MODULE 5

OPERATING SYSTEM SECURITY:

File Protection Mechanisms

Until now, we have examined approaches to protecting a general object, no matter the object's nature or type. But some protection schemes are particular to the type. To see how they work, we focus in this section on file protection. The examples we present are only representative; they do not cover all possible means of file protection on the market.

Basic Forms of Protection

We noted earlier that all multiuser operating systems must provide some minimal protection to keep one user from maliciously or inadvertently accessing or modifying the files of another. As the number of users has grown, so also has the complexity of these protection schemes.

All "None Protection

In the original IBM OS operating systems, files were by default public. Any user could read, modify, or delete a file belonging to any other user. Instead of software- or hardware-based protection, the principal protection involved trust combined with ignorance. System designers supposed that users could be trusted not to read or modify others' files, because the users would expect the same respect from others. Ignorance helped this situation, because a user could access a file only by name ; presumably users knew the names only of those files to which they had legitimate access.

However, it was acknowledged that certain system files were sensitive and that the system administrator could protect them with a password. A normal user could exercise this feature, but passwords were viewed as most valuable for protecting operating system files. Two philosophies guided password use. Sometimes, passwords were used to control all accesses (read, write, or delete), giving the system administrator complete control over all files. But at other times passwords would control only write and delete accesses , because only these two actions affected other users. In either case, the password mechanism required a system operator's intervention each time access to the file began .

However, this all-or-none protection is unacceptable for several reasons.

- Lack of trust . The assumption of trustworthy users is not necessarily justified. For systems with few users who all know each other, mutual respect might suffice; but in large systems where not every user knows every other user, there is no basis for trust.
- All or nothing . Even if a user identifies a set of trustworthy users, there is no convenient way to allow access only to them.
- Rise of timesharing . This protection scheme is more appropriate for a batch environment, in which users have little chance to interact with other users and in which users do their thinking and exploring when not interacting with the system. However, on timesharing systems, users interact with other users. Because users

choose when to execute programs, they are more likely in a timesharing environment to arrange computing tasks to be able to pass results from one program or one user to another.

- Complexity . Because (human) operator intervention is required for this file protection, operating system performance is degraded. For this reason, this type of file protection is discouraged by computing centers for all but the most sensitive data sets.
- File listings . For accounting purposes and to help users remember for what files they are responsible, various system utilities can produce a list of all files. Thus, users are not necessarily ignorant of what files reside on the system. Interactive users may try to browse through any unprotected files.

Group Protection

Because the all-or-nothing approach has so many drawbacks, researchers sought an improved way to protect files. They focused on identifying groups of users who had some common relationship. In a typical implementation, the world is divided into three classes: the user, a trusted working group associated with the user, and the rest of the users. For simplicity we can call these classes user, group, and world . This form of protection is used on some network systems and the Unix system.

All authorized users are separated into groups. A group may consist of several members working on a common project, a department, a class, or a single user. The basis for group membership is need to share . The group members have some common interest and therefore are assumed to have files to share with the other group members. In this approach, no user belongs to more than one group. (Otherwise, a member belonging to groups A and B could pass along an A file to another B group member.)

When creating a file, a user defines access rights to the file for the user, for other members of the same group, and for all other users in general. Typically, the choices for access rights are a limited set, such as {read, write, execute, delete}. For a particular file, a user might declare read-only access to the general world, read and write access to the group, and all rights to the user. This approach would be suitable for a paper being developed by a group, whereby the different members of the group might modify sections being written within the group. The paper itself should be available for people outside the group to review but not change.

A key advantage of the group protection approach is its ease of implementation. A user is recognized by two identifiers (usually numbers): a user ID and a group ID. These identifiers are stored in the file directory entry for each file and are obtained by the operating system when a user logs in. Therefore, the operating system can easily check whether a proposed access to a file is requested from someone whose group ID matches the group ID for the file to be accessed.

Although this protection scheme overcomes some of the shortcomings of the all-or-nothing scheme, it introduces some new difficulties of its own.

- Group affiliation . A single user cannot belong to two groups. Suppose Tom belongs to one group with Ann and to a second group with Bill. If Tom indicates that a file is to be readable by the group, to which group(s) does this permission refer? Suppose a file of Ann's is readable by the group; does Bill have access to it?

These ambiguities are most simply resolved by declaring that every user belongs to exactly one group. (This restriction does not mean that all users belong to the same group.)

- Multiple personalities . To overcome the one-person one-group restriction, certain people might obtain multiple accounts, permitting them, in effect, to be multiple users. This hole in the protection approach leads to new problems, because a single person can be only one user at a time. To see how problems arise, suppose Tom obtains two accounts, thereby becoming Tom1 in a group with Ann and Tom2 in a group with Bill. Tom1 is not in the same group as Tom2, so any files, programs, or aids developed under the Tom1 account can be available to Tom2 only if they are available to the entire world. Multiple personalities lead to a proliferation of accounts, redundant files, limited protection for files of general interest, and inconvenience to users.
- All groups . To avoid multiple personalities, the system administrator may decide that Tom should have access to all his files any time he is active. This solution puts the responsibility on Tom to control with whom he shares what things. For example, he may be in Group1 with Ann and Group2 with Bill. He creates a Group1 file to share with Ann. But if he is active in Group2 the next time he is logged in, he still sees the Group1 file and may not realize that it is not accessible to Bill, too.
- Limited sharing . Files can be shared only within groups or with the world. Users want to be able to identify sharing partners for a file on a per-file basis, for example, sharing one file with ten people and another file with twenty others.

Single Permissions

In spite of their drawbacks, the file protection schemes we have described are relatively simple and straightforward. The simplicity of implementing them suggests other easy-to-manage methods that provide finer degrees of security while associating permission with a single file.

Password or Other Token

We can apply a simplified form of password protection to file protection by allowing a user to assign a password to a file. User accesses are limited to those who can supply the correct password at the time the file is opened. The password can be required for any access or only for modifications (write access).

