

Jiankun Hu

## Contents

<b>13.1 Background Material</b> .....	<b>236</b>
13.1.1 Basics of Computer Operating Systems	236
13.1.2 Basics of Networking .....	237
13.1.3 Basic Concepts in Network Security...	239
<b>13.2 Intrusion Detection System</b> .....	<b>239</b>
13.2.1 Basics of IDS .....	240
13.2.2 Network IDS .....	240
13.2.3 Host-Based Anomaly IDS .....	242
<b>13.3 Related Work on HMM-Based Anomaly Intrusion Detection</b> .....	<b>245</b>
13.3.1 Fundamentals of HMM .....	245
13.3.2 How to Apply the HMM to HIDS? ....	247
<b>13.4 Emerging HIDS Architectures</b> .....	<b>250</b>
13.4.1 Data Mining Approach Combining Macro-Level and Micro-Level Activities .....	250
13.4.2 Two-Layer Approach .....	252
13.4.3 New Multi-Detection-Engine Architecture .....	253
<b>13.5 Conclusions</b> .....	<b>254</b>
<b>References</b> .....	<b>254</b>
<b>The Author</b> .....	<b>255</b>

Network security has become an essential component of any computer network. Despite significant advances having been made on network-based intrusion prevention and detection, ongoing attacks penetrating network-based security mechanisms have been reported. It is being realized that network-based security mechanisms such as firewalls or intrusion detection systems (IDS) are not effective in detecting certain attacks such as insider

attacks and attacks without generating significant network traffic. The trend of network security will be to merge host-based IDS (HIDS) and network-based IDS (NIDS). This chapter will provide the fundamentals of host-based anomaly IDS as well as their developments. A new architectural framework is proposed for intelligent integration of multiple detection engines. The novelty of this framework is that it provides a feedback loop so that one output from a detection engine can be used as an input for another detection engine. It is also illustrated how several schemes can be derived from this framework. New research topics for future research are discussed. The organization of this chapter is as follows. Section 13.1 is about background material. It provides a brief introduction to computer (host) operating systems and networking systems, which are needed to understand computer and computer network security and IDS. Section 13.2 presents the basic concepts in HIDS and their developments. Practical examples are provided to illustrate the implementation procedures in a step-by-step approach. Section 13.3 introduces powerful hidden Markov models (HMM) and HMM-based anomaly intrusion detection schemes. Section 13.4 discusses emerging HIDS architectures. It also proposes a new theoretic framework for designing new IDS architectures. Conclusions are given in Sect. 13.5. Much material on HIDS schemes is drawn from the author's own published research work. This chapter is suitable as a text for final-year undergraduate students or postgraduates in advanced security courses. It is also useful as a reference for academic researchers intending to conduct research in this field.

## 13.1 Background Material

The intrusion detection problem and its solutions have involved many disciplines. In this part, we will introduce the basics of computer operating systems, the basics of networking, and the basic concepts of network security, which will provide the necessary preliminaries for the discussion of host-based anomaly intrusion detection in the remaining parts.

### 13.1.1 Basics of Computer Operating Systems

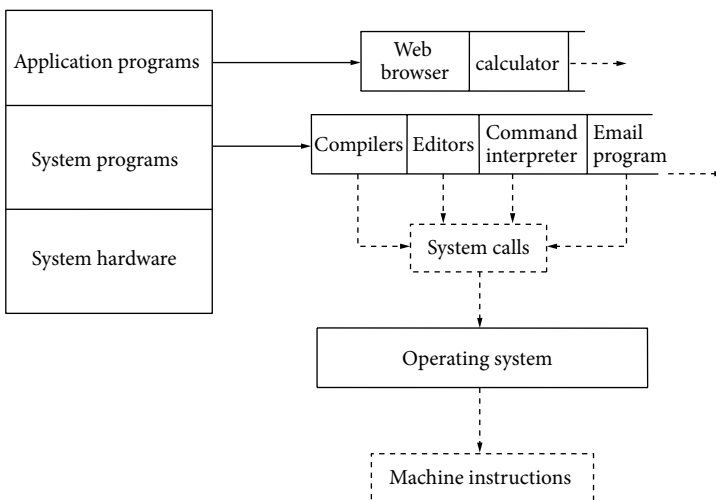
It is well known that what really works in a computer is software. Computer software can perform many functions, such as playing music, sending e-mail, and searching the Internet. Generally computer software can be divided into two categories: system programs that manage the operation of the computer itself, and application programs that perform the actual work the users want. The operating system is the fundamental system program whose task is to control all the computer's resources that application programs need [13.1]. Basically, a computer system consists of one or more computer processing units (CPUs), memories, network interfaces, and input/output devices. An operating system is deployed to control the operation of these components. An architectural structure of computer systems is de-

