VICTORIA UNIVERSITY OF WELLINGTON

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

Protecting Computer Systems from Attack

Simon McAuliffe and Ewan Tempero

Technical Report CS-TR-95/7
July 25, 1995

Abstract

The operation of many organizations in New Zealand increasingly depends on computer systems. Consequently, it is important to protect such systems, particularly from deliberate attack. However security is a complex problem and requires a good understanding of the issues involved. In this paper, we present a framework for the security of computer systems. We discuss a number of familiar and not so familiar issues in computer security in the context of our framework. Our goal is to provide system administrators with a systematic approach to evaluating the security of their computer systems.

1 Introduction

The operation of many organizations in New Zealand increasingly depends on networked personal computer systems. These systems are often entrusted with information that is important to the successful operation of an organization, or information that is confidential to the organization or its clients. Consequently, the security of an organization's computer system is of great importance.

There are several parts to computer security: reliability — whether the software and hardware components are performing correctly, integrity — ensuring that no part of the system is damaged or missing, policy — how the system is managed, secrecy — hiding information from unauthorised people, and authentication — ensuring that those accessing the system are who they say they are. Security can also be divided into problems caused by accident, and problems caused by deliberate attack. In this paper, we are concerned with securing computer systems against the latter, and so will concentrate on secrecy and authentication.

Despite its importance, the security of many computer systems is inadequate. Sometimes this is due to lack of proper attention. Often there has been a conscientious effort to install appropriate security measures, but the safeguards are inadequate due to the complexity of securing the system. This paper is intended to help system administrators manage the complexity of computer security. Our goal is to discuss the issues and organize them so as to provide a systematic process to evaluating the security of computer systems.

Security does not come for free. Increasing security means increasing the resources (equipment and personnel) devoted to security. The level of security an organization needs depends on the value of the information on its system (or the cost to the organization or clients should the information be stolen). The value of the information partly dictates the level of *threat*, that is, the likelihood that an attack will take place and the effort the attacker will exert. However just spending money will not ensure good security; the best retina scanner in the world is no good if there is no protection on the modem. The security measures must completely cover all possible forms of attack for the expected level of threat.

Deciding what security measures to install requires identifying the kinds of attack that are to be prevented (see Appendix A for some examples). There is no way to list all possible attacks for every system because every system is different and even the list for a specific system would be huge. Nevertheless, creating such a list is a necessary part of system security and it is very important that it be complete. The size of the resulting list makes it difficult to be sure there are no gaps. The only way to create such a list is to do it systematically. Doing so requires a some understanding of the issues involved.

One of the goals of our work is the development of such a system. To do so, we have created a framework to help organise the issues involved. We present this framework in section 5. The remainder of this paper is devoted to summarising the main issues regarding computer security so that our framework can be applied correctly. In section 2, we discuss the main issues of developing an adequate encryption system. In section 3 we discuss the various alternatives for authentication. Ultimately, the security of a system depends on the system's physical security. Information is not very secure if an intruder can walk off with the hard disk. The best retina scanner in the world cannot prevent unauthorised access if it can be bypassed by shorting the wires between it and the system. We discuss issues associated with physical protection in section 4.

2 Secrecy

For thousands of years, people have tried to keep information secret [6]. The problem is that information must be available to be useful, and that means it must be communicated between parties, or at least stored between uses. Should either the communication or storage medium be uncontrolled in any way, then the secrecy of the information may be compromised. The most common technique used to protect the secrecy of information is *encryption*. Encryption is the process of converting information (referred to as the *plaintext*) into an incomprehensible form (referred to as the *ciphertext*). The particular method for performing encryption is the encryption algorithm, or *cipher*. *Cryptography* is the study of creating ciphers. The reverse process of encryption is *decryption*, and it is usually done using the inverse of the cipher. Discovering how to reverse the encryption without full knowledge of the cipher is referred to as *breaking* the cipher, and the study of breaking ciphers is called *cryptanalysis*. The general study of ciphers, both creating and breaking them, is called *cryptology*.

The most useful ciphers are *keyed* — that is, the ciphertext depends on some additional information, and without this, the encryption cannot easily be reversed. When both the encryption mechanism and decryption mechanism use the same key, or when the decryption key can be derived from the encryption key, the system is called a symmetric cipher. Up until recently, almost all ciphers in practical use were symmetric ciphers and this area of cryptography is therefore often referred to as *conventional cryptography*. Encryption systems have recently been developed that use different keys for encryption and decryption. These asymmetric or *public key* ciphers also offer some interesting possibilities for authentication. We explain these ciphers and discuss their benefits and problems in section 2.1.

