

Chapter 2

COMPUTER SECURITY AND INTRUSION DETECTION

The scenario in the previous section described an exemplary threat to computer system security in the form of an intruder attacking a company's web server. This chapter attempts to give a more systematic view of system security requirements and potential means to satisfy them. We define properties of a secure computer system and provide a classification of potential threats to them. We also introduce the mechanisms to defend against attacks that attempt to violate desired properties.

Before one can evaluate attacks against a system and decide on appropriate mechanisms to fend off these threats, it is necessary to specify a *security policy* [Tanenbaum and van Steen, 2002]. A security policy defines the desired properties for each part of a secure computer system. It is a decision that has to take into account the value of the assets that should be protected, the expected threats and the cost of proper protection mechanisms. A security policy that is sufficient for the data of a normal home user may not be sufficient for a bank, as a bank is obviously a more likely target and has to protect more valuable resources.

1. Security Attacks and Security Properties

For the following discussion, we assume that the function of a computer system is to provide information. In general, there is a flow of data from a source (e.g., a host, a file, memory) to a destination (e.g., a remote host, another file, a user) over a communication channel (e.g., a wire, a data bus). The task of the security system is to restrict access to this information to only those parties (persons or processes) that are authorized to have access, according to the security policy in use.

The normal information flow and several categories of attacks that target it are shown in Figure 2.1 (according to [Stallings, 2000]).

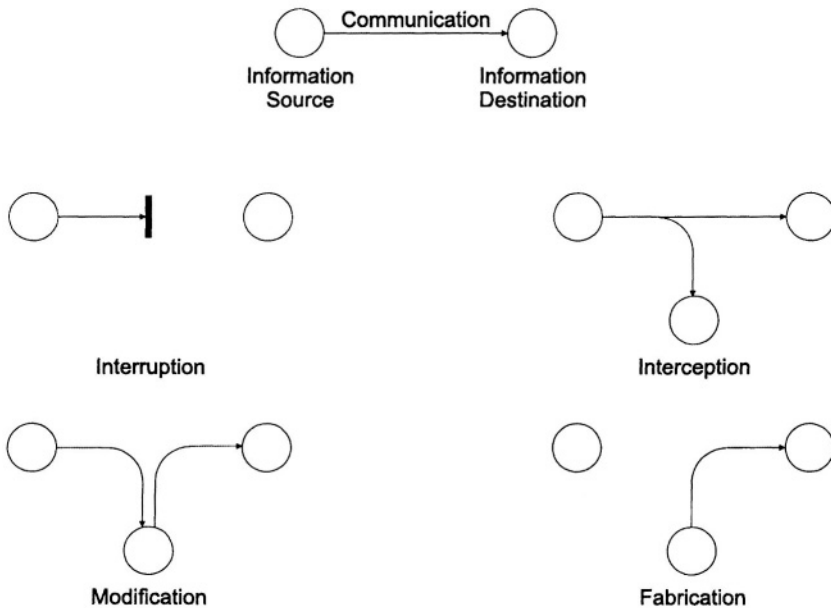


Figure 2.1. Security Attacks

- 1 **Interruption:** An asset of the system gets destroyed or becomes unavailable. This attack targets the source or the communication channel and prevents information from reaching its intended target (e.g., cutting the wire or overloading the link so that the information gets dropped because of congestion). Attacks in this category attempt to perform a kind of *denial of service (DOS)*.
- 2 **Interception:** An unauthorized party gets access to the information by eavesdropping into the communication channel (e.g., by wiretapping).
- 3 **Modification:** The information is not only intercepted, but modified by an unauthorized party while in transit from the source to the destination. (e.g., by modifying the message content).
- 4 **Fabrication:** An attacker inserts counterfeit objects into the system without having the sender doing anything. When a previously intercepted object is inserted, this process is called *replaying*. When the attacker pretends to be the legitimate source and inserts her desired information, the attack is called *masquerading* (e.g., replaying an authentication message or adding records to a file).

The four classes of attacks listed above violate different security properties of the computer system. A security property describes a desired feature of a system with regard to a certain type of attack. A common classification is listed below [Coulouris et al., 1996; Northcutt, 1999].

- **Confidentiality:** This property covers the protection of transmitted data against its release to unauthorized parties. In addition to the protection of the content itself, the information flow should also be resistant against traffic analysis. Traffic analysis is used to gather other information than the transmitted values themselves from the data flow (e.g., the parties involved, timing data, or frequency of messages).
- **Integrity:** Integrity protects transmitted information against modifications. This property assures that a single message reaches the receiver as it has left the sender, but integrity also extends to a stream of messages. It means that no messages are lost, duplicated, or reordered and it makes sure that messages cannot be replayed. As destruction is also covered under this property, all data must arrive at the receiver. Integrity is not only important as a security property, but also as a property for network protocols. That is, message integrity must also be ensured in case of random faults, not only in case of malicious modifications.
- **Availability:** Availability characterizes a system whose resources are always ready to be used. Whenever information needs to be transmitted, the communication channel is available and the receiver can cope with the incoming data. This property makes sure that attacks cannot prevent resources from being used for their intended purpose.
- **Authentication:** Authentication is concerned with making sure that the information is authentic. A system implementing the authentication property assures the recipient that the data is from the source that it claims to be. The system must make sure that no third party can masquerade successfully as another source.
- **Non-repudiation:** This property describes the mechanism that prevents either sender or receiver from denying a transmitted message. When a message has been transferred, the sender can prove that it has been received. Similarly, the receiver can prove that the message has actually been sent.

