Security in an Operating System

Operating systems need to protect against two types of security violations:

Accidental security violations, e.g.:

- a program accidentally overwrites a file;
- a user issues “rm -rf *”, unaware of the current working directory.

Intentional security violations, e.g.:

- a user reads data that he/she is not supposed to read;
- a user tries to gain control of another user's process in order to perform operations for which he/she does not have permission.

Accidental security violations are easier to deal with, because they are usually limited to unintentional write operations.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Security in UNIX

Security permissions in UNIX: **R**ead, **W**rite, e**X**ecute.

Every object has a *user* (the *owner*) and a *group*. Different security permissions may be specified for *user*, *group*, and *others*.

Types of objects:

- files;
- directories;
- mounted file systems;
- shared memory segments;
- message queues;
- ...

In addition, Access Control Lists (ACLs) allow to grant more specific permissions, e.g., the permission to change permissions.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Principle of Least Privilege

In UNIX, a process by default has the same rights as its owner.

Problem:
What if the program contains a bug that corrupts some data completely unrelated to the program itself?

Solution: Protection domains.

A protection domain is a set of objects (e.g., list of files) with associated access permissions. The objects do not need to be listed explicitly.

The protection domain of a process may be different from the protection domain of its user.

UNIX does not explicitly support protection domains, but:
setuid, setgid, setfsuid, chroot, ...

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Principle of Least Privilege

Violations of the principle of least privilege are ubiquitous.

Examples:

- Rights management in UNIX: Almost every user process runs with all of the user's privileges.
- Monolithic kernels: Every part of the kernel has access to the entire system.
- System administrator: Does the admin really need to have access to every process's address space in order install a new software package?
- When machine *A* mounts a file system on machine *B* via NFS, should root on *A* have the same access privileges as root on *B*?

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Authentication

How does a user convince the computer that she really is who she is?

- password-based authentication;
- biometrical authentication, e.g., retina scan, fingerprint;
- cryptographical authentication;
- physical tokens.

All four types of authentication have advantages and disadvantages.

Which one is the best?

In addition to the above four, there are some bogus mechanisms, such as my bank asking me for the name of my first pet whenever I do on-line banking from a previously unseen IP address.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Biometrical Authentication

Biometrical authentication makes it very difficult for a person to impersonate another person.

retina scan, voice recognition, fingerprint, ...

However:

- Biometrical authentication only works if the computer that performs the authentication is trustworthy.
- If user *U* establishes her authenticity with computer *A*, how is *A* going to convince computer *B* that *U* is really *U*?
- Biometrical authentication is expensive and inherently error-prone (it is a physical measurement after all).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Token-Based Authentication

Tokens can be stolen.

Therefore, token-based authentication is usually combined with one of the other three authentication types.

For example:

- smart card with built-in fingerprint recognition device;
- debit card with PIN (password authentication);
- credit card with physical signature (*really secure!*).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Password-Based Authentication

Password-based authentication is the oldest type of authentication and is the easiest to realize.

Passwords are no physical items and therefore cannot be stolen.

Unfortunately, it is possible to guess a password (and to forget it...).

Also possible: Rubber-hose attack (or *rubber-hose cryptanalysis*).

> *...the rubber-hose technique of cryptanalysis (in which a rubber hose is applied forcefully and frequently to the soles of the feet until the key to the cryptosystem is discovered, a process that can take a surprisingly short time and is quite computationally inexpensive)*
>
> (Marcus J. Ranum on sci.crypt, 1990-10-16)

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Password-Based Authentication

Ask a question

Friday October 22, 2004 Previous | Next

Dear Yahoo!:

What's the most popular password?

Dave
Salem, Oregon

Dear Dave:

On average, the human brain can hold only five to nine "random bits of information" in short-term memory. Considering the brain's limited capacity and the sheer number of secret names, codes, and words a person needs to remember in this password-protected age, it's no surprise that the most common password is simply "password."

Besides serving as an easy-to-remember code for less-creative computer users, "password" is often used as the default password for many web sites and programs, making it extremely common and not at all secure. In other words, "password" is a bad password.

Other perennial favorites include "God," "sex," "money," and "love." Passwords based on the names or birthdays of partners, children, or pets are also quite common. Here's a pretty lengthy list of **common passwords**. Make sure to scan it and look for yours. If yours made the list, it's probably a good idea to change it.

Related Links

- Ask Y!: What are the five most popular names for pets?
- Y! Full Coverage: Computer Security

More Questions About

- **Computers & Internet**

Get Ask Your Way

- Email
- Yahoo! Toolbar
- XML MY YAHOO!

