Chapter 9

System (in)security

*One of my sons was taught stranger-danger at his school. We were asked to quiz him afterwards, so we asked him if he should accept a lift in a car with a stranger. He immediately replied "**No way!**". We then asked: "**What if he offered you sweets?**", but he still replied "**No way!**". Finally we asked: "**Why not?**", to which he replied "**Because you might not get any!**"*

9.1 Ethical concerns

A mature ethical sense normally develops as you age. We recognize that young people are not capable of fully comprehending the world around them. Lawrence Kohlberg, a Harvard psychologist, formalizes moral development into various stages:

Stage 1: *Obedience and punishment* - deference to superior power or prestige.

Stage 2: *Naively egoistic* - a *right* action satisfying the self's needs and occasionally others.

Stage 3: *Good-boy/good-girl* - an orientation to approval, to pleasing and helping others, with conformity to stereotypical images of majority or natural role behavior.

Stage 4: *Authority and social-order-maintaining* - an orientation to "doing duty" and to showing respect for authority and maintaining the given social order for its own sake.

Stage 5: *Contractual/legalistic* - defined in terms of laws or institutionalized rules.

Stage 6: *Individual principles of conscience* - an orientation not only toward existing social rules, but also toward the conscience as a directing agent, mutual trust and respect, and principles of moral choice involving logical universalities and consistency. If one acts otherwise, self-condemnation and guilt result.

It is my expectation, and requirement, that you are able to maturely evaluate rights and wrongs. Why? Because in these sections of the course, I will be outlining systems which demonstrate poor cryptographic techniques, and as a result, can be defeated.

A more cynical view might be that *I am teaching hacking*[1]. This view is certainly not my intent, and I only discuss *hacks/cracks* within a framework of *how you can fix it*.

9.1.1 Why study this?

Many common views related to insecurity are promoted by the popular press, driven by commercial interests: "*Use 128-bit encryption - guaranteed secure*", "*Use NT - secure your network*", and so on. These sort of slogans give a false sense of security, and are commonly incorrect.

An uninformed belief in the safety of computer systems leads to insecure systems, and is reason enough to study this area. However, if you wish more justifications:

- An awareness of the nature and limits of cryptography is essential to computer people in business. Computer systems are becoming more critical, and directly relate to the core operations of many businesses[2].
- Network administrators commonly need to attempt to hack their own systems to assure themselves of the security of those systems.
- A forgotten encryption password may need to be recovered.

9.1.2 Ethics and computing

In general computer based systems do not introduce any new ethical dilemmas. In most cases it is relatively easy to draw a parallel with existing non-computer systems. Here are some sample areas:

Software duplication: It is very easy to duplicate software at no cost. However, doing so can only be viewed as *theft*.

Using information: It is often easy to recover information from computer systems - for example a programmer may become aware of her employer's proprietary algorithms and then make use of this knowledge to make money. This is known as insider trading and is considered a crime.

[1] The perjorative term *hacking* has a proud history - it originally meant *"a codesmith"*, but has been perverted to mean *"someone who breaks into computer systems"*. I prefer to use the term *cracker*, rather than *hacker*.

[2] It is interesting to note that of those businesses that were unable to restore their computer systems after the San Francisco earthquake, 50% failed within the next year.

E-mail abuse: E-mail is no different from any other communication, and most countries already have laws that inhibit reading, tampering, changing or intercepting mail. Abuse over email is no different from any other form of (non-contact) abuse.

Network administrator's dilemma: Network administrators often come to learn things about their 'clients' that they did not intend. However, without asking the client, they should not make use of that information. The documents that most computer users sign when they are given access to a computer system do not over-ride their legal rights. This leads to the network administrator's dilemma: How to control bad-guys without trampling over their rights.

Perhaps the only significant difference is that the computer crimes are so easy.

9.1.3 Professional codes of ethics

Most professional bodies[3] have formal written codes of ethics, along with committees to deal with abuses of the ethical standards set. The computer industry has yet to develop a single standard code of conduct, and if computer crime continues to rise, codes may be imposed on it.