2. Security Mechanisms

Different security mechanisms can be used to enforce the security properties defined in a given security policy. Depending on the anticipated attacks, different means have to be applied to satisfy the desired properties. Three main classes of measures against attacks can be identified, namely attack prevention, attack avoidance, and attack detection. They are explained in detail in the following sections.

2.1 Attack Prevention

Attack prevention is a class of security mechanisms that contains ways of preventing or defending against certain attacks before they can actually reach

and affect the target. An important element in this category is access control, a mechanism which can be applied at different levels such as the operating system, the network, or the application layer.

Access control [Tanenbaum and van Steen, 2002] limits and regulates the access to critical resources. This is done by identifying or authenticating the party that requests a resource and checking its permissions against the rights specified for the demanded object. It is assumed that an attacker is not legitimately permitted to use the target object and is therefore denied access to the resource. As access is a prerequisite for an attack, any possible interference is prevented.

The most common form of access control used in multi-user computer systems are access control lists for resources that are based on the user and group identity of the process that attempts to use them. The identity of a user is determined by an initial authentication process that usually requires a name and a password. The login process retrieves the stored copy of the password corresponding to the user name and compares it with the presented one. When both match, the system grants the user the appropriate user and group credentials. When a resource should be accessed, the system looks up the user and group in the access control list and grants or denies access as appropriate. An example of this kind of access control can be found in the UNIX file system, which provides read, write and execute permissions based on the user and group membership. In this example, attacks against files that a user is not authorized to use are prevented by the access control part of the file system code in the operating system.

A firewall [Cheswick and Bellovin, 1994] is an important access control system at the network layer. The idea of a firewall is based on the separation of a trusted inside network of computers under single administrative control from a potential hostile outside network. The firewall is a central choke point that allows enforcement of access control for services that may run at the inside or outside. The firewall prevents attacks from the outside against the machines in the inside network by denying connection attempts from unauthorized parties located outside. In addition, a firewall may also be utilized to prevent users behind the firewall from using certain services that are outside (e.g., surfing web sites containing pornographic content).

2.2 Attack Avoidance

Security mechanisms in this category assume that an intruder may access the desired resource but the information is modified in a way that makes it unusable for the attacker. The information is preprocessed at the sender before it is transmitted over the communication channel and post-processed at the receiver. While the information is transported over the communication channel, it resists attacks by being nearly useless for an intruder. One notable exception

are attacks against the availability of the information, as an attacker could still interrupt the message. During the processing step at the receiver, modifications or errors that might have previously occurred can be detected (usually because the information can not be correctly reconstructed). When no modification has taken place, the information at the receiver is identical to the one at the sender before the preprocessing step.

The most important member in this category is cryptography, which is defined as the science of keeping messages secure [Schneier, 1996]. It allows the sender to transform information into what may seem like a random data stream to an attacker, but can be easily decoded by an authorized receiver (see Figure 2.2).

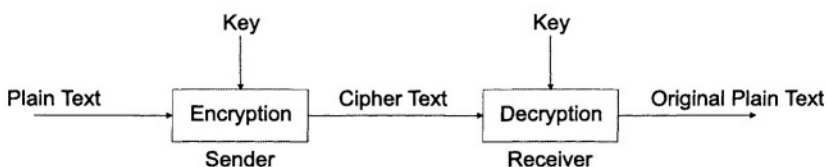


Figure 2.2. Encryption and Decryption

The original message is called *plaintext* (sometimes also cleartext). The process of converting this message through the application of some transformation rules into a format that hides its substance is called *encryption*. The corresponding disguised message is denoted as *ciphertext*, and the operation of turning it back into cleartext is called *decryption*. It is important to notice that the conversion from plain to ciphertext has to be lossless in order to be able to recover the original message at the receiver under all circumstances.

The transformation rules are described by a cryptographic algorithm. The function of this algorithm is based on two main principles: *substitution* and *transposition*. In the case of substitution, each element of the plaintext (e.g., bit, block) is mapped into another element of the used alphabet. Transposition describes the process where elements of the plaintext are rearranged. Most systems involve multiple steps (called rounds) of transposition and substitution to be more resistant against cryptanalysis. Cryptanalysis is the science of breaking the cipher, i.e., discovering the substance of the message behind its disguise.

When the transformation rules process the input elements in a continuous fashion, the mechanism is called a *stream cipher*. When the transformation operates on fixed sized input blocks, the corresponding encryption algorithm is called a *block cipher*.

If the security of an algorithm is based on keeping the way how the algorithm works (i.e., the transformation rules) secret, it is called a restricted algorithm. Those algorithms are no longer of any interest today because they do not allow standardization or public quality control. In addition, when a large group of

users are involved, such an approach cannot be used. A single person leaving the group makes it necessary for everyone else to change the algorithm.