Because obtaining the information can be done by anyone in possession of the decryption key, it is important that such keys do not fall into unfriendly hands. This is an important problem, known as *key distribution*, and we discuss some of the issues in section 2.2.

Encryption has a certain mystique attached to it, presumably because of the secrecy that usually surrounds its use. As a consequence, it is generally assumed by many people that encryption is "the answer", that once encryption is employed, there is no need to worry about the secrecy of the information. This assumption is held despite many famous examples where the encryption has been broken. Perhaps this assumption is based on the reasoning that it takes experts with very powerful computers a long time to break encryption systems. This is true for some forms of encryption, however the techniques that the experts have developed are now available to less expert people. Furthermore, there are many weak encryption systems that can be fairly easily broken, and even the best encryption system's protection can be nullified by poor management.

Interestingly, despite the perception that only experts can break encryption systems, there is also the perception that anyone can *create* an encryption system. The problem seems to be a simple matter to the encryption novice; just "muck up" the information and nobody will be able to recover the original information. This leads many organisations to have their regular programmers whip up an encryption routine to go with their software. However designing a cryptographic system is not as simple as it sounds — something seemingly unrelated can have a huge impact on security. Consequently, encryption systems developed this way always results in a product that is *not* secure. Almost all commercial word processors, databases and spreadsheets have their encryption systems built in this way and so offer no real security.

Unfortunately there is currently no practical way to truly prove the security of a cipher and the only way they are evaluated is by submitting them to extensive tests at the hands of expert cryptanalysts. This may take years, and so even those organizations who understand that creating encryption systems is not easy cannot afford to properly test their systems. Given the problems in designing ciphers, it is not generally a recommended practice, but rather to use a tried and trusted method.

Should an organization decide to develop its own cipher, here is a very incomplete list of the issues that must be dealt with. We include it more to indicate the magnitude of the task than to help assure the usefulness of the result. As we have indicated above, any development must be based on sound or tested principles.

- Novices often create algorithms based on random number generators. This type of cipher should be avoided without a good understanding of the issues involved. Special random number generators and uniqueness techniques are required and these are almost as hard to design as the ciphers themselves.
- Do not depend on the secrecy of the algorithm for security. Algorithms can be reverse engineered from their implementations (even when supplied as a "black box", such as in ROM). Once the algorithm is known (and it will become known) the entire system is compromised. The security of the cipher must depend on the secrecy of the key.
- The key size should be sufficiently large to prevent a "brute force" attack (an exhaustive key-space search). If there are n bits of key, then it will take 2^n trials to test all keys. Note that choosing certain algorithms may allow the number of trials to be reduced (because some keys are made ineffective).
- When choosing a key size, the useful lifetime of information should be taken into account. If your organisation has information that would be problematic if obtained within the next 50 years, then the cipher should be strong enough to protect it that long. What's more, the design should take into account technological advances and increasing computer power. The designer should give enough lee-way to cope with advances in cryptanalysis.
- Many ciphers apply to "blocks" of text of a fixed size. The block size should be sufficiently large so that a database of the results of encrypting a commonly occurring fragment of information with all keys cannot be cataloged.
- In situations where each block of text need not be encrypted independently from the rest, streaming, feedback and other related techniques can significantly improve security. However, care must be taken. There are many ways to implement streaming incorrectly that result in a cipher that has little or no security.
- A common technique is to create a cipher by altering an existing one. Designing a new cipher based on another cipher requires as much care as designing a cipher from scratch.
- The algorithm should be reasonably fast, otherwise it is not much use in many situations. However, security should not be traded in for speed. The cutoff point should lean towards the security side rather than the speed side, otherwise it will surely come back to haunt the users. Security doesn't come for free and a small performance loss is one of the common costs.
- The result of encryption of any information should result in output that has no statistical biases. Poor distribution or unsuspected correlation provide a lot of leverage to a cryptanalyst.
- It is very important that the cipher exhibit text *avalanche* properties. This means that any small change in the input data produces a large change in the output data. Ciphers without *key* avalanche are vulnerable to special types of attacks. Verifying that a cipher has the appropriate avalanche properties is non-trivial.