picted in Fig. 13.1 [13.1,2]. There are two functional sublayers in system programs. In the upper functional layer are programs that are application-independent, such as system compilers and editors, which are used to support the above-mentioned application programs. This type of system program shares a common feature with the application program: i.e., they both run in user mode. Users can write and install these programs. The upper sublayer system programs differ from application programs in two ways:

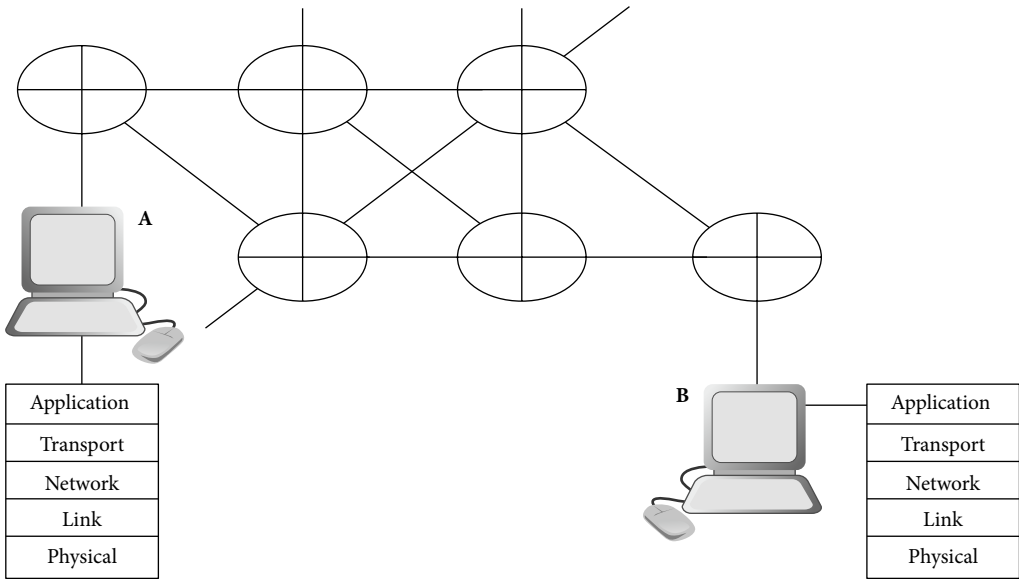
1. System programs are more generic and application-independent.
2. These system programs can interact directly with the lower functional sublayer system program, which is called the operating system. Therefore, they are sometimes called privileged programs.

Usually emphasis is placed on the second characteristic. Hence, programs such as sendmail (e-mail) and lpr (printing) are considered as privileged system programs.

The activities specified in the application programs will be converted into instructions that will interact with the lower sublayer of system programs. This lower sublayer of system programs is called the operating system and the instructions fed into the operating system are called system calls. The operating system directly interacts with system hardware such as the CPU, memory, and registers. The sys-



**Fig. 13.1** Architecture of a computer system



**Fig. 13.2** Architecture of a Transmission Control Protocol (TCP)/Internet Protocol (IP) wide-area network

tem calls will be converted into system hardware instructions, i.e., machine instructions. The software in the operating system runs in kernel mode or supervisor mode, which cannot be accessed directly by users.