Modern crypto systems solve this problem by basing the ability of the receiver to recover encrypted information on the fact that she possesses a secret piece of information (usually called the *key*). Both encryption and decryption functions have to use a key and they are heavily dependent on it. When the security of the crypto system is completely based on the security of the key, the algorithm itself may be revealed. Although the security does not rely on the fact that the algorithm is unknown, the cryptographic function itself and the used key, together with its length, must be chosen with care. A common assumption is that the attacker has the fastest commercially available hardware at her disposal in her attempt to break the ciphertext.

The most common attack, called *known plaintext attack*, is executed by obtaining ciphertext together with its corresponding plaintext. The encryption algorithm must be so complex that even if the code breaker is equipped with plenty of such pairs, it is infeasible for her to retrieve the key. An attack is infeasible when the cost of breaking the cipher exceeds the value of the information, or the time it takes to break it exceeds the lifespan of the information itself.

Given pairs of corresponding cipher and plaintext, it is obvious that a simple key guessing algorithm will succeed after some time. The approach of successively trying different key values until the correct one is found is called *brute force* attack because no information about the algorithm is utilized. In order to be useful, it is a necessary condition for an encryption algorithm that brute force attacks are infeasible.

Depending on the keys that are used, one can distinguish two major cryptographic approaches - public-key and secret-key crypto systems.

Secret-Key Cryptography

Secret-key cryptography is the type of cryptography that has been used for the transmission of secret information for centuries, long before the advent of computers. These algorithms require that the sender and the receiver agree on a key before communication is started.

It is common for this variant (which is also called single-key or symmetric encryption) that a single secret key is shared between the sender and the receiver. It needs to be communicated in a secure way before the actual encrypted communication can start and has to remain secret as long as the information is to remain secret. Encryption is achieved by applying an agreed function to the plaintext using the secret key. Decryption is performed by applying the inverse function using the same key.

The classic example of a secret-key cipher which is widely deployed today is the *Data Encryption Standard (DES)* [DES, 1977]. DES has been developed in

1977 by IBM and adopted as a standard by the US government for administrative and business use. Recently, it has been replaced by the *Advanced Encryption Standard (AES - Rijndael)* [AES, 2001]. DES is a block cipher that operates on 64-bit plaintext blocks and utilizes a 56-bit key. The algorithm uses 16 rounds that are key dependent. Although there are no known weakness of the DES algorithm itself, its security has been much debated. The small key length makes brute force attacks possible and several cases have occurred where DES protected information has been cracked. A suggested improvement called 3DES uses three rounds of the simple DES with three different keys. This extends the key length to 168 bits while still relying on the very secure DES base algorithm.

Public-Key Cryptography

Before the advent of public-key cryptography, the knowledge of the key that is used to encrypt a plaintext also allowed the inverse process, the decryption of the ciphertext. In 1976, this paradigm of cryptography was changed by Diffie and Hellman [Diffie and Hellman, 1976] when they described their public-key approach. Public-key cryptography utilizes two different keys, one called the *public key*, the other one called the *private key*. The public key is used to encrypt a message while the corresponding private key is used to do the opposite. The innovation is the fact that it is infeasible to retrieve the private key given the public key. This makes it possible to remove the weakness of secure key transmission from the sender to the receiver. The receiver can simply generate her public/private key pair and announce the public key without fear. Anyone can obtain the public key and use it to encrypt messages that only the receiver, with the corresponding private key, is able to decrypt.

Mathematically, the process is based on the *trapdoor* of *one-way* functions. A one-way function is a function that is easy to compute but very hard to inverse. That means that given x it is easy to determine $f(x)$ but given $f(x)$ it is hard to get x . Hard is defined as computationally infeasible in the context of cryptographically strong one-way functions. Although it is obvious that some functions are easier to compute than their inverse (e.g., square of a value in contrast to its square root) there is no mathematical proof or definition of one-way functions. There are a number of problems that are considered difficult enough to act as one-way functions, but this is more an agreement among crypto analysts than a rigorously defined set (e.g., factorization of large numbers). A one-way function is not directly usable for cryptography, but it becomes so when a trapdoor exists. A trapdoor is a mechanism that allows one to easily calculate x from $f(x)$ when some additional information y is provided.

A common misunderstanding about public-key cryptography is thinking that it makes secret-key systems obsolete, either because it is more secure or because it does not have the problem of secretly exchanging keys. As the security

of a crypto system depends on the length of the key used and the utilized transformation rules, there is no automatic advantage of one approach over the other. Although the key exchange problem is elegantly solved with a public-key system, the process itself is very slow and has its own problems. Secret-key systems are usually a factor of 1000 (see [Schneier, 1996] for exact numbers) faster than their public-key counterparts. Therefore, most communication is still secured using secret-key systems and public-key systems are only utilized for exchanging temporary secret keys which are then used to encrypt the communication. This hybrid approach is the common design, which benefits from the high speed of conventional cryptography and from the simplification of the key distribution process provided by public-key systems.

A problem in public-key systems is the authenticity of the public key. An attacker may offer the sender her own public key and pretend that it originates from the legitimate receiver. The sender then uses the fake public key to perform her encryption and the attacker can simply decrypt the message using her private key. This technique may be used to set up a man-in-the-middle attack in which a third party is able to monitor and modify the communication between two parties, even when encryption is used.

