Selected Aspects of Computer Security in Open Systems

Danny Smith
Australian Computer Emergency Response Team
c/- Prentice Centre
The University of Queensland
Qld. 4072. Australia

+61 7 365 4417
auscert@auscert.org.au

8 November, 1993

Abstract

The past five years have seen a lot more attention being focused on computer and network security. This heightened awareness has given rise to a better understanding of the types of threats a computer system might face. It has also seen the development of new tools and techniques to combat those threats.

This report examines those security threats, and details what tools and techniques are available to combat them. These threats are categorised using standard criteria. In addition, an examination of common authentication techniques is provided, including a comparison between X.509 Strong Authentication and Kerberos.

The suite of available tools is complemented, through the development of a one time password system for OpenVMS. This development is also detailed.

1. Introduction

In November 1988 a "worm" program was released to the Internet, which (although not entirely intended by the worm's author) brought much of the network to a standstill [Spa88]. Many sites disconnected from the network until a solution could be found. This had an added disadvantage as many of the first fixes for the problem were distributed via the network.

In March 1991 a ship was lost in the Bay of Biscay. A weather forecasting satellite was not working as the integrity of a computer system at the European Weather Forecasting Centre in Bracknell, Berkshire had been broken by intruders[1]. The storm that caused the ship to sink had not been predicted [Aus93].

In mid-1993, a number of sensitive medical test results were changed from negative to positive by intruders. Various people around that country were then led to believe that they had cancer. It was up to Police to advise them that the results had been changed and were incorrect [Aus93].

Often when the topic of computer security is discussed, particularly when it relates to intruders gaining unauthorised access, a wide range of viewpoints can be expressed. Some people view intruders as "fighters for freedom of information"; others view them as "joyriders in the electronic space"; others view them as "intellectual joyriders". Mostly the intruders do not cause harm, but when intruders perform acts like the sample described above, it is difficult to tell the difference between the "good" intruder and the "bad" intruder. Therefore, these intruders must now be treated as criminals.

This report examines the topic of computer security for Open Systems. In this report, an Open System is one that is connected to a global network without any network access restrictions. There are many other more restrictive definitions for an Open System, but this one will suffice for the purposes of this report.

Over the past five years, computer vulnerabilities and the way they are exploited has changed character. In 1988, many intruders were simply exploiting poor password choices, poor system configuration, or software vulnerabilities [CER92]. However, in 1993, those types of attacks are still common, with the addition of people viewing network information, either through modifying the network software or looking at data that traverses the network links. Intruders are now also actively examining source code for operating systems and utilities in an attempt to determine any potential vulnerabilities, or attempting to exploit poor protocol design to try and fool servers into performing some action.

---

[1]This report uses the term "intruder" to indicate a person who enters a computer system, or who accesses or modifies data, and does not contain the appropriate authorisation to do so. Other terms are used in the public domain such as "hacker" and "cracker", and these terms are continually debated for appropriateness. The term intruder is chosen here, as it indicates the criminal and antisocial intent of the activity, as viewed by the author. Other terms which would also been seen to be acceptable to the author are "criminal", "vandal", or "data terrorist".

This report examines the common threats faced by computer systems, and looks at various tools and techniques that have been developed to combat these threats. Its aim is to examine what tools and techniques are available to combat the known threats. As a result of this examination, an extension is provided to the security on one open system (OpenVMS) where a deficiency is perceived.

Whilst this report concentrates heavily on the Unix operating system, many of the vulnerabilities and techniques to combat them apply to all operating systems. The approach taken is not a formal one, but a comparison of services provided from a user perspective. There are many papers that formalise the analysis of security (for example, see [LAB92]).

This report commences with an examination of authentication techniques, as authentication forms the backbone which other security services build upon. It is for this reason that the author believes that more attention should be devoted to it. This examination is found in Chapter 2.

Chapter 3 examines common threats faced by computer systems today. These threats involve more than authentication services, and special attention is devoted to categorising them based upon standard criteria. In particular, this chapter captures the wider spectrum of computer security, as some of the detailed threats bypass authentication, rendering them potentially more dangerous than they might first appear.

Chapter 4 contains an examination and evaluation of what tools are readily available to combat the common threats outlined. These tools have been developed to address the problems of computer security.

Once the general problems of security are detailed and solutions proposed, an extension is proposed to an existing system (OpenVMS) for which a common solution (one time passwords) to a standard problem (network monitoring) is not freely available. A design and implementation is supplied in Chapter 5 for the proposed extension. All source code and header files are found in Appendix A.

2. Authentication Techniques

This chapter reviews computer security services in broad terms, and indicates why authentication is important. It examines the threats that authentication faces, and then compares some of the commonly used authentication techniques in use, looking at their strengths and weaknesses. These techniques generally involve the use of a password to effect the authentication.

Some of the newer techniques use public and private key encryption techniques to protect and enhance the authentication sequence. These authentication techniques are also examined, looking at their strengths and weaknesses.

2.1. What is Security?

Businesses today require the processing and exchange of information to conduct their business dealings. The Twentieth Century has been described as "The Information Age". Security of that information is now paramount if business is to survive in this new world [CLS91], [Edw90].

In order to protect against perceived threats, various security services need to be provided. These can include [ISO92]:

- *Authentication*: the act of identifying that an entity is who or what they are claiming to be;
- *Access Control*: the act of deciding whether an entity has the rights or authority to perform the task they are seeking to do;
- *Data Confidentiality*: the act of protecting data from unauthorised disclosure;
- *Data Integrity*: the act of protecting data from manipulation;
- *Non-repudiation*: the act of proving the integrity and origin of data in an unforgeable relationship which can be verified by a third party at any time.

When imposing access controls, judgment must be reliable and predictable , and the contents of an audit trail should be accurate to resolve any problems that might occur. Data Integrity and Confidentiality may be questioned if the identity of the entity being communicated with is in doubt [JM91]. Correct identification is impossible if the act of authenticating the entity is deficient, and doubts are cast on the integrity of the authentication[2].

2.2. What are the Threats to Data Security?

The act of authenticating an entity must be trustworthy if there is to be any security in the computer system at all. This means that the authentication facility must correctly identify the entity in all cases, regardless of the role played by the entity, the location of the entity, and the presence or absence of potential enemies. The commonly

[2] Then again, can the authentication be believed if the access control and data integrity are insufficient to protect the authentication data?

known threats [ISO92] to data security include identity interception, masquerade, reply, data interception, manipulation, repudiation, denial of service, misrouting, and traffic analysis. These threats are more fully defined in Section 3.1.

This remainder of this chapter examines a number of authentication schemes and indicates if any protection is provided for Identity Interception, Masquerade, Replay, and Repudiation threats.

One of the aspects of authentication that is often overlooked is that it is a two-way street. When a user authenticates to a computer, it is often just as important that the computer authenticates to the user. This has been highlighted in recent times with people starting "trojan horse" programs that imitate the login sequence, but in reality capture the account and password for use by an enemy. Another example is an automatic teller machine that captures account numbers and PINs for later use. The ability to authenticate the machine will help to prevent these types of attack.

2.3. How is Authentication Achieved?

Authentication is generally achieved through one or more of three items [CLS91]. These are:

> Something you *know*. This could for example be a password or Personal Identification Number (PIN);
>
> Something you *possess*. This could be an identity card, an access card for electronic locks, or a smart card;
>
> Something you *are*. This involves something personal about the subject such as a retina scan. It may also involve the role played when seeking authentication. For example, a person's rights to perform an action are vastly different if they acting as a Lecturer in a Department, as opposed to acting as the Head of that Department. The role of Head may be delegated via authentication.