Password access creates for a user the effect of having a different "group" for every file. However, file passwords suffer from difficulties similar to those of authentication passwords:

- Loss . Depending on how the passwords are implemented, it is possible that no one will be able to replace a lost or forgotten password. The operators or system administrators can certainly intervene and unprotect or assign a particular password, but often they cannot determine what password a user has assigned; if the user loses the password, a new one must be assigned.
- Use . Supplying a password for each access to a file can be inconvenient and time consuming.
- Disclosure . If a password is disclosed to an unauthorized individual, the file becomes immediately accessible. If the user then changes the password to

reprotect the file, all the other legitimate users must be informed of the new password because their old password will fail.
- Revocation . To revoke one user's access right to a file, someone must change the password, thereby causing the same problems as disclosure.

Temporary Acquired Permission

The Unix operating system provides an interesting permission scheme based on a three-level user “group “world hierarchy. The Unix designers added a permission called set userid (suid) . If this protection is set for a file to be executed, the protection level is that of the file's owner , not the executor . To see how it works, suppose Tom owns a file and allows Ann to execute it with suid . When Ann executes the file, she has the protection rights of Tom, not of herself.

This peculiar-sounding permission has a useful application. It permits a user to establish data files to which access is allowed only through specified procedures.

For example, suppose you want to establish a computerized dating service that manipulates a database of people available on particular nights. Sue might be interested in a date for Saturday, but she might have already refused a request from Jeff, saying she had other plans. Sue instructs the service not to reveal to Jeff that she is available. To use the service, Sue, Jeff, and others must be able to read and write (at least indirectly) the file to determine who is available or to post their availability. But if Jeff can read the file directly, he would find that Sue has lied. Therefore, your dating service must force Sue and Jeff (and all others) to access this file only through an access program that would screen the data Jeff obtains. But if the file access is limited to read and write by you as its owner, Sue and Jeff will never be able to enter data into it.

The solution is the Unix SUID protection. You create the database file, giving only you access permission. You also write the program that is to access the database, and save it with the SUID protection. Then, when Jeff executes your program, he temporarily acquires your access permission, but only during execution of the program. Jeff never has direct access to the file because your program will do the actual file access. When Jeff exits from your program, he regains his own access rights and loses yours. Thus, your program can access the file, but the program must display to Jeff only the data Jeff is allowed to see.

This mechanism is convenient for system functions that general users should be able to perform only in a prescribed way. For example, only the system should be able to modify the file of users' passwords, but individual users should be able to change their own passwords any time they wish. With the SUID feature, a password change program can be owned by the system, which will therefore have full access to the system password table. The program to change passwords also has SUID protection, so that when a normal user executes it, the program can modify the password file in a carefully constrained way on behalf of the user.

Per-Object and Per-User Protection

The primary limitation of these file protection schemes is the ability to create meaningful groups of related users who should have similar access to one or more data sets. The access control lists or access control matrices described earlier provide very flexible

protection. Their disadvantage is for the user who wants to allow access to many users and to many different data sets; such a user must still specify each data set to be accessed by each user. As a new user is added, that user's special access rights must be specified by all appropriate users.

MODELS OF SECURITY-Bell-LaPadula Model

The Bell-LaPadula model is a classical model used to define access control. The model is based on a military-style classification system (Bishop). With a military model, the sole goal is to prevent information from being leaked to those who are not privileged to access the information. The Bell-LaPadula was developed at the Mitre Corporation, a government funded organization, in the 1970's (Cohen).

The Bell-LaPadula is an information flow security model because it prevents information to flow from a higher security level to a lower security level. The Bell-LaPadula model is based around two main rules: the simple security property and the star property. The simple security property states that a subject can read an object if the object's classification is less than or equal to the subject's clearance level. The simple security property prevents subjects from reading more privileged data. The star property states that a subject can write to an object, if the subject's clearance level is less than or equal to the object's classification level. What the star property essentially does is it prevents the

lowering of the classification level of an object. The properties of the Bell-LaPadula model are commonly referred to as "no read up" and "no write down", respectively.

The Bell La-Padula model is not flawless. Specifically, the model does not deal with the integrity of data. It is possible for a lower level subject to write to a higher classified object. Because of these short comings, the Biba model was created. The Biba model in turn is deeply rooted in the Bell La-Padula model.

Integrity

Integrity deals with the correctness of data. According to Matt Bishop, "integrity refers to the trustworthiness of data or resources, and it is usually phrased in terms of preventing improper or authorized change". Bishop also gives the following goals of integrity:

1. Preventing unauthorized users from making modifications to data or programs.
2. Preventing authorized users from making improper or unauthorized modifications.
3. Maintaining internal and external consistency of data and programs.

A military model such as the Bell-LaPadula works well in some environments and not so well in others. In a commercial environment, the integrity of the data is often more important than who can actually view it. What good is it to store large amounts of data in a database if that data is not correct? The data in a database is virtually worthless if its integrity is not maintained. For instance, a bank would require the strict enforcement of integrity. It would be catastrophic if the customers of a bank were able to make unauthorized alterations to an account. They would be able to credit their accounts with funds that would be unaccounted for. There are numerous other examples of how crucial it is that the integrity of data is maintained. In many cases, integrity is more important than confidentiality.

Various security models have been created to enforce integrity. Some of the more popular models that have been proposed to enforce integrity are Biba Model, Lipner's Integrity Matrix Model, and Clark-Wilson Model. Each of these models takes a different approach to supporting integrity.

Biba Model

The Biba integrity model was published in 1977 at the Mitre Corporation, one year after the Bell La-Padula model (Cohen). As stated before, the Bell-LaPadula model gurantees confidentiality of data but not its integrity. As a result, Biba created a model to address the need of enforcing integrity in a computer system. The Biba model proposed a group of integrity policies that can be used. So, the Biba model is acutally family of different integrity policies. Each of the policies uses different conditions to ensure information integrity (Castano). The Biba model, in turn, uses both discretionary and non-discretionary policies.

The Bell La-Padula model uses labels to give subjects clearance levels and objects classification levels. Similarly, the Biba model also uses labels to define security, but it takes a different approach. The Biba model uses labels to give integrity levels to the subjects and objects. The data marked with a high level of integrity will be more accurate and reliable than data labeled with a low integrity level. The integrity levels are in turn used to prohibit the modification of data.