The Australian Computer Society proposes a code of ethics which include the following sections:

1. I will serve the interests of my clients and employers, my employees and students, and the community generally, as matters of no less priority than the interests of myself or my colleagues.
   (a) I will endeavour to preserve continuity of computing services and information flow in my care.
   (b) I will endeavour to preserve the integrity and security of others' information.
   (c) I will respect the proprietary nature of others' information.
   (d) I will advise my client or employer of any potential conflicts of interest between my assignment and legal or other accepted community requirements.
   (e) I will advise my clients and employers as soon as possible of any conflicts of interest or conscientious objections which face me in connection with my work.
2. I will work competently and diligently for my clients and employers .
   (a) I will endeavour to provide products and services which match the operational and financial needs of my clients and employers.
   (b) I will give value for money in the services and products I supply.
   (c) I will make myself aware of relevant standards, and act accordingly.

[3]For example: Medical boards.

   (d) I will respect and protect my clients' and employers' proprietary interests.
   (e) I will accept responsibility for my work.
   (f) I will advise my clients and employers when I believe a proposed project is not in their best interests.
   (g) I will go beyond my brief, if necessary, in order to act professionally.
3. I will be honest in my representations of skills, knowledge, services and products.
   (a) I will not knowingly mislead a client or potential client as to the suitability of a product or service.
   (b) I will not misrepresent my skills or knowledge.
   (c) I will give opinions which are as far as possible unbiased and objective.
   (d) I will give realistic estimates for projects under my control.
   (e) I will not give professional opinions which I know are based on limited knowledge or experience.
   (f) I will give credit for work done by others where credit is due.
4. I will strive to enhance the quality of life of those affected by my work.
   (a) I will protect and promote the health and safety of those affected by my work.
   (b) I will consider and respect people's privacy which might be affected by my work.
   (c) I will respect my employees and refrain from treating them unfairly.
   (d) I will endeavour to understand. and give due regard to, the perceptions of those affected by my work, whether or not I agree with those perceptions.
   (e) I will attempt to increase the feelings of personal satisfactions, competence, and control of those affected by my work.
   (f) I will not require, or attempt to influence, any person to take any action which would involve a breach of this Code.
5. I will enhance my own professional development, and that of my colleagues, employees and students.
6. I will enhance the integrity of the Computing Profession and the respect of its members for each other.

Within a general framework of ethical and moral responsibility, codes such as this one can help clarify *grey* areas of concern.

9.2 Insecurity - threats and protection

The dangers of the use of insecure systems cannot be underestimated. Supposedly secure systems at the CIA, the Pentagon and the DOD have all been hacked. For example:

- Pentagon machines were repeatedly corrupted by unknown intruders during the Gulf war. The intruders appeared to be doing it as part of a contest.
- German hackers demonstrated on TV a method of transferring money into their own accounts using ActiveX controls downloaded to an unsuspecting person's machine.
- Estimates of computer theft in the US range from 1 to 30 $billion/year - most of which goes unreported.

There have been various attempts to provide a taxonomy of insecurity, but each new attack seems to add new levels to the structure. We start of course with the obvious:

- physical insecurity, and
- password insecurity

Some of the security of modern systems is provided through cryptographic techniques (particularly password storage), and this course concentrates on *these* insecurities.

9.2.1 Non-cryptographic cracking

General *hacking/cracking* is not limited to cryptographic methods, and may often be done much more quickly. For the sake of completeness, here are some of the general strategies employed in hack attacks - many can be used either by internal attackers or by remote (external) attackers.

Misconfiguration: If excessive permission exist on certain directories and files, these can lead to gaining higher levels of access. For example, on a UNIX system, if /dev/kmem is writable it is possible to rewrite your UID to match root's.

Poor SUID: Sometimes there are scripts (shell or Perl) that perform certain tasks and run as root. If the scripts are writable by you, you can edit it and run it.

Buffer overflow: Buffer overflows are typically used to spawn root shells from a process running as root. A buffer overflow could occur when a program has a buffer for user-defined data and the user-defined data's length is not checked before the program acts upon it.

Race conditions: A race condition is when a program creates a short opportunity for attack by opening a small window of vulnerability. For example, a program that alters a sensitive file might use a temporary backup copy of the file during its alteration. If the permissions on that temporary file allow it to be edited, it might be possible to alter it before the program finishes its editing process.

Poor temporary_files: Many programs create temporary files while they run. If a program runs as root and is not careful about where it puts its temporary files and what permissions these files have, it might be possible to use links to create root-owned files.

Attacks using these methods can be launched locally on the target machine, or often remotely, by exploiting *services* with loopholes. In this context, a *service* may be a web server, a file server, an ftp server, or even a security password server.

