Computer Science Tripos part 2 – Security

Lecture 3 – Unix and Internet Security

Ross J. Anderson

1 Introduction

We now move on from the question of what we are trying to achieve with our security systems to the engineering aspects of how we are going to implement the protection.

Recall that in the 1b introductory course we discussed:

- the problems of choosing good passwords
- Trojan horse attacks on password entry
- that once we have authenticated our user, we can map users to file access permissions using an *access control matrix*; if we store this matrix by rows they are called *access control lists* (ACLs) while if we store it by columns they are called *capabilities*
- that in real systems, these ACLs or capabilities will usually refer to roles or groups rather than individual users
- the fact that access control could be implemented using a single bit to signify 'authorised state' (which in the IBM mainframe implementation meant the ability to change memory segment registers)
- that most security failures in operating systems have historically been caused by bugs

In this lecture, I will talk some more about the particular problems of protecting networked Unix systems.

2 Unix

Unix and internet security is a fast moving field; the attacks that are catching the headlines tend to change significantly from one year to the next. It is also an increasingly specialised field. To keep up, you should not just read books like Cheswick and Bellovin's *Firewalls and Internet Security* and Garfinkel and Spafford's *Practical Unix and Internet Security*; you should also follow the latest advisories from CERT and elsewhere.

However, although one of the more volatile areas of computer security technology, it is also one of the most newsworthy (one might even say 'fashionable'). Regardless of whether you work in a field that is explicitly security related or not, internet security issues are likely to have an impact on your professional work in the next few years. A familiarity with the basic issues is important for the working systems engineer.

2.1 Background

Understanding Unix security depends on understanding the models of computation in its authors' minds at the time when various versions of it were produced.

It was originally designed as an operating system to be used by a number of people in a department who were sharing a single machine. In this environment the function of the security mechanisms is more than anything else to contain mistakes; to prevent one user's typing errors or program crashes from deleting or overwriting another user's files. The original security mechanisms, as described in the paper by Grampp and Morris (appended), were quite adequate for this purpose.

However, Unix security has suffered the problems of the operating system's success; its functionality was repeatedly extended without proper consideration always being given to how the protection mechanisms also needed to be extended. The Berkeley extensions (`rlogin, rsh, rcp, rhosts, lpd` etc) were based on an extension from a single machine to a network of machines that were all on one LAN and all under one management. However mechanisms such as `rhosts` were based on a tuple *(username.hostname)* rather than just a username, and saw the beginning of the transfer of trust.

The Internet RFCs (telnet, ftp, DNS, SMTP), which grew out of Arpanet in the 1970's, were written for mainframes on what was originally a secure WAN. Mainframes were autonomous; the network was outside the security protocols; and there was no transfer of authorisation. Thus remote authentication, which the Berkeley model was starting to make prudent, was simply not supported.

The Sun contributions (NFS, NIS, RPC etc) were based on a workstation model of the universe, with a multiple LAN environment with distributed management (but still usually in a single organisation).

Of course, mixing all these different models of computation together has resulted in chaos. Some of their initial assumptions still apply partially, but none of them apply globally any more. The Internet of nowadays has millions of workstations, hundreds of thousands of LANs, many interconnected WANs, and many independent managements. Some workstations of course have no management at all.

The moral is of course that designing and analysing systems in isolation is a bad idea. But what can we actually do to deal with the situation in which we now find ourselves?

Things may improve in the medium-to-long term as a result of the introduction of IPv6 with its inbuilt authentication mechanisms. However the design of this isn't complete (of which more later) and even when it is it will probably take a decade at least to roll out. In the meantime we may expect a steady stream of Unix security problems. These often have a complex origin – partly from the continually changing environment, and partly from blunders in implementation. Nor is it reasonable to expect other operating systems, such as Windows, to be any different. The short-to-medium term fixes for these problems may involve various firewall techniques. We will deal with these topics in turn.

2.2 Basic mechanisms

As you are probably aware, Unix programs can run as kernel (system) or as user (application). In kernel mode, a program has complete access to the entire machine; applications (programs) have their accesses to disks, files, etc mediated by the kernel. There is a similar dichotomy of users, between root (superuser) and user (mortal). The former, with identifier zero, is allowed to do whatever she likes – access any file, become any user, etc.