Labels

In a computer system there are a set of subjects and a set of objects. Subjects are the active components in the system such as processes created by the users. On the other hand, objects are a set of protected entities in the system such as files. The Biba model requires that each subject and object is given an integrity label. The labels by themselves do not provide protection to data. The labels must be complemented with a security mechanism in order to provide protection (RFC 1457). Usually the level of a security label remains constant, but there are exceptions to this rule; some of the policies in the Biba model support dynamic labels. The Biba model can use both static and dynamic labels. Dynamic labels allow the integrity levels to vary.

An integrity label consists of two parts, a classification and a set of categories. The classification of integrity forms a hierarchical set. For example, the classification could be crucial, important, and insignificant. Crucial would have the highest classification level and insignificant would have the lowest classification level. The name for the classification levels can be what ever is chosen for the implementation as long as it is consistent. In this case: crucial > important > insignificant.

The second part of the label will consist of a set of categories also known as a compartment. The set of categories contained in the label will be a subset of all the sets in the system. The classification of the set of categories is non-hierarchical. An example of two categories are category X = {Detroit, Chicago, New York} and category Y = {Detroit, Chicago}. In this case $X \geq Y$ (X dominates Y), because Y is a subset of X. If there were a third compartment Z containing {Detroit, Chicago, Miami}, compartment Z and X in this case are non-comparable because the third element in the sets are different (Frost).

Each integrity level will be represented as L = (C, S) where L is the integrity level, C is the classification and S is the set of categories. The integrity levels then form a dominance relationship. For instance, integrity level $L_1 = (C_1, S_1)$ dominates ($\geq$) integrity level $L_2 = (C_2, S_2)$ if and only if this relationship is satisfied: $C_1 \geq C_2$ and $S_1 \supseteq S_2$ (Castano).

Integrity labels tell the degree of confidence that may be placed in the data (RFC 1457). The confidence placed in data never increases but it is possible for the integrity level of an object to decrease. For example, if a data of high integrity level is sent across a network that has a low integrity, the trust in the data will not be the same as before it was sent across the network. The integrity level of the data could be lowered as a result of

this. The data in this instance would then be relabeled to a lower classification level. The Biba model has a number of low-watermark policies that enforce this dynamic labeling.

In the case of a networked system, each system in the network must support the use of integrity labels. If some systems in the network do not support integrity labels there is a loss of confidence in the integrity of the data. Currently, there are no network protocols that support integrity labeling (RFC 1457). This creates a problem of using integrity labels in a network environment.

Access Modes

The Biba Model consists of group access modes. The access modes are similar to those used in other models, although the may use different terms to define them. The access modes that the Biba model supports are:

1. **Modify**: allows a subject to write to an object. This mode is similar to the write mode in other models.
2. **Observe**: allows a subject to read an object. This command is synonyms with the read command of other models.
3. **Invoke**: allows a subject to communicate with another subject.
4. **Execute**: allows a subject to execute an object. The command essentially allows a subject to execute a program which is the object.

Policies Supported by the Biba Model

The Biba model can be divided into two types of policies, those that are mandatory and those that are discretionary. Within these two divisions, there are a number of policies that can be selected based on the security needs. Most literature on the Biba model refers to the model as being the Strict Integrity Policy, although there are a number of other policies that can be used in the model.

Mandatory Policies:

1. Strict Integrity Policy
2. Low-Water-Mark Policy for Subjects 3. Low-Water-Mark Policy for Objects
4. Low-Water-Mark Integrity Audit Policy
5. Ring Policy

Discretionary Policies:

1. Access Control Lists
2. Object Hierarchy
3. Ring

Mandatory Biba Policies

The Strict Integrity Policy is the first part of the Biba model. The policy states:

1. Simple Integrity Condition: $s \in S$ can observe $o \in O$ if and only if $i(s) \leq i(o)$.

2. Integrity Star Property: $s \in S$ can modify to $o \in O$ if and only if $i(o) \leq i(s)$. 3. Invocation Property: $s_1 \in S$ can invoke $s_2 \in S$ if and only if $i(s_2) \leq i(s_1)$.

The Strict Integrity Policy is the most popular policy in the Biba model. The first part of the policy is known as the simple integrity property. The property states that a subject may observe an object only if the integrity level of the subject is less than the integrity level of the object. The second rule of the strict integrity property is the integrity star property. This property states that a subject can write to an object only if the objects integrity level is less than or equal to the subject's level. This rule prevents a subject from writing to a more trusted object. The last rule is the invocation property, which states that a subject s_1 can only invoke another subject s_2, if s_2 has a lower integrity level than s_1.

The strict integrity policy enforces "no write-up" and "no read-down" on the data in the system, which is the opposite of the Bell-LaPadula model. This policy restricts the contamination of data at higher level, since a subject is only allowed to modify data at their level or a low level. The "no write up" is essential since it limits the damage that can be done by malicious objects in the system. For instance, "no write up" limits the amount of damage that is done by a Trojan horse in the system. If the malicious code was hidden in a subject it would only be able to write to objects at its integrity level or lower. This is important because it limits the damage that can be done to the operating system. The "no read down" prevents a trusted subject from being contaminated by a less trusted object.

The strict integrity property is the most restrict of the policies that make up the Biba model. The strict integrity property succeeds at enforcing integrity in a system, but it is not without its weaknesses. Specifically, the strict integrity property restricts the reading of lower level objects which may be too restrictive in some cases. To combat this problem, Biba devised a number of dynamic integrity polices that would allow trusted subjects access to an un-trusted objects or subjects. Biba implemented these in a number of different low-water mark policies.

The low-watermark policy for subjects is the second part of the Biba model. The policy states:

1. Integrity Star Property: $s \in S$ can modify $o \in O$ if and only if $i(o) \leq i(s)$.
2. If $s \in S$ examines $o \in O$ the $i'(s) = min(i(s),i(o))$, where $i'(s)$ is the subjects integrity level after the read.
3. Invocation Property: $s_1 \in S$ can invoke $s_2 \in S$ if and only if $i(s_2) \leq i(s_1)$.

What the low-watermark policy for subjects in essence does is lowers the integrity level of the subject to the lowest integrity level of the subject and object involved. The first rule of this policy is integrity star property which enforces "no write up". This prevents the modification of more trusted objects. The second rule of the policy states that if a subject is to read a less trusted object, the integrity level of the subject will drop to that of object. This prevents an object from contaminating a subject because the subject's integrity level will be reduced to that of the object. Last, the low-watermark policy for subject uses the invocation property.

