Computer Science Tripos part 1b — Security

Lecture 2 — Introduction to Access Control

Ross J. Anderson

1 Passwords

We will usually not want every user of a computer system to have access to every resource. We will want most users to be restricted to a subset of the filespace, and perhaps forced to access certain files using particular programs. So we need to consider how to control access.

The first thing we will have to do is authenticate the user. There are three approaches to this: something you have (such as a key), something you know (such as a password), and something you are (such as a fingerprint or iris scan). For reasons of cost, most systems take the second option; and even where we use a physical token such as a hand-held password generator, it is common to use a password as well (to arm it).

So passwords are important, and a number of points need to be made.

1. Many users choose passwords which are easy to guess. Left to their own devices, they will use spouses' names, single letters, or even just a carriage return as their password. For this reason, some systems require minimum lengths or even check user entered passwords against a list of bad choices. But this almost ensures that many users will write their passwords down.
2. The password file is an obvious target for intruders. One way of protecting it, which is used in Unix, is to encrypt the passwords using a one-way algorithm (this idea was invented in the 1960's by Needham and Guy).
3. However, some systems make the encrypted password file world readable (Unix does this — a design error now too well entrenched to change easily). This means that an opponent can fetch this file and then try to break passwords offline using a dictionary. This kind of attack will typically find 10-20% of passwords.
4. Government systems tend to issue randomly selected passwords to users in order increase the entropy of the password space. A password might be of the form CVCNCVCN (eg fur5leb8) — but the entropy is still only $4.62 \,.\, 10^8$, which can still be searched offline, and the danger of passwords being written down is real.
5. Some products, such as PGP, allow you to use a passphrase instead of a password. The entropy of this has been estimated at $O(10^{11})$, so it is still not beyond the scope of offline search.

The ability to do offline password guessing more or less guarantees that an attacker will be able to compromise at least some accounts on any system. The obvious remedy is to stop offline guessing, e.g. by keeping the password file secret and/or using better logon protocols, and enforcing a policy of allowing only three guesses before the account is frozen. However, systems such as Unix already have such a large installed base of software that changing the password mechanism is often impractical.

2 Multistate Machines

Many cpus have some simple security mechanism supported in the hardware, on which the higher level access control mechanisms are built.

For example, early IBM mainframes had a two state cpu: the machine was either in authorised state or it was not. In the latter case, the program was restricted to a memory segment allocated by the operating system; in the latter, it could go across memory and alter other programs and their data at will. The cpu told whether a program was authorised or not from the library it was in, so system security depended on keeping bad code (whether malicious or buggy) out of the authorised libraries. These mainframes were completely at the mercy of staff, such as system programmers, who had access to authorised libraries.

Much the same happens with Unix systems, where a program can run as root or as a user. Here, too, we are at the mercy of all code running as root, and many security holes have resulted from bugs in privileged code (or which allowed users to run their code with root privileges). As it is often easier to write programs that run as root, we have a serious problem or root bloat.

Most security holes have always been caused by bugs, and so we might try to minimise the amount of code which we have to trust by concentrating our security critical functions in a subset of the system, often called the Trusted Computing Base or TCB. If the TCB has much less code, then hopefully it will have far fewer bugs. An example of a TCB is the banking security module used to handle ATM PINs; like an IBM mainframe, this is two state, with 'authorised state' having to be set (by two console passwords) if clear PINs are to be printed, or if clear key components are to be either input or output. In this case, the two state architecture is used to support the banking security policy of dual control.

The Intel architectures are a bit messier. Although the 80386 and subsequent processors will run multiple virtual 80286's and support multiuser operating systems, they will not fully virtualise themselves (unlike the IBM S/360 processors). This is why one cannot run Linux and Windows simultaneously on the same processor. So designers of PC access control systems designed to cope with both 80286 and subsequent processors faced a choice: they could control access to DOS applications only, add extra hardware (such as an encryption chip in the disk controller, or plump for security-through-obscurity. Most did the latter, and could be broken by disassembly attacks.

On the other hand, if future Intel CPUs can be fully virtualised, then this might make it in principle impossible for an application to detect whether its execution was being examined by a disassembler, which could make a number of copy protection schemes (such as tamper-resistant soft DVD implementations) more vulnerable.

We conclude that a CPU should be multistate if we want to provide mutual security to multiple users; that one bit of state is in theory enough; but that in practice there may be some quite complex trust issues.

3 Typical Access Control Mechanisms

Given this one bit of state, we can build an entire access control system by writing an authorised program to manage users. Other mechanisms may be used too; some systems have a fence address — a boundary in the hardware below which only the operating system has access — while in the IBM architecture, the most important extra mechanism is that the registers which store offsets for code and data addresses can only be changed by an authorised program. However, the machine still needs at least one bit of state to know whether it is currently authorised or not.