Most authentication techniques make use of one of these items (such as knowing a password, possessing a key to an office, or signing of a name as the Head of Department). Whilst in many circumstances this is sufficient, the authentication can be enhanced by combining two or more of these items together. Examples of this might be possessing an identity card indicating the status of Head of Department which must be shown to security personnel, plus possessing a key to that office. If an attempt to enter the office of Head of Department with an identity card that indicates the status of a lecturer is attempted, then the security personnel should forbid this.

Another example in common use is the possession of a magnetic stripe card from the bank, and knowing the PIN for that card.

The following sections investigate different authentication techniques, describing their attributes according to this criteria, and looks at how well they cope with the threats detailed above.

2.4. Passwords

2.4.1. Normal Passwords

This is by far the most common authentication technique used on computer systems today. It is categorised by "something you know" which is a user account and the password for that account. If this information is revealed in any way, then an enemy can authenticate to the computer system, pretending to be a legitimate user.

Passwords are stored and handled in many different ways by different computer systems.

2.4.1.1. Plain Text, Non-world Readable

International Business Machine's VM/CMS stores its passwords in the directory file in plain text (EBCDIC), but to access this file requires special privileges and permissions which are normally only granted to a small number of users [IBM89].

Strengths

This system is very easy to implement, easy to maintain, and easy to logically prove that it is behaving correctly (that is, it can be trusted).

Should a user forget their password, it is relatively easy for an authorised person to find out what the password was, and reveal this information to the user.

Weaknesses

If access is gained to the password file by an unauthorised person, then *every* account and password on the system is revealed. This means that the Virtual Machine concept that so effectively keeps users separated is compromised, and any and all data on the system is available to the enemy.

Since all passwords are stored in plain text, then security personnel for that system will have access to every password. This would include passwords for the Head of Department, leading researchers, system administrators, and general users. It would be relatively easy to access any information on the system, masquerading as the owner of that information, thus bypassing the access control and auditing facilities.

2.4.1.2. World Readable, but Encrypted

This is the technique mostly used on UNIX systems. The password is encrypted using a one way function (one that is computationally difficult to reverse), and the encrypted version stored in a world readable file. The encryption is further protected by "salting" the algorithm with a seed chosen from 4096 possible values. This means that a single password may produce up to 4096 possible encryptions [GS91].

Strengths

The algorithm to encrypt passwords operates by encrypting a null string several times over with an altered version of the DES [FIP77] algorithm using the plain text password as the key. To date, nobody has successfully reversed this algorithm in a reasonable time.

By having the password file world readable and the algorithm publicly available, it simplifies the design of applications that require authentication services. Therefore, it is simpler to prove that these applications are performing correctly (that is, they can be trusted).

The UNIX authentication system has been subject to much attention due to its public nature. It has been studied, targeted, tested, and analysed by many people. This should give a user confidence that the algorithm is computationally strong enough for the present moment (provided a good password is chosen - see [SER93]).

Weaknesses

The *major* weakness of this authentication technique is the combination of the world readable file, and human nature. Whilst the encryption algorithm has not been reversed, much success has been gained by guessing the plain text password, encrypting it, and checking the encrypted value against the encrypted password stored in the password file. These password guessing attacks are notoriously successful because human users often pick passwords that are easy to remember, which often means that they are easy to guess [CLS91].

Another weakness also involves brute force attacks. With the speed of computers ever increasing in leaps and bounds, it may be feasible one day to just try all possible passwords against the encrypted version. The current estimates for encryption speeds indicate that 6,400,000 passwords encryptions may be achieved *per second*. This enables all combinations of six character passwords to be tested in less than one hour [Mad93]. The ability to check all eight character (maximum length) passwords may not be all that far in the future[3]. Therefore, it is imperative that the password be changed regularly, well chosen, and not reused.

2.4.1.3. Non-world Readable Passwords

One of the major problems with UNIX style passwords is that the encrypted password is freely available. Many attacks are performed by taking a copy of the encrypted password to another machine, and applying attacks such as dictionary attacks in an attempt to guess the password. Often the remote machine belongs to another site, and has been compromised previously. The crackers use the resources of one organisation to gain access to another organisation.

One mechanism to prevent the encrypted password from becoming known is to store it in a non-world readable file. This concept is called shadow passwords on UNIX systems. The utilities that require access to the encrypted password require special privileges to perform their tasks. The encrypted passwords can still be read by privileged users, but by protecting the shadow password file appropriately, then the normal system user cannot gain access to the information. This means that even if a

[3]At this speed, the time required to check all eight character passwords is 31 years.

password is guessed, and access gained to the system, the rest of the encrypted passwords are still secure.

Digital Equipment Corporation's OpenVMS has had the concept on non-readable authorisation files for many years now [DEC90]. Access can be gained to the authorisation file through the use of special system calls and utilities. Access to some parts of the authorisation file are freely available, whilst access to others is restricted to privileged users.

This still does not prevent password guessing attacks. However, since the password file is now not world readable, the password guessing attacks will need to be performed on the machine under attack. Both OpenVMS and UNIX implement mechanisms to prevent or limit the success of such attacks[4].

Ultimately, the success of the password mechanism is up to the individual user to ensure that their password is well chosen, and not easy to guess. Analysis of how to form a well chosen password can be found in many places. Some examples are [SER93], [Cur90], and [Bis92a].

Strengths

It now requires privileged access to copy the password file and take it away to another machine to run password guessing programs on it. This means that if one user allows access to the system via an easy to guess password, the rest of the passwords are not at risk by that user's actions.

Since the encrypted password is no longer freely available, password guessing attacks need to be performed on the local machine (by attempting to login to the account using different passwords). This allows the machine to take action against this type of attack, and enforce a consistent security policy.

Weaknesses

Now that it requires privileged code to access the encrypted password, it requires special care must be taken to ensure that extra security holes are not created. This means that it is becoming more difficult to prove that the security system is behaving as expected (that is, it can be trusted).

Such systems are sometimes not standard, as shipped with the original operating system code. Therefore, they are security "add-ons" which are provided by a third party. This means that they may not have received a thorough testing, or may be inconsistent with existing operating system code. Generally however, most of these types of security systems have been around long enough to have the major bugs fixed in them.

The password is still transmitted in plain text on the network, thus rendering this scheme susceptible to replay attacks.

---

[4]These facilities are beyond the scope of this report.

2.4.2. Passwords - Secure?

Are passwords all that secure then? It is possible to protect them from being revealed, either while they are stored on the system or being transmitted as part of an authentication sequence.

Klein [Kle90] completed a study of password security by attempting to "guess" passwords that were in use on various machines. After 12 CPU months, more than 25% of the passwords were guessed[5]. Of those passwords guessed, more than half of them were six characters or less. This, coupled with the huge advancements in computer speed almost brings a brute force attack into the realms of reality - no password will be safe [Alv90]!

Guessing just ONE password is enough to allow access to a system. Having achieved this, an intruder has taken the first step to gaining further access to the computer system [Che92]. Some suggest that a hit rate of over 50% is not uncommon the first time a password guessing program is run [KC90]. Of five documented security incidents at Columbia University, all were as a result of bad passwords [BKS90].