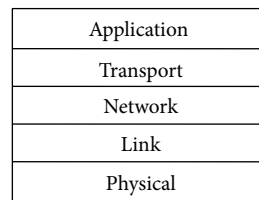
### 13.1.2 Basics of Networking

As shown in Fig. 13.2, a Transmission Control Protocol (TCP)/Internet Protocol (IP) wide-area network has a network core consisting of interconnected routers. Communication between two network-connected devices A and B is via exchange of messages over the network. The message is encoded in packets and delivered via the TCP/IP protocol shown in Fig. 13.3. The TCP/IP protocol stack is a layered structure. The application layer handles application-specific issues. The transport layer handles the flow of data between two hosts. There are two transport protocols, TCP and User Datagram Protocol (UDP), for data flow control in the TCP/IP suite. TCP provides reliability of data flow by using the mechanism of acknowledging the received data. UDP provides a much simpler service by sending packets of data from host to host without concern whether the data sent has been received or not. The network layer is responsible for delivering,

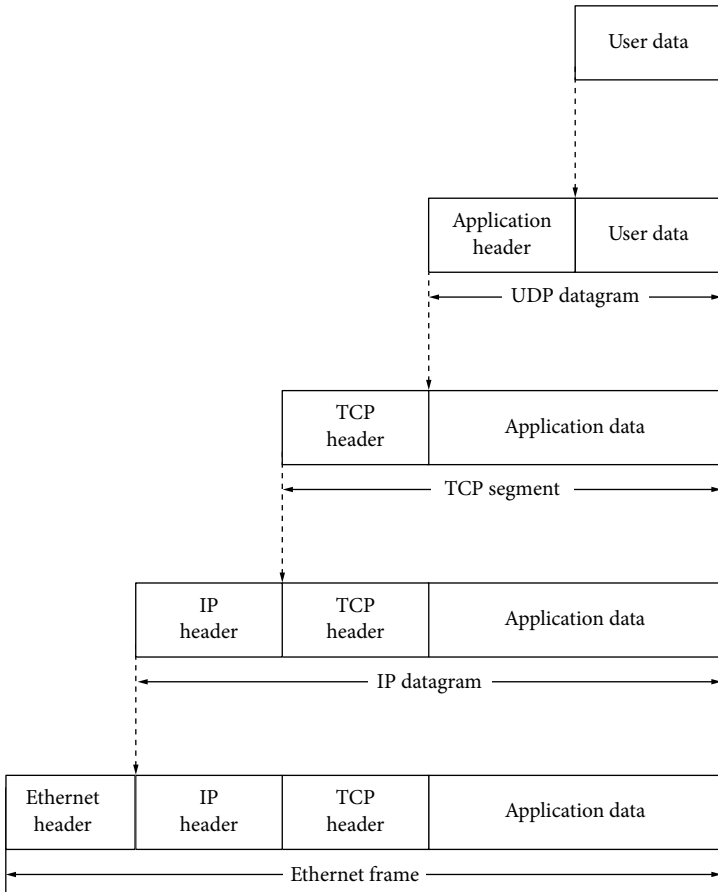
or routing, the packets to the correct destination over the network. The link layer deals with issues related to connection link properties such as Ethernet link and Wi-Fi link.

The physical layer handles issues related to the interfacing to the actual transmission medium of the link such as twisted-pair copper wires. Both TCP and UDP use 16-bit port numbers to identify applications. For example, a File Transfer Protocol (FTP) server provides this service on TCP port 21. The telnet server is on TCP port 23. The functions of each layer are encoded via each layer header. A typical example of encapsulation of data as it goes down the protocol stack is shown in Fig. 13.4 [13.3, 4].

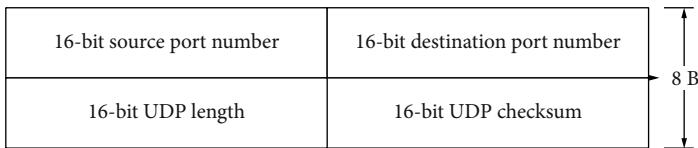
As shown in Fig. 13.5, the header of the UDP uses port numbers to represent what applications are running on the host and the recipient.