As we mentioned in 1b, the standard file security mechanism is the access control list (ACL). This consists of only a few bits of state stored for each file. Although primitive, this can often be used to achieve the desired effect, sometimes using subdirectories and groups. Note though that whereas on older unices a process only holds one group id at a time and must use a privileged program to access other groups, newer unices put a process simultaneously in all groups that the user is in.

A feature of Unix that is often used carelessly is set-user-id (setuid, suid) and its group counterpart (setgid). A user (or root) can give a program privilege so that whenever it runs it has file access rights of its owner instead of the user running it. The program can then usually choose to switch between accessing as owner and as user. The unwise use of setuid has led to a number of well known security holes.

2.3 Common implementation problems

One of the more serious is kernel bloat. Under Unix, all device drivers, filesystems etc. must be in the kernel. This means that large quantities of code are put inside the ultimately trusted security perimeter. Other systems, such as CAP, allowed proper compartmentation – most of the operating system ran in user mode; and yet others such as MVS are starting to introduce mechanisms that decrease the level of trust needed by many utilities. However the means to do this in Unix are few and relatively nonstandard.

Even more seriously, application developers often make their programs run as root. This may be easier, as it avoids permission problems; it also often introduces horrible vulnerabilities which are quite unnecessary, as more limited privilege could have been used with only a modicum of thought and a minor redesign.

Some applications do need root privilege. For example, MTAs (mail transfer agents) must be able to deal with user mailboxes. However, while a prudent designer would restrict this privilege to a small part of the application, most MTAs are written so that the whole program needs to run as root. The classic example is sendmail, which has a long history of serious security holes; but many other MTAs also have problems. The general effect is that a bug which ought to compromise only one person's mail may end up giving root privilege to an outside attacker.

More examples come from network applications. To bind to a low-numbered (<1024) TCP port you have to be root, and this has led many people to write network applications with too much privilege. Instead of writing programs that have to be started as root, it is preferable to have a small 'startup' program.

More seriously, there are systems such as `lpr/lpd` – the lineprinter subsystem – which does not need to run as root but does anyway on most systems. This has also been a source of security failures in the past (e.g., getting lpd to spool to `/etc/passwd`).

Sometimes the cure is almost as bad as the disease. Some programmers avoid 'root bloat' and the difficulty of getting non-root software installed and working securely by simply leaving important shared data structures and resources accessible to all users. Examples include that to allow 'write' and similar programs to work, most systems allow any user to send characters to another user's terminals. However, many terminals allow special escape codes to cause data (e.g. a screendump) to be sent back!

Another common example is the mail spool. Many systems (including CUS) store mail in a file per user in a world-writeable (sticky bit) directory. Mail forgery becomes fairly easy. There are also problems with symbolic links. Fixing these once the initial design decisions have been made can be troublesome.

A final example is `utmp` (the list of users logged in). This is frequently used for security checking of various kinds, but is also frequently world-writeable !

In general, much better design and coding is required. Rather than happily using setuid and root, designers should create groups with limited powers, and be clear about what the compromise of that group should mean for the rest of the system. Programs which require privilege should only be run as privileged when necessary. Default configurations should be safe.

2.4 Memory Overwriting Attacks

Programmers are often careless about checking the size of arguments. A classic example is a vulnerability in the `finger` daemon that was exploited by the Internet Worm (we will describe the Worm below). A widespread implementation of this command would accept an argument of any length, although only 256 bytes had been allocated for this argument by the program. The result was that when an attacker used the command with a longer argument, the trailing bytes of the argument ended up being executed by the system.

Many of the vulnerabilities reported routinely by CERT and other advisories relate to memory overwriting attacks. There is really no excuse for this problem to continue, but it does.

2.5 SUID

Suid programs are very difficult to do right. They run with all the privileges of the creating user, not just the ability to do certain limited operators. Suidness is inherited by subprocesses. So if an attacker can subvert a program a little bit she gets complete access as if she were the creating user. If the program is suid root the system is compromised completely.