In order to thwart an attacker that attempts to substitute her public key for the victim's one, *certificates* are used. A certificate combines user information with the user's public key and the digital signature of a trusted third party that guarantees that the key belongs to the mentioned person. The trusted third party is usually called a *certification authority (CA)*. The certificate of a CA itself is usually verified by a higher level CA that confirms that the CA's certificate is genuine and contains its public key. The chain of third parties that verify their respective lower level CAs has to end at a certain point that is called the root CA. A user that wants to verify the authenticity of a public key and all involved CAs needs to obtain the self-signed certificate of the root CA via an external channel. Web browsers (e.g. Netscape Navigator, Internet Explorer), for example, ship with a number of certificates of globally known root CAs. A framework that implements the distribution of certificates is called a public key infrastructure (PKI). An important issue is revocation, the invalidation of a certificate when the key has been compromised.

The best known public-key algorithm and textbook classic is RSA [Rivest et al., 1978], named after its inventors Rivest, Shamir and Adleman. It is a block cipher in which the security of the messages is based on the infeasibility of the factorization of large integers, as the public and the private keys are functions of large primes. A well known protocol for public key management is X.509 [x509, 2002].

An interesting and important feature of public-key cryptography is its possible use for authentication. In addition to making the information unusable for attackers, a sender may utilize cryptography to prove her identity to the

receiver. More precisely, by encrypting a message with her own private key a user can prove to another user that she is in fact the source of the message. The receiver can verify the identity of the sender by decrypting the message with the sender's public key. If the operation succeeds, the receiver can be confident that the message was sent by the sender. The process of encrypting a message with a user's private key is called *signing* a message.

2.3 Attack Detection

Attack detection assumes that an attacker can obtain access to her desired targets and is successful in violating a given security policy. Mechanisms in this class are based on the optimistic assumption that, most of the time, the information is transferred without interference. When undesired actions occur, attack detection has the task of reporting that something went wrong and to react in an appropriate way. In addition, it is often desirable to identify the exact type of attack. An important facet of attack detection is recovery. Often it is enough to just report that malicious activity has been detected, but some systems require that the effects of an attack be reverted or that an ongoing and discovered attack is stopped. On one hand, attack detection has the advantage that it operates under the worst-case assumption that the attacker gains access to the communication channel and is able to use or modify the resource. On the other hand, detection is not effective in providing confidentiality of information. When the security policy specifies that interception of information has a serious security impact, then attack detection is not an applicable mechanism.

The most important members of the attack detection class are intrusion detection systems. Because this book is focused on intrusion detection and alert correlation, the remaining sections of this chapter are dedicated to a more detailed introduction to intrusion detection.

3. Intrusion Detection

An intrusion is defined as a sequence of related actions performed by a malicious adversary that results in the compromise of a target system. It is assumed that the actions of the intruder violate a given security policy. The existence of a security policy that states which actions are considered malicious and should be prevented is a key requisite for an intrusion detection system. Violations can only be detected when actions can be compared against given rules.

Intrusion detection (ID) is the process of identifying and responding to malicious activities targeted at computing and network resources. This definition introduces the notion of intrusion detection as a process, which involves technology, people, and tools. Intrusion detection is an approach that is complementary with respect to mainstream approaches to security, such as access control and

cryptography. The adoption of intrusion detection systems is motivated by several factors:

- 1 Surveys have shown that most computer systems are flawed by vulnerabilities, regardless of manufacturer or purpose [Landwehr et al., 1994], that the number of security incidents is continuously increasing [CERT, 2003], and that users and administrators are generally very slow in applying fixes to vulnerable systems [Rescorla, 2003]. As a consequence, many experts believe that computer systems will never be absolutely secure [Bellovin, 2001].
- 2 Deployed security mechanism, e.g., authentication and access control, may be disabled as a consequence of misconfiguration or malicious actions.
- 3 Users of the system may abuse their privileges and perform damaging activities.
- 4 Even if an attack is not successful, in most cases it is useful to be aware of the compromise attempt.

Intrusion detection systems (IDSs) are software applications dedicated to detect intrusions against a target network. IDSs have been designed to address the issues described above. As such, they are not intended to replace traditional security methods, but rather to complete them. According to [Dacier et al., 1999], an intrusion detection system has to fulfill the following requirements.

- *Accuracy* - An IDS must not identify a legitimate action in a system environment as an anomaly or a misuse (a legitimate action which is identified as an intrusion is called a *false positive*).
- *Performance* - The IDS performance must be sufficient enough to carry out real time intrusion detection (real time means that an intrusion has to be detected before significant damage has occurred – according to [Ranum, 2000] this should be under a minute).
- *Completeness* - An IDS should not fail to detect an intrusion (an undetected intrusion is called a *false negative*). One has to admit that it is rather difficult to fulfill this requirement because it is almost impossible to have a global knowledge about past, present, and future attacks.
- *Fault Tolerance* - An IDS must itself be resistant to attacks.
- *Scalability* - An IDS must be able to process the worst-case number of events without dropping information. This point is especially relevant for systems that correlate events from different sources at a small number of dedicated hosts. As networks grow bigger and get faster, such nodes become overwhelmed by the increasing number of events.

3.1 Architecture

There exist many IDSs, based on different conceptual frameworks. It is still possible, however, to recognize a common architecture that underlies all intrusion detection systems. We will present the main components of IDSs and their functionality.