The problem with passwords is not the password itself, but the people choosing them [Sto89]. If all passwords were a MINIMUM of eight characters, and chosen well, then this type of attack would at best, yield minimal results [Spa92].

In the past, it was always considered poor practice to write a password down. Now instead of writing it down, it is simpler for a user to choose a password that is easy to remember (which most likely means it is also easy to guess). Maybe it is time to start rethinking the policies, and suggest that it *is* alright to write a password down, provided that the password is stored in a safe place. The literature is full of papers where passwords are guessed, but no article indicated that someone's wallet was stolen to retrieve the password. Even if it is, the owner will know that their password has been compromised, rather than relying on chance to discover that someone has guessed their password [Bra90].

The use of default passwords and poorly chosen passwords has caused many problems over the years [CER92]. This is the major flaw that allowed the WANK and OILZ worms to operate [LS93].

The choice of passwords can be improved by preventing users from changing their password to a known "bad" password. The checks for a bad password are often done at the time the user changes their password, using a form of pro-active password checker (Some examples are [Bis92a], [Kur90], and [Spa92]).

2.4.3. One Time Passwords

One-time password generators generally employ a combination of hardware and software to effect the authentication. This technique employs "something you possess" and in some systems, "something you know". The hardware devices are

[5] 21% were guessed in the first week, and 368 (2.7%) in the first 15 minutes of testing.

usually known as "tokens" or "password tokens". There are several techniques in use [Ell92], [CER92], [Lau92].

One system operates by issuing a challenge which consists of a seven digit number (in phone number format). The user enters their PIN number and the challenge number into the hand held device which returns a seven digit response code to reply with.

Other systems use a changing, non-reusable password system. Each time the user authenticates, a new password is supplied by the hand held device. There is no challenge-response system, and the user must keep in synchronisation with password usage to prevent a denial of service. Some of these systems can support single use password generation for up to eight separate host systems. Some of these systems require that a PIN be entered before the next password is issued.

Another system displays the password continuously, changing it every minute or so. The host must not only keep the user's key (for generating the same sequence), but also a synchronised clock. This system is vulnerable to replay attacks if the password is captured and replayed prior to it changing. Some systems combat this by only allowing each password to be used once.

Some of the systems that employ the use of a PIN allow a second PIN to be specified. This may be useful for example to alert security staff that the user is being held and forced to access the computer system against their will. The second PIN may allow the user access (in a restricted sense), but also signal alarms!

S/Key is a software only system that requires the user to carry a list of generated passwords with them [KHW93]. S/Key is discussed in Section 4.4.9 and Chapter 5.

Strengths

The one-time password system is extremely effective in preventing replay attacks, provided the enemy does not know the sequence of generated passwords (either by guessing, or possession of a similar device and key). Many attacks in the present time will be instigated by monitoring authentication sequences [CER92].

Extra techniques can be employed that allow limited access, whilst also generating an alarm to indicate that this authentication sequence is being forced by an enemy.

Weaknesses

One of the major disadvantages is that to authenticate, the user must carry the hardware or password list with them at all times. If they do not possess the hand held device, then authentication cannot take place.

Some of the systems have a requirement for synchronised clocks. This suffers the same problems that Kerberos does, with denial of service due to clocks slipping, or an attacker targeting the clock synchronisation protocols to set the time to any desired value.

One particular system used generated keys that were so short, that it was easier to guess the password "1111" than it was to guess the correct password for the generated key [Bra90].

Systems that make use of a password that changes at time intervals may be prone to replay attacks if special precautions are not taken to prevent them.

2.5. Kerberos

The following analysis is drawn from [Ste90], [BM91], and [KCS90].

The Kerberos authentication system was produced at MIT as a part of Project Athena. It is a system that uses protocols which allow authentication to take place, even under the assumption that the network is under the complete control of an enemy. Kerberos uses a private key cryptosystem to protect the information from disclosure and modification. The user interface is the same as that for normal passwords, so this authentication technique is categorised as "something you know".

Kerberos authenticates a principal by providing evidence of that principal's identity. The principal can be a user or a particular service on some machine. A principal consists of the tuple:

<primaryname, instance, realm>

If the principal is a user (a person), then the primary name is the login identifier, and the instance is either null or represents particular attributes of the user. For a service, the service name is used as the primary name and the machine name is used as the instance. The realm is used to distinguish among different authentication domains. It is therefore possible to have several Kerberos databases serving an organisation rather than one large and universally trusted one.

In the following discussion, the symbols used are defined as:

c	client principal
s	server principal
tgs	ticket granting server
K_x	private key of "x"
$K_{c,s}$	session key for "c" and "s"
$\{ info\}K_x$	"info" encrypted in key K_x
$\{ T_{c,s}\}K_s$	Encrypted ticket for "c" to use "s"
$\{ A_c\}K_{c,s}$	Encrypted authenticator for "c" to use "s"
addr	client's IP address

Kerberos principals may obtain tickets for services from a special server known as the ticket-granting server or TGS. A ticket contains assorted information identifying the principal, encrypted using the private key of the service. `timestamp1` indicates the time the ticket was issued, and `lifetime` the amount of time the ticket is valid for.

$$\{ T_{c,s}\}K_s = \{ s, c, addr, timestamp1, lifetime, K_{c,s}\}K_s$$

Since only Kerberos and the service share the private key K_s, the ticket is known to be authentic. The ticket contains a new private session key, $K_{c,s}$, known to the client as well (when it obtains the ticket); this key may be used to encrypt transactions during the session which protects them from disclosure and modification.

To guard against replay attacks, all tickets presented are accompanied by an authenticator:

$$\{ A_c\}K_{c,s} = \{ c, addr, timestamp2\}K_{c,s}$$

This is a string encrypted in the session key and containing `timestamp2` (the current time); if the time does not match the current time within the (predetermined) clock skew limits, the request is assumed to be fraudulent. This clock skew limit is usually in the order of five minutes.

For services where the client needs bidirectional authentication, the server can reply with:

$$\{ timestamp2 + 1\}K_{c,s}$$

This demonstrates that the server was able to read timestamp from the authenticator, and hence that it knew $K_{c,s}$; that in turn is only available in the ticket, which is encrypted using the server's private key.

Tickets are obtained from the TGS by sending a request:

$$s, \{ T_{c,tgs}\}K_{tgs}, \{ A_c\}K_{c,tgs}$$

In other words, an ordinary ticket/authenticator pair is used; the ticket is known as the ticket-granting ticket. The TGS responds with a ticket for server `s` and a copy of $K_{c,s}$, all encrypted with a private key shared by the TGS and the principal:

$$\{ \{ T_{c,s}\}K_s, K_{c,s}\}K_{c,tgs}$$

The session key $K_{c,s}$ is a newly-chosen random key.

The key $K_{c,tgs}$ and the ticket-granting ticket itself, are obtained at session-start time. The client sends a message to Kerberos with a principal name; Kerberos responds with:

$$\{ K_{c,tgs}, \{ T_{c,tgs}\}K_{tgs}\}K_c$$

The client key K_c is derived from a non-invertible transform of the user's typed password. Thus all privileges depend ultimately on this one key (the user's password).

Note that servers must possess private keys of their own, in order to decrypt tickets. These keys are stored in a secure location on the server's machine. The security of the tickets and ultimately, the session key depends upon the security of the server private keys.

Strengths

The major strength of Kerberos is that the password is never transmitted on the network in plain text. This reduces the likelihood of the password being captured and replayed.

