Computer Science Tripos part 2 — Security

Lecture 4 – Further Security Topics

Ross J. Anderson

1 Malicious Code

We have already discussed the Internet Worm, and seen how viruses can be used to breach multilevel secure systems. Malicious code does not have to be self-replicating to be dangerous. Bad applets are being seen more and more on the web, while it has been known for students to write computer games with a Trojan horse inside that checks to see if the program is running as root, and if so creates an extra privileged account with a known password (we once even had to discipline a student for re-ROMming the keyboards of some terminals in the user area in order to harvest passwords). For this reason, a prudent system administrator will use a root logon as little as possible.

However, self replicating code is more interesting, and is more likely to deny service than to attempt to breach confidentiality. Service denial was the (unintended) effect of the Internet Worm, because it replicated so profusely. However, most self-replicating malicious code consists of viruses that infect PCs or applications such as Word rather than multiuser systems.

A virus will typically have two components – a replication mechanism and a payload. The commonest way for a virus to replicate is to append itself to an executable file and patch itself in, so that the execution path jumps to the virus code and then back to the original program. It will usually have some means of recognising itself so that it does not infect the same file twice.

Some early antivirus software 'immunised' files, by patching in enough of the virus to fool it into thinking that the file was already infected. However this is not effective against a large virus population, and the next generation were 'scanners' – programs which searched through the early part of each executable file's execution path for a string of bytes known to be from an identified virus.

Virus writers responded in various ways, such as by delaying the virus's entry point in the host file code and thus forcing scanners to check the entire filespace for infection; and most recently with polymorphic viruses. These are mostly encrypted, but have a small header which contains code to decrypt them. The encryption is often not much more than a byte-oriented Caesar or Vigenère, but even so is enough to greatly slow down the scanner, as the virus re-encrypts itself under a different key, and may also insert a few 'NOP's into the decryptor, each time it replicates.

The other main technical approach to virus prevention is the checksummer. This is a piece of software which keeps a list of all the authorised executables on the system, together with checksums of the original versions of these files. However, one leading commercial product merely calculates cyclic redundancy checks using two different polynomials – a technique which could easily be defeated by a smart virus writer.

Viruses can be written not just to attack operating systems such as DOS and Unix, but also applications such as Lotus and emacs which have sufficiently powerful macro languages. One can even write malicious code in postscript and thus infect an organisation's laser printers (perhaps to suppress any hidden copyright messages in print files) although it is hard to see how this code could get back out of the printers and into the network again.

Once one starts looking at a specific platform, such as DOS, there are a range of additional tricks available to the virus writer. For example, if the target system has a file called ACCOUNTS.EXE, one could introduce a file called ACCOUNTS.COM, which DOS will execute first. This is called a companion virus.

DOS viruses may also attack the boot sector, the partition table, and even some specific antivirus programs. They may be stealthy, in that they watch out for operating system calls of the kind used by antivirus programs and hide themselves while a check is being done; and there are even printable viruses – viruses all of whose opcodes are printable ASCII characters, so that they can even propagate on paper. (Printable code has also been used in stack-overflow attacks.)

The second component of a virus is the payload. This will usually be activated by a trigger, such as a date, and may then do one or more of a number of bad things:

- make selective or random changes to the machine's protection state (this is what we worried about with multilevel secure systems)
- make selective or random changes to user data (e.g., trash the disk)
- lock the network (e.g., start replicating at maximum speed)
- steal cpu resources for some nefarious task (e.g. DES keysearch)

By far the most damaging payloads are those that do their damage slowly and imperceptibly, such as by occasionally swapping words in documents or blocks in files. By the time this kind of damage comes to the system administrator's attention, the virus may have managed to corrupt all extant generations of backup.

Various writers have also proposed benevolent payloads, such as to perform software upgrades in a company, to enforce licensing terms, or even to roam the world looking for cheap airline tickets for you (so-called 'intelligent agents'). There may be legal problems though as in many countries (including the UK) it is an offence to make an unauthorised modification to a computer system.

In the practical world, the most important protective measure is managerial discipline. This means controlling all software loaded on the organisation's machines, and providing a central reporting point for all incidents. One also needs the discipline to deal with infected backups, and with staff abusing their privileges, e.g. by taking a corporate PC home supposedly for work but in reality for their children to play computer games. As with the prevention of the organic kind of disease, prevention is preferable to cure; and software hygiene can be well integrated with controls on illegal software copying and unauthorised private use of equipment.