The low-watermark policy for subjects is a dynamic policy because it lowers the integrity level of a subject based on the observations of objects. This policy is not without its problems. One problem with this policy is if a subject observes a lower integrity object it will drop the subject's integrity level. Then, if the subject needs to legitimately observe another object it may not be able to do so because the subject's integrity level has been lowered. Depending on the times of read requests by the subject, to observe the objects, a denial of service could develop.

The low-watermark policy for objects is the third part of the Biba model. This policy is similar to the low-watermark policy for subject. The policy states:

1. $s \in S$ can modify any $o \in O$ regardless of integrity level.
2. If $s \in S$ observe $o \in O$ the $i'(o) = min(i(s),i(o))$, where $i'(o)$ is the objects integrity level after it is modified.

This policy allows any subject to modify any object. The objects integrity level is then lowered if the subject's integrity level is less than the objects. This policy is also dynamic because the integrity levels of the objects in the system are changed based on what subjects modify them. This policy does nothing to prevent an un-trusted subject from modifying a trusted object. This policy is not very practical. The policy provides no real protection in a system, but lowers the trust placed in the objects. If a malicious program was inserted into the computer system, it could modify any object in the system. The result would be to lower the integrity level of the infected object. It is possible with this policy that, overtime, there will be no more trusted objects in the system because their integrity level has been lowered by subjects modifying them.

The low-watermark integrity audit policy is the fourth mandatory policy under the Biba model. The policy states:

1. $s \in S$ can modify any $o \in O$, regardless of integrity levels.
2. If a subject modifies a higher level object the transaction is recorded in an audit log.

This policy is similar to the low-watermark policy for objects. Like the low-watermark policy for objects it does nothing to prevent the improper modification of an object. The low-watermark integrity audit policy simply records that an improper modification has taken place. The audit log must then be examined to determine the cause of the improper modification. The drawback to this policy is that it does nothing to prevent an improper modification of an object to occur.

The Ring Policy is the last mandatory policy in the Biba Model. This policy is not dynamic like the first three policies. Integrity labels used for the ring policy are fixed, similar to those in the strict integrity policy. The Ring Policy states:

1. Any subject can observe any object, regardless of integrity levels.
2. Integrity Star Property: $s \in S$ can modify $o \in O$ if and only if $i(o) \leq i(s)$.

3. Invocation Property: $s_1 \in S$ can invoke $s_2 \in S$ if and only if $i(s_2) \leq i(s_1)$.

The Ring Policy allows any subject to observe any object. The policy is only concerned with direct modification. A subject can write to an object only if the integrity level of object is less than or equal to the integrity level of the subject, which is the integrity star property. The last part of the ring policy is the invocation property.

The ring policy is not perfect; it allows improper modifications to take place. A subject can read a low level subject, and then modifies the data observed at its integrity level (Castano). An example of this would be, a user reading a less trusted object, then remembers the data they read and then, at a later time, writing that data to an object at their integrity level. The ring policy allows indirect modification of trusted data.

Discretionary Biba Policies

The Biba model has a number of discretionary policies. These policies are not used a much as the mandatory policies in the Biba model. Most literature on the Biba model does not even mention the discretionary policies. The book, entitled Database Security, gives three discretionary policies that make up the Biba model. The first discretionary policy is an access control list, which can be used to determine which subjects can access which objects. The access control list can then be modified by the subjects with the correct privileges. Second, integrity can be enforced by using an object's hierarchy. With this method, there is root and objects that are ancestors to the root. To access a particular object, the subject must have the observe privileges to that objects and all the other ancestor objects all the way up to the root. Last, discretionary policy is the ring policy, which numbers the rings in the system with the lower number being a higher-privilege. The access modes of the subject must fall within a certain range of values to be permitted to access an object. These are the three discretionary policies that make up the Biba model. These policies are not as common as the mandatory policies. Most literature on the Biba model fails to mention that the model contains these discretionary policies.

Current Implementations of the Biba Model

A theoretical model is of little use if it cannot be implemented. One instance of where the Biba model is currently used is in FreeBSD 5.0. The TrustedBSD MAC framework is a new kernel security framework in FreeBSD 5.0 (Watson). The kernel of FreeBSD can support MAC and various other security models which are part of the TrustedBSD MAC framework. To use this option the kernel must be properly configured by adding "options MAC" to the kernel configuration. Depending on the policies selected, different configurations are used. For instances, policies relying on file system or other labels may require a configuration step that involves assigning initial labels to system objects or creating a policy configuration file (Watson).

The TrustedBSD project included a number of different policies which it supports. Each of the policies is part of a separate model that can be loaded into the kernel by supplying the correct kernel option for that module. Some of the Policies that are part of TrustedBSD are:

- Biba Integrity Policy
- File System Firewall Policy
- Low-Watermark Mandatory Access Control
- Multi-Level Security Policy
- MAC Framework Test Policy

The Biba Integrity Policy is in the module mac_biba.ko. The Biba policy provides for hierarchical and non-hierchial labeling of all system objects with integrity data and the strict enforcement of information flow policy to prevent the corruption of high integrity subjects by low integrity subjects (Watson).

Advantages & Disadvantages

There are a number of benefits that come from using the Biba model. The first benefit of the model is that it is fairly easy to implement. It is no harder to implement the strict integrity policy in the Biba model, compared to the Bell-LaPadula model. Another advantage is that the Biba model provides a number of different policies that can be selected based on need. If the strict integrity property is too restricting, one of the dynamic policies could be used in its place.

The Biba model is not without its drawbacks. The first problem with this model is selecting the right policy to implement. The model gives a number of different policies that can be used. On one hand, it provides more flexibility and, on the other hand, the large number of polices can make it hard to select the right policy. Another problem is the model does nothing to enforce confidentiality. For this reason, the Biba model should be combined with another model. A model such as the Bell-LaPadula could be used to complement it. The Lipner model is one such model that has been developed to meet these requirements; it, in turn, combines both the Bell-LaPadula and Biba models together. Also, the Biba model doesn't support the granting and revocation of authorization.