The tickets and authenticators include a timestamp which goes a long way to preventing replay attacks. If an authenticator is captured and replayed, it will not be valid the second time around provided sufficient time has passed by. To use this type of attack successfully requires either breaking the encryption algorithm, or possessing the password, the session key, or the server key.

This style of authentication was designed with the distributed or networking environment in mind. It is well suited to the client-server model often used in networking applications.

Since both the ticket and authenticator contain the network address of the client, another workstation cannot use stolen copies without changing their network address to be the same as the one that owns the stolen copy.

Weaknesses

The timestamps are critical to the successful operation of Kerberos. The times on the source and target machines must be closely aligned, or it will be possible that a valid ticket will be rejected as fraudulent because the time is incorrect. Typically, a clock drift of five minutes will cause a denial of service.

Relying on the time to be in synchronisation means that one should also protect the protocols that set the time, so that an enemy cannot adjust the time to their will via this mechanism.

Tickets are reusable. However, after having been authenticated for a long period (typically eight hours), it is necessary to generate a new ticket. This is because every ticket has a lifetime so once it expires, a new ticket must be generated by entering the login name and password to Kerberos again.

Within the Project Athena environment (and hence, Kerberos), the primary need is for user to server authentication. That is, when a user sits down at a workstation, that person needs access to private files residing on a server. The workstation itself has no such files, and hence has no need to contact the server or even identify itself. This is in marked contrast to a typical UNIX system's view of the world. Such systems do have an identity, and they do own files. Assorted network daemons transfer files in the background, clock daemons perform management functions, electronic mail and news is transferred, and so on. If such a machine relied on servers to store its files, it would have to prove its identity when talking to these servers. This means that Kerberos is designed to authenticate the end-user - a human being sitting at a keyboard. It is not a peer to peer system, intended to be used by one computer's daemons when contacting another computer.

In a workstation environment, it is quite simple for an intruder to replace the login command with a version that contains a trojan horse (captures accounts and passwords). Such an attack negates the primary strength of Kerberos, that passwords are not transmitted in plain text over a network. While this problem is not restricted to Kerberos environments, the Kerberos protocol makes it difficult to employ the standard countermeasure: one-time passwords.

The authenticator relies on the use of a timestamp to prevent against replay. Given that the lifetime of an authenticator is typically five minutes, a window of opportunity exists where a stolen live authenticator could be used to fraudulently gain access to a server. It has been suggested that the proper defence is for the server to store all live authenticators so that a replay could be detected. However, on UNIX systems, TCP-based servers generally operate by forking a separate process to handle each incoming request. Since the child and the parent do not share any memory, it is not convenient to communicate to the parent (or any other child processes) the value of any authenticator that is presented. UDP-based query servers generally use a single process to handle all incoming requests, but may have problems with legitimate retransmissions of the client's request if the answer gets lost.

Whilst the Kerberos system guards against having to send the password in plain text, the passwords chosen are much the same as the standard UNIX password, and therefore suffer the same fate of normal passwords in a password guessing attack. At least the encrypted password is not freely available (it is stored on the Kerberos server), so the success of such an attack is much the same as the system where the passwords are not world readable. To succeed with such an attack would require the password to be guessed, rather than just replaying the authentication sequence again. This would allow an enemy to masquerade as another user, without using replay.

Tickets are based upon a system's IP address. On multi-homed systems (systems with more than one network interface and IP address), this may cause a problem as the ticket will only be valid through one of those interfaces.

Tickets are stored in `/tmp` which does not work very well for multi-user systems.

2.6. X.509 Authentication

X.509 is a CCITT Recommendation (as well as being Part 8 of ISO International Standard 9594) [ISO92]. This standard forms the Authentication Framework, initially for use by the X.500 Directory Service. The concepts and facilities provided by this framework could be used by other OSI applications, or by network applications in general that require such facilities.

The strong authentication techniques of X.509 make use of public key cryptosystems (PKCS), unlike Kerberos. The recommendation states the requirements of the PKCS, without dictating any particular algorithm. It also includes the mechanisms for specifying the algorithm in use, and therefore no single algorithm is required to successfully use X.509 services. One particular algorithm that is commonly known, and satisfies the criteria dictated by X.509 is RSA [RSA78].

Whilst X.509 could be categorised by "something you know" (the password), it is possible that the secret key for the PKCS (when using strong authentication) could be so complex that it is necessary to store it on something like a "smart card". This system might require a smaller password to unlock the smart card. In this case, the authentication is categorised by "something you possess" and "something you know".

The recommendation describes more facilities than just authentication (such as Certificates, Certification Authorities, and Digital Signatures), but the scope of this report is limited to the use of digital signatures and authentication facilities. The authentication facilities range from very simple authentication, to extremely strong mechanisms.

2.6.1. Digital Signatures

Information can be *signed* by appending to it an encrypted summary of the information. The summary is produced by means of a one-way hash function while the encrypting is done by applying a public key cryptosystem and using the secret key of the signer[6]. This can be shown as [ISO92]:

$$X\{\text{Information}\} = \text{Information}, Xs[h(\text{ Information})]$$

The signature can be verified by applying the one-way hash function to the information, and comparing the result with that obtained by decrypting the signature using the public key of the signer[7] [Law93].

2.6.2. Simple Authentication

Simple Authentication is intended to provide local authorisation based upon the name of a user (their username) and an agreed password. Utilisation of Simple Authentication is primarily intended for local use only, and will not be very effective in a wider scope. There are two means by which Simple Authentication can be achieved:

<u>2.6.2.1. Unprotected Simple Authentication</u>

The procedure for achieving unprotected simple authentication is:

1. an originating user A sends their username and password to a recipient user B (the target machine);
2. B consults a local database of usernames and passwords where the password is checked against that held for user A;
3. the success or failure of authentication may be conveyed to A.

Unprotected Simple Authentication involves the transfer of the username and password in clear text to the recipient for evaluation. It is therefore prone to attack

[6]It is possible to use secret key encryption to implement digital signatures, but in general they are not as efficient. This report only addresses digital signatures that use public key encryption.

[7]It is not strong enough to encrypt a hash value of the information if the hash value is itself weak. Data manipulation may still occur, producing the same value for the hash function. In this case, the signature will be verified (providing data origin authentication), but the message will have been changed (voiding the data integrity requirements).

from network traffic monitors. As well, the passwords are stored in plain text, and may be discovered if privileged access to the database is gained.

Unprotected Simple Authentication could be compared with the plain text, non-world readable password mechanisms currently in use, such as that used in VM/CMS. If the password is stored in the database in encrypted form, then this mechanism would be comparable to the authentication systems found in Operating Systems such as UNIX and OpenVMS.

2.6.2.2. Protected Simple Authentication

The procedure for achieving protected simple authentication is:

1. an originating user *A* sends their protected identifying information (`Authenticator1`) to a recipient user *B* (the target machine).

   Authenticator1 = timestamp1, random1, username, Protected1
   Protected1 = f1(timestamp1, random1, username, password)

   where `random1` is a random number with an optional counter included (where used), `timestamp1` consists of timestamps (where used), `username` is the login identifier for the user seeking authentication, and `Protected1` is `timestamp1`, `random1`, `username`, and `password`, all processed through the one-way function f1;
2. *B* verifies the protected identifying information offered by *A* by generating (using the information included in `Authenticator1`, plus a local copy of the password for *A*) a local copy of `Protected1`. *B* compares for equality the locally generated copy of `Protected1` with the one supplied by *A*;
3. *B* confirms or denies the identity of *A* based upon this verification.