To this end, we use the terminology introduced by the working group on the Common Intrusion Detection Framework (CIDF) [Porrás et al., 1998]. CIDF models an IDS as an aggregate of four components with specific roles:

- *Event boxes (E-boxes)*. The role of event boxes is to generate events by processing raw audit data produced by the computational environment. A common example of an E-boxes is a program that filters audit data generated by an operating system evaluated at the C2 level of TCSEC [U.S. Department of Defense, 1985]. Another example is a network sniffer that generates events based on the network traffic.
- *Analysis boxes (A-boxes)*. The role of an analysis box is to analyze the events provided by other components. The results of the analysis are sent back to the system as additional events, typically representing alarms. Usually A-boxes analyze simple events supplied by E-boxes. Some A-boxes analyze events produced by other A-boxes and operate at a higher level of abstraction.
- *Database boxes (D-boxes)*. Database boxes simply store events, guaranteeing persistence and allowing postmortem analysis.
- *Response boxes (R-boxes)*. Response boxes consume messages that carry directives about actions to be performed as a reaction to a detected intrusion. Typical actions include killing processes, resetting network connections, and modifying firewall settings.

Figure 2.3 presents an IDS system where two E-boxes monitor the environment and deliver audit events to two A-boxes. These A-boxes analyze the audit data and provide their conclusions to a third A-box that correlates the alerts, stores them in a D-box and controls an R-box. Dashed lines are used to indicate exchange of events, solid lines represent the exchanging of raw audit data.

Note that components are logical entities which produce or consume events. The CIDF model does not mandate what their implementation should be. It only states their roles and interactions. They can be realized as a single process on a single computer or as a collection of cooperating processes spanning multiple computers.

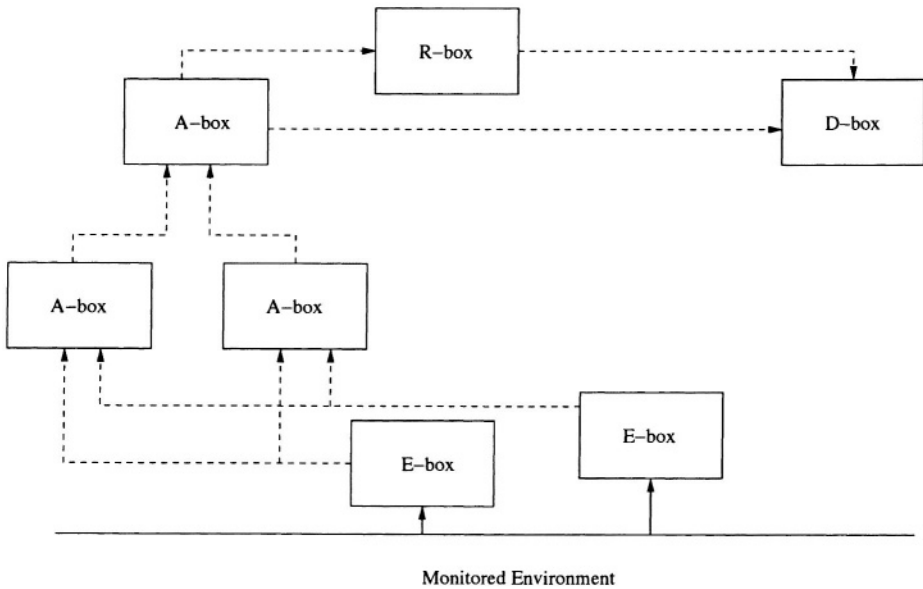


Figure 2.3. CIDF Description of an IDS System

3.2 Taxonomy

Intrusion detection systems may be classified according to different characteristics [Lunt, 1993; Allen et al., 2000; Bace and Mell, 2001]. The following ones seem to be particularly characterizing [Dacier et al., 1999]:

- *Detection method.* It defines the philosophy on which the A-box is built. Two approaches have been proposed. When the IDS defines what is “normal” in the environment and flags as attacks deviations from normality, it is qualified as *anomaly-based*. When the IDS explicitly defines what is “abnormal”, using specific knowledge about the attacks in order to detect them, it is called *misuse-based*.
- *Behavior on detection.* It defines a characteristic of the R-box. It is said to be passive if the system just issues an alert when an attack is detected. If more proactive actions are taken (e.g., disconnecting users, shutting down network connections), it is said to be active.
- *Audit source location.* It specifies where the E-box takes audit data from. We distinguish between host-based IDSs, which deal with audit data generated on a single host, e.g., a C2 audit trail; application-based IDSs, which work on audit records produced by a specific application; and network-based IDSs, which monitor network traffic.

- *Usage frequency.* It discriminates between systems that analyze the data in real time and those that are run periodically (offline). It specifies how often the A-box analyzes data collected by other parts of the system.

We will analyze these characteristics more in detail in the following sections.

3.3 Detection Method

Detection can be performed according to two complementary strategies:

- 1 defining what is the manifestation of an attack and searching for an occurrence of the attack (misuse-based) or
- 2 defining what is the normal behavior on the system and searching for activities that deviate from it (anomaly-based).

Misuse-based Systems

Misuse-based systems are equipped with a database of information (the knowledge base) that contains a number of attack models (sometimes called “signatures”). The audit data collected by the IDS is compared with the content of the database and, if a match is found, an alert is generated. Events that do not match any of the attack models are considered part of legitimate activities.

