Knowledge - Passwords

Extremely common way of gaining unauthorised access to computer systems.

Password guessing

- brute force – try every possible combination

The system should spend several seconds before replying and deny access after a few attempts.

The greater the number of symbols and the length of the password the harder it is.

Many Unix systems have a maximum password length of 8.

- intelligent search
  - try default passwords (unfortunately common)
  - sometimes the user hasn't even set a password
  - words associated with the user – names of friends, relatives, pets, dates, hobbies, phone numbers, and the same things backwards.
  - common passwords – "sesame", "password", …
  - dictionary attacks – word lists

Most commonly done when password files are extracted from a site. Can find a very high percentage of hits.

  - Substituting numbers for similar letters e.g. 3 for E.
  - Typing one row higher or lower.

Stealing passwords

Shoulder surfing

Video cameras

Snooping on a network for plain text passwords

Keyloggers

Trojan horse login screen

Keyboard sniffing

wireless keyboards

but also

can use the electromagnetic radiation emitted when keys are pressed on wired keyboards

Making passwords safer

1. Don't write them down.

2. Use mixed upper and lower case letters with numbers and symbols.

   Better to use the first letters of a phrase

3. Change them regularly (but only in some cases).

This can be enforced by the system, along with other common password requirements.

Usually prevent the user from using an earlier version.

Unfortunately this makes it harder to remember and hence the user is more likely to write it down.

4. Have system produced passwords.

Random but pronounceable

If people forget their passwords they need the sys admin to give them a new one.

This also requires authentication.

Many passwords have been bullied out of sys admins over the phone for example.

See http://xkcd.com/936/ "correct horse battery staple"

Password files

The password has to be kept somewhere.

Either keep the password file secret or one-way encrypt its data (preferably use both).

If the file is readable they can be broken by dictionary attacks.

One-time passwords

a new one produced at the end of a session

security tokens - time based / algorithm based

challenge/response

Rather than memorising a password an algorithm can be the secret.

System issues a challenge e.g. an integer.

The user responds with the value of using that integer as input to the algorithm.

Can be made one-time by using secret seeds that are generated each use. In this case the user needs the algorithm (and seed) on another protected computer or smart card.

Two-factor authentication - e.g. PIN and message sent to your phone

Single sign-on

With distributed environments we don't want people to have to sign-on to every machine they use during a session.

- enter a password at the computer
- enter a password to use the network
- enter a password to access a server
- enter a password to use a database
- enter a password to open a table in the database

We can use a single sign-on service – which remembers our passwords and supplies them to the systems that need them.

We must ensure we don't store passwords in cleartext and that they aren't transmitted in cleartext.

Kerberos

Uses tickets granted by central security servers.

Principals – users and servers.

Kerberos Authentication Server (KAS) – checks principals at login and issues tickets for ticket granting servers.

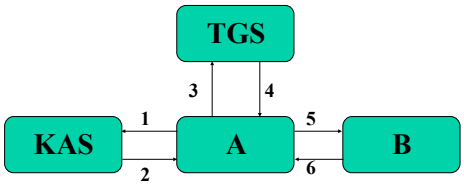
Ticket Granting Server (TGS) – issues tickets to network services

Remember tickets are time expiring capabilities.

We want two principals (A & B) to mutually authenticate themselves.

Based on the Needham-Schroeder protocol. But A needs to authenticate itself to a TGS on the way to getting the service it wants from B.

Kerberos authentication



1. A asks KAS for a session key (for private communication between A & TGS) and ticket to TGS.

2. KAS returns $Ticket_{a,tgs}$ (includes A's ID, network address, valid period and $K_{a,tgs}$) along with the key $K_{a,tgs}$ for A and TGS to use. (All encrypted with A's secret key.)

$Ticket_{a,tgs}$ is encrypted with TGS's secret key, this stops A (or any one else) modifying it.

Kerberos authentication cont.

3. A asks TGS for a session key and a ticket (capability) to talk to B. It includes $Ticket_{a,tgs}$ and an authenticator proving it comes from A (encrypted with $K_{a,tgs}$).

4. TGS replies with $Ticket_{a,b}$ (along with the key $K_{a,b}$ for A and B to use. (This is encrypted with $K_{a,tgs}$.)

$Ticket_{a,b}$ is encrypted with B's secret key, this stops A (or any one else) modifying it. It includes the key for A and B to use.

Timestamps are included in the authenticators and random challenges (nonces) are sent as well.