(http://ask.yahoo.com/20041022.html)

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Password-Based Authentication

Further shortcomings of password-based authentication:

- If I use a password to convince a computer that I am I, then what prevents the computer (or the computer's administrator) to pretend it is me when communicating with other computers?
- If I login to a computer remotely, how do I make sure the password can only be read by the target machine and not by any other computer on the network packet's way to the target machine?
- How does the computer check that the password is actually correct? Needs to maintain a persistent (on-disk) database of all passwords – but what if the hard drive gets stolen (or thrown away)?

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Cryptographical Authentication

From a mathematical point of view, cryptographical authentication is the strongest form of authentication.

General idea:

The user proves to the computer that she has knowledge of a certain fact *(*referred to as *secret key* or *private key*), but does not reveal the fact itself.

Problem:

Cryptographical authentication involves complex calculations that can easily take many hours if performed by a human. Thus, the human needs the help of the computer, but can the computer be trusted?

The secret fact must be difficult to guess, which usually makes it rather difficult to remember it.
⇒ Use a passphrase to encrypt the secret.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Encrypted Communication

Assume we want to transmit a password between two computers *A* and *B*.

There are known mechanisms (e.g., the Advanced Encryption Standard – AES) to encrypt the communication taking place between *A* and *B*. But all such methods require *A* and *B* to share a common secret, the *session key*, that let's them encrypt and decrypt messages.

Maybe *A* and *B* agreed on a session key ahead of time, but for many applications (e.g., online banking), this is not possible.

How to transmit the session key?

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Diffie-Hellman Key Exchange

Invented by Diffie and Hellman in 1976 (and by some British spook a few years earlier).

Fix a prime number P and a generator G.

Generator means that for every $0 \leq X < P$ there is a Y such that $G^Y \bmod P = X$. A generator always exists.

Neither P nor G need to be kept secret.

A picks a random number R_A, $0 \leq R_A < P$, and sends $G^{R_A} \bmod P$ to B.
B picks a random number R_B, $0 \leq R_B < P$, and sends $G^{R_B} \bmod P$ to B.

The secret key is $(G^{R_A})^{R_B} \bmod P = (G^{R_B})^{R_A} \bmod P = (G^{R_A * R_B}) \bmod P$.

An eavesdropper can hear G^{R_A} and G^{R_B}, but neither R_A nor R_B.

Finding X, given $G^X \pmod P$ is called the *Discrete Logarithm* problem (DL). Nobody knows how to solve it.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Man-In-The-Middle Attacks

The problem with Diffie-Hellman is that it assumes that G^{R_A} (or G^{R_B}) actually originated from *A* (or *B*).

However, in an unsecure network we cannot be sure of anything. A third computer might pretend to be *A* when talking to *B* and *B* when talking to *A*.

SSL addresses this problem by introducing key fingerprints:

```
[stu@stu slides]$ ssh sbuettcher@student.cs.uwaterloo.ca
The authenticity of host 'student.cs.uwaterloo.ca (129.97.152.10)' can't
  be established.
RSA key fingerprint is a7:56:ce:63:48:f7:6e:73:8d:67:07:3e:1b:5d:4a:d8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'student.cs.uwaterloo.ca,129.97.152.10' (RSA)
  to the list of known hosts.
sbuettcher@student.cs.uwaterloo.ca's password:
```

Problem: It is too convenient to simply type “yes”.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Asymmetric Cryptography

Asymmetric cryptography is also referred to as *public-key cryptography*.

It is asymmetric because sender and receiver use different keys to encrypt and to decrypt a message.

Every user *U* has a *private key* that only the user herself knows.

Everybody who wants to communicate with *U* can look up her *public key* in a directory (e.g., key server) and use the public key to send encrypted messages to *U*.

Only *U* herself can decrypt the messages encrypted with her public key.

Knowing the public key, it is impossible (i.e., computationally infeasible) to infer the private key.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

RSA

RSA was invented by Rivest, Shamir, and Adleman in 1977 (and by some British spook in 1973). It is one of the most popular public-key cryptography systems.

Like the Diffie-Hellman key exchange, it relies on computations performed using modular arithmetic.

RSA is particularly interesting because it cannot only be used to encrypt a message, but also to issue a digital signature that allows the receiver to verify the authenticity of a message.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

RSA

Generating the private key

Alice picks two random prime numbers P and Q and a random number A: $0 < A < (P\text{-}1)*(Q\text{-}1)$.