The procedure can be further protected by passing `Protected1` through another one-way function f2, and including a second random number and/or timestamp.

Authenticator2 = timestamp1, timestamp2, random1, random2, Protected2.
Protected2 = f2(timestamp2, random2, Protected1)

There is no requirement for f1 and f2 to be the same, or different. The principle for verifying the identity of *A* is the same as before.

The use of random numbers and timestamps is to minimise the chance of a successful replay attack, and to conceal the password from a cryptographic attack.

Protected Simple Authentication provides similar facilities to Kerberos in that it prevents the password from being transmitted in plain text, and provides some protection against replay attacks. However, Protected Simple Authentication does not provide bidirectional authentication which Kerberos can.

Strengths

One of the strengths of this technique is that it is relatively easy to implement, and to prove that it is functioning correctly without vulnerabilities. The more complex the system, the harder it is to determine the lack of vulnerabilities.

Another advantage is that it is quick to determine if the authentication attempt is successful or not. For transaction style operations, the time to authenticate a user may become an important issue.

The use of timestamps and random numbers reduces the chance of a successful replay attack.

Weaknesses

Since the username and password are transmitted in plain text, simple authentication can suffer from network monitoring and replay attacks.

The password is stored in the password database in plain text, so it is relying on the database security to protect the password from being revealed.

2.6.3. Strong Authentication

Strong Authentication is defined as "Authentication by means of cryptographically derived credentials" [ISO92]. The approach taken by X.509 makes use of public-key cryptosystems. These systems use two keys, one secret and one public, rather than a single key in private key cryptosystems.

Each user is identified by their possession of the secret key. A second user or system is able to determine if a partner is in possession of the secret key without either learning that secret key, nor revealing their own secret key to anyone.

Strong Authentication can be 1-way, 2-way, or 3-way as described below.

2.6.3.1. One-way Strong Authentication

One-way authentication is used when user *A* wishes to authenticate to user *B* (a system or user). *A* sends the following message:

$$B \rightarrow A, A\{ t^A, r^A, B\}$$

where $B \rightarrow A$ is a chain of certificates (certification path from *B* to *A*) which ultimately verifies the public key for *A*, t^A is a timestamp consisting of one or two dates (generation time and expiry time), r^A is a non-repeating number, and $A\{ t^A, r^A, B\}$ is the authentication information digitally signed by user *A* (by appending an encrypted summary to the end). The timestamps are used to determine the life of the token, and help to detect replay attacks. r^A is valid until the expiry date indicated in t^A.

It is possible to include extra information such as `sgnData` which will be used to provide data origin authentication by digital signatures, and `encData` which will be used as a secret key for the cryptosystem specified in `sgnData`. `encData` is encrypted using the public key of *B*, so that only *B* can read it.

$$B \rightarrow A, A\{ t^A, r^A, B, sgnData, Bp[encData]\}$$

Note: This last form of message (authentication token) in certain circumstances is subject to a form of manipulation attack. For example, suppose a token accompanies a confidential message which is encrypted with a key that is contained in `encData`. An opponent *C* might intercept the transmission of the token plus encrypted message from *A* to *B* and substitute a transmission consisting of the token:

$$C, C\{ t^C, r^C, B, sgnData, Bp[encData]\}$$

plus the original encrypted message. *B*, upon receipt of this substitute transmission might, in its response to *C*, disclose confidential information from the original encrypted message (which would have otherwise be inaccessible to *C*). One method that can avert such a threat is to include additional information in the encrypted message that accompanies the token (e.g., A, t^A, and r^A).

2.6.3.2. Two-way Strong Authentication

Two-way authentication is used when user *A* wishes to authenticate to user *B* (a system or user), and wishes *B* to authenticate to *A* in return. The initial authentication sequence is the same as one-way strong authentication, but *B* then returns to *A*:

$$B\{ t^B, r^B, A, r^A\}$$

where t^B is a timestamp defined in the same way as t^A, r^B is a non-repeating number used for similar purposes to r^A, and $B\{ t^B, r^B, A, r^A\}$ is the authentication information digitally signed using *B*'s secret key.

It is possible to include extra information such as `sgnData` which will be used to provide data origin authentication by digital signatures, and `encData` which will be used as a secret key for the cryptosystem specified in `sgnData`. `encData` is encrypted using the public key of *A*, so that only *A* can read it.

$$B\{ t^B, r^B, A, r^A, sgnData, Ap[encData]\}$$

Note that this means that separate keys may be used to provide data confidentiality in each direction of data transfer.

2.6.3.3. Three-way Strong Authentication

Three-way authentication is used to establish the same properties as two-way authentication, but does so without the need for association timestamp checking. It involves a further transfer from *A* to *B*.

The first two exchanges between *A* and *B* are the same for two-way authentication except that the timestamps may be zero, and they need not be checked. *A* then sends to *B*:

$$A\{ r^B, B\}$$

B needs to check that the received r^B is the same as the one sent. $A\{ r^B, B\}$ is the authentication information digitally signed using the secret key of *A*.

Strengths

One of the strengths of X.509 is that it is not hampered by existing systems in its operation and interface. It has the ability to use any facility at its disposal to perform its function well. However, this is a double-edged sword in that by trying to specify a system that is usable by all, it may be that the system is not usable by anybody. This may be reflected by the lack of widespread deployment of this system.

X.509 protects the authentication sequence from replay through the use of timestamps and random numbers. The inclusion of the random number means that it is not required that the token be valid for any longer than necessary. The authentication instance is defined by a combination of timestamp and random number, which will be unique in a defined time period. All that is required is that the entity performing the authentication cache the random numbers for at least the period of validity of the timestamp.

X.509 protects the authentication sequence from disclosure through the use of public key encryption. This allows a third party to verify the correct credentials of a purported entity, without that entity ever having to exchange any secret information, or revealing that entity's secret key. Thus authentication can be undertaken between mutually distrusting parties, provided a trusted entity verifies that the public keys of those parties are correct.

X.509 does not mandate the use of any particular public key encryption algorithm. One algorithm that matches the requirements of X.509 is RSA [RSA78]. This algorithm has to date been difficult to break, provided that keys of a suitable size are chosen. One advantage of the RSA algorithm is that should a certain key size be determined to be insufficient, it is possible to increase the key size to attain the desired level of security. This would also have the effect of slowing the encryption down, and increasing complexity.

Weaknesses

The only public key encryption algorithm that has been currently used with X.509 is RSA [RSA78]. This algorithm uses keys that can be difficult to manage. Private key encryption can use keys that are only 56 bits long, and therefore, can be easy to remember ASCII text. RSA works on large prime numbers, which can be 100 digits long! Some people can be hard pressed remembering seven digit phone numbers!