SYSTEM SECURITY

INTRUDERS

One of the two most publicized threats to security is the intruder (the other is viruses), generally referred to as a hacker or cracker. In an important early study of intrusion, Anderson [ANDE80] identified three classes of intruders:

- **Masquerader:** An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account

- **Misfeasor:** A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges.
- **Clandestine user:** An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection

The masquerader is likely to be an outsider; the misfeasor generally is an insider; and the clandestine user can be either an outsider or an insider. Intruder attacks range from the benign to the serious. At the benign end of the scale, there are many people who simply wish to explore Internets and see what is out there. At the serious end are individuals who are attempting to read privileged data, perform unauthorized modifications to data, or disrupt the system.

Attempts to copy the password file (discussed later) at a rate exceeding once every other day Suspicious remote procedure call (RPC) requests at a rate exceeding once per week .Attempts to connect to nonexistent "bait" machines at least every two weeks. Benign intruders might be tolerable, although they do consume resources and may slow performance for legitimate users. However, there is no way in advance to know whether an intruder will be benign or malign. Consequently, even for systems with no particularly sensitive resources, there is a motivation to control this problem.

An analysis of this attack revealed that there were actually two levels of hackers. The high level were sophisticated users with a thorough knowledge of the technology; the low level were the "foot soldiers" who merely used the supplied cracking programs with little understanding of how they worked. This teamwork combined the two most serious weapons in the intruder armory: sophisticated knowledge of how to intrude and a willingness to spend countless hours "turning doorknobs" to probe for weaknesses. One of the results of the growing awareness of the intruder problem has been the establishment of a number of computer emergency response teams (CERTs). These cooperative ventures collect information about system vulnerabilities and disseminate it to systems managers. Unfortunately, hackers can also gain access to CERT reports. In the Texas A&M incident, later analysis showed that the hackers had developed programs to test the attacked machines for virtually every vulnerability that had been announced by CERT. If even one machine had failed to respond promptly to a CERT advisory, it was wide open to such attacks. In addition to running password-cracking programs, the intruders attempted

to modify login software to enable them to capture passwords of users logging on to systems. This made it possible for them to build up an impressive collection of compromised passwords, which was made available on the bulletin board set up on one of the victim's own machines.

INTRUSION TECHNIQUES

The objective of the intruder is to gain access to a system or to increase the range of privileges accessible on a system. Generally, this requires the intruder to acquire information that should have been protected. In some cases, this information is in the form of a user password. With knowledge of some other user's password, an intruder can log in to a system and exercise all the privileges accorded to the legitimate user. Typically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no protection, then it is an easy matter to gain access to it and learn passwords. The password file can be protected in one of two ways:

- **One-way function:** The system stores only the value of a function based on the user's password. When the user presents a password, the system transforms that password and compares it with the stored value. In practice, the system usually performs a one-way transformation (not reversible) in which the password is used to generate a key for the one-way function and in which a fixed-length output is produced.

- **Access control:** Access to the password file is limited to one or a very few accounts.

  If one or both of these countermeasures are in place, some effort is needed for a potential intruder to learn passwords. On the basis of a survey of the literature and interviews with a number of password crackers, reports the following techniques for learning passwords:

1. Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.

2. Exhaustively try all short passwords (those of one to three characters).Try words in the system's online dictionary or a list of likely passwords. Examples of the latter are readily available on hacker bulletin boards.

3.Collect information about users, such as their full names, the names of their spouse and children, pictures in their office, and books in their office that are related to hobbies.

4. Try users' phone numbers, Social Security numbers, and room numbers.

5. Try all legitimate license plate numbers for this state.

6. Use a Trojan horse to bypass restrictions on access.

7. Tap the line between a remote user and the host system.

The first six methods are various ways of guessing a password. If an intruder has to verify the guess by attempting to log in, it is a tedious and easily countered means of attack. For example, a system can simply reject any login after three password attempts, thus requiring the intruder to reconnect to the host to try again. Under these circumstances, it is not practical to try more than a handful of passwords. However, the intruder is unlikely to try such crude methods. For example, if an intruder can gain access with a low level of privileges to an encrypted password file, then the strategy would be to capture that file and then use the encryption mechanism of that particular system at leisure until a valid password that provided greater privileges was discovered. Guessing attacks are feasible, and indeed highly effective, when a large number of guesses can be attempted automatically and each guess verified, without the guessing process being detectable. Later in this chapter, we have much to say about thwarting guessing attacks.

The seventh method of attack listed earlier, the Trojan horse, can be particularly difficult to counter. A low-privilege user produced a game program and invited the system operator to use it in his or her spare time. The program did indeed play a game, but in the background it also contained code to copy the password file, which was unencrypted but access protected, into the user's file. Because the game was running under the operator's high-privilege mode, it was able to gain access to the password file. The eighth attack listed, line tapping, is a matter of physical security. It can be countered with link encryption techniques, discussed in Other intrusion techniques do not require learning a password. Intruders can get access to a system by exploiting attacks such as buffer overflows on a program that runs with certain privileges. Privilege escalation can be done this way as well. **Detection** is concerned with learning of an attack, either before or after its success. **Prevention** is a challenging security goal and an uphill battle at all times. The difficulty stems from the fact that the defender must attempt to thwart all possible attacks, whereas the attacker is free to try to find the weakest link in the defense chain and attack at that point.