She then computes B such that $A * B \equiv 1 \pmod{(P\text{-}1)*(Q\text{-}1)}$, i.e., $A * B = 1 + k * (P\text{-}1) * (Q\text{-}1)$, for some integer k (ext. Euclidean Alg.).

Let $N := P * Q$.

Then Alice's private key is: (A, N). Her public key is: (B, N).

Given a message X (a number between 0 and N), Bob can encrypt the message by computing $Y = X^B \pmod{N}$.

Alice can decrypt the message by computing $X = Y^A = X^{B*A} \pmod{N}$.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

RSA

Why does RSA work?

Two prime numbers P, Q, and $N := P * Q$.

Two numbers A and B such that $A * B = 1 + k * (P\text{-}1) * (Q\text{-}1)$.

Then for any number X, we have:

$$X^{A*B} = X^{1+k*(P-1)*(Q-1)} = X \pmod{P}$$
$$\text{and} \quad X^{A*B} = X^{1+k*(P-1)*(Q-1)} = X \pmod{Q}.$$

This is because of Fermat's theorem: $X^{P-1} = 1 \pmod{P}$ for every X.

The *Chinese Remainder Theorem* states that if $X = a \pmod{P}$ and $X = a \pmod{Q}$ for some a, then $X = a \pmod{N}$ for $N = P * Q$.

If you really want to understand what is going on here: CS487 (maybe) or C&O 485.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

RSA and Digital Signatures

We have seen that a message X can be encrypted by computing

$$Y := X^B \pmod{N},$$

where B is Alice's public key.

What if Alice takes a message X and encrypts it with her private key A?

$$Y := X^A \pmod{N}.$$

Then everybody who knows her public key B can decrypt the message: $Y^B = (X^A)^B = X \pmod{N}$. By doing so and making sure that the result is meaningful, it is possible to verify that the original message was in fact encrypted using Alice's private key. This is called a *digital signature*.

In practice, X will not be the entire message that is to be signed, but a hash value of the original message.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

RSA and Factorization

Breaking RSA is as difficult as solving the integer factorization problem.

Integer Factorization (simplified version): Given a large integer N ($>2^{1024}$) that is a product of two primes, find N's prime factors.

Like for the Discrete Log problem, nobody knows an efficient algorithm to solve Integer Factorization.

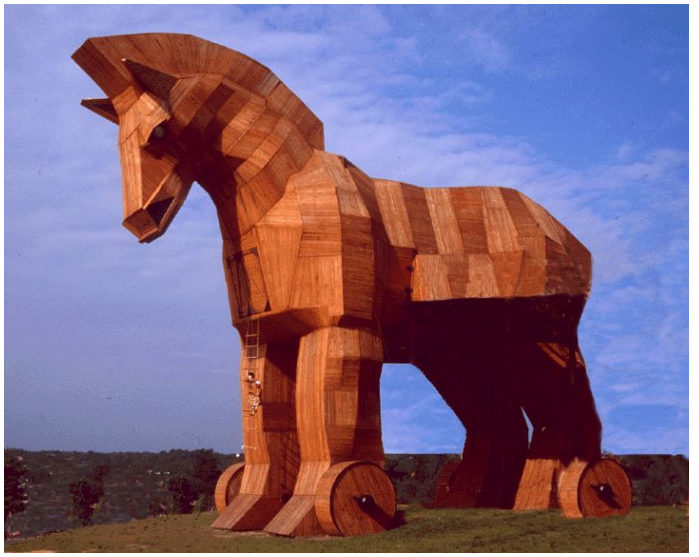
It can be shown that if there is an efficient (polynomial-time) algorithm to solve the Discrete Log problem, then there is an efficient (polynomial-time) algorithm (randomized) to solve the Integer Factorization problem.

⇒ Discrete Log is at least as difficult as Integer Factorization.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Trojan Horses

A Trojan Horse (sometimes simply referred to as “Trojan”) is a program that pretends to do something else than what it actually does.