The main advantage of misuse-based systems is that they usually produce very few false positives: attack description languages usually allow for modeling of attacks at such fine level of detail that only a few legitimate activities match an entry in the knowledge base.

However, this approach has drawbacks as well. First of all, populating the knowledge base is a difficult, resource intensive task. Furthermore, misuse-based systems cannot detect previously unknown attacks, or, at most, they can detect only new variations of previously modeled attacks. Therefore, it is essential to keep the knowledge base up-to-date when new vulnerabilities and attack techniques are discovered.

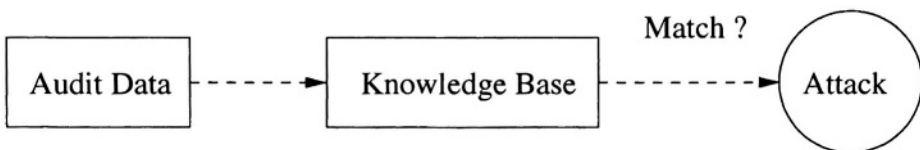


Figure 2.4. Block Diagram of a Typical Knowledge-Based IDS

A number of different techniques have been proposed for performing misuse-based intrusion detection [Ilgun et al., 1995; Lindqvist and Porras, 1999; Kumar and Spafford, 1994; Roesch, 1999; Neumann and Porras, 1999]. Before briefly

discussing the most common misuse-based techniques, it is worth discussing how these systems can be differentiated on the basis of state maintenance (and the importance of this characteristic on the overall intrusion detection system).

A *stateless* intrusion detection system treats each event independently of others. When the processing of the current event is completed, every information regarding this event is discarded.

This approach simplifies the design of the system, specifically of the A-box, because it does not need to allocate and maintain memory to keep information about past activity. Stateless systems usually have very good performance in terms of speed of processing because the analysis step is reduced to simple lookups in the knowledge base and matches against the current event, with no additional operations needed.

However, stateless systems have important limitations. In the first place, they are not complete, in the sense that there exist classes of attacks that cannot be detected. For example, multistep attacks are malicious activities that require a series of steps to be completed. Actions involved in each step are not *per se* intrusive and become malicious only when executed in the correct and complete sequence. Because the system has no memory of past events, it lacks the possibility to keep track of steps. Note that it would be possible to model the entire attack on the basis of the last step, thus considering the action involved in this step as malicious, and inserting it in the attack knowledge base. However, because this action alone is not intrusive, it is likely to be performed also as part of legitimate activities. Thus, this approach would possibly generate an unacceptable number of false positives, thwarting the main advantage of misuse-based intrusion detection.

Furthermore, stateless systems are subject to a class of attacks whose aim is to trigger the generation of a flood of alerts. A number of tools have been proposed that analyze the database of attacks and automatically generate events or series of events that conform to the attack descriptions. The event stream synthesized in this manner forces intrusion detection systems to generate a large number of detection alerts. The resulting “alert storm” has been used to desensitize intrusion detection system administrators and hide attacks in the event stream [Patton et al., 2001; Mutz et al., 2003; Snot, 2004; Stick, 2004].

Stateful intrusion detection systems maintain information about past events. As a consequence, the effect of a certain event on the system is not independent of its position in the event stream. While this approach adds an additional level of complexity to the design and implementation of A-boxes, it provides significant advantages. In particular, stateful tools are able to model and detect attacks that involve many steps. Furthermore, they are less prone to the alert storm attacks described in the previous paragraph, because it is more difficult to develop a program that is able to exactly reproduce the steps of a modeled attack. However, these systems are all vulnerable to state-based attacks, where an

attacker forces the IDS to maintain large amount of information about ongoing attacks, effectively affecting the overall performance of the system.

In signature-based systems, abstractions (“signatures”) of attacks are stored in the knowledge base. Signatures are used to summarize in a compact format all the information necessary to describe attacks. For example, a network intrusion detection system may store in the knowledge base the content of network packets involved in known attacks. Usually, signatures are stored in a format that allows straightforward comparison with information found in the event stream. During operation, events are checked against entries in the signature file: if a match is found, the system raises an alarm.

Signature based systems have gained popularity because they are easy to develop, give accurate feedback on alarms, and are usually requires little computational resources.

However, they also have disadvantages:

- The description of an attack is usually a very low level description and thus, difficult to determine or interpret.
- Every attack or variation of an attack requires a new signature to be added to the knowledge base, thus its size can become very large.
- The more specific a signature is, the less false positives are generated. But also, the more specific a signature is, the easier it is for an attacker to create a slightly different version of an attack that do not match the signature and thus goes unnoticed. Such an attack is said to be *polymorphic*.

Example of signature-based systems include Snort [Roesch, 1999], EMERALD [Neumann and Porras, 1999] and many commercial products.

One technique that can be used to describe complex attack signatures are state transitions [Ilgun et al., 1995]. State transitions model an activity on the system as a series of state transitions, where a *state* is a snapshot of the system representing the value of all the memory locations of the system. Malicious activities move the state of the system from an initial safe state to a final compromised state, possibly passing through a number of intermediate states. This technique requires the analyst to identify those transitions that are critical in leading the system into a compromised state. The IDS will then search for such transitions. Of course, state transition systems are stateful tools.