INTRUSION DETECTION

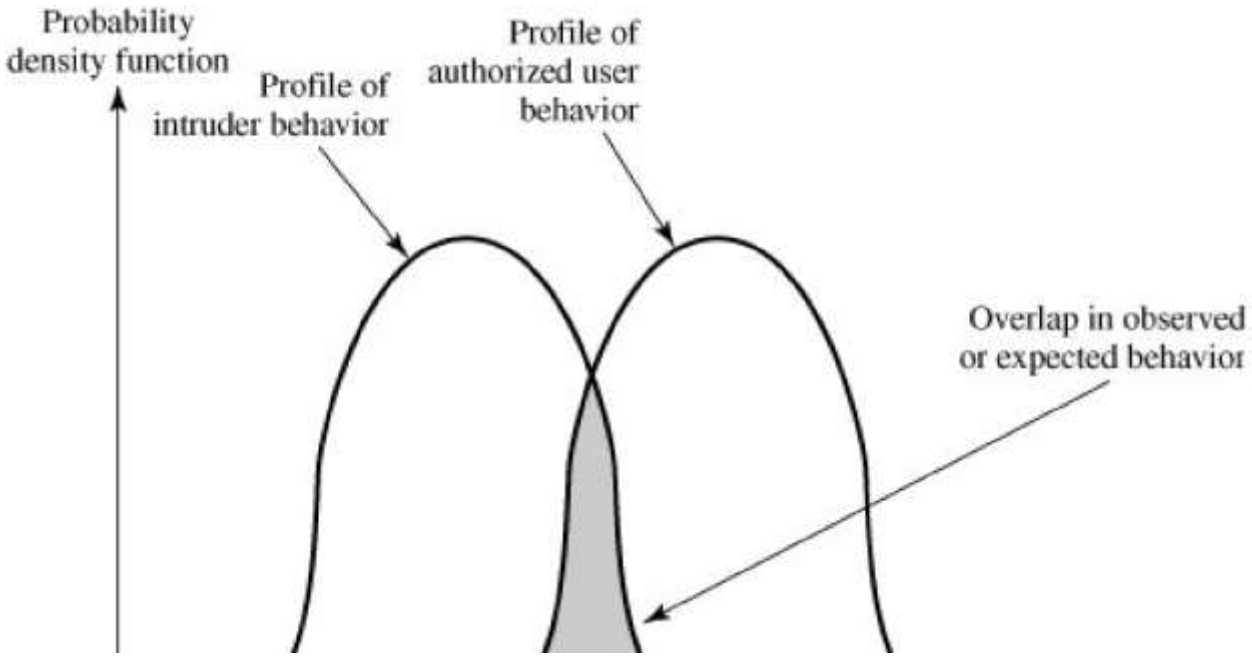
Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

- If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Even if the detection is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less the amount of damage and the more quickly that recovery can be achieved.

- An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

- Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified. Of course, we cannot expect that there will be a crisp, exact distinction between an attack by an intruder and the normal use of resources by an authorized user. Rather, we must expect that there will be some overlap.

Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.



In Anderson's study, it was postulated that one could, with reasonable confidence, distinguish between a masquerader and a legitimate user. Patterns of legitimate user behavior can be established by observing past history, and significant deviation from such patterns can be detected. Anderson suggests that the task of detecting a misfeasor (legitimate user performing in an unauthorized fashion) is more difficult, in that the distinction between abnormal and normal behavior may be small. Anderson concluded that such violations would be undetectable solely through the search for anomalous behavior. However, misfeasor behavior might nevertheless be detectable by intelligent definition of the class of conditions that suggest unauthorized use. Finally, the detection of the clandestine user was felt to be beyond the scope of purely automated techniques. These observations, which were made in 1980, remain true today.

[PORR92] identifies the following approaches to intrusion detection:

- **Statistical anomaly detection:** Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.
- Threshold detection: This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.

- Profile based: A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.
- **Rule-based detection:** Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.

a. Anomaly detection: Rules are developed to detect deviation from previous usage patterns.

b. Penetration identification: An expert system approach that searches for suspicious behavior.

In a nutshell, statistical approaches attempt to define normal, or expected, behavior, whereas rule-based approaches attempt to define proper behavior. In terms of the types of attackers listed earlier, statistical anomaly detection is effective against masqueraders, who are unlikely to mimic The behavior patterns of the accounts they appropriate. On the other hand, such techniques may be unable to deal with misfeasors. For such attacks, rule-based approaches may be able to recognize events and sequences that, in context, reveal penetration. In practice, a system may exhibit a combination of both approaches to be effective against a broad range of attacks.

Audit Records

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity by users must be maintained as input to an intrusion detection system. Basically, two plans are used:

Native audit records: Virtually all multiuser operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that the native audit records may not contain the needed information or may not contain it in a convenient form.

Detection-specific audit records: A collection facility can be implemented that generates audit records containing only that information required by the intrusion detection system. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine. A good example of detection-specific audit records is one developed by Dorothy Denning . Each audit record contains the following fields:

Subject: Initiators of actions. A subject is typically a terminal user but might also be a process acting on behalf of users or groups of users. All activity arises through commands issued by subjects. Subjects may be grouped into different access classes, and these classes may overlap.

Action: Operation performed by the subject on or with an object; for example, login, read, perform I/O, execute.

Object: Receptors of actions. Examples include files, programs, messages, records, terminals, printers, and user- or program-created structures. When a subject is the recipient of an action, such as electronic mail, then that subject is considered an object. Objects may be grouped by type. Object granularity may vary by object type and by environment. For example, database actions may be audited for the database as a whole or at the record level.

Exception-Condition: Denotes which, if any, exception condition is raised on return.

Resource-Usage: A list of quantitative elements in which each element gives the amount used of some resource (e.g., number of lines printed or displayed, number of records read or written, processor time, I/O units used, session elapsed time).

Time-Stamp: Unique time-and-date stamp identifying when the action took place.

Most user operations are made up of a number of elementary actions. For example, a file copy involves the execution of the user command, which includes doing access validation and setting up the copy, plus the read from one file, plus the write to another file.

Statistical Anomaly Detection

As was mentioned, statistical anomaly detection techniques fall into two broad categories: threshold detection and profile-based systems. Threshold detection involves counting the number of occurrences of a specific event type over an interval of time. If the count surpasses what is considered a reasonable number that one might expect to occur, then intrusion is assumed.

Threshold analysis, by itself, is a crude and ineffective detector of even moderately sophisticated attacks. Both the threshold and the time interval must be determined. Because of the variability across users, such thresholds are likely to generate either a lot of false positives or a lot of false negatives. However, simple threshold detectors may be useful in conjunction with more sophisticated techniques. Profile-based anomaly detection focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert. The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. An analysis of auditrecords over a period of time can be used to determine the activity

profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior. Examples of metrics that are useful for profile-based intrusion detection are the following:

Counter: A nonnegative integer that may be incremented but not decremented until it is reset by management action. Typically, a count of certain event types is kept over a particular period of time. Examples include the number of logins by a single user during an hour, the number of times a given command is executed during a single user session, and the number of password failures during a minute.

Gauge: A nonnegative integer that may be incremented or decremented. Typically, a gauge is used to measure the current value of some entity. Examples include the number of logical connections assigned to a user application and the number of outgoing messages queued for a user process.

Interval timer: The length of time between two related events. An example is the length of time between successive logins to an account.