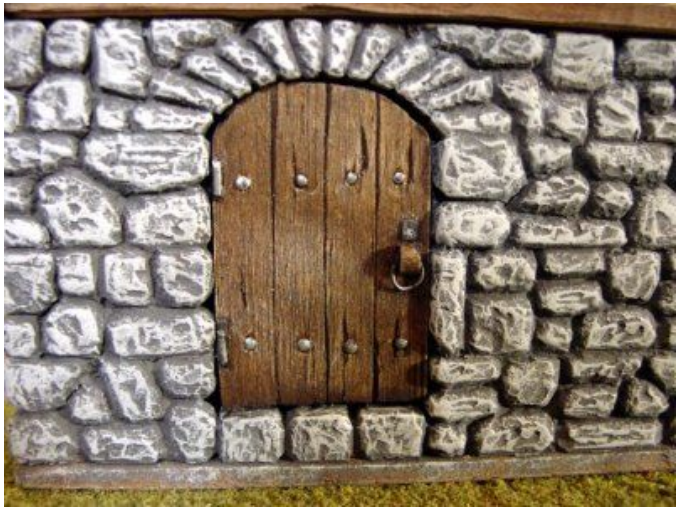
Classical example:
A program that looks like the machine's login screen and asks the user to enter her password.

Named after the wooden horse (filled with warriors) that the Greeks left behind when they pretended to stop their siege of Troy.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Back Doors

A *back door* (or *trap door*) is a special rule that a programmer leaves in the code and that allows her to circumvent security restrictions.



This includes master passwords for ATMs (usually approved of by the bank for some reason) and back doors that the employer is unaware of.

There were some cases where a programmer tried to blackmail her former employer after getting fired because there was a back door in the system that the programmer could use to do evil.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Computer Viruses

Not a free-standing program, but attached to some legitimate program.

Like a real virus, cannot survive on its own, but relies on its host.

Very common in the 80s/90s.

An .EXE file that was infected would leave the virus in memory even after it terminated. When other programs were started, they were automatically infected by the virus.

Traditional viruses are very rare these days and have mostly been replaced by internet worms or script viruses (MS Word, ActiveX, ...).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Computer Worms

A *worm* is a program that replicates itself from one system to another, usually via the Internet.

Sometimes, a worm exploits a buffer overflow in an application or in the kernel.

Other worms use scripting languages or security holes in ActiveX to replicate.

One of the first worms: The Morris worm (1988). Spread by using legitimate access of compromised user accounts (.rhosts files). Authors was sentenced to 3 years on probation, 400 hours community service, and 10,000$.

In many cases, worms are used to take control of a large number of computers in order to use them to send spam or to launch distributed denial-of-service attacks (dDoS).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Buffer Overflows

If a program received some input data and copies it into a fixed-size buffer without checking whether the buffer is large enough to hold the data, then this causes a *buffer overflow*.

The data received might contain executable code that accidentally gets executed at some later point, or they might just contain some carefully engineered “garbage” data.

If you are lucky, the program just crashes.

If you are less lucky, the attacker gains control of your account.

Most common form of buffer overflows: Buffers on the stack that may corrupt the function's return address (because stack grows from high to low).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Firewalls

The Internet is dangerous.

As soon as you connect a computer to the Internet, somebody will start trying to break into it. This *somebody* is usually a worm.

CMP TechWeb The Business Technology Network

FOCAL POINTS: Sponsored Links
Enterprise VoIP, Business Innovation, WAN Acceleration

SEARCH Enter term(s) go

NEW techseArch Try it now!

Blogs | News | Mobile | Software | Security | E-Business & Management | Networking | Hardware

August 18, 2004 (10:31 AM EST)

Unpatched PC "Survival Time" Just 16 Minutes

techseArch for related stories

TechSearch for more articles on:

Search Now

Unpatched PC "Survival Time" Just 16 Minutes

By Gregg Keizer,

The average unpatched Windows PC lasts less than 20 minutes on the Internet before it's compromised, according to data from the Internet Storm Center.

TECHWEB MARKETPLACE

(Sponsored Links)

(http://www.techweb.com/wire/30000109)

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Firewalls

How to connect a computer to an untrusted network?

Put a *firewall* between them.

All network traffic must flow through the firewall.

The firewall

- limits network access (in- & outbound) – bad news for worms;
- monitors all traffic and logs it for later analysis;
- possibly even some real-time analysis with automatic warning messages to the administrator.

The firewall may even reside on the same machine (e.g., Linux iptables).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Firewalls

Since firewalls perform some extra computation on all data, they might be subject to denial-of-service attacks.

A firewall usually cannot deal with a tunneling attack:

- inbound port 80 (HTTP) – potentially;
- outbound port 80 – no way to detect whether an application is piggybacking something on HTTP.

Some networks have a *demilitarized zone* (DMZ) for which the firewall allows all incoming traffic, while for the rest of the network incoming traffic is only allowed if it is part of an established connection initiated from within the trusted network.

Stefan Buettcher
<sbuettch@uwaterloo.ca>