State transition models are appealing because they allow high-level, even graphical, description of attacks. They have very high expressive power, meaning that arbitrarily complicated attacks can be modeled and detected. In particular, multistep attacks fit very well the state transition modeling technique. It also provides very detailed feedback on the generated alerts, because the entire sequence of actions that caused the alarm to be triggered can be easily provided. Furthermore, it allows to deploy a response before an attack reaches its final

step, thus allowing to effectively prevent the attack rather than only detecting it.

The main disadvantage of the state transition technique is that its computational requirements can be high if the system has to keep track of many concurrent attacks.

Anomaly-based Systems

Anomaly-based systems are based on the assumption that all anomalous activities are malicious¹. Thus, they first build a model of the normal behavior of the system (i.e., profile) and then look for anomalous activities, i.e., activities that do not conform to the established model. Anything that does not correspond to the system profile is flagged as intrusive [Helman and Liepins, 1993]. Many systems have been built following this approach, e.g., [Hofmeyr et al., 1998; Ghosh et al., 1998; Tan and Maxion, 2002; Ko et al., 1997].

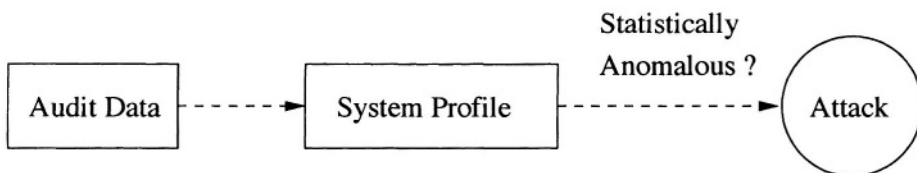


Figure 2.5. Block Diagram of a Typical Behavior-Based IDS

The main advantage of systems based on this approach is that, in theory, they are able to detect previously unknown attacks. On the other hand, anomaly-based systems are prone to the generation of many false positives, that is, their accuracy is typically low.

One class of anomaly detection systems use machine learning techniques to determine the parameters of normal system behavior (i.e., system profiles). These systems may suffer from specific problems:

- The amount of time required to teach to the system the normal behavior might be long.
- The behavior of the monitored environment might change during a period of time, requiring the system to be retrained.

¹ Note that “All malicious activities are anomalous” is different from “All anomalous activities are malicious”. The former implies that there do not exist malicious activities that are not anomalous and thus assumes that systems that detect every anomaly also detect every attack. On the other hand, the latter leaves open the possibility that there exist attacks that do not appear as anomalous activity. Thus, the “all malicious is anomalous” assumption postulates that anomaly-based systems can detect every possible attacks. The “all anomalous is malicious” asserts that attacks might go undetected by an anomaly-based IDS.

- If the training set contains attacks, the system will consider malicious behavior normal.
- It may be possible to perform an attack within the boundaries of “normality”.

Statistical analysis has shown that, under reasonable assumptions, the probability $P(\text{Intrusion}|\text{Alarm})$, i.e., the probability that an alarm really indicates an intrusion, is dominated by the false alarm rate rather than the true positive detection rate [Axelsson, 1999]. This is a consequence of the *base rate fallacy*: typical traffic shows a huge number of benign activities and only a few malicious events. The problem is that most systems today have poor performance in terms of suppression of false alarms.

A number of studies have indicated that most network systems are intrinsically characterized by a high rate of change and diversity [Floyd and Paxson, 2001]. Moreover, ambiguities in underlying protocols and differences in application programs exist [Bellovin, 1993]. Thus, detecting anomalous behavior in certain environments is very difficult, because anomalies are characteristic of the environment itself.

Finally, some new attack strategies have emerged that mimic legitimate activities and thus go undetected [Wagner and Soto, 2002; Tan et al., 2002]. While these studies were applied to a specific anomaly-based IDS, they are based on general concepts that seem to be applicable to whole classes of anomaly-based IDSs.

3.4 Type of Response

Most intrusion detection tools are passive: when an attack is detected, an alert is generated without trying to oppose the attack any further. This requires the security officer to manually inspect all alerts and take appropriate actions. Therefore, there can be a significant delay in the process of dealing with an intrusion.

A number of IDSs have proactive capabilities, e.g., they can change the security posture of a protected network to react to the detected attack. For example, such systems may modify file permissions, add firewall rules, kill processes, or shutdown network connections. It is important to note, however, that automatic countermeasures could, in certain cases, be leveraged by the attacker to damage the system itself or to cause a denial of service.

3.5 Audit Source Location

Intrusion Detection Systems can be characterized according to the source of the events they analyze. Typical system classes include host-based IDSs, application-based IDSs, network-based IDSs, as well as correlation systems. These are discussed in detail in the next paragraphs.