9.2.2 Protection

Can you protect yourself against attacks? - Yes - but only up to a point. You can reduce your vulnerability by continual re-examination of your computer systems. The following points are often made:

- **Hack/crack yourself:** A common activity of network administrators is to attempt to hack.crack their own systems, and to encourage friendly colleagues to do the same.
- **Be vigilant:** There are new exploits discovered every day, and you can keep relatively up-to-date by subscribing to BugTraq mailing lists.
- **Reduce reliance:** Don't rely totally on the security of the machines.
- **Use more secure systems:** If you are concerned about security, use more secure systems. Enforce encrypted communications, inhibit plaintext passwords and so on.
- **Update systems:** More recent revisions of the software normally have better security features.

Finally: "*Its not the end of the world!*" If your system is damaged, its not the end of the world. Fix the flaw, fix the damage and get back to work.

9.3 CERT - Computer Emergency Response Team

CERT describes itself in the following way:

> *The CERT Coordination Center is the organization that grew from the computer emergency response team formed by the Defense Advanced Research Projects Agency (DARPA) in November 1988 in response to the needs identified during the Internet worm incident. The CERT charter is to work with the Internet community to facilitate its response to computer security events involving Internet hosts, to take proactive steps to raise the community's awareness of computer security issues, and to conduct research targeted at improving the security of existing systems.*
>
> *The CERT/CC offers 24-hour technical assistance for responding to computer security incidents, product vulnerability assistance, technical documents, and courses. In addition, the team maintains a mailing list for CERT advisories, and provides a web site (www.cert.org) and an anonymous FTP server, (ftp.cert.org) where security-related documents, CERT advisories, and tools are available.*
>
> *The CERT Coordination Center is part of the Networked System Survivability (NSS) program at the Software Engineering Institute (SEI), a federally funded research and development center (FFRDC) at Carnegie Mellon University (CMU).*

If you are ever involved in a computer security incident it is useful to get in touch with CERT. They provide incident reports and advisories, and can liaise with other system administration people if the attack on your system comes from outside your organization.

9.3.1 CERT Incident Note IN-99-04

Here is an excerpt from an incident report:

Similar Attacks Using Various RPC Services

Thursday, July 22, 1999

Overview

We have recently received an increasing number of reports that intruders are using similar methods to compromise systems. We have seen intruders exploit three different RPC service vulnerabilities; however, similar artifacts have been found on compromised systems.

Vulnerabilities we have seen exploited as a part of these attacks include:

- CA-99-08 - Buffer Overflow Vulnerability in rpc.cmsd
  http://www.cert.org/advisories/CA-99-08-cmsd.html
- CA-99-05 - Vulnerability in statd exposes vulnerability in automountd
  http://www.cert.org/advisories/CA-99-05-statd-automountd.html
- CA-98.11 - Vulnerability in ToolTalk RPC Service
  http://www.cert.org/advisories/CA-98.11.tooltalk.html

Description

Recent reports involving these vulnerabilities have involved very similar intruder activity. The level of activity and the scope of the incidents suggests that intruders are using scripts to automate attacks. These attacks appear to attempt multiple exploitations but produce similar results. We have received reports of the following types of activity associated with these attacks:

- Core files for rpc.ttdbserverd located in the root "/" directory, left by an exploitation attempt against rpc.ttdbserverd
- Files named callog.* located in the cmsd spool directory, left by an exploitation attempt against rpc.cmsd
- Exploitations that execute similar commands to create a privileged back door into a compromised host. Typically, a second instance of the inetd daemon is started using an intruder-supplied configuration file. The configuration file commonly contains an entry that provides the intruder a privileged back door into the compromised host. The most common example we have seen looks like this:

  /bin/sh -c echo 'ingreslock stream tcp wait root /bin/sh -i' > > /tmp/bob;/usr/sbin/inetd -s /tmp/bob