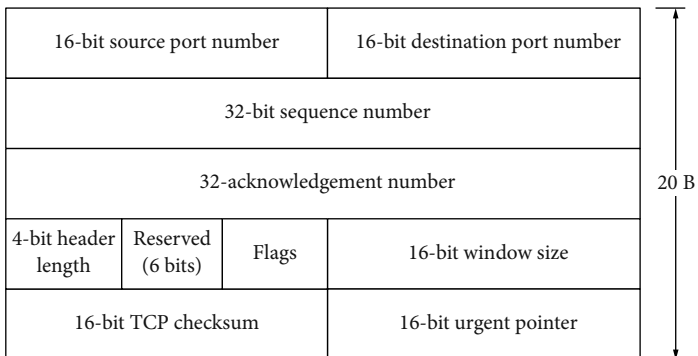
**Fig. 13.3** TCP/IP protocol stack



**Fig. 13.4** Protocol data unit process [13.3]. *UDP* User Datagram Protocol



**Fig. 13.5** UDP header



**Fig. 13.6** TCP header

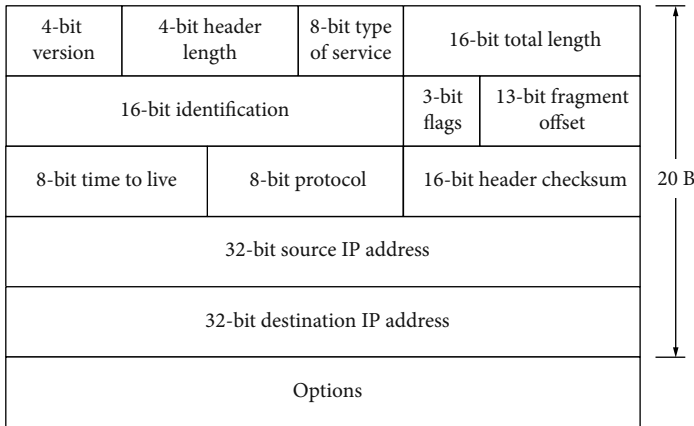


Fig. 13.7 IP header

In the TCP header shown in Fig. 13.6, TCP uses the mechanism of acknowledging packets received to improve the reliability of data transmission. Also the window size can be used by the receiver to indicate the maximum data rate it can handle at the time, which can help control data flow between end-to-end points.

The IP header shown in Fig. 13.7 provides source and destination addresses which are needed to route the packet to the destination timely and correctly.

### 13.1.3 Basic Concepts in Network Security

Security has attributes of confidentiality, integrity, and availability. The commonly used security services in a networked environment are confidentiality, authentication, integrity, nonrepudiation, and availability [13.5, 6].

**Confidentiality** This refers to the secrecy characteristics which prevent unauthorized access to the sensitive information. Confidentiality of data is often achieved by cryptographic encryption, i.e., a mathematical transformation to make the transformed data not intelligible to those who do not possess the decryption key.

**Authentication** This refers to verifying that the communicating partner is who it claims to be. Conventionally this is achieved by applying cryptographic authentication protocols. Conventional cryptography is either a knowledge-based mechanism, i.e., based on “what you know”, such as

a password or a personal identification number (PIN), or a possession-based mechanism such as token possession. The combination of a PIN and a token is also used. However, all of these mechanisms have a fundamental flaw in identifying genuine users. The PIN can be forgotten or discovered and the token can be lost or stolen and there is no way to identify who is presenting the token and the PIN. It is well known that the face, fingerprint, etc. possess very unique identity characteristics of an individual. Biometrics-based authentications and biocryptography are emerging as promising solutions. Interested readers are referred to [13.7] and Chap. 6 in [13.8].

**Integrity** This refers to the absence of improper alterations of data or information.

**Nonrepudiation** This refers to the fact that once a person has created and sent a message, he or she cannot deny having sent the message and being the creator of the message.

**Availability** This refers to the readiness to provide a set of predefined services at a given time [13.9].

Security and dependability are closely related. There is a trend to integrate them within the same framework. Interested readers are referred to [13.9] for the latest development on this topic.

## 13.2 Intrusion Detection System

In this part, we introduce the basic and general concepts in IDS. The principles of NIDS are described.

The weakness of NIDS is discussed, which leads to the section on solutions addressing this weakness. In this section, a popular system-call-based IDS is introduced where detailed implementation procedures are illustrated using an example. This will pave the way to understand a more advanced HMM for anomaly IDS in Sect. 13.3.