If the decision is to choose an existing cipher, then the following points must be kept in mind.

- Do not accept a commercial cipher on faith. Ideally, the system should be reviewed (at least briefly) by an expert. Many systems have already been reviewed so finding past reviews can be very valuable. Commercial systems that refuse to reveal details of the design of the system (such as its algorithm) should not be trusted.

- Keep up to date. A cipher that was once good, may not be good any more. New research may have found weaknesses in the cipher that allow it to be broken more easily or the key size and block size may be too small to keep up with the pace of technology. An example of this is the popular Data Encryption Standard (DES) [1]. For high threat situations, it is not sufficient since the speed of current technology is such that it can be broken in a reasonable amount of time.

- Do not judge a cipher on key size alone. Many commercial systems with proprietary encryption systems advertise the cipher just by the key size. They will quote breaking times in the millions or billions of years. However it does not matter how big the key is, it will be no good if the cipher is weak in some other respects, in which case the length of time it takes to try every possible key is meaningless.

2.1 Public Key Ciphers

Public key [5] cryptographic methods have recently been receiving a great deal of attention and have been hailed as the solution to the world's security problems. While it is certainly true that many new security techniques can be achieved with public key methods, contrary to popular belief, they still suffer from some serious problems.

Let us first review the principles of public key systems. In conventional cryptography, we have an encryption function, $\mathrm{E}(m, k)$, which is used to encrypt the message m with key k, and a decryption function,[1] $\mathrm{D}(c, k)$, which is used to decrypt the ciphertext c with the (same) key k, so that it must be the case that $\mathrm{D}(\mathrm{E}(m, k), k) = m$.

Anybody that encrypts a message must know k and can therefore also decrypt the message. It seems reasonable that if you can encrypt a message, then surely you should be able to decrypt it! However, consider the following example.

Suppose Alice wants to receive private messages from a number of people. She chooses a key k and gives it to her friends Bob and Eve. Now, if Bob encrypts a message using k, not only Alice can decrypt it, but so can Eve. Alice could solve the problem by giving Bob and Eve different keys, but this has the disadvantage that she has to keep a list of keys for everybody she wants to communicate with, as does anybody else involved.

Public key cryptography solves this problem more cleanly by using encryption and decryption functions that require different keys. Anyone can encrypt a message m with the encryption key, k_e, that is, $\mathrm{E}(m, k_e)$, but because of the nature of the encryption function, $\mathrm{D}(\mathrm{E}(m, k_e), k_e) \neq m$. Instead the decryption key, k_d, must be used, that is, $\mathrm{D}(\mathrm{E}(m, k_e), k_d) = m$. For this to be secure, it must not be feasible to determine k_d given k_e.

Now Alice can give her encryption key k_{eA} to Bob and Eve. Bob and Eve can encrypt messages for Alice using this key, but since they do not have k_{dA} (Alice's secret decryption key), Eve can no longer read Bob's messages to Alice.

This solution is a little nicer, since Alice doesn't have to remember a key to decrypt messages from all her friends. Of course, if she wants to be able to reply, she still needs to remember an encryption key for each of her friends. Because the encryption function is

[1] Sometimes the encryption and decryption functions are the same.

one-way, and anybody with it only has the capability to send messages, not to decipher them, a simple solution is for everybody to publish their encryption keys, making them completely public (hence the name public key cryptography). Unlike conventional cryptography, nobody other than the owner should know the decryption key, whereas everyone can know the encryption key.

Elegant as this form of encryption may seem, it is not impenetrable to attack. Here is an outline of a possible attack by Eve, trying to eavesdrop on a message from Alice to Bob. Since Alice wants to send a message to Bob, she must obtain a copy of his public key so she sends Bob some mail asking for a copy, knowing that it doesn't matter if somebody reads the mail — after all, that's the point of a public key. Now Bob sends mail back to Alice containing his public key k_{eB}. Eve realizes this and intercepts the mail before it reaches Alice, changing the mail to contain her key k_{eE}. Now, when Alice tries to send private mail to Bob, she unwittingly sends it as $\mathrm{E}(m, k_{eE})$ rather than the $\mathrm{E}(m, k_{eB})$ she had hoped for. Eve intercepts this message too, and using her decryption key k_{dE}, decrypts Alice's message, revealing all. In order to prevent Bob from becoming suspicious, Eve simply re-encrypts the message using Bob's genuine public key and sends it to Bob (addressed from Alice) and Bob will also decrypt the message. Of course, Eve could modify the message in the process too, potentially causing additional damage. This form of attack is often call *spoofing*.