In the academic world, research centers on making smarter antivirus products by exploiting analogies with biology. IBM is automating its techniques for identifying and culturing viruses, with a view to shipping its corporate clients a complete 'path lab'; researchers at the University of New Mexico mimic the immune system by generating a lot of random 'antibodies' and then removing those which try to 'kill' the system's own tissue.

One possible advantage of such techniques is that with decentralised immune responses, not all organisations will have one of the same four or five standard anti-virus defences, as they do at present, and so the organisation will only lose a proportion of its systems when a virus strikes. In general, a diversity of defences is a good thing, so long as the benefits of central incident reporting are not lost.

2 Intrusion Detection

Antivirus mechanisms are a special case of intrusion detection. This is one of the fastest growing fields of security research, with US military funding for research and development having grown from almost nothing to \$500 m in the space of a few years. This growth has been prompted by the realisations that, firstly, attacks happen; secondly, many systems have made essentially no use of log and audit data (in some cases the audit formats were not documented and tools to read them were not available); and thirdly, that there are many ways in which one can detect an attack in progress.

Simultaneously, there have been significant investments by credit card companies and mobile phone operators in systems whose function is to detect fraud in progress by examining transaction patterns, and some stock markets have installed systems to try to detect insider trading. Although these systems are trying to perform very similar tasks, the researchers and developers do not talk to each other and one sees the wheel being re-invented again and again.

The simplest intrusion detection methods involve sounding an alarm when a threshold is passed. Three or more failed logons, a credit card expenditure of more than twice the moving average of the last three months, or a mobile phone call lasting more than six hours, might all flag the account in question for attention. More sophisticated systems generally fall into two categories.

Misuse detection systems operate on the basis of a model of the likely behaviour of an intruder. An example would be a banking system that alarms if a user draws the maximum permitted amount from a cash machine on three successive days. Another would be a Unix intrusion detection system that looked for a user's account being taken over by someone with a much more sophisticated command of the system; thus an account whose user previously only used simple commands such as `cp` and `rm` would alarm if the log showed use of a compiler. An alarm might also be triggered by specific actions such as an attempt to download the password file.

Anomaly detection systems attempt the much harder job of looking for anomalous patterns of behaviour in the absence of a clear model of the attacker's modus operandi. Systems of this type often use AI techniques – neural networks are particularly fashionable.

The dividing line between misuse and anomaly detection is somewhat blurred. Many of the above examples could be represented as either; and in the few cases where a detected anomaly has turned out to highlight a real misuse, it can thereafter be detected more efficiently using conventional rather than AI techniques. An example was a phone company whose system learned to be suspicious of mobiles which roamed away from home without calling home regularly: this was discovered using a neural network but subsequently hard coded in the toll ticket database using SQL.

Another approach is the use of techniques drawn from information retrieval to index the audit trail, so that whenever a new misuse is detected by whatever means, systems can be rapidly and efficiently interrogated to find any instances of it.

The experience to date is that general intrusion detection is a difficult problem, and effective systems use a lot of information about the application area in question. They will also use a variety of techniques to detect different kinds of intrusion.

3 Application Level Controls

Intrusion detection systems will therefore often run at the application rather than operating system level. They will usually not be the only such controls: the mechanisms already present in the operating system (authentication, access control, backup/disaster recovery) often have to be duplicated for granularity or other pragmatic reasons; and there are also a number of features such as delegation and the sanity checking of transactions which are hard to put anywhere else.

A very important application level security feature is ease of safe use. This means that systems should be designed with failsafe defaults and in the full appreciation of the various mistakes which people make.