Suid programs inherit the environment from their parent process, which is controlled by the user. The PATH variable, file descriptors, terminals and all the arguments are all completely untrusted. This makes almost any operation risky – especially given the Unix philosophy of reusing programs where possible, and the fact that library routines are often loaded from directories specified by environment variables, and often call subprograms without the programmer's knowledge.

More complex problems include race conditions with suid shell scripts and interactions; for example, many programs do complex transformations on their arguments, and if the caller of such a program is suid then simple mistakes can have serious consequences. Some kernel bugs can allow tracing and/or other debugging of setuid programs, so that the attacker can subvert the program using a debugging tool. Finally, signals can be sent by the caller of a suid program, e.g. to corrupt files and can prevent logging of attempted attacks.

Yet sometimes suid programs simply must be used, for example to connect to an otherwise-inaccessible socket so that the peer process can identify the user. In short, the situation is quite unsatisfactory

3 Network security aspects

Unix traditionally comes with a very large range of network services, many of which are enabled by default, and/or shipped with configurations that make "plug and play" working easy – for the attacker as well as the legitimate user. We will look at local area network aspects in this section and Internet issues in the next. A common theme is that mapping methods (between addresses, filenames, etc) provide many of the weak points.

Of course, anyone with physical access to a LAN can install a packet sniffer device that will harvest passwords. However there are a number of more subtle attacks, which are worth knowing about as they may be executed on standard hardware by malicious software that has, for example, been uncautiously downloaded and run.

3.1 ARP

Ethernet has a separate address space from IP, and the two are mapped via ARP. Thus a machine running malicious code can masquerade as another machine by giving wrong answers to ARP messages. The victim machine might notice if alert, but the attacker can always wait until it is down – or take it down by using another attack.

Note that it is quite common for actual attacks to use a combination of basic techniques.

3.2 Subnet masks

Originally, IP numbers were 32 bit values with the first 3 bits specifying the split between the network address and the host address. Now they are interpreted as addressing network, subnetwork and host, with a variable 'network mask'. Diskless workstations, when booting, broadcast a request for a subnet mask; many of them will apply any subnet mask they receive at any time.

Thus ICMP messages can be used to make networks unusable; this may be a component in a larger attack. They can also be used to make a host broadcast an ARP for a nonexistent machine, to which the attacker replies; or to make machines vanish, in which case the attacker can masquerade as a helpful router. We will talk more about IP routing and ICMP redirection attacks below.

3.3 NFS

NFS – Sun's Network File System – is the de-facto standard for file sharing between Unix systems, and provides a number of well-known points of attack. When a volume is first mounted, the client requests from a special daemon on the server a 'root filehandle'. This is a 'filehandle' which refers to the root directory of the mounted filesystem. Filehandles referring to other directories are obtained by making requests to the NFS server. The value of the root filehandle doesn't depend on time, server generation number, or any such. It can't be revoked. The same is true for other filehandles.

There is no mechanism for per-user authentication: the server must trust a client completely or not at all. Also, NFS servers often reply to requests from a different network interface to the one on which the request arrived. Most clients therefore don't check the source addresses on replies to their NFS requests.

A number of attacks have therefore involve manipulating filehandles in various ways.

4 The Internet

The insecurity of the Internet has filled many column inches of newspapers in the last few years. While much of the coverage is generated by the publicity agents of companies that sell firewalls, there are still some underlying problems of which you should be aware.

The fundamental problem is the lack of integrity or confidentiality checking. This is particularly manifest at the lower level TCP/IP protocols, though often there is no checking at higher levels either. For example, the Berkeley protocols assume that the lower levels are secure; checking might typically be a verification of the hostname against an IP address using DNS. So someone who can forge IP addresses, or break DNS (which can be done using TCP sequence number guessing) can abuse facilities like rlogin and rhosts.