### 13.2.1 Basics of IDS

Although cryptography has provided a powerful tool for computer and computer network security, it focuses more on attack prevention [13.6]. Unfortunately prevention of all possible attacks is impossible. Successful attacks have been happening and will always happen. Therefore, a second line of defense is needed where the IDS comes to play an important role. Intrusion refers to unauthorized activity including unauthorized access to data or a computing service [13.10]. Typical intrusion examples are:

- Unauthorized login: attackers can attempt to log in by password-guessing or explore networking protocol vulnerabilities. For example, attackers attacking SUNOS 4.1.x can explore vulnerabilities related to its file sharing protocol to gain unauthorized network login [13.10].
- Data theft: a spy-agent introduced via a Web download or a Trojan embedded in an e-mail can collect data from the affected host/server.
- Denial of service: attackers can generate an enormous amount of network traffic to congest the normal operation of the network server.

IDS attempts to identify that such intrusion activity has been attempted, is occurring, and/or has already occurred. Several benefits of IDS are:

- It can generate alarms and trigger either a manual or an automated response to prevent further damage.
- It can help assess the damage done and provide court evidence of intruders, which in turn provides a deterrence to attackers.

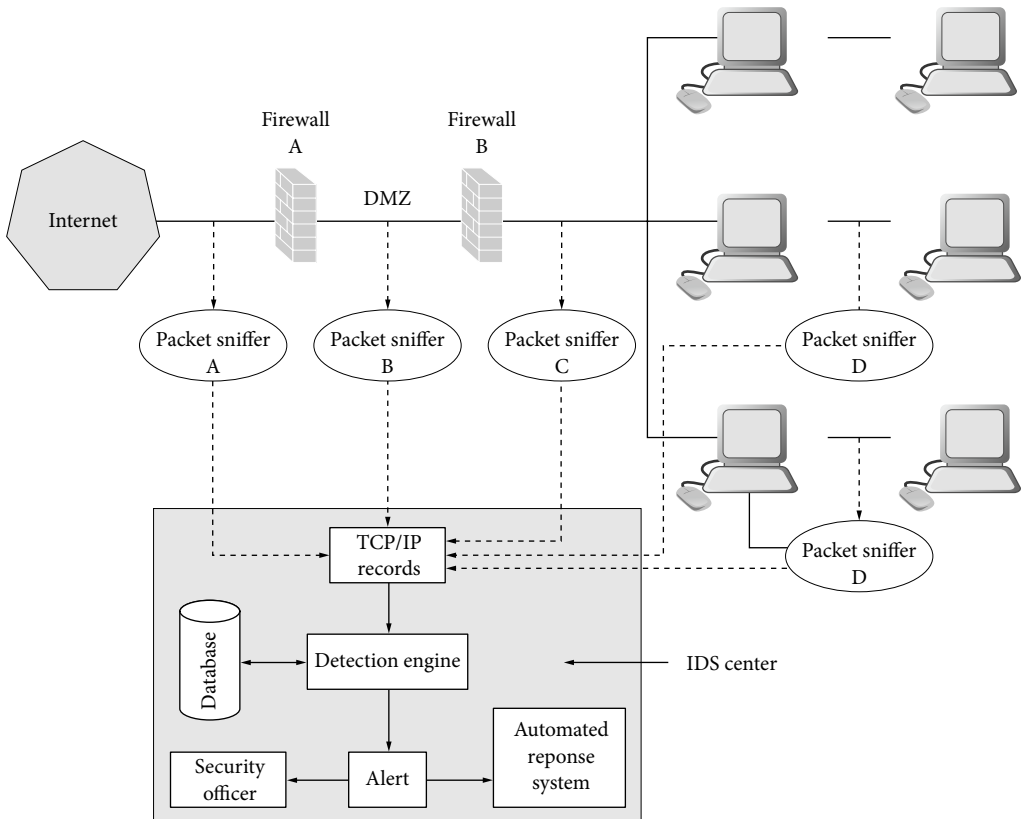
There are several ways to classify IDS. One way is classifying IDS into two categories, namely, NIDS and HIDS (HIDS), which focuses on what physical targets the IDS tries to protect. A NIDS mainly inspect network activities such as network packet traffic and network protocols present via those pack-

ets. A HIDS inspects computing activities happening within a host such as file access and execution of files. Another method of IDS classification is to classify IDS into misuse detection IDS and anomaly detection IDS. Misuse IDS inspect a suspicious event against a large a priori built attack signature database to find a match. This mechanism is very effective for attacks whose characteristics are known a priori. Anomaly IDS inspect whether an event is abnormal or not. It is a promising mechanism for detecting attacks whose characteristics are not known a priori.