Resource utilization: Quantity of resources consumed during a specified period. Examples include the number of pages printed during a user session and total time consumed by a program execution.

Given these general metrics, various tests can be performed to determine whether current activity fits within acceptable limits. [DENN87] lists the following approaches that may be taken:

- Mean and standard deviation
- Multivariate
- Markov process
- Time series
- Operational

The simplest statistical test is to measure the **mean and standard deviation** of a parameter over some historical period. This gives a reflection of the average behavior and its variability. The use of mean and standard deviation is applicable to a wide variety of counters, timers, and resource measures. But these measures, by themselves, are typically too crude for intrusion detection purposes.

A **multivariate** model is based on correlations between two or more variables. Intruder behavior may be characterized with greater confidence by considering such correlations (for example, processor time and resource usage, or login frequency and session elapsed time).

A **Markov process** model is used to establish transition probabilities among various states. As an example, this model might be used to look at transitions between certain commands.

A **time series** model focuses on time intervals, looking for sequences of events that happen too rapidly or too slowly. A variety of statistical tests can be applied to characterize abnormal timing.

Finally, an **operational model** is based on a judgment of what is considered abnormal, rather than an automated analysis of past audit records. Typically, fixed limits are defined and intrusion is suspected for an observation that is outside the limits. This type of approach works best where intruder behavior can be deduced from certain types of activities. For example, a large number of login attempts over a short period suggests an attempted intrusion.

Rule-Based Intrusion Detection

Rule-based techniques detect intrusion by observing events in the system and applying a set of rules that lead to a decision regarding whether a given pattern of activity is or is not suspicious. In very general terms, we can characterize all approaches as focusing on either anomaly detection or penetration identification, although there is some overlap in these approaches.

Rule-based anomaly detection is similar in terms of its approach and strengths to statistical anomaly detection. With the rule-based approach, historical audit records are analyzed to identify usage patterns and to generate automatically rules that describe those patterns. Rules may represent past behavior patterns of users, programs, privileges, time slots, terminals, and so on. Current behavior is then observed, and each transaction is matched against the set of rules to determine if it conforms to any historically observed pattern of behavior.

As with statistical anomaly detection, rule-based anomaly detection does not require knowledge of security vulnerabilities within the system. Rather, the scheme is based on observing past behavior and, in effect, assuming that the future will be like the past. In order for this approach to be effective, a rather large database of rules will be needed.

Rule-based penetration identification takes a very different approach to intrusion detection, one based on expert system technology.The key feature of such systems is the use of rules for

identifying known penetrations or penetrations that would exploit known weaknesses. Rules can also be defined that identify suspicious behavior, even when the behavior is within the bounds of established patterns of usage. Typically, the rules used in these systems are specific to the machine and operating system. Also, such rules are generated by "experts" rather than by means of an automated analysis of audit records. The normal procedure is to interview system administrators and security analysts to collect a suite of known penetration scenarios and key events that threaten the security of the target system. Thus, the strength of the approach depends on the skill of those involved in setting up the rules.

A simple example of the type of rules that can be used is found in NIDX, an early system that used heuristic rules that can be used to assign degrees of suspicion to activities . Example heuristics are the following:

1. Users should not read files in other users' personal directories.

2. Users must not write other users' files.

3. Users who log in after hours often access the same files they used earlier.

4. Users do not generally open disk devices directly but rely on higher-level operating system utilities.

5. Users should not be logged in more than once to the same system.

6. Users do not make copies of system programs.

The penetration identification scheme used in IDES is representative of the strategy followed. Audit records are examined as they are generated, and they are matched against the rule base. If a match is found, then the user's *suspicion rating* is increased. If enough rules are matched, then the rating will pass a threshold that results in the reporting of an anomaly. The IDES approach is based on an examination of audit records. A weakness of this plan is its lack of flexibility. For a given penetration scenario, there may be a number of alternative audit record sequences that could be produced, each varying from the others slightly or in subtle ways. It may be difficult to pin down all these variations in explicit rules. Another method is to develop a higher-level model independent of specific audit records. An example of this is a state transition model known as USTAT [ILGU93]. USTAT deals in general actions rather than the detailed specific actions recorded by the UNIX auditing mechanism. USTAT is implemented on a SunOS systemthat provides audit records on 239 events. Of these, only 28 are used by a preprocessor, which maps these onto 10 general actions . Using just these actions and the parameters that are invoked with

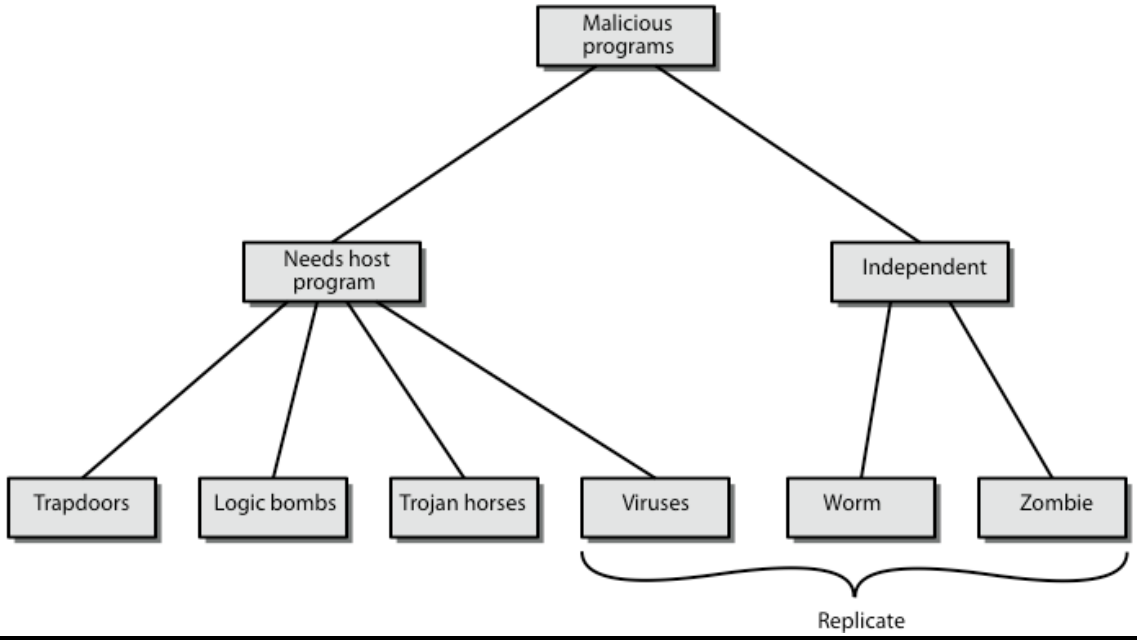
each action, a state transition diagram is developed that characterizes suspicious activity. Because a number of different auditable events map into a smaller number of actions, the rule-creation process is simpler. Furthermore, the state transition diagram model is easily modified to accommodate newly learned intrusion behaviors.