  If successfully installed and executed, this back door may be used by an intruder to gain privileged (e.g., root) access to a compromised host by connecting to the port associated with the ingreslock service, which is typically TCP port 1524. The file names and service names are arbitrary; they may be changed to create an inetd configuration file in a different location or a back door on a different port.
- In many cases, scripts have been used to automate intruder exploitation of back doors installed on compromised hosts. This method has been used to install and execute various intruder tools and tool archives, initiate attacks on other hosts, and collect output from intruder tools such as packet sniffers.
  One common set of intruder tools we have seen is included in an archive file called neet.tar, which includes several intruder tools:
  - A packet sniffer named update or update.hme that produces an output file named output or output.hme
  - A back door program named doc that is installed as a replacement to /usr/sbin/inetd. The back door is activated when a connection is received from a particular source port and a special string is provided. We have seen the source port of 53982 commonly used.
  - A replacement ps program to hide intruder processes. We have seen a configuration file installed at /tmp/ps_data on compromised hosts.
- Another common set of intruder tools we have seen is included in an archive file called leaf.tar, which includes serveral intruder tools:
  - A replacement in.fingerd program with a back door for intruder access to the compromised host
  - eggdrop, an IRC tool commonly installed on compromised hosts by intruders. In this activity, we've seen the binary installed as /usr/sbin/nfds
  - Various files and scripts associated with eggdrop, many of which are installed in the directory /usr/lib/rel.so.1
  - A replacement root crontab entry used to start eggdrop

It is possible that other tools and tool archives could be involved in similar activity.

In some cases, we have seen intruder scripts remove or destroy system binaries and configuration files.

9.4 NSA - National Security Agency

NSA describes itself in the following way:

> *The National Security Agency is the USA's cryptologic organization. It coordinates, directs, and performs highly specialized activities to protect U.S. information systems and produce foreign intelligence information. A high technology organization, NSA is on the frontiers of communications and data processing. It is also one of the most important centers of foreign language analysis and research within the Government.*
>
> *Signals Intelligence (SIGINT) is a unique discipline with a long and storied past. SIGINT's modern era dates to World War II, when the U.S. broke the Japanese military code and learned of plans to invade Midway Island. This intelligence allowed the U.S. to defeat Japan's superior fleet. The use of SIGINT is believed to have directly contributed to shortening the war by at least one year. Today, SIGINT continues to play an important role in maintaining the superpower status of the United States.*
>
> *NSA employs the country's premier codemakers and codebreakers. It is said to be the largest employer of mathematicians in the United States and perhaps the world. Its mathematicians contribute directly to the two missions of the Agency: designing cipher systems that will protect the integrity of U.S. information systems and searching for weaknesses in adversaries' systems and codes.*

In 1943, SIGINT, a forerunner of the National Security Agency, began a very secret program, codenamed VENONA. The object of the VENONA program was to examine encrypted Soviet diplomatic communications. In October 1943, weaknesses were discovered in the cryptographic system of the Soviet trade traffic.

During 1944, the skills of other expert cryptanalysts were brought to bear on the message traffic to see if any of the encryption systems of the messages could be broken. One of these cryptanalysts made observations which led to a fundamental break into the cipher system used by the KGB. The messages were double-encrypted and were extremely difficult to crack. It took almost two more years before parts of any of these KGB messages could be read or even be recognized as KGB rather than standard diplomatic communications.

Almost all of the KGB messages between Moscow and New York, and Moscow and Washington in 1944 and 1945 that could be broken at all were broken between 1947 and 1952.

NSA continue this sort of work actively, but more recent work is classified.

9.5 C2 security

The NSA created various criteria for evaluating the security behaviour of machines. These criteria were published in a series of documents with brightly coloured covers, and hence became known as the *Rainbow* series.

The document DOD 5200.28-STD[4] - "Department of Defense Trusted Computer System Evaluation Criteria", has been developed to serve a number of purposes:

- To provide a standard to manufacturers as to what security features to build into their new and planned, commercial products in order to provide widely available systems that satisfy trust requirements (with particular emphasis on preventing the disclosure of data) for sensitive applications.
- To provide DoD components with a metric with which to evaluate the degree of trust that can be placed in computer systems for the secure processing of classified and other sensitive information.
- To provide a basis for specifying security requirements in acquisition specifications.

The term ***C2*** comes from these documents, and describes a set of desireable security features related to controlled access, that were considered by the US Department of Defense when the documents were developed. Many of the elements of a C2-secure, system are just those functions that should ***not*** be enabled on a system, and so making a system C2-secure includes turning off some features of a system.

For example, C2 requires that:

> *The TCB*[5] *shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.*

And so on...