This problem arose, not because the keys were made public, but because the information was *changed*. In conventional (symmetric) cryptography we had to make sure our keys were distributed without being seen by anybody, while in public key (asymmetric) cryptography, we had to make sure our keys were distributed without change. In both cases, the problem lies in *key distribution*.

2.2 Key Distribution

In many high risk situations there is no substitute for really distributing the keys yourself (for example, by trusted courier), but for most applications it is preferable to distribute keys within the system (this usually means electronically).

The full key distribution problem is the problem of secretly distributing keys while also making sure they aren't modified. This problem can be reduced just to that of the latter (that is just integrity) by using public key cryptography.

If public key cryptography is being used, the problem is already just integrity as mentioned above. To transfer a key secretly for use in a symmetric system, all that needs to be done is first of all arrange a public key system and then transmit the key using that system.

If the sender of the message (for example, when sending a public key) can be correctly identified, and the receiver is sure they are who they say they are, then there is no chance of the spoofing attack outlined earlier. To do this requires authentication of all exchanged messages.

3 Authentication

Authentication is about confirming the identity of those trying to access the system. The main techniques used are *passwords*, *tokens*, and *biometric devices*. There are situations where one computer must authenticate itself to another. For example, when a user on a workstation requests a file from the server, the server must be sure that the request really is coming from that workstation. Some authentication systems require each computer to verify which other computer it is communicating with, and so require some form of *computer-computer* authentication.

3.1 Passwords

A password is a secret shared between an authorised user and the system. Users prove their identity by confirming they know the secret associated with them. Password authentication is compromised whenever the information is no longer a secret, and there are many ways to try to learn the secret. Some methods involve trying to pry the secret from the user (such as masquerading as a system administrator and asking the user to change their password). The user must communicate with the system to confirm they know the secret, and there are a number of methods for attacking this part of the authentication mechanism (such as installing password grabbers on the terminal, or eavesdropping on the LAN). Finally, the system's record of the passwords is also subject to attack (stealing the password file to break off-line).

3.2 Tokens and Biometric devices

Tokens used for the purpose of authentication are becoming commonplace. Access cards — one form of authentication token — have been used for many years to authenticate users to a physical security system but only relatively recently have they been applied to computer security. When an organisation uses both physical and computer security, a single card can be used for authentication in both systems resulting in a clean and simple system (from the user's point of view). There are many type of access cards, each with their own merits and their own disadvantages; many of the other tokens are effectively equivalent to access cards.

Biometric devices such as fingerprint scanners, retina scanners and voice matchers are becoming quite sophisticated and affordable. Fingerprint readers have been known to cause severe problems in several instances when attackers removed fingers or hands from valid users and used those to authenticate themselves. Some modern systems require warmth and pulsing blood vessels that solves the problem to some extent. Voice matching may not yet be sophisticated enough to guarantee uniqueness and it is open to simple replay attacks (that is, recording a previous authentication attempt and then playing the recording back to the authentication system). Retina scanners are effective but they are still fairly new and commercial versions are expensive.

In addition to a security token or biometric device it is sensible to include something similar to a password that only the owner knows, such as a PIN. If a computer keyboard is available, then standard password systems are quite suitable, but if the keyboard is locked off completely until a user is authenticated, a separate PIN pad is necessary. PIN pads are vulnerable in that fingerprints usually leave traces of which keys were pressed. If there are n digits in a PIN number then there are $n! = 2 \times 3 \times \cdots \times (n-1) \times n$ total possible combinations assuming no repeated digits. There are also chemical methods that will reveal the order of the keystrokes. There are techniques that help prevent trial of these combinations, but some better approaches solve the problem completely (such as PIN pads that re-arrange the number order).

In a high threat situation, access tokens have a number of problems. Most tokens are easily and cheaply forged, and they are usually vulnerable to replays. If a small transmitter (which can be as small as a grain of rice, and cheap) is installed either in or near the token reader, an attacker can record the data digitally (including PIN or password) that can be replayed to the system to falsely authenticate the attacker.

