

# 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.
  - Unfortunately 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 – words in dictionaries

# Making passwords safer

1. Don't write them down.
2. Use mixed upper and lower case letters with numbers and symbols.
3. Change them regularly.
  - 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.
5. 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.
6. Single use passwords – a new one produced at the end of a session.

## 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.

### **Algorithms as passwords**

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.

## 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 dbms
- enter a password to open a table in the database

We can use a single sign-on service – which remembers our password(s) and supplies it to the systems that need it.

Unfortunately this needs a cleartext password file.

# 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.

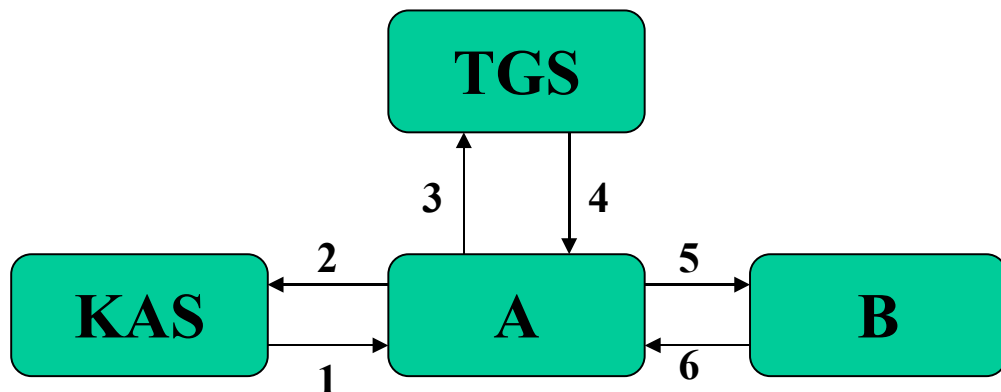
Ticket Granting Server (TGS) – issues tickets to network services

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  $\text{Ticket}_{a,tgs}$  along with the key for A and TGS to use. (This is encrypted with A's secret key.)

$\text{Ticket}_{a,tgs}$  is encrypted with TGS's secret key, this stops A (or any one else) modifying it. It includes the key for A and TGS to use.

## Kerberos authentication cont.

3. A asks TGS for a session key and a ticket (capability) to talk to B. It includes an authenticator proving it comes from A.
4. TGS replies with  $\text{Ticket}_{a,b}$  along with the key for A and B to use. (This is encrypted with the  $a, \text{tgs}$  key.)

$\text{Ticket}_{a,b}$  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.
6. B replies with its own authenticator. (This is encrypted with the  $a,b$  key.)

# 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 – presenting login screens is an example of a Trojan horse.

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

## Trap doors

Leaving hidden access to the programmer.

Disgruntled employees.

Similarly time-bombs (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 are not 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.

# EALs

Evaluation Assurance Level (1999) the  
Common Criteria project – replaced the  
Orange Book levels, DoD Trusted  
Computer Security Evaluation Criteria –  
1985

How much can you trust a computer system?

Specifications for different levels of security.

- (EAL 0) D – minimal protection  
MS-DOS, early Macintosh, Windows 3.1
- (EAL 2,3) C – discretionary protection,  
“need to know”
- (EAL 4,5,6) B – mandatory protection
- (EAL 7) A – verified protection

# Discretionary protection

## C1 – cooperating users of the same level of integrity

Users have to be authenticated.

Discretionary access control based on users and/or groups. It doesn't have to be used.

Early versions of UNIX.

## C2 – controlled access protection

Like the above but access can be controlled at the individual level. i.e. Each subject can have its own permissions on each object.

Object reuse (e.g. memory) should not give users access to information from other users.

Audit trails have to be kept.

Some versions of UNIX and some installations of Windows NT.

# Mandatory protection

## B1 – labelled security protection

Labels for subjects and objects – showing the security classification.

Label integrity is protected.

Hierarchy of subjects and objects.

Output is labelled too – e.g. when printing a sensitive document the pages are labelled with the security classification.

There has to be a model of the security policy.

Much more thorough testing and documentation.

Some Unix and dbms systems have been certified B1.

## B2 – structured protection

Mandatory access control to devices as well.

Users have to be notified of changes to their security levels.

Login and authentication must be via a trusted path.

A formal model is required.

# Pretty secure

## B3 – security domains

Auditing mechanisms issue automatic warnings of suspicious activity.

Trusted recovery after a system failure.

Modules are excluded that are not security relevant (can't be used for ordinary activity).

Consistency can be convincingly shown with the formal model.

## A1 – verified design

Formal methods to prove security.

I/O channels have to be justified and bandwidth may have to be limited.

# Before next time

## Read from the textbook

19.3 – Program Threats

19.4 – System Threats

19.5 – Securing Systems and Facilities (*Threat Monitoring*)

19.8 – Computer Security Classifications

19.9 – An Example Security Model: Windows NT