3.1 Access control matrices

The access control system is designed to limit access by users to system resources. The main system will typically be either part of, or a tightly coupled addition to, the operating system. As the operating system works at the granularity of files, this will usually be the smallest object which the access control mechanism can deal with, and so there may be secondary access control systems implemented in database and other software.

However, let us set aside these concerns for the time being and consider what an access control system does. Its effect can often be modelled by a matrix of access permissions, with columns for files and rows for users:

	File 1	File 2	File 3	File 4
Sam	rwx	rwx	rw	r
Alice	x	x	rw	0
Bob	rx	r	r	r

In this simplified example, we can imagine that file 1 is the operating system, file 2 is the application, file 3 contains the data, and file 4 is the audit trail. Sam is the system administrator and has universal access (except to the audit trail, which even he should only be able to read); Alice, the manager, needs to execute the operating system and application, and read and write the data; Bob the auditor can read everything.

Recall that the Clark-Wilson security policy limits access to CDIs (constrained data items) to authorised TPs (transformation procedures) which write an audit trail. If we wanted to implement such a policy in our system, we might make file 2 the TP and file 3 the CDI. Our access control matrix would then look something like

	File 1	File 2	File 3	File 4
Sam	rwx	rwx	rw	r
Alice	x	x	0	0
File 2	x	-	rw	w
Bob	rx	r	r	r

Access control matrices can be used to implement protection mechanisms as well as just model them. However, they do not scale well: a bank with 50,000 users and 300 applications would have an access control matrix of 1,500,000 entries. This is too large to cache, and if the operating system had to look it up with every disk access, the I/O bandwidth requirement would double.

3.2 Access control lists

One solution is to break up the access control matrix, and store it a column at a time. In this case, we would have an access control list or ACL for file 2, the application, of:

	File 2
Sam	rwx
Alice	x
Bob	r

We could store this with the file itself. This is how Unix works: each file has `rwx` attributes for its owner, the group, and the world. With most Unices, we have difficulty implementing an audit trail as a file that Sam cannot modify.

ACLs have a number of advantages and disadvantages as a means of managing security state. They are quite suited to environments where users manage their own file security — so-called discretionary access control, such as we have at the laboratory. You can make all your files world readable and writeable if you wish; there is no central mechanism to stop you. On the other hand, ACLs are not very good at coping with more complex operations such as one user's delegating her authority to run a particular program to another user.

ACLs are also not the most efficient way of doing security checking at runtime, as the typical operating system knows which user is running a particular program, rather than what files it has been authorised to access since it was invoked. The operating system must either check the ACL at each file access, or keep track of the active access rights in some other way. Finally, if we have a large and constantly changing population allowed to access a given file, then managing its ACL will be tiresome — and if this population can access more than one file, then ACL-based access control may become too tedious to use.

3.3 Capabilities

The other way to break up the access control matrix and make it more manageable is obviously to store it by rows. These are called capabilities, and in our example above, Bob's capabilities would be:

	File 1	File 2	File 3	File 4
Bob	rx	r	r	r

From some points of view this is preferable. Runtime security checking is much more efficient, and we can do delegation without much difficulty: Bob could create a certificate saying 'Here is my capability and I hereby delegate to David the right to read file 4 from 9am to 1pm, signed Bob'. On the other hand, changing a file's status can suddenly become more tricky; it can be very difficult even to find out which users have access. This can be tiresome when we have to investigate an incident or prepare evidence of a crime.

4 The Real World

All software is buggy, and security software is no exception. If we build an access control structure in several layers of software on top of a two state machine, then we will expect that security flaws will be found from time to time. For example, the PDP-10 TENEX operating system had the bug that the program address could overflow into the next bit of the process state word which was the privilege-mode bit; this meant that a program overflow could put a program in supervisor state.

4.1 Putting more of the security in the hardware

So if we want provable security, we might implement a bit more protection in the hardware. This was done in the early 1970's, when a capability based machine — the CAP — was built here at the Computer Laboratory. The CAP had special hardware to cache capabilities, and thus gave provable security with tolerable efficiency. In fact, most of its operating system ran in user mode. It was an example of a tagged architecture, in which security information is bound with objects in memory.

You may also see Multics mentioned in the literature. Multics was designed at the same time as CAP, and was widely used until recently in US defence networks. Multics machines also had more than one protection state — each process had a number from 0 upwards; ring 0 programs had complete access to disk, supervisor states ran in ring 2, and user code at various less privileged levels.