Windows NT Workstation vs 3.5 with U.S. Service Pack 3 was the first Microsoft product that has completed C2 testing, and is only certified if using the same hardware, and installed software, and does not include any network connection. The NT utility ***c2config.exe*** sets up an NT system to pass the C2 tests. Many UNIX systems have also got C2 certification, and come configured this way from the manufacturer.

[4] This document may be found at http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html.
[5] Trusted Computing Base.

The 1998 attacks on the Pentagon involved theft and modification of data, as well as denial-of-service. The attacked machines were C2-secure Windows NT machines.

Many UNIX systems have also got C2 certification, and come configured this way from the manufacturer.

There are numerous examples of hacked UNIX systems found on the Internet. In 1996, a site I managed in New Zealand was the target of a malicious attack by intruders from Australia and Belgium.

Given all this, C2 certification is probably not a good guide as to the security of your system.

9.6 Summary of topics

In this section, we introduced the following topics:

- Ethical concerns
- Security institutions

Further study

- Material on ethical development at http://www.ilt.columbia.edu/publications/artistotle.html or http://caae.phil.cmu.edu/Cavalier/80130/part2/Kohl_Gilligan.html.

Chapter 10

OS Insecurity case studies

We present here a series of brief insecurity case studies related to operating systems. We can characterize OS insecurity in many ways. For example we might be concerned with hacking, and categorize the hacks into *internal* and *external* hacks.

- An internal hack is one performed by a valid OS user (i.e. someone who can log in).
- An external hack is one performed by an outsider.

Another hacking characterization might be to categorize the hacks into types (buffer overflows, loopholes...).

We take a wide view of security, and include OS features intended to protect users from each other and the OS integrity in the face of malicious or just broken programs.

10.1 UNIX password security

UNIX systems are traditionally open systems, given their background in university environments. As such, the security on them is often minimal. It is common for UNIX accounts to be made available relatively freely. For example, at the MIT Media lab[1] all computers have been password-free until recently - an expression of academic freedom.

UNIX systems are vulnerable to a wide range of attacks, particularly internal attacks. All Unix systems have a ***root*** account. This account has a UID and GID of zero, and once root access is obtained on a UNIX system, there is very little that *cannot* be done. Account passwords are constructed to meet the following requirements:

[1] MIT - home of Kerberos!

- Each password has at least six characters.
- Only the first eight characters are significant.

There are many other accounts found on Unix systems, not just those for clients. For example the following accounts are commonly found:

sysadm - A System V administration account, and

daemon - A daemon process account, and

uucp - The UUCP owner, and

lp - The print spooler owner.

When protecting a UNIX system, we must protect all these accounts as well - not just the valid-user accounts.. If a hacker managed to get unauthorized access to one of these accounts, then since the account has some capabilities on the system, then the hacker may be able to use this to make further attacks. However the principal account to protect is the **root** account.

Account information is kept in a file called /etc/passwd. It normally consists of seven colon-delimited fields, which may look like the following:

hugo:aAbBcJJJx23F55:501:100:Hughs Account:/home/hugo:/bin/tcsh

The fields are:

hugo: The account or user name.

aAbBcJJJx23F55: A one-way encrypted (hashed) password, and optional password aging information.

501: The UID - Unique user number

100: The GID - Group number of the default group that the user belongs to.

Hughs Account: Account information. In some versions of UNIX, this field also contains the user's office, extension, home phone, and so on. For historical reasons this field is called the GECOS field.

/home/hugo: The account's home directory

/bin/tcsh: A program to run when you log in - usually a shell

UNIX uses a DES-like algorithm to calculate the encrypted password. The password is used as the DES key (eight 7-bit characters make a 56 bit DES key) to encrypt a block of binary zeroes. The result of this encryption is the hash value. Note: the password is not encrypted, it is the key used to perform the encryption!

A strengthening feature of UNIX is that it introduces two random characters in the alogrithm (the salt). This ensures that two equal passwords result in two different hash values. From viewing the UNIX password file you can not deduce whether two persons have the same password. Even if they do, the salt introduced in the algorithm ensures that the hash values will be different.

When you log in with your account name and password, the password is encrypted and the resulting hash is compared to the hash stored in the password file. If they are equal, the system accepts that you've typed in the correct password and grants you access.