The X.509 standard details some other potential threats that are specific to the strong authentication method [ISO92]. These are:

- *Compromise of the user's secret key*: The identity of an entity is determined by possession of the secret key. Compromise of that key would mean that a rogue user could authenticate themself fraudulently;
- *Compromise of the Certification Authority's (CA) secret key*: Certification Authorities are a trusted source that can be consulted to

determine a user's public key. This key is transmitted inside a certificate (signed using the CA's secret key). If an invalid certificate is issued, than a rogue user could use an invalid secret/public key pair to authenticate as another user, and present what appears to be a valid certificate verifying the incorrect public key thus tricking an entity into believing that the invalid secret key in possession is the real one;

- *Misleading a CA into producing an invalid certificate*: In general, CAs are kept off-line to the main processing, and this offers some protection. It is up to the CA to verify the credentials of a user before issuing them a certificate that verifies their public key;
- *Collusion between a rogue CA and user*: This represents a basic breakdown of the trust relationship required in the X.509 method. There must be at least one entity that can be trusted to verify the credentials of all other (delegated) authorities. There are techniques to combat this type of threat. The reader is referred to [HY92] and [Smi93].
- *Forging of a certificate*: As the certificate is signed by the CA, the only way a certificate can be forged is if the CA's secret key is compromised, or the encryption technique is broken;
- *Forging of a token*: As the token is signed by the sender, the only way a token can be forged is if the sender's secret key is compromised, or the encryption technique is broken;
- *Replay of a token*: This is protected against by using random numbers and timestamps;
- *Attack on the cryptographic system*: "*the likelihood of effective cryptanalysis of the system, based on advances in computational number theory and leading to the need for a greater key length are reasonably predictable.*" [ISO92].

Lack of practical experience with X.509 means that it is difficult to prove that it does provide the services it promises effectively and efficiently. A system such as Kerberos has been in use in real situations for many years now, and its deficiencies are relatively well known. X.509 *may* contain hidden weaknesses that are yet to surface.

2.7. Authentication Techniques versus Threats

This section details which of the authentication techniques above provide protection against a subset of the threats outlined in Section 3.1. The threats chosen from the detailed list are Identity Interception, Replay, Masquerade, and Repudiation. Data Interception and Manipulation are viewed as extensions of the previous list, and are not considered. Denial of Service, Misrouting, and Traffic Analysis form a special type of threat which generally only applies to systems with extreme security (such as Government, or Military).

Table 1 shows which authentication scheme provides protection for a subset of defined security threats.

Table 1 - Threats handled by authentication schemes

	Identity Interception	Replay	Masquerade	Repudiation
plaintext passwords - world readable	x	x	x	x
encrypted passwords - world readable	x	x	x	x
encrypted passwords - read protected	x	x	x	x
one time passwords	x	Ã	Ã	x
Kerberos	x	Ã	Ã	x
X.509 simple unprotected	x	x	x	x
X.509 simple protected	x	x	x	x
X.509 strong - 1-way	Ã	Ã	Ã	Ã (see Note 1)
X.509 strong - 2-way	Ã	Ã	Ã	Ã (see Note 1)
X.509 strong - 3-way	Ã	Ã	Ã	Ã (see Note 1)

Note 1: Non-repudiation is not provided by the authentication facilities of X.509 alone, but is, when coupled with other facilities such as Digital Signatures. Public key cryptosystems (under certain circumstances) can help to provide non-repudiation [Tan89] (as opposed to private key). By having a user sign data using their secret key, data integrity and data origin services are provided. This coupled with ensuring that only the specified user can view the data (by encrypting it using that user's public key - only they can decrypt it with their secret key) and having them reply using some part of that message signed by them, provides two way non-repudiation. The sender cannot claim not to have sent it (as only they could have signed it), and the receiver cannot claim that it was corrupted (it was signed) nor that they didn't receive it (as they replied using the contents of the message).

2.8. Comparison of Kerberos and X.509 Strong Authentication

X.509's one-way and two-way strong authentication techniques provide basically the same services that Kerberos provides, but uses different techniques. The basic differences between the two systems are shown in Table 2.

Table 2 - Comparison between Kerberos and X.509

Kerberos	X.509 Strong Authentication
Uses private key encryption	Uses public key encryption

Requires keys to be exchanged prior to authentication	Requires a trusted third party to verify the public key of the entity seeking authentication
Private key encryption relatively "easy" and "fast" to use	Public key encryption relatively "difficult" or "expensive" to use
Key management more difficult	Key management easier
Requires synchronised clocks	Three-way strong authentication does not require synchronised clocks (one-way and two-way still require synchronised clocks)
Uses clocks to prevent replay attacks - a window of opportunity exists where an attack may be successful - can be reduced if the server caches previously used information	Uses clocks and random numbers to prevent replay attacks - does not provide a window of opportunity provided the server caches previously used information
Is in current use in a variety of situations	Currently not many implementations
Required to fit into an existing framework of operation - the interface is determined by pre-existing systems	Provides services that clients must determine how best to use - no preconceptions on the interface

A number of development implementations and projects using concepts from X.509 have been undertaken over the last few years, but these systems are currently not in wide use. The reader is referred to [UCL92], [OSI92] and [TAP90] for further information.

3. Security Threats in Existing Operating Systems

This section describes the types of threats that a computer system may face. Note that it simply describes the class of attack, but does not detail how to actively exploit particular examples of this type of attack. This is deliberately done in the interests of those systems that may not be fully secure.

3.1. Classifying Threats

The X.509 standard details a number of threats that computer networks may face, particularly relating to authentication exchanges. These are detailed below [ISO92].

Identity Interception - this is the threat that the identity of one or more of the users involved in some form of communication is observed for later misuse. Generally, this is not a serious problem unless the nature of the parties communicating is in the political arena (for example, the leader of a rival company is observed to be communicating with one of the top employees in another company).

Masquerade - this is the threat where one user pretends to be another user. This usually takes place during an authentication sequence. Once the user has convinced the remote person or computer that they possess a different identity, the user then inherits all the access rights of the masqueraded user. When that masqueraded user is privileged, then this type of attack can be devastating.

Replay - this is a very simple form of attack where some sequence of events or commands is observed, and then replayed in the attempt to trick the user or machine into performing some act. This is mostly seen during authentication, by replaying another user's username and password, thus ultimately performing a masquerade attack. This is a special form of masquerade attack as it is the most common form in use; hence it is in a category of its own.

Data Interception - this type of threat can be observed at any time during network communications. The threat here is that some form of user data may be observed by another user that is unauthorised to see that data. This has many implications for confidentiality, or maintaining a secret during an basic authentication sequence.

Manipulation - if data can be manipulated either during transmission, or whilst it is stored on disk, then the potential for destructive action is enormous. In this case the integrity of all the data (not just that portion which was changed) is brought into question.

Repudiation - this is the denial of one or many of the users in having participated in a communication. This is more a problem in the business world where directives or agreements need the ability to be proven to have taken place.

Denial of Service - A denial of service attack is insidious as it often conflicts with other forms of security protection mechanisms. Denial of service can be categorised as the prevention or interruption of access to a service, or the delay of time-critical operations.

Misrouting - this is where a communication intended for one person is rerouted to another person.

Traffic Analysis - this is when the communication between two parties is observed, and information about that communication is obtained (such as absence/presence, frequency, direction, sequence, type, amount, and so on). This might be of importance for example to the military or government, particularly in an international arena.

The next section examines different types of attacks that have occurred in the past, and details how they they may be categorised using the above criteria.

3.2. Characteristics of Common Threats

3.2.1. Morris Worm

"*On the evening of 2nd November 1988, a program was executed on one or more hosts connected to the Internet. This program collected host, network, and user information, then broke into other machines using vulnerabilities present in existing software. After breaking in, the program would replicate itself, and the replica would also attempt to infect other machines. The program spread quickly, and the scope of the breakins came as a great surprise to almost everyone.*" [Spa88].