However, one protection bit is sufficient; one can (at least in theory) write a microkernel for a two-state machine which would enable it to emulate either the CAP or a Multics computer.

4.2 Hybrid systems

In an ideal system, we would manage users by capabilities and files by ACLs. Kerberos does something like this, in that users are authenticated by one security server (the key distribution centre) while files are managed by another (the ticket granting service).

The lab's MSSA system which Jean Bacon and Ken Moody have developed with their research students also uses a hybrid: ACLs are used to express security policy, and are checked once for each set of accesses. Thereafter, a transient capability is generated which allows efficient runtime checking with temporary delegation if required.

4.3 Groups and Roles

In a large organisation such as a bank, we may have tends of thousands of users and hundreds of applications, so we may have millions of bits of security state. Whether these are stored by rows or columns, we still have a lot of bits and have to manage them somehow.

When we look at real organisations, we usually find that most staff fit into one or other of a small number of categories (teller, chief teller, branch accountant, branch manager, ...). The remainder (security manager, chief foreign exchange dealer, ...) may amount to only a few dozen people. So rather than having several dozen people employed full-time to set up either ACLs or capabilities, we would rather have a small number of pre-defined functional descriptions to which staff could be assigned.

Some writers use the words 'group' and 'role' interchangeably in this context. However the emerging usage is that while group membership is fairly stable, a role is something that a user assumes for a period of time. The classic example of a role is the officer of the watch on a ship. There is exactly one watchkeeper at any one time, and it would normally be one of the ship's officers; but given enough casualties, the watch could end up being stood by an able seaman.

Groups and roles can be combined; an example of a group of roles would be 'the officers of the watch of all ships currently at sea'. An example from banking might be that the manager of the Cambridge branch might have his privileges expressed by membership of the group 'manager' and assumption of the role 'manager of Cambridge branch'.

4.4 Granularity

One practical problem is that an increasing number of applications are built using 4GL or database tools. All the application data may be bundled together in one file, and access control mechanisms which operate at the level of the operating system must either grant or deny a user access to the lot.

What this means in practice is that if you develop your branch accounting system under a database product such as Oracle, then you may have to manage one access mechanism at the operating system level, and another at the database or application level. Many real problems result. For example, the O/S and DBMS security administration may be performed by different departments, which do not talk to each other; and often user pressure drives DP departments to put in crude hacks which make the various access control systems seem to work as one, but which open up serious holes.

A serious quagmire is 'single sign-on'. As people in a company get more and more logons to different systems and the cost of administering these logons escalates, many companies look for ways to give each employee a single logon to all the machines on the network. A crude solution is to endow their PCs with a menu of hosts to which a logon is allowed, and hide the necessary userids and passwords in scripts; more sophisticated solutions may involve a single security server through which all logons must pass. Bugs in such systems are commonplace. But if users are allowed to choose passwords to a number of systems, then many of them will not only choose bad ones but will use the same password for many systems. The effect is that an attacker can find out passwords from weak systems and then try them on the others.

4.5 Dual control

As we saw, dual control is a frequent security requirement — it underlies many commercial system security policies. However, many access control systems support it poorly, and developers resort to hacks such as putting transactions authorised by manager A into a special file, from which manager B then moves them to further processing. It is tedious in Unix systems to ensure that files can only be accessed by a designated application program, and in almost all systems the administrator can override the dual control by creating two suitable accounts.

So where dual control is critical, it may be sensible to use multiple systems. One might have two servers, each under the control of a separate team, and give each of them a part of the key used to sign messages. Similar systems are used in military command and control, so that if a commander is killed in a decapitation attack, control passes to some combination of junior officers. The same idea is also used by the resilient systems community; secure processors are sometimes constructed from sets of less secure devices endowed with secure voting protocols.

4.6 Application dependance

In some systems, the information needed to create access control lists is derived from the applications themselves. For example, in a system used at a number of British hospitals, medical record access is allowed to the consultants responsible for the patient's care (physician, surgeon, haematologist, ...), to the junior doctors and administrative staff working for them, and to staff on the ward during the patient's stay and for the next 90 days.

Here, the direct entries on the access control list could be set by an administrator directly, and so could the entries for their staff; but the essentially geographical controls on access by nurses and the like make access controls dependent on the relevant administrative systems. The solution adopted is that all the access control decisions, other than those for consultants, are made by reference to the administrative system. This arrangement is not in practice a problem, as the latter system must have high-quality data for the hospital to work at all. This may be a useful example of how one cannot always insist on a minimal, operating-system-level TCB.

5 Failure Modes

In general, typical real world attacks consist of the opportunistic exploitation of bugs that are descovered by chance. For example, one top selling dial access system in the 1980's had a default software support user id of 999999 and a password on 9999. It also had a default supervisor id of 777777 with a password of 7777. A surprising number of these systems were penetrated by hackers in the mid 1980's!