More sophisticated systems require some form of cryptographic computation that is outside the system itself. Smart cards or smart tokens (with an embedded microprocessor in them) provide a good solution, however special techniques are still required to perform a secure authentication, otherwise the smart cards are also equivalent to conventional non-interactive access tokens. A commercial smart-token authentication system should not be considered better just because it is “smart”.

Another alternative that is popular and fairly cheap is the one-time password solution (usually based on a challenge-response system). In these systems the user typically carries a small device with a display and keypad. The user enters a PIN number into the device to enable it. To authenticate the user, the system issues a challenge (which is just a number). The user enters the number into their device that will calculate another number (generally by using a cryptographically strong keyed hash or strong encryption system). When this number is returned to the system, the system can be sure the user is who they say they are since only they had the device with the correct key in it.

3.3 Computer-Computer Authentication

There are a number of methods of performing authentication between computers, perhaps the most popular and useful today is the public key based digital signature.

Public key based digital signatures can be generated in a number of ways. With some public key systems, such as RSA [8] the same encryption techniques can also be used for signing messages. Suppose Alice wants to send Bob a signed message (so Bob can be sure it's really from Alice). She encrypts her message with her secret key k_s, giving $\mathrm{E}(m, k_s)$. As before, Alice needs to publish a copy of her public key k_p. Anybody wishing to read her message uses her public key to decrypt the message, $\mathrm{D}(\mathrm{E}(m, k_s), k_p) = m$. The message will only decrypt correctly with Alice's public key if it was encrypted with her secret key, and as mentioned earlier, there is no practical way an attacker can calculate the secret key given the public key. Therefore anybody reading the message can be sure Alice sent the message.

Detached signatures are another alternative in which anybody can read the message without having to decrypt it first, and check the signature only if they wish to. These methods are based on using cryptographically strong one-way hash functions. Such a hash function takes as input an arbitrary sized message and outputs a fixed sized block of data that depends only on the input. Given the output it is not feasible to determine any information about the original message, and it is also not feasible to generate a different message that produces the same output.

The message to be signed is hashed with a cryptographically strong one-way hash function and the result of the hash is encrypted in the same fashion as Alice's message was. Anybody wishing to verify the message originator can decrypt the encrypted data to obtain the hash value and then compare that with the result of hashing the data themselves. If the two match, the message is genuine, otherwise it is a forgery.

In some public key ciphers, such as RSA, the k_s used to sign the data can be the same as k_d, the key used for decryption of an encrypted message (so Alice could have used k_{dA} for signing messages) and the key k_p used to check the signature can be the same as k_e, the encryption key (so Alice could have used k_{eA}. This is convenient since it allows a single system to be used for both secrecy and authentication however with many systems, such use will compromise the key. In addition, many systems that can be used for secrecy cannot be directly used for authentication, and vice versa.

Now that we have a mechanism that lets us encrypt and authenticate data securely, it would seem the problem is solved. However there are still further problems, especially for computer networks that perform the encryption and authentication automatically. To authenticate a message, a computer must acquire the public key associated with the sender, which requires solving the key distribution problem, which, as we observed above, leads back to authentication. Clearly there is a circularity problem here that cannot be resolved. In short, secure key distribution cannot be achieved within the system. Furthermore, there is also the problem that, since workstations may be left unattended, the public key could be modified, allowing a similar attack to Eve's attack mentioned earlier. The only solution is that at some point in any practical system, a key's delivery and storage must be physically protected.

4 Physical Protection

To protect data (such as public keys) that must be located at each machine on a network, the data can be encrypted or stored on floppy disks that are only ever put into the machine by authorised users. This is not a complete solution for a number of reasons:

- An authorised user can obtain and modify data for future users. This could be used for gain additional privileges for example.
- Software could be installed on the hard drive that captures passwords or modifies keys, and so on. One way of solving this problem is to always boot the machine from a floppy. Another is to use diskless workstations.
- The ROM could be replaced to record passwords or modify public keys. There is no way to easily detect such modifications. Checksums and other ROM validations are useless because the ROM can also modify software intended to perform such checks since it gains control of the computer first.

Especially because of the last problem (which can be performed easily, quickly and cheaply), some kind of physical protection of the machine is needed to prevent modifications inside the computer.