The "worm" exposed some security flaws in standard system services. These flaws are now patched, with the patches widely available. The first flaw was a bug in the `fingerd` daemon. This program made a call to the `gets()` routine. `gets()` makes no checks on the bounds of the buffer involved, and hence it is possible to overflow that buffer (`scanf()`, `sscanf()`, `fscanf()`, and `sprintf()` also have this problem). The particular problem in question allowed the input buffer to be overflowed thus by careful selection of the input data, a privileged login shell could be obtained and any commands executed.

A simple change to prevent unbounded input was all that was required. This particular vulnerability is a typical example of poor coding practices. When a program is to be executed by a privileged or an unauthenticated user, extreme care must be taken to minimise any damage as a result of error conditions or extraneous events[8]. In this case, the buffer overflow was not detected and a security risk was the result.

The second vulnerability exploited by the Morris Worm was in `sendmail`. `sendmail` is a mailer designed to route mail in a heterogeneous network. It is extremely large and complex. Configuring the sendmail system is a long and difficult task. Therefore, the sendmail developers allowed a DEBUG feature to be turned on to aid with configuring the system. Unfortunately, it was such a useful feature that it was often left turned on by default. When DEBUG was enabled, it allowed a user to issue a set of commands instead of the user's address as the recipient

[8]This does not mean that a program be executed by an unprivileged, authenticated user does not also require the same level of care!

of a message. Since `sendmail` was running as a privileged program, this allowed those commands to operate in a privileged environment.

This is a classic example of poor configuration, particularly relating to extremely complex computer subsystems. Despite this being a well documented problem, automated attacks targeting this vulnerability are still being seen today, suggesting that perhaps some limited success is still being found using this mechanism [CER93].

The key attack of the worm involved attempts to discover user passwords. Since this is such a common problem, it is addressed as a separate issue in a later section.

Simple precautions would have prevented the rapid and widespread deployment of the Morris Worm. However in 1988, computer security did not have the high profile that it has today.

The Morris Worm was aimed at Unix based systems, specifically those based around the BSD variants. The next section shows that this type of attack is not a Unix-only problem.

3.2.2. WANK and OILZ Worms

The following analysis is drawn from [LS93].

During October and November of 1989, a different work attacked the SPAN and HEPnet networks, targeting VMS systems linked via DECnet. Whenever a system was infected with WANK or OILZ, the damage ranged from annoying users with messages and practical jokes to more major damage in terms of Trojan Horses and penetrated system accounts. After about two weeks, the attacks by WANK subsided. However, a new version of this worm, OILZ, attacked hundreds of additional systems. OILZ fixed a number of errors and problems that were present in WANK.

The WANK worm would use the network to examine remote systems with a view of learning what some usernames were on those systems. It would then attempt to guess a password for those accounts, initially only trying a small subset of passwords to help prevent detection. The worm would determine if it had penetrated a privileged account and further invade the system if possible.

The OILZ worm used the same techniques (correcting a few errors in the WANK worm), but also used the DECnet default account for penetration. This allowed the worm to spread to hundreds of additional systems.

Once the worm had penetrated the system using a privileged account, it would install back doors back into the system by editing any command file it could find to modify the password and privileges of a set account each time that command file was executed, it would disable mail to the system account (probably to prevent any notification of eradication scripts or other information), and it would modify user accounts to play a practical joke (which resulted in users deleting their own data).

These worms exploited standard attacks using already known vulnerabilities. The attacks that allowed the worms to propagate were the result of poor password choices, and default user accounts. Once again, it highlights that this is an extremely important security mechanism.

Similar worms have existed in the past which also exploited poor password choices and vendor supplied default passwords (such as the Father Christmas worm in 1988 [Cly93]).

A formal definition of worms can be found in [Coh92].

3.2.3. Trusted Systems

The discussion on this type of attack relates specifically around Unix, although it could easily apply to other Operating Systems (such as OpenVMS using the Network Proxy facility [Cly93] and DECnet [DEC88a]). Using a trusted system attack, many systems are compromised by using a trusted system on their local network that has itself been compromised [Cur90]. Turning this around, it means that once one system is compromised at a site, it is usually very easy to compromise many further systems at that site. There are a number of facilities that enable this to happen. The first is the file `/etc/hosts.equiv`. This file contains a list of all known hosts that this system trusts. In the past, many versions of the operating system were shipped by default as trusting all hosts on the global network. This was usually done as many of the workstations were being installed on closed networks, and not connected to the global network. It resulted in a system that was easy to install and get running in the local network. In the global network, this type of assumption is entirely inappropriate.

Another aspect of trusted systems attacks involves the use of the `.rhosts` file. This file works much the same as the `hosts.equiv` file, but only acts for specific users, rather than all users. The unfortunate part of this situation is that the file is entirely under the control of the user. This means that any individual user may lower the security of a system by allowing access to their account via any other system on the network.

The `.rhosts` file is used to grant access via the "r" commands (`rlogin`, `rcp`, `rsh`, `rexec`, `rdist`, and so on) [Sun90b]. However there are further vulnerabilities concerning these commands that have been discovered and rectified in the past. It was possible that the protocol exchange could be exploited to gain unauthorised access. This has the problem that the detection of this situation is usually out of the hands of the system administrator, and the system administrator now relies on third parties (such as an Incident Response Team) to detect and take responsibility for organising a software fix, and then disseminating the information to the sites.

The `.rhosts` file is used to identify the trusted system, by name. When all the host name to host address mappings were stored on the local system (for example, in the `/etc/hosts` file), this worked reasonably well. Now that these mappings are being performed by name servers that are usually out of the control of the local system administrator, it is possible to change the address to name binding to fool a system into thinking that the connection is coming from a trusted host [Ven92]. This represents a fundamental flaw in the design of these "r" commands.

The final style of vulnerability that falls under a trusted systems attack comes from inappropriate setting of the "secure" attribute on terminals [CER92]. This allows anyone using that terminal to login as "`root`" (the system privileged account on Unix). If the terminal is not set to `secure`, then the user must login on their own account first, and then change their identity to `root`, thus passing through more security checks and audit trails. Some operating systems by default had all of their terminals set to secure as it helped the system administrators when initially setting up their systems. Once again, this highlights the need for sensible default configurations when installing new software.

3.2.4. tftp

`tftp` stands for trivial file transfer protocol [Sun90b], [RFC783]. This works much the same as the normal file transfer protocol, but the word trivial is used to indicate that a number of key services are not implemented (such as user authentication!). `tftp` is mainly used by diskless workstations or smart terminals for boot information, configuration loading, or font loading. The major problem with `tftp` is that if care is not taken, it is possible to configure it in an inappropriate manner, thus allowing any world readable file on the system to be copied (generally, an intruder cannot overwrite a file, but even this is still possible under extreme circumstances). A typical `tftp` attack involves an intruder taking a copy of the `/etc/passwd` file to a remote site, and then attempting to guess the passwords. It is possible to automate these attacks, resulting in some widescale attempts at accessing password files (one single incident covered 65,000 individual hosts!) [CER92].

3.2.5. NIS

Sun Microsystem's Network Information Service (NIS)[9] is a distributed database system that lets many computers share password files, group files, host tables, and other files over the network [Sun90a]. The files are stored centrally on a NIS server, and are made available to the NIS clients. The files on the clients contain a special entry (that begins with a "+") to specify that the NIS server needs to be contacted for the rest of the information. This can have some unexpected effects. For example, the `/etc/passwd` file has an entry that looks like:

```
+::0:0:::
```

On some older versions of NIS, it was possible that this line would be treated as a normal password file line if the NIS server was unavailable [GS91], [Sho93]. Therefore, logging in with an account of "+" resulted in a privileged login without requiring a password [Spa93]. Another danger is that should the "+" ever be accidentally removed or changed, the line becomes a valid password file entry, allowing a privileged login without a password.

