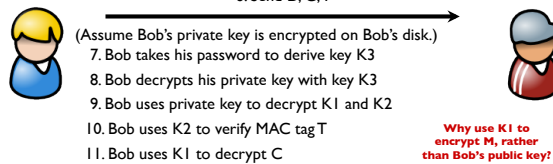
© 2013 Gribble, Lazowska, Levy, Zahorjan

34

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
4. Alice authenticates (MACs) C with key $K2$; call the result T
5. Alice encrypts $K1$ and $K2$ with Bob's public key; call the result D
6. Send D, C, T



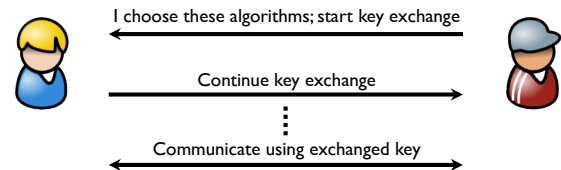
Why use $K1$ to encrypt M , rather than Bob's public key?

© 2013 Gribble, Lazowska, Levy, Zahorjan

35

Interactive Communications

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



Again, life is never this simple!

© 2013 Gribble, Lazowska, Levy, Zahorjan

36

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 ...)

© 2013 Gribble, Lazowska, Levy, Zahorjan

37

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 billion. 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 billion.

"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 "emailpassword," he could seize any person's Passport account and change the password associated with it.

© 2013 Gribble, Lazowska, Levy, Zahorjan

38

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

© 2013 Gribble, Lazowska, Levy, Zahorjan

39

Additional modern security problems

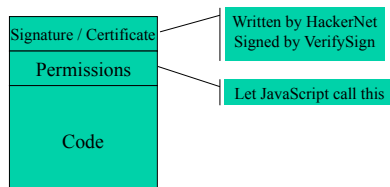
- Confinement
 - How do I run code that I don't trust?
 - e.g., RealPlayer, Flash
 - How do I restrict the data it can communicate?
 - What if trusted code has bugs?
 - e.g., Internet Explorer
- Solutions
 - Restricted contexts – let the user divide their identity
 - ActiveX – make code writer identify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

© 2013 Gribble, Lazowska, Levy, Zahorjan

40

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



© 2013 Gribble, Lazowska, Levy, Zahorjan

41

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

© 2013 Gribble, Lazowska, Levy, Zahorjan

42

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
```

© 2013 Gribble, Lazowska, Levy, Zahorjan

43