### 13.2.2 Network IDS

A standard NIDS architecture is shown in Fig. 13.8 [13.10]. In this architecture, one or more network packet sniffers capture network traffic entering and leaving the protected network. These packets are then sent to the IDS center for processing. In the IDS center, the network packets received will be classified into various TCP/IP traffic records, e.g., TCP traffic records and UDP traffic records. Then these data will be fed into the network intrusion detection engine for intrusion analysis. A database can log those raw TCP/IP records. The database can also provide intrusion signatures so that the detection engine can search for a match between the stored intrusion patterns and retrieved patterns from incoming network traffic. This is the main operational process for misuse intrusion detection. If the detection engine has found sufficient evidence of intrusion attacks, it will generate alarms which will be sent to the system operators and/or trigger the automated response system.

As shown in Fig. 13.8, there are several popular ways of deploying the network sniffer [13.10]. Packet sniffer A detects attacks originating outside the organizational networks such as denial of service, mapping, scans, and pings. Packet sniffer B is located in the demilitarized zone between the inner and the outer firewalls. It detects attacks that penetrate the outer firewall of the organization. Packet sniffer C is placed inside the firewall and the network traffic exit of the internal organizational networks. It monitors unauthorized or suspicious network traffic leaving the internal organizational networks. Packet sniffer D is placed within the organizational networks to monitor suspicious network traffic flowing between internal systems, which is particularly useful for de-



**Fig. 13.8** A standard network-based intrusion detection system architecture [13.10]. DMZ demilitarized zone

20:40:20:904814 EnGarde.com.80 > EnGarde.com.80:17112001:1711552021 90) win 4096 <mss 1460>

**Fig. 13.9** Broadcast attack example [13.10]

tecting attacks launched from inside the network. For instance, a host could be compromised and then become a platform for launching new attacks.

The core of the NIDS is the detection engine. A detection engine can be of signature-based misuse IDS or anomaly IDS or a combination of both. The signature-based detection engine in the NIDS will check the network packet traffic pattern against the prestored intrusion packet traffic pattern. An alarm will be produced if there is a match. In the broadcast attack example shown in Fig. 13.9, a network packet has been sent to EnGarde.com.80 with the source address being EnGarde.com.80, which is exactly the same as the destination address. In most IP implementations this will cause operation of the

TCP/IP stack to fail, which leads to the machine crashing [13.10]. Therefore, the identical source and destination addresses will become a pattern of broadcast attacks. This pattern will be prestored in the database. When the captured packet is sent to the detection engine, the detection engine will check whether it matches this pattern or not to detect such broadcast attacks. Similarly, the detection engine can check the database against many other patterns.

New network attacks are occurring constantly. A signature-based NIDS is very effective in handling network attacks with known patterns but is very poor at detecting new network attacks. An anomaly NIDS is promising in handling such attacks. An sno-

maly NIDS establishes a nominal network traffic profile using clean data and then detects whether there is a substantial difference between the test data and the nominal profile. An intrusion alarm is triggered if a threshold has been reached. The majority of NIDS work on the statistical distribution of TCP and UDP traffic with attributes such as volume, destination, source, and connection time.

### Summary of Benefits/Weaknesses of NIDS

In general, a NIDS is very useful in detecting network-related attacks. The timely detection of network intrusions can help generate timely automated or manual responses and notification such as paging NIDS operators, reconfiguring the routers/firewalls, and shutting down targets being attacked. It is very useful in damage control. As advanced NIDS can help trace back the source of attacks and provide court evidence, it serves both as a deterrent against attacks and evidence supporting court proceedings.