Other problems include the ability for any user outside of the organisation to obtain copies of the database files exported by the NIS server, by guessing the name of the

[9]Formerly "Yellow Pages".

NIS domain, binding to it using the `ypset` command, and requesting the databases [Mor90]. One major vulnerability here is that the password file may be disclosed, allowing password crackers to be exploited.

There are design flaws in the code of some NIS implementations that allow a user to reconfigure and fool the NIS system [GS91]. One of the major ways that this is performed is to establish a `ypserv` program that acts like a real `ypserv` program and responds to `ypbind` requests. The `ypbind` daemon can then be instructed to use that program rather than the real `ypserv` daemon. In this way, an attacker can supply their own version of the password file (containing any information they think is appropriate!).

Another form of attack is performed by analysing the NIS data packets as they pass on the network, and providing a valid response to NIS requests [Mor90]. This requires detailed knowledge of the NIS protocol. There are techniques around that ensure that the reply is received by the client before the valid reply. Since it is a datagram protocol, once the client receives a reply, it ceases listening, and the second reply is lost. This allows an attacker to enter a computer system by supplying any username (the client doesn't know it isn't a valid username), and that user can then become privileged (despite the fact that the terminals have been set secure, or that only a subset of people are allowed to execute the `su` command!).

Many implementations of NIS have been tidied up, and a lot of these problems have been addressed. The final problem relating to NIS is a simple configuration error with the password files. The "+" entry in the `/etc/passwd` file indicates that NIS is running and should be consulted for further passwords. A common mistake is to also include a "+" line at the end of the password file that is being exported by the NIS server. This has the effect of validating the username "+" as a real user account [GS91].

3.2.6. NFS

The Network File System (NFS) allows different computers to share files over the network [Sun90a], [RFC1094]. NFS works using a client-sever model. Using NFS, clients can mount the disks on the server as if they were physically connected to themselves. Unlike other remote filesystems, NFS allows client users to read and change the contents of files stored on the server without ever having to log into the server or supply a server password. This characteristic is at the heart of NFS's security problems.

NFS can be made secure, however it is easy to incorrectly configure the system, often without even knowing. Most forms of NFS attacks take place on systems that are incorrectly configured. NFS unfortunately has a few "silent" features that may result in allowing any client on the network to mount the disk, if the configuration is not 100% correct. Inappropriate configuration allows any client to modify files on the server's disk, possibly installing malicious programs. The `showmount -e` command can be used to check a systems configuration to verify that what is set is appropriate.

Another security problem with NFS concerns file handles used to identify files [GS91]. These file handles consist of a filesystem id and an inode number, so it is relatively easy to guess valid file handles. The standard NFS server is unable to distinguish between falsified file handles from file handles that were created by the `mount` daemon. Therefore, the files on any NFS server are potentially accessible over the network by anyone who has the ability and determination to search for valid file handles.

3.2.7. Passwords

By far, the major strength (and weakness) in any system is the first line of defence - the authentication into the system. Any facility that allows *someone* access has the potential to allow *anyone* access. It is for this reason alone that special attention must be given to the authentication services. Unfortunately, this is not universally performed, and time after time, computer systems are being compromised as a result of a weak authentication service; in most cases a poor choice of password [CER92], [BKS90]. Sometimes the passwords are simply guessed, and with a small amount of knowledge it is usually easy to guess an inexperienced person's password. Mostly however, the password file is captured through some other means (perhaps `tftp`, or it may even be a local user with read access to that password file), taken to a remote site (often one that has been compromised!), and a variety of password guessing programs (password crackers) is run against it. It is a common belief that a young system intruder will never have sophisticated hardware to run password crackers on a password file. It is known that often the first thing the intruder does when compromising a system is to run a password cracker as a benchmark. If they decide the system is of sufficient speed, they will be back with a captured password file for cracking, possibly during the night hours only.[10]

Many systems support the concept of a "guest" account, generally to make the system administrator's life a little easier when they are required to generate a temporary account for someone who is temporarily visiting. Unfortunately, the passwords chosen for these guest accounts tend to be extremely poor, more so than normal user accounts. It is not uncommon for the password of a "guest" account to be "guest". This represents a major flaw in the system security as it not only allows access to the computer system, but also prevents any valid audit trails, as it can be easily argued that anyone could be using that account when it contains such an easy password to guess. [Gro93]

The final vulnerability relating to passwords is usually perpetrated by vendors and software manufacturers. Sometimes the vendor software requires an account and password to operate. Usually (always?), the choice of password for this account is arbitrary, and generally not required by any human user. Unfortunately, many vendors and manufacturers in the past tended to install a "default" password, which made their life easier. Many customers when installing products were either not aware of the problem, or never got around to rectifying it. This allowed many intruders to simply access the system with a known account and password, often with

---

[10]This information is referenced in files that contain other information that is sensitive, and hence cannot be published.

full system privileges. It is fortunate that now many vendors are becoming more security conscious, and this practice is falling into disfavour. It does still occur however. Special care must be taken after applying upgrades as well. It is possible that this vulnerability was closed by changing the account's password, but the upgrade silently set it back to a known value.

One reason that passwords remain a major source of vulnerability is that they are out of control of the system administrator. Passwords are chosen by users, and if those users are not educated on what forms a strong password, then they will tend to choose a password that is easy to remember (which is also easy to guess or crack). It is therefore not surprising that there are a large number and variety of tools which are solely aimed at improving password security (for example, [Bis92b] and [Hoo90]). Ultimately, the solution to the password problem is to make use of these tools coupled with educating users that security is a community response and they have just as much responsibility if the computer system is compromised.

3.2.8. Software Vulnerabilities

There are many techniques that a programmer must employ when writing privileged code (or at least, code that will run as the Id of another user) [Far91b]. In essence, it is important to ensure that the user cannot control the environment in which the program executes, and that all error status returns are checked and handled appropriately. Failure to do this may result in the software integrity being compromised. These techniques are not reserved for privileged code, but are generally good coding practices. They include simple tasks such as initialising the operating environment to a known state, checking the status returns of all system calls, and parsing all arguments internally and not trusting a third party to do it correctly. Much of the rest of this section is drawn from [Bis87] and [GS91].

One particular type of attack involves the IFS shell variable (Input Field Separator). This variable is used to indicate what characters separate input words to the shell. Whilst its functionality has been largely superseded, it lives on to cause unexpected results. For example, if a program calls the `system()` or `popen()` functions to execute a command, then that command is parsed by the shell first. If the user has control over the IFS environment variable, then they may cause unexpected results. A typical scenario might be if the program executes the following code:

```
system( "/bin/ls -l ");
```

If the IFS variable has been set to contain the "/" character and a malicious program called "bin" is placed in the path of the user executing the program, then that program will be executed as the shell will have parsed the line as:

```
bin ls -l
```

which executes the program `bin` (in the current path) passing two arguments `ls` and `-l`. It is for this reason that a program should not get the shell to parse command lines by using the `system()`, `popen()`,

Offending Command = 14s
Error = nametype : undefined
this name is not defined in a dictionary
Stack =
1175
2863
240
exec103 (m4 SB 14) 4272 SB 82 SB 300 2473 201887 1466 SB 3s