The second problem is that access control usually depends on passwords. These are transmitted in the clear (e.g. telnet, ftp) so snooping is very easy; backbone snooping attacks have been done for years by intelligence agencies, and are starting to be done by others as well. As NSA person remarked that the main culture change brought by the net is that they had assumed that sophisticated attacks were essentially the preserve of national governments; yet nowadays one finds not just password snooping attacks but also more subtle routing attacks being done by kids for fun. The critical change here is that people write the necessary exploit software and then post it on sites like `www.rootshell.com` from which 'script kiddies' can download it and use it. In effect, hacking has been de-skilled.

4.1 Routing attacks

Routing attacks come in a variety of flavours. The basic attack involves Charlie telling Alice and Bob that a convenient route between their sites passes through his. Source level routing was originally fielded to help get around bad routers. The underlying assumption – that "hosts are honest" – no longer holds, and the only short term solution is to block source routing. However it continues to be used for network diagnosis.

Another approach involves ICMP redirect messages, which are based on the same false assumption. These effectively say "you should have sent this message to the other gateway instead', and are generally applied without checking. They can be used to do the same subversion as source-level routing.

As we have seen, mail forgery is often trivial; rerouting is harder since mail routing is based on DNS, but it is getting easier as the number of service providers goes up and their competence goes down.

4.2 Masquerade

The common masquerade attacks involve password guessing, whether using a small number of highly likely passwords online or by getting the `\etc\passwd` file and trying a much larger number of passwords offline. However, there are more subtle attacks.

Consider for example the 3-way handshake used by TCP to initiate a connection.

A $\rightarrow$ B:	SYN; my number is X
B $\rightarrow$ A:	ACK; you're X
	SYN; my number is Y
A $\rightarrow$ B:	ACK; you're Y
	(start talking)

If the attacker, Charlie, can find out Y somehow then he can masquerade as A. Sequence numbers are often easy to guess – they are meant to change rapidly but many implementations change them, for example, once a second.

4.3 Service denial attacks

The Internet's predecessor, the Arpanet, was designed to provide a very high degree of resilience against service denial attacks. In fact, the goal was that it should survive a strategic nuclear strike. So we may be particularly interested in attacks that result in denial of service.

We have already seen how this can be done using ICMP, but the most famous case of a service denial attack was the Internet worm of November 1988. This was a program written by Robert Morris jr (son of the Unix security architect) which exploited a number of vulnerabilities to spread from one machine to another. Some of these were general (e.g. 432 common passwords were used in a guessing attack, and opportunistic use was made of `.rhosts` files), and others were system specific (problems with sendmail, and the `fingerd` bug mentioned above). This code was so successful at replicating that it completely clogged up the net.

The most recent service denial attack to have been widely fielded involves SYN flooding. This brought down Panix, the top New York ISP, for several days in 1996. The attack is quite simply to send a large number of SYN packets and never acknowledge any of the replies. This leads to buffer overflow, or at least swamps other attempts to communicate.

4.4 Firewalls

The most widely sold solution to the problems of Internet security is the firewall – a machine which stands between a local network and the Internet and filters out traffic that might be harmful. These come in a number of flavours, with the simplest simply filtering packet addresses. This can ensure, for example, that no packet that appears to come from a host on the local network is allowed to come from outside, and that only standard TCP/UDP port numbers are used.

Simple packet filtering is already available in some unices but can be defeated by a number of tricks. For example, a packet can be fragmented in such a way that the initial fragment (which passes the firewall's inspection) is overwritten by a subsequent fragment, thereby replacing the source address with one that violates the firewall's security policy.

More complex firewalls, called circuit gateways, reassemble and examine all the packets. This still does not prevent attacks at the application level, such as malicious code, and so there are still more complex products that provide application level protection; these act as proxies for mail, ftp, http, and so on. These may still be simpler to install and maintain than fixing the fundamental problems would be. They can also provide added functionality, such as providing a virtual private intranet over the Internet by doing encryption from firewall to firewall, and screening out black-listed web sites or newsgroups (there have been reports of Asian governments building national firewalls for this purpose).