Another good example was a bug in a new release of VAX/VMS a few years ago. This had the effect that if you tried to log on with the wrong password, the system would tell you this; however, if you then hit carriage return, it would log you in regardless. This bug was discovered fairly quickly, and DEC rushed a patch to all their licensed users. However, the Chaos computer club in Hamburg discovered that there were many VAX sites which were not licensed users and had not received the patch. These allegedly included a number of US military sites in West Germany, through which they were able to penetrate various government networks.

Unix is no different; bugs are continually being discovered. For example, '`mkdir`' used to work in two steps: firstly, the storage was allocated, and secondly, ownership was transferred to the user. Since these steps were separate, a user could initiate a '`mkdir`' in background, and if this completed only the first step before being suspended, a second process could be used to replace the newly created with a link to the password file. Then the original process would resume, and change ownership of the password file to the user. By deleting the password for root, the user could now gain superuser status.

Many attacks involve entering inputs that are too long or otherwise outside the range for which the programmer allowed. For example, the '`chfn'` function enables a user to change his name on the system, which entails writing to the password file. However, until about 1988, `chfn` failed to check the length of the string the user entered: if it was too long, the write to the password file failed in such a way that a blank line was inserted — creating a null-named account with no password and root privileges.

Finally, we will mention the famous case R v Gold and Schifreen. The two defendants (and others) dialed into BT's Prestel development system, and found that the welcome screen had a superuser account number and password displayed on it. They tried this logon to the live system, and it worked! They proceeded to hack into the Duke of Edinburgh's electronic mail account, and sent mail 'from' him to people they didn't like announcing the award of honours.

6 Intrusion Detection

Since bad things happen, it is prudent to have some means to detect them and recover. It is also economic; rather than trying to tighten access control mechanisms to the point that they interfere with business it is often realistic to accept that intruders will occasionally get in. Combining access control with a mechanism for intrusion detection or auditing makes even more sense for the many systems whose attacks will likely come from insiders.

Essentially an intrusion detection system is a mechanism for examining system logs and audit trails and drawing attention to suspicious events. The simplest examples tend to be threshold based and alarm when a certain count is exceeded: 'three incorrect password attempts on account X', 'credit card spend exceeds twice running total for last three months', or 'three sex line numbers called this month where none called before'. More sophisticated systems may be model based (where we have a clear idea of what attacks will look like) or anomaly based (where we do not). The latter often AI. Another approach is to use information retrieval techniques to index audit trails.

Logging is not entirely straightforward. Logs may be highly sensitive, as one can search for records of failed logons — if someone tries to log on to a nonexistent account keVin12, then it is a fair guess that this is someone's password and that he got the (userid,password) pair out of phase.

7 Malicious Code

Among the attacks that the access control system designer must cope with are those based on malicious code. A Trojan Horse program may mimic the logon program and capture user passwords; viruses may move from one machine to another and cause various kinds of havoc. Historically, viruses have been more likely to deny service than to breach confidentiality or to initiate fraudulent transactions, but with the rapid growth of the net this may change.

A virus will typically have two components — a replication mechanism and a payload. The commonest way for a virus to replicate is to patch itself into other executable files that it finds in its environment. It will usually have some means of recognising itself so that it does not infect the same file twice. The payload may be a routine that does serious mischief, such as reformatting the hard drive or (worse) slowly corrupting all the data there.

There are various antivirus programs, whose writers engage in an arms race with the virus writers. The most recent development has been that while most viruses up till 1996 were DOS based, the advent of Win95/NT and the net has caused macro viruses (especially viruses written as Word macros) to take pole position. Malicious code can be written in any sufficiently powerful language, including the macro languages of many word processors, spreadsheets, graphics packages and so on. This makes it particularly hard to create a single product. The best strategy in my experience is prevention rather than cure; one should practice 'safe computing' as one would safe sex — not running anything on one's machine unless fairly sure of its origins.

8 Problems

1. What precautions would you take to avoid losing your password to a Trojan Horse program?
2. Discuss how a Trojan Horse might be inserted into a compiler.
3. How would you implement a Clark-Wilson security policy in a Unix application?
4. What measures would you take to prevent an attacker who had obtained root status from deleting all record of his intrusion from logs and audit trails?
5. What problems would you expect to encounter when adapting a credit card intrusion detection system to ATMs? To mobile phone call records?
6. Unlike Unix, VAX/VMS allows you to give a user execute access to a file without getting read access. However, he will still get write access to support dynamic linking. Show how the write access can be abused to read at least parts of an arbitrary file.