This protection would typically involve interfacing computers to physical alarm systems so that if a computer is opened an alarm is activated. The problem is that the alarm communication (either over wire, or wireless) is susceptible to the same communication attacks. In a simple case, the alarm circuit can be cut or shorted to prevent the alarm from activating. More sophisticated systems such as biased circuits can still be circumvented easily with a special type of bridge circuit.

The only effective solution to communication with the alarm system is through the use of cryptography yet again. A typical system might continually encrypt a counter and communicate that to the alarm system. If an unexpected result arrives, or communication stops the alarm can activate. In this case we do not have the circularity problem of before since the protection system protects itself too.

5 Framework

The level of security of a computer system is dictated by the value of the information it contains, hence not all organizations will want to protect their systems against all forms of attack. This means their security system will contain some deliberate gaps. The trick in designing their security is to ensure that in leaving open some areas, no other areas are accidently left unguarded. We have developed a framework for evaluating the security of a system that we believe reduces the chance of such accidental gaps. It is a framework in the sense that we do not describe how to do the evaluation, but we describe how to create a procedure for the evaluation.

5.1 Information Flow

Our approach to evaluating the security of a system is based on the observation that any attack on the system must target *information flow*, that is, the movement of information between different parts of the system. This means that identifying the parts of the system that are vulnerable to attack is the same as identifying the ways that information can flow in the system. Once the focus is on information flow it quickly becomes clear why computer security is so difficult: there are many ways information can flow and some of them are not very obvious.

To identify information flows, the first step is to determine where information can reside in the system. The exact list will vary from system to system, but we give a representative list below:

- workstation
- server
- Standalone (non-networked) machine
- Peripherals (keyboard, monitor, mouse, printer, network cards, etc.)
- Cables (power, network, printer cable, telephone, etc.)
- communications (routers, multiplexers, terminal servers, modems, microwave transmitters, etc.)
- system components (disk drives, CMOS RAM, etc)
- date files (password files, key repositories, etc) (1)
- software (boot routines, device drivers, etc) (1)
- file system (1)
- authorised users (2)
- privileged users (2)
- management (2)
- paperwork (e.g., account request forms)
- documentation
- environment (3)

Notes: (1) Although data files, software, and filesystems seem similar, the kind of information they represent is different and so we must distinguish between them. (2) The people associated with the system have to be considered part of it as well. Not only do they have information about the system, they will have to be part of some of the information flows. Furthermore, some important information flows may happen between them (for example, a user requesting their password be reset). (3) The environment is needed to allow the representation of information flowing[2] outside the computer system.

The next step is to identify how information can leave each repository and where it goes. This step will probably bring to light further places where information may lurk, and so there will likely be some iteration required to ensure the list is complete. Given the number of repositories that may be present, the number of combinations will generally be quite large. At this stage, it is tempting to make decisions as to what is unimportant to system security in an effort to reduce the amount of work. Not only is this not a good idea in general (the decision may be wrong) but, as we will see below, such "unimportant" flows may become important if other vulnerabilities are exploited.

Having identified the information flows, the next step is to represent them. We have chosen to use a (directed) graph, where the nodes are the information repositories and edges

[2] Perhaps "leaking" would be a better word!