10.1.1 Crypt code

Sample crypt code from LINUX uClibc. The code has the following structure:

```
extern char * crypt(const char *key, const char *salt) {
      /* First, check if we are supposed to be using the MD5 replacement
      /* instead of DES...  */
   if (salt[0]=='$' && salt[1]=='1' && salt[2]=='$')
       return __md5_crypt(key, salt);
   else
       return __des_crypt(key, salt);
}
```

To prevent crackers from simply encrypting an entire dictionary and then looking up the hash, the salt was added to the algorithm to create a possible 4096 different conceivable hashes for a particular password. This lengthens the cracking time because it becomes a little harder to store an encrypted dictionary online as the encrypted dictionary now would have to take up 4096 times the disk space. This does not make password cracking harder, just more time consuming.

10.1.2 Brute force cracking

Brute force password cracking is simply trying a password of A with the given salt, folowing by B, C, and on and on until every possible character combination is tried. It is very time consuming, but given enough time, brute force cracking ***will*** get the password.

The hashed passwords are compared with the entry in the */etc/passwd* file. You cannot try to exhaustively log in using all the possible passwords, as UNIX systems enforce 10 second timeouts after three consecutive login failures.

10.1.3 Dictionary cracking

Dictionary password cracking is the most popular method for cracking Unix passwords. The cracking program will take a word list, and one at a time try to crack one or all of the passwords listed in the password file. Some password crackers will filter and/or mutate the words as they

try them, such as substitute numbers for certain letters, add prefixes or suffixes, or switch case or order of letters.

A popular cracking utility is called ***crack.*** It can be configured by an administrator to periodically run and send email to users with weak passwords, or it may be run in manual mode. Crack can also be configured to run across multiple systems and to use user-definable rules for word manipulation/mutation to maximize dictionary effectiveness.

10.1.4 UNIX base security fix

The susceptibility of UNIX systems to dictionary attacks has been known for many years, and a system known as ***shadow*** passwords is used to fix the problem. Most modern UNIXes either use ***shadow*** passwords out-of-the-box, or can be configured to use them by running a utility.

Once the password hashes are moved to the shadow file, its permissions are changed as follows:

```
opo 35# ls -l /etc/shadow
-r--------   1 root     sys      3429 Aug 20 14:46 /etc/shadow
opo 36#
```

These permissions ensure that ordinary users are unable to look at the password hashes, and hence are unable to try dictionary attacks.

10.2 Microsoft password security

Two one-way password hashes are stored on NT systems:

- a LanManager hash, and
- a Windows NT hash.

The LanManager hash supports the older LanManager protocol originally used in Windows and OS/2. In an all-NT environment it is desirable to turn off LanManager passwords, as it is easier to crack. The NT method uses a stronger algorithm and allows mixed-cased passwords.

The database containing these hashes on an NT system is called the SAM (Security Access Manager) database and is used for all user authentication as well as inter-process authentication. If you have administrative access[2], the program ***pwdump*** can extract the hashes. The hashes may also be directly captured from a local area network using a sniff utility such as ***readsmb***[3].

[2] Originally, *anyone* could extract the hashed passwords from the SAM, as Microsoft believed that "If they didn't tell anyone the algorithms they used, no-one could discover what they had done". Security through obscurity is not a safe strategy, and Jeremy Allison was able to de-obfuscate the SAM entries relatively quickly.

[3] The security strategies used by Microsoft have been uncovered by the SAMBA team, to allow their development of an open source SMB file and print service. Some parts of this section have been extracted from the SAMBA documentation.

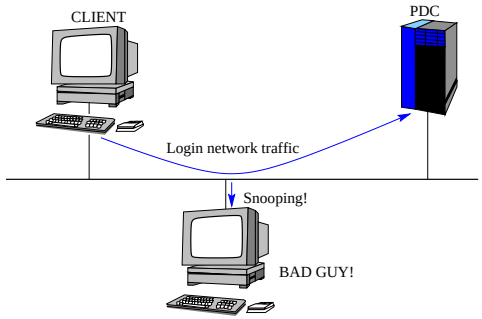

Figure 10.1: Network login traffic snooping.

In figure 10.1, we see network login traffic between a Windows client and a PDC (Primary Domain Controller). If the network media is a shared bus (such as ethernet), then the login traffic may be sniffed (or snooped) by a third party.

Microsoft does not *salt* during hash generation, so once a potential password has generated a hash it can be checked against ***all*** accounts. The crack software takes advantage of this.