The downside is that application relays can turn out to be serious bottlenecks; they can also get in the way of users who wish to run the latest applications. An example is the NHS wide network; this was conceived as a private intranet for NHS users with a single firewall to the outside world. What the designers didn't anticipate was that fully 40% of traffic was international from the start, and that everyone would want to use SMTP rather than the government approved X.400. (In fact a number of applications and utilities couldn't be made to run over X.400.) There was also the minor philosophical question of how one could consider a network with a million users to be secure, especially given that most attacks on most systems are internal.

Back in 1994–5 there was much debate in the Internet security community about whether firewalls were a good thing. Surely it would be better to run a well configured site, perhaps with a simple filtering router at most, and tackle problems such as password snooping using one-time password generators. However, such an approach requires skill and care; the alternative of a 'solution in a box' has great appeal for nervous organisations that are new to the Internet, and even experienced organisations may find that it is just too expensive to fix all the security holes that were tolerable on a local network but would not be with an Internet connection. There is also the problem that often an organisation's most critical applications run on the least secure machines, as administrators have not dared to apply operating system upgrades and patches for fear of losing service.

Whatever the merits of the arguments, by now the firewall industry's momentum appears to be unstoppable. Of course, there are firewalls and firewalls. Many of the products crowding the market are not much good, as one would expect of an industry most of whose members entered the market since 1994. The experience of military laboratories is that capable attacks are only stopped by multilevel secure machines that not only implement the Bell-LaPadula security policy but have been subjected to severe testing by independent evaluators. Even when they are available to commercial users, such systems may be passed over on grounds of cost.

4.5 Emerging problems and strategic solutions

It is a general principle in computer security that there are no 'silver bullets'. Any given set of measures may be adequate for a time, but will eventually be overtaken by technology. The passwords and access control lists used in the earliest unices are a good example of this.

The limits of the firewall approach may be illustrated by the recent problems with the security of Java. Here the problem is that users demand to use an unsafe mechanism; Java programs can break security in a number of ways. The problem is not really one of communications security and cannot be effectively stopped by a firewall (unless it simply banned the import of any web pages incorporating Java code – but that would not be acceptable to users). The problem is more akin to that posed by viruses in Word documents. There, anti-virus software writers have started to suggest that the countermeasures should be incorporated in Word itself, as there are limits to what can be done by a third party supplier without access to proprietary file format and other information. Even when we move from a proprietary product such as Word to a partially open one, such as ActiveX, it may still be preferable to build a reasonable level of protection into the product itself. We will discuss viruses further in a later lecture.

In the long term, the real solution may be the authentication mechanisms proposed for IPng (IPv6), and those parts of them that can be retrofitted to IPv4 (specifically IPSEC). However there are serious political arguments about the extent to which governments should permit the use of cryptographic technologies that could seriously affect their signals intelligence capabilities. We will discuss this topic later.

Meanwhile, net users who wish to communicate securely have really no choice but to supply the security at higher layers, such as by using PGP for secure email and SSH for connecting to remote hosts. However it is worth always bearing in mind that even if PGP's cryptographic security is indeed 'pretty good', it will be pretty little use if it is run on a Unix system on which a passphrase sniffer has been installed by an attacker using some combination of the attack techniques described here.

5 Further Reading

The recommended books for Unix security are *'Firewalls and Internet Security — Repelling the Wily Hacker'* by Cheswick and Bellovin, and *'Practical Unix* *and Internet Security'* by Simson Garfinkel and Eugene Spafford. As mentioned above, the advisories from CERT and elsewhere are also worth following if you wish to remain abreast of events.

Acknowledgement: These notes are based on Unix security lecture notes by Ian Jackson and a talk on the subject by James Davenport.

6 Problems

1. What Unix command will list all the world writeable files in your directories?
2. What modifications would you make to Unix to ensure that the sysadmin cannot bypass or modify an audit trail?
3. How would you go about writing a mail transfer agent that adheres to the principle of least privilege?
4. Discuss in detail how you might go about hijacking a telnet session. How could you prevent such attacks?
5. Discuss in detail how you might go about hijacking an X session. How could such an attack be stopped?
6. Discuss in detail how you might go about attacking DNS. What defences are possible?
7. How would you prevent SYN flooding attacks?