5. A sends the ticket to B along with an authenticator (encrypted with $K_{a,b}$).

6. B replies with its own authenticator. (This is also encrypted with $K_{a,b}$.)

Aspects of Kerberos

How are rights (tickets) revoked under Kerberos?

KAS and TGS databases need updating.

But tickets stay valid until they expire.

(KAS tickets about one day. TOCTTOU problem.)

There is a trade-off with TGS tickets – short expiry times means more requests to the servers. Long expiry times means less control but helps if some TGS servers are occasionally out of action.

A Kerberos *realm* is a number of servers under one administrative domain.

- All principals have to be registered with the KAS.
- The TGSs have to have access control information.

There can be hierarchies of realms.

The KAS and TGS servers have to be trusted, since they generate the session keys.

Keys and tickets are held on local machines (so each machine must be able to keep these secret from other local processes).

Program threats

When a user runs a program written by another user there is always the potential for misuse.

Trojan horse

A program that has hidden side-effects.

Trojan horses can be hidden in search paths.

Spoofing attacks and phishing e.g. man-in-the-middle or presenting fake login screens.

- This can be stopped with non-trappable key sequences or reporting the number of unsuccessful login attempts.

Backdoors

Leaving hidden access to the programmer.

Disgruntled employees.

Compromised compilers - producing compromised compilers.

Logic bombs

Goes off under particular circumstances.

If I don't login every week wipe all files.

Root-kits

Replace standard commands with versions which hide the presence of the kit. Usually used to keep access hidden.

System threats

Worms – replicate and spread and can bring systems to a standstill.

Performance is affected. Can lead to …

… DoS – Denial of Service (flooding) attacks on networks.

Viruses – spread and do damage.

Viruses should not be as dangerous on multi-user systems as usually only the individual user's files can be infected. Not the system files - unfortunately not true if the user is the superuser.

Mobile Security

Extra problems.

Portable devices

Always with us, easy to lose/steal

Convenience required

We don't want to make using the device difficult

How can we make it safe?

- minimise the attack surface
- code signing
- user permissions to do things
- sandboxing

Attack Surface

What code can be run without authentication?

The less code which is reachable from outside the better.

Many security risks have been found in 3rd party layers such as Java and Flash environments in browsers, and components in pdf readers.

- Getting rid of these immediately increases the difficulty that attackers have to find exploits (bugs which give them access that should not be allowed).
- Removing this type of functionality is reducing the attack surface.

Checking before running

Only running code which has been checked strongly reduces the chance of running malware.

Apple's App Store and Google's Google Play

They perform checks on submitted apps (playing the role of an anti-virus program).

AV programs are mostly ineffective on mobile devices (and pretty much unnecessary on iOS)

Code Signing

Code signing is used for very different purposes on iOS and Android.

In iOS Apple signs every app (all apps must come from the App store)

This means that you can be sure the app you install and run is the same code that Apple verified. It has not been altered.

The app is only installed if properly signed and when the program runs each page of memory is checked to make sure it has not been altered.

In Android apps are signed by the developer (there is no need to use a CA). Apps can come from anywhere (much safer if you only get them from Google Play).

The code signing is so that updates can be verified to come from the same developer and establishing trust between apps.

Jailbreaking removes most of the checks on code signing (actually allows other signers, including self signed certs).

Permissions

Usually not a good idea to allow all apps access to all of the services on the device. e.g. Granting read access to messages to an app that doesn't need them.

Android asks the user at install time.

Presents a list of privileges it wants. Remember that the app is only checked by Google if you use Google Play.

Supposed to be minimal but it is up to the developer.

No explanations are given.

Most users just click "accept".

The program is not installed if rejected.

Apple didn't use to ask users except for location information

If an app was passed by the App Store it could do anything it wanted.

No longer true - now as apps request access to photos or messages the user is asked to allow or not.

If the user turns down a permission the program is still supposed to run (but obviously it can't do everything).

Sandboxing

If an app goes bad or allows an external exploit the last protection is the sandbox.

In Android the default is files in internal storage are only accessibly by the creating app. (Can be made accessible to others.)

Files on external storage (SD cards) are globally readable and writable. Not so on iOS (trick question, why not?).

In iOS all files can only be created in the sandboxed area and there is very limited ability to pass information from one app to another. (Now there are controlled ways since iOS 8 to share data.)

Before next time

Read from the textbook

- 13.3 Application I/O Interface
- 18.8 Input and Output
- 13.5 Transforming I/O to Hardware Operations