\ Viruses and Related Threats

- Viruses
- Computer viruses have got a lot of publicity
- One of a family of **malicious software**
- Malicious software is software that is intentionally included or ins
erted in a system for a harmful purpose
- Effects usually obvious
- They have figured in news reports, fiction, movies
- often exaggerated
- getting more attention than deserve

Malicious Programs

- Dependent program
- They Need host programs
- They cannot exist independent of some actual application
- E.g.: Viruses, Logic bomb, Backdoor
- Independent program
- They can be scheduled and run by the OS
- E.g.: Worm, Zombie



Backdoor or Trapdoor

- A secret entry point in a program

- It allows those who know access bypassing usual security procedures
- It have been commonly used by developers
- A threat when left in production programs allowing exploited by attackers
- It is very hard to block in O/S
- It requires good s/w development & update

Logic Bomb
- One of oldest types of malicious software
- Code embedded in legitimate program
- It is activated when specified conditions met
- Example
- presence/absence of some file
- particular date/time
- particular user
- When triggered typically damage system
- E.g., modify/delete files/disks, halt machine, etc

Trojan Horse
- A program (or some part of a program) with hidden side-effects
- Trojan horse is usually attractive to run
- E.g., freeware game, s/w upgrade, etc
- when runs, it performs some additional tasks
- allows attacker to indirectly gain access they do not have directly
- E.g., destroy/modify data, …
- It often used to propagate a virus/worm or install a backdoor Zombie

- A program which secretly takes over another networked computer
- Then, the attacker uses the zombies to indirectly launch attacks (to the target host)
- => Zombies often used to launch distributed denial of service (DDoS) attacks
- Zombie exploits known flaws in network systems

The Nature of Viruses
- Viruses: a piece of self-replicating code attached to some other code
- Cf. biological virus
- Both (biological/computer virus) carry a payload and propagate itself
- Payload contains code to make copies of itself as well as code to perform some covert task

Virus phases:
- Dormant – waiting on trigger event
- Propagation – replicating to programs/disks
- Triggering – activated by event to execute payload
- Execution – performing the functions in the payload
- Detailed phases usually depend on machine/OS specific
- exploiting features/weaknesses

Structure

program V :=
{go to main :

```
1234567;
subroutine infect-executable :=
{loop:
file:=get-random-executable-file;
if( first-line-of-file = 1234567 )
then goto loop
else prepend V to file;}
subroutine do-damage :=
{whatever damage is to be done}
subroutine trigger-pulled :=
{return true if some condition holds}
main : main-program :=
{infect-executable;
if trigger-pulled then do-damage;
goto next;}
next;
}
```

- A simple virus
- This virus is easily detected because an infected version of a program is longer than the corresponding uninfected one

Email Virus

- Email viruses are spread using email with attachment containing a macro virus
- cf Melissa
- They are triggered when user opens attachment, or worse even when mail viewed by using scripting features in mail

agent

- => hence propagate very quickly
- Usually targeted at Microsoft Outlook mail agent & Word

/Excel documents

- We need better O/S & application security

Worms

- Originally, worms are self-replicating programs but not infecting ones.
- Typically, spread over a network
- E.g., Morris Internet Worm in 1988, which led to creation of CERTs
- Worms propagate by using users' distributed privileges or by exploiting system vulnerabilities
- Recently, worms are widely used by hackers to create **zombie PC's**, subsequently used for further attacks, esp. DoS (Denial-of-Services) attack.
- Major issue is lack of security of permanently connected systems, esp. PC's

It exhibits the same characteristics as a computer virus

- The propagation phase performs the following functions :
- Search for other systems to infect by examining host tables
- Establish a connection with a remote system.
- Copy itself to the remote system and cause the copy to be run.
- It may also disguise its presence by naming itself as a system process or using some other name that may not be noticed by a system operator.

Morris Worm

- best known classic worm
- released by Robert Morris in 1988
- targeted Unix systems
- using several propagation techniques
- simple password cracking of local pw file
- exploit bug in finger daemon
- exploit debug trapdoor in sendmail daemon
- if any attack succeeds then replicated self

Recent Worm

- Attacks new spate of attacks from mid-2001
- Code Red - used MS IIS bug
- probes random IPs for systems running IIS
- had trigger time for denial-of-service attack
- 2nd wave infected 360000 servers in 14 hours
- Code Red 2 - installed backdoor
- Nimda - multiple infection mechanisms
- SQL Slammer - attacked MS SQL server
- Sobig.f - attacked open proxy servers
- Mydoom - mass email worm + backdoor

Virus Countermeasures

- Anti-Virus Approaches
- **first-generation**
- scanner uses virus signature to identify virus
- or change in length of programs
- **second-generation**
- uses heuristic rules to spot viral infection
- or uses crypto hash of program to spot changes
- **third-generation**
- memory-resident programs identify virus by actions
- **fourth-generation**
- packages with a variety of antivirus techniques
- eg scanning & activity traps, access-controls
- arms race continues

Advanced Anti-Virus Techniques

- Generic decryption
- It use CPU simulator to check program signature & behavior before actually running it
- It start the simulator to simulate the file execution
- Note that all polymorphic virus should decrypt itself to activate.
- By periodically scanning the memory, decrypted virus code can be detected.

Digital Immune System

- It is a comprehensive approach to virus protection developed by IBM
- The objective of this system is to provide rapid response time so that viruses can be stamped out almost as soon as they are introduced
- It uses general purpose emulation & virus detection

- Any virus entering org is captured, analyzed, detection/shielding created for it, removed