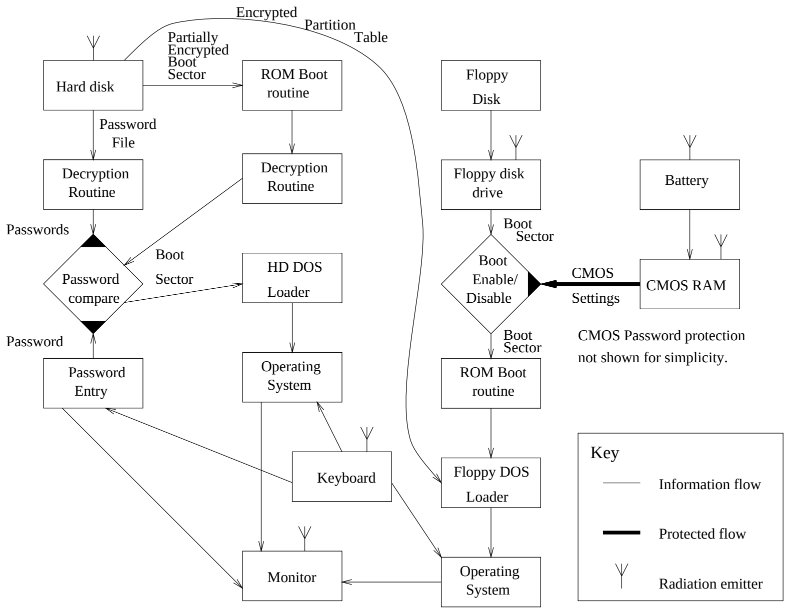

Figure 1: Information flow for a commercial PC data security product

connecting the nodes represent information flows. Nodes that are associated with authentication can enable or disable information flow and so we distinguish them in our graphs. We also annotate the edges in cases where the information is of a specific kind (such as partition tables for a disk) or in a specific form (such as encrypted).

For example, figure 1 shows a small part of the graph when applied to a personal computer system that has a security package installed for protecting the hard disk, that is, preventing unauthorised people from getting information from the disk. The authentication nodes are the diamond-shaped nodes, and the antennas indicate radiation emitters.

Once the graph has been created, every edge must be examined to determine how vulnerable that information flow is to attack. The kind of attacks that are possible include:

- Passive (reading information)
- Active (reading and changing information)
- Vandalism (corrupting information, or changing it without knowing what it was before or after the attack.)
- Denial (disrupting information flow)

Even at this stage, this analysis must be done without any making any assumptions about the likelihood of that kind of attack taking place, or the cost of protecting against that kind of attack.

If vulnerable information flows are detected, then the graph must then be reevaluated under the assumption that those vulnerabilities have been exploited. This may result in further vulnerabilities being detected and so will require further interactions until there is no change.

The final stage is evaluating how the vulnerabilities that have been detected affect the security of the system. This requires identifying the nodes that contain sensitive information (for example, password files, medical records) and those nodes that represent a threat to the system (typically the environment nodes, but some systems may also want to include some users). If there is a sequence of edges that have been labeled as vulnerable between a node representing sensitive information and a threat node, then that is a potential security hole.

1. Identify information repositories
2. Identify information flows
3. Create information flow graph
4. For each edge of the graph (information flow), determine whether it is vulnerable to attack. Repeat this step under the assumption that a vulnerability has been exploited.
5. For each node representing sensitive information, and each node representing an attacker, determine whether there is a path of vulnerable edges between them. This is a potential security hole.
6. Decide what to do about the security holes.

Figure 2: Evaluating computer security

In the example, if the system is in an insecure room, then flow **3** (which indicates that the system cannot be booted from a floppy disk) is vulnerable. If it can be interrupted (for example, by removing the battery), then it disables the authentication check and thus means that the flows labeled **4** (which represents what happens when a machine is booted from floppy disk) to proceed uninterrupted. If flow **4** is vulnerable, then flows **1** (the hard disk partition table) and **2** (other data, which is normally not available without the partition table but can often be interpreted with appropriate low-level software) become vulnerable. While **1** is protected by encryption, **2** is not, and so is also vulnerable. As a consequence, any sensitive information on the disk can reach anyone with access to the keyboard and monitor.

Once all the potential security holes have been determined, they can be evaluated as to how likely they are to attack and how expensive they are to protect. Should any new security measures be installed, the process must then be repeated. Figure 2 summarises this procedure.

6 Conclusions

We have given an overview of the issues involved in computer security for computer systems, focusing on the most common errors made in designing or managing computer security. There are a number of excellent books for readers interested in learning more [2, 3, 4, 7].

When designing a security system, it is important to understand how the system may be vulnerable to attack. We have described a framework to help make the analysis of the system more systematic. Our framework is based on a simple characterisation of attack: targeting information flows. Because this characterisation is simple, we believe it is easier to use it to identify possible security holes than techniques based on sophisticated classification techniques. Because our framework is systematic, it reduces the concern that something may have been forgotten, as is the case when an evaluation is based on subjective criteria. Our framework also has the advantage that it can be used to record both the information gathering process and the resulting decisions.

Our framework suffers from two major problems. The first is managing the huge amount of information that is required to use it. One possibly is that it appears to be amenable to computer automation, and we intend to explore this in future work. The second problem is that it is still difficult to identify information flows, so while it may benefit those with a lot of experience in computer security, it is not clear that it is useful to those with only a modest amount of experience.[3] We hope to develop better guidelines for identifying information flows as we acquire more experience with this approach.