10.2.1 LanManager encryption

LanManager encryption is created by taking the user's plaintext password, capitalising it, and either truncating to 14 bytes, or padding to 14 bytes with null bytes. This 14 byte value is used as two 56-bit DES keys to encrypt a *magic* eight byte value, forming a 16 byte value which is stored by the server and client. This value is known as the ***hashed password***.

10.2.2 NT encryption

Windows NT encryption is a higher quality mechanism, consisting of doing an MD4 hash on a Unicode version of the user's password. This also produces a 16 byte hash value that is non-reversible.

MD4 is a one-way hashing function developed by Ron Rivest (The R in RSA). It takes 512-bit blocks as input and outputs a 128-bit fingerprint of the input data. It is described in ***rfc1320***, complete with source code detailing the algorithm.

10.2.3 Challenge-response protocol

When a client wishes to use an SMB resource, it first requests a connection and negotiates the protocol that the client and server will use. In the reply to this request the server generates and appends an 8 byte, random value - this is stored in the server after the reply is sent and is known as the ***challenge***. It is different for every client connection.

The client then uses the hashed password (16 byte values described above), appended with 5 null bytes, as three 56 bit DES keys, each of which is used to encrypt the challenge 8 byte value, forming a 24 byte value known as the ***response***. This calculation is done on *both* hashes of the user's password, and *both* responses are returned to the server, giving two 24 byte values.

The server then reproduces the above calculation, using its own value of the 16 byte hashed password and the challenge value that it kept during the initial protocol negotiation. It then checks to see if the 24 byte value it calculates matches the 24 byte value returned to it from the client. If these values match exactly, then the client knew the correct password and is allowed access. If not then the client did not know the correct password and is denied access.

There are good points about this:

- The server never knows or stores the *cleartext* of the users password - just the 16 byte hashed values derived from it.
- The cleartext password or 16 byte hashed values are never transmitted over the network - thus increasing security.

However, there is also a bad side:

- The 16 byte hashed values are a "password equivalent". You cannot derive the users password from them, but they can be used in a modified client to gain access to a server.
- The initial protocol negotiation is generally insecure, and can be hijacked in a range of ways. One common hijack involves convincing the server to allow clear-text passwords.

Despite functionality added to NT to protect unauthorized access to the SAM, the mechanism is trivially insecure - both the hashed values can be retrieved using the network sniffer mentioned before, and they are as-good-as passwords.

10.2.4 Attack

The security of NT systems relies on a flawed mechanism. Even *without* network access, it is possible by various means to access the SAM password hashes, and *with* network access it is easy. The hashed values are password equivalents, and may be used directly if you have modified client software.

The attack considered here is the use of either a dictionary, or brute force attack directly on the password hashes (which must be first collected somehow).

L0phtCrack is a tool for turning Microsoft Lan Manager and NT password hashes back into the original clear text passwords. It may be configured to run in different ways.

Dictionary cracking: L0phtCrack running on a Pentium Pro 200 checked a password file with 100 passwords against a 8 Megabyte (about 1,000,000 word) dictionary file in under one minute.

Brute force: L0phtCrack running on a Pentium Pro 200 checked a password file with 10 passwords using the alpha character set (A-Z) in 26 hours.

As the character sets increase in size from 26 characters to 68 the time to brute force the password increases exponentially. This chart illustrates the relative time for larger character sets.

Character set size	Size of computation	Relative time taken
26	$8.353 * 10^9$	1.00
36	$8.060 * 10^{10}$	9.65
46	$4.455 * 10^{11}$	53.33
68	$6.823 * 10^{12}$	816.86

So if 26 characters takes 26 hours to complete, a worst-case scenario for 36 characters (A-Z,0-9) would take 250 hours or 10.5 days. A password such as ***take2asp1r1n*** would probably be computed in about 7 days.

10.2.5 Microsoft base security fix

A range of steps may be taken to reduce exposure due to the hash insecurity.

- Disable the use of Lan Manager passwords.
- Don't log in over network as any user you do not wish to compromise.
- Encrypt all network traffic (to be discussed later in the section on use of ***ssh***).
- Use long passwords, and all allowable characters, to slow down the crack.
- Use an alternative login system (PAM supports multiple login methods, and there are more secure systems).
- Use an unsniffable network cabling system.

10.3 Summary of topics

In this section, we introduced the following topics:

- Simple OS password security

Further study

- Textbook, Section 12.2
- Textbook, Section 12.3