Although there are still many issues in the research of NIDS, the following two issues appear to be the most challenging. The first challenge is the high false alarm rate in the anomaly NIDS, which has formed a hurdle for practical applications; therefore, how to reduce this high false alarm rate has become an intensive ongoing research topic. The second challenge is to detect attacks that tend to generate little network traffic and attacks originating from inside the protected network. There are some detrimental network attacks that do not generate significant network traffic. Very recently, the Zotob worm disabled thousands of computer systems, bringing business to a halt. The Australian car manufacturer Holden lost \$6 million in this Zotob worm attack and the other victims of the attack reported in the press include the Financial Times, CNN, ABC, the New York Times, UPS, General Electric, Canadian Bank of Commerce, DaimlerChrysler, General Electric, SBC Communications, and CNN [13.11, 12]. The Zotob worm exploits the Microsoft Windows plug and play buffer overflow vulnerability on TCP port 445 and installs a FTP server on the victim's machine to download its malicious code, which can repeatedly shut down and reboot the machine. Once it gets on a corporate network, it can pass from machine to machine. Because of the variant nature of the worm, it has penetrated the firewall. It

can also pass NIDS as it does not generate a large amount of traffic. It is observed that many network attacks, even including some network traffic attacks, are from compromising a machine and propagating to other machines on the network. Network traffic statistics profiles are infeasible for this task. Therefore, an effective IDS scheme is to build a host-based anomaly IDS, to complement the NIDS, which is the focus of this chapter.

### 13.2.3 Host-Based Anomaly IDS

HIDS can also be classified into misuse HIDS and anomaly-based IDS. A misuse HIDS detects intrusions by inspecting the patterns of computing activities such as the usage of the CPU, memory and file access against the prestored signatures of host-based intrusions. Many commercial virus-checking software programs falls into this category. Similarly a signature-based HIDS is very effective in detecting attacks that are already known and is poor at detecting new attacks, which occur daily. This will require anomaly HIDS.

Historically the most fundamental principle in IDS including NIDS originated from anomaly HIDS research based on Denning's pioneering work [13.13]. The principle is a hypothesis that security violations can be detected by monitoring a system's audit records for abnormal patterns of system usage. It is suggested that profiles are used to represent the behavior of subjects using statistical measures. Although the first intrusion detection model was a HIDS, extensive research activities have been shifted to NIDS. Several factors have been driving this phenomenon [13.14]:

1. Networking factor: In the Internet age, an overwhelming number of computing applications are network-based. Many security problems that have not been observed before are introduced from this new environment. Examples are denial of service attacks and attacks exploiting other security loopholes related to networking protocols.
2. Real-time and computing resource restraint: Ideally intrusions should be detected as soon as they happen, which can help minimize the potential damage. However, audit data collection and processing for detecting intrusion can involve a large amount of computing re-

sources [13.15]. Therefore, a dedicated hardware and software IDS component is most efficient.

However, as discussed in Sect. 13.2.1, NIDS have encountered the challenge of detecting some non-traffic-sensitivity attacks, which requires the deployment of HIDS. While efforts in NIDS extending to packet content inspection can help address non-traffic-sensitivity attacks, the trend of adopting end-to-end encryption such as the IPSec mechanism makes the task of inspecting packet content by the NIDS infeasible. At the same time, computing power has been dramatically increased in recent years; it is time to invest more research effort into HIDS.

Signature-based IDS are mature and very effective in detecting known attacks. Therefore, this chapter focuses on anomaly IDS. For anomaly HIDS, a number of techniques such as data mining, statistics, and genetic algorithms have been used for intrusion detection on the user-activity and program-activity levels individually [13.14, 16–23]. In [13.20], a novel framework, called MADAM ID, for mining audit data for automated models for intrusion detection, is proposed. This framework uses data mining algorithms to compute activity patterns from system audit data and extracts features from the patterns. Then machine-learning algorithms are applied to the audit records, which are processed according to the feature definitions and generate intrusion detection rules. The data-mining algorithms include meta-classification, association rules, and frequent-episode algorithms. The test results in 1998 conducted by DARPA Intrusion Detection Evaluation showed that the model was one of the best performing of all the participating systems in off-line mode. To detect user anomalies, normal user activity profiles are created and a *similarity score range* (*upper and lower bound*) is assigned to each user's normal pattern set. When in operation, the IDS computes the *similarity score* of the current activity's patterns. If this score is not in the *similarity score range*, then the activity is considered as abnormal. Such user-behavior-based approaches can adapt to slowly changing user behavior, but fail to distinguish intrusion behavior and rapidly changing behavior of the legitimate user.