Host-based IDSs

Host-based IDSs (HIDS) detect attacks against a specific host by analyzing audit data produced by the host operating systems. Audit sources include:

- System information. Operating systems make available to processes in user space information about their internal working and security-relevant events. There exist programs that collect and show this information, e.g., *ps*, *vmstat*, *top*, *netstat*. The information provided is usually very complete and reliable because it is retrieved directly from the kernel. Unfortunately, few operating systems provide mechanism to systematically and continuously collect this information.
- Syslog facility [Lonvick, 2001]. Syslog is an audit facility provided by many UNIX-like operating systems. It allows programmers to specify a text message describing an event to be logged. Additional information, like the time when the event happened and the host where the program is running, is automatically added. Because of their simplicity, syslog events are used extensively by applications. However, applications usually log information valuable for debugging purposes that is not necessarily tailored to the needs of intrusion detection. Furthermore, a specific audit format is not imposed by the facility but changes according to the program that uses it. Thus, it may be difficult to extract audit data from logs and perform sophisticated analysis. Finally, the logged information can be easily polluted by messages crafted by an attacker to cover her tracks.
- C2 audit trail. Some operating systems comply with the C2 level of the TCSEC standard and thus monitor the execution of system calls. The data obtained is accurate because it comes directly from within the kernel.

Application-based IDS

Application-based IDSs detect attacks against a specific application. The audit information necessary to perform intrusion detection is usually obtained either using the already described syslog facility or instrumenting the application with specific audit mechanisms. The following discussion will focus on the latter approach.

Adding audit mechanisms to an existing application requires the modification of the application so that it produces audit information in response to security-relevant events. This can be accomplished in different ways:

- Directly modifying the application source code to include auditing code. This approach requires that the application code be available and modifiable.
- Interposing code responsible to extract audit information in correspondence of interfaces used by the application, e.g., the system call or the standard C

library interface. One disadvantage of this approach is that applications other than the one to be monitored could be affected, e.g., in terms of performance loss, by the use of the interposition mechanism.

- Using extension hooks provided by the application itself to implement the audit collection functionality. This approach guarantees great flexibility, but not all the applications offer extension hooks.

A description of a tool performing intrusion detection at the application level can be found in [Almgren and Lindqvist, 2001]. This system leverages the extension mechanism of the Apache web server to implement an audit data source. The collected audit information is used to monitor the behavior of the web server.

Network-based IDSs

Network-based IDSs (NIDSs) detect attacks by analyzing the network traffic exchanged on a network link. Therefore, in network-based IDSs, the E-boxes are network sniffers.

The analysis of the content of network packets can be performed at different levels of sophistication, e.g., performing simple pattern matching on the header and/or the payload of a packet, or exploiting knowledge about the protocol followed by the communication. Higher-level analysis supports more sophisticated analysis of the data, but it is usually slower and requires more resources.

Network-based IDSs are very appealing because they are easy to deploy, and have a minimal or no impact on the monitored hosts. On the other hands, changes in network technology could impair the usefulness of NIDSs:

- High-speed networks might increase the network throughput beyond the capabilities of sniffers.
- Switched networks make it more difficult to choose the location where the NIDSs should be placed.
- The adoption of encryption of the communications reduces or completely prevents NIDSs from accessing the contents of network connections.

Furthermore, studies have shown that, if particular care is not taken, NIDSs are vulnerable to a class of attacks (called insertion and evasion) that take advantage of the physical and logical separation of the NIDSs from their monitored hosts to undermine their detection capability. [Ptacek and Newsham, 1998; Malan et al., 2000; Handley et al., 2001].

3.6 Usage Frequency

Dynamic intrusion detection tools analyze in real time the activity of the system to be protected. Audit data is examined as soon as it is produced. The advantage of this approach is that system activities can be analyzed timely and thus, a proper response can be issued when an attack is detected. However, real-time collection and analysis of audit data may introduce a significant overhead.

Static tools are run offline at specific intervals. They analyze a snapshot of the system state and produce an evaluation of the security of that state. They do not provide any security in between two consecutive runs, and therefore, in case of a successful attack, they can be used only for postmortem analysis. However, by running only occasionally, they may perform a more thorough analysis without having an unacceptable impact on the performance of the monitored system.

3.7 IDS Cooperation and Alert Correlation

A recent trend in intrusion detection is to use, at the same time, different intrusion detection systems and correlate the analysis results and corresponding alerts. This approach aims at achieving higher-level descriptions of attacks or a more condensed view of the security issues highlighted during the analysis.

A number of requirements and possible application scenarios for IDS interoperation have been described [Kahn et al., 1998]. For example, two IDSs might cooperate in the following ways:

- Analyzing each other's generated alerts. This would be especially useful if they use different detection techniques, e.g., one IDS is anomaly-based and the other one is misuse-based.
- Complementing each other's coverage, e.g., two IDSs may be both host-based but located on two different hosts involved in an attack. These IDSs would produce an aggregate report on the malicious activity.
- Reinforcing each other's alerts to keep the number of false positives low.

Alert correlation is a multistep process that receives alerts from different intrusion detection systems as input and produces a high-level description of the malicious activity on the network. The core of this book describes the different phases of the correlation process and the challenges associated with each phase. We emphasize open problems and discuss current approaches to solve some of the issues. After a detailed discussion of the correlation phases, we present an approach to perform distributed correlation in Chapter 7 that addresses scalability requirements for correlation in very large network installations.



<http://www.springer.com/978-0-387-23398-7>

Intrusion Detection and Correlation
Challenges and Solutions

Kruegel, C.; Valeur, F.; Vigna, G.

2005, XIV, 118 p., Hardcover

ISBN: 978-0-387-23398-7