We hope we have convinced the reader that designing computer security is not for the inexperienced. However we hope that our discussion of the issues, and in particular our framework, will help system administrators to better evaluate both existing and proposed security mechanisms for their system.

References

[1] Data Encryption Standard. FIPS PUB 46, January 1977. National Bureau of Standards, Washington, DC.

[2] Donald Watts Davies and Wyn L. Price. *Security for Computer Networks: an introduction to data security in teleprocessing and funds transfer.* J. Wiley and Sons, 2nd edition edition, 1989.

[3] D. E. R. Denning. *Cryptography and Data Security.* Addison-Wesley Publishing Company, 1982.

[4] Peter J. Denning. *Computers Under Attack: intruders, worms, and viruses.* Addison-Wesley, 1990.

[5] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions in Information Theory*, IT-22(6):644–654, November 1976.

[6] David Kahn. *The codebreakers; the story of secret writing.* Macmillan, 1967.

[7] Peter G. Neumann. *Computer Related Risks.* Addison-Wesley, 1995.

[8] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key crypto-systems. *Communications of the ACM*, 21:120–126, 1978.

Appendix A Example Threats or Attacks

- A multiuser system is put into single-user mode.
- A workstation is put into promiscuous mode to monitor keystrokes going across the LAN.
- Attacker eavesdrops on cell phone modem connection.
- Attacker finds written records (account request forms, logs of failed login attempts) containing passwords in rubbish.
- Attacker interacts with authentication system.
- Attacker intercepts data on LAN using portable computer.
- Attacker intercepts data on fibre-optic LAN using portable computer.
- Attacker is an ex-staff member whose access has not been disabled.
- Attacker masquerades as a user and asks for password to be changed.
- Attacker masquerades as operations staff and asks user to change password.
- Attacker monitors a microwave link.

[3] We do not recommend that inexperienced people carry out security evaluations.

- Attacker monitors a part of a LAN that goes off-site.
- Attacker monitors keystrokes or video signal through electromagnetic emissions ("TEMPEST" attack).
- Attacker monitors microwave link.
- Attacker monitors network through telecommunications provider's equipment.
- Attacker uses default passwords.
- Attacker watches user enter password (through a window for example).
- Authentication is based solely on machine addresses and can be attacked with forged packets.
- Authentication is bypassed by booting from a floppy disk.
- Boot software is modified to capture passwords, either before installation or via software patches.
- Bugs in security relevant code, or even seemingly irrelevant code.
- Bypass mechanisms (such as `rshd`) exploited by forging packets or various other problems.
- Code sent from boot servers can be manipulated.
- Equipment is modified to capture or transmit keystrokes or passwords.
- File systems that are mounted across the network are incorrectly set up allowing access from unauthorised terminals.
- Files on the file system have the wrong permissions, allowing them to be read and/or modified.
- Infect system with a virus that grabs keystrokes or passwords.
- Modify hardware (e.g., by replacing ROMs) to capture passwords.
- Modify hardware (for example, by replacing ROMs) to capture passwords.
- Password file is acquired (although it may be encrypted and so require further efforts), for example, from a BBS site.
- Public Information Services (such as World Wide Web) incorrectly configured.
- Public/guest logins incorrectly configured.
- Terminal/workstation software is modified to capture keystrokes.
- The LAN is monitored to observe keystrokes.
- The attack comes through a non-local network (such as the Internet). This kind of attack includes trying to log on using default or weak passwords, or forging mail or even packets.
- The attack consists of going through the organization's rubbish in the City Dump ("trashing").
- The attack consists of trying default or weak passwords, or based on the assumption that there is no protection on dialup access.
- The ciphers are weak enough that a brute force attack is possible.
- The ciphers are weak enough that they can be reversed.
- The hard disk is stolen from the server.

- The system's knowledge of the authentication system (such as the password file) is available to any privileged user.
- The system's knowledge of the passwords is available to any authorised user (whether in plaintext or not).
- The user's keystrokes are monitored using standard remote access software (such as is available through the X-windows system).
- User tells the attacker their password, or overhears user tell someone else their password, or finds user's record of password.
- User's password is name, birthdate, or word that can be found in a dictionary.
- Workstation ROMS that are involved with authentication are replaced.