### System-Call-Based Patterns

On the program-activity level (micro-level), anomaly detection systems based on system calls have

received growing attention by many researchers since the successful initiative of Forrest et al. [13.24]. Forrest et al. [13.24] proposed defining a normal profile by short-range correlations in system calls of privileged processes. There are several advantages of this approach over user-behavior-based approaches. First, root processes are more dangerous than user processes. Second, they have a limited range of behavior which is more stable over time. In principle, each program is associated with a set of system call sequences that it can generate. The execution paths through the program will determine the ordering of these sequences [13.2, 24]. One challenge is that a normal program is associated with a huge set of system calls and different execution of the program may produce different system call sequences. Forrest et al. discovered that the local ordering of system calls appears to be very consistent. Therefore, such a short system call sequence ordering serves as a good representation of program behavior. Their results show that short sequences of system calls define a stable signature that can detect some common sources of anomalous behavior in the system event stream. Within this framework, each system generates its own normal database based on the software and hardware configuration and usage patterns. These normal databases will be compared against abnormal events collected during operations. For convenience, the IDS scheme checking short system call sequences against the prestored patterns of short system call sequences that have been generated during clean normal operation condition is called the system call database approach [13.25].

### Illustrative Intrusion Detection Example Using the System Call Database Approach [13.24]

**Step 1: Building a Nominal Database** Use a sliding window of size  $k + 1$  sliding across the trace of system calls that have been produced during normal and clean operation conditions. Record calls ordering within the window. If  $k = 3$ , the following system call sequences are produced during the normal operational conditions:

Open, read, mmap, mmap, open,  
getrlimit, mmap, close.

As we slide the window size across the sequence, we record all different call sequences following each

**Table 13.1** Short sequence ordering calculated from the first window

Call	Position 1	Position 2	Position 3
open	read	mmap	mmap
read	mmap	mmap	
mmap	mmap		
mmap			

**Table 13.2** Short sequence ordering calculated from the second window

Call	Position 1	Position 2	Position 3
read	mmap	mmap	open
mmap	mmap	open	
mmap	open		
open			

**Table 13.3** Short sequence ordering calculated from the third window

Call	Position 1	Position 2	Position 3
mmap	mmap	open	getrlimit
mmap	open	getrlimit	
open	getrlimit		
getrlimit			

**Table 13.4** Short sequence ordering calculated from the fourth window

Call	Position 1	Position 2	Position 3
mmap	open	getrlimit	mmap
open	getrlimit	mmap	
getrlimit	mmap		
mmap			

call within the window. For the subsequent five windows, the database illustrated via Tables 13.1–13.6 is produced.

Table 13.6 is the database of normal patterns. When a new trace of system calls is produced, we repeat the same procedure using the same sliding window. For instance, suppose a new trace of system calls is produced by replacing the mmap call with the open call in the fourth position of the sequence, i.e.,

Open, read, mmap, open, open,  
getrlimit, mmap, close.

Then the short sequence patterns shown in Table 13.7 are produced.

**Table 13.5** Short sequence ordering calculated from the fifth window

Call	Position 1	Position 2	Position 3
open	getrlimit	mmap	close
getrlimit	mmap	close	
mmap	close		
close			

**Table 13.6** Combining Tables 13.1–13.5, where different short sequences no longer than 4 are recorded

Call	Position 1	Position 2	Position 3
open	read	mmap	mmap
	getrlimit	mmap	close
read	mmap	mmap	open
mmap	mmap	open	getrlimit
	open	getrlimit	mmap
	close		
getrlimit	mmap	close	
close			

**Table 13.7** Short sequence patters derived from the new trace

call	osition 1	Position 2	Position 3
<i>open</i>	<i>read</i>	<i>mmap</i>	<i>open</i>
<i>open</i>	<i>open</i>	<i>getrlimit</i>	<i>mmap</i>
<i>open</i>	<i>getrlimit</i>	<i>mmap</i>	<i>close</i>
<i>read</i>	<i>mmap</i>	<i>open</i>	<i>open</i>
<i>mmap</i>	<i>open</i>	<i>open</i>	<i>getrlimit</i>
	<i>close</i>		
getrlimit	mmap	close	
close			