According to the motor industry, human error can be classified as follows:

Type of Human Behaviour	**Error rate**
Extraordinary errors — hard to conceive how they could occur. Stress free, with powerful cues pointing to success	10^{-5}
Errors in regularly performed simple tasks with minimum stress	10^{-4}
Errors such as pressing the wrong button or reading the wrong display. Complex tasks, little time, some cues needed	10^{-3}
Errors of omission depending on situation and memory. Complex, unfamiliar task with little feedback and some distraction	10^{-2}
Highly complex task, considerable stress, little time available	10^{-1}
Problems involving creative thinking and unfamiliar, complex operations where time is short and stress is high	$1 - 10^{-1}$

The experience of the computer industry is not quite so optimistic. A raw error rate of 1 in 100 to 1 in 200 can be expected for data entry, and it is rare to find even fielded programs with less than 1 bug remaining per 100 lines of code. Some complex operations, such as the design of cryptographic protocols, are got wrong by most people; and management problems can greatly compound the problem.

However, error rates can have a nasty sting in the tail. In banking, for example, the introduction of computers cut error rates from about 1 transaction in 500 to 1 transaction in 10,000. A side effect of this is that procedures for dealing with customer complaints have fallen into disuse; this in turn has led to a number of the problems mentioned in the article 'Why Cryptosystems Fail' (appended to the 1b notes).

4 Denial of Service

When eavesdroppers are a significant part of our threat model – as with military communications – we will aim for confidentiality and may use the Bell-LaPadula model. When fraudsters are the main threat, as in banking, we will make integrity a priority and our model will probably be Clark-Wilson or something similar. But what when our opponent seeks to deny service, and our main security goal is availability?

The classic civilian example here is burglar alarm signaling. If there is a burglar in my vault, then I don't care who tells me (no integrity requirement) or who else gets told (no confidentiality requirement); but I do very much care that an attempt to tell me is not frustrated. A paper on the subject by Roger Needham is appended.

There are many military examples. It is common to try to jam radar, missile guidance signals and fighter control traffic, for example, and here there may be confidentiality or integrity requirements as well. There are also hybrid civilian examples, such as the systems which sell tokens for prepayment metering equipment. When a vending station is given the authority to produce tokens that have value, we must think about what might happen if it is stolen or its records are deliberately destroyed by a dishonest operator.

To return to the case of alarms, attacks can vary from simply cutting the wire to more sophisticated attempts to substitute bogus alarm equipment or a computer that mimics it. Such attacks can be detected if the alarm company sends periodic random challenges to the alarm controller in the vault; these are encrypted and returned. Some care is needed in the design of such challenge-response protocols so that they are not vulnerable to various kinds of message manipulation attacks.

However, even if the cryptography works, things are still not as straightforward as they might seem. The classic attack on alarm signalling involves reducing confidence in its reliability. For example, it is common for criminals to cut the telephone line to a vault, and retire a few hundred yards. They count the number of men who arrive, and the number who depart; if the two numbers are equal, then it is a fair guess that the custodian has said, 'Oh bother, we'll fix it in the morning', or words to that effect, and an entry can now safely be forced.

More sophisticated attacks may involve manipulation of the traffic so that a number of potential targets appear to have unreliable alarm systems. This is helped by the UK chief police officers' policy that after four false alarms, commercial premises are put on 'keyholder response only' for a year. Thus phone company employees are in a position to deny alarm service by creating a suitable level of false alarms, unless the communications can be made anonymous enough to prevent a service denial being selective.

Global service denial is always possible by blowing up the telephone exchange, and this has happened from time to time; for this reason, very large risks need multiple independent communications systems (e.g., leased line plus cellular telephone). However, anonymous communications may provide a more economic solution for the great majority of small to medium sized risks, and this connection between anonymity and availability is a subject of current research. We will discuss anonymity further in the next lecture.

5 Further Reading

Fred Cohen's *'A Short Course on Computer Viruses'* is the standard reference on malicious code, while *'Database Security'* by Castano and others has a section on intrusion detection.

The Motor Industry Software Reliability Association's booklet *'Development*

Guidelines for Vehicle Based Software' is a good introduction to the software and human factors engineering problems in safety critical systems.

6 Problems

1. Give concrete examples of how a virus might be propagated in word processing documents, spreadsheets, postscript files, web pages and other objects that are interpreted rather than executed.
2. Could you write a companion virus to infect a Unix system?
3. Show that the problem of detecting an arbitrary virus is undecidable.
4. Show that for every program, there is a virus that will infect it.
5. What problems would you anticipate in adapting a credit card intrusion detection system to ATMs? How would you deal with them?
6. How would you prevent viruses being used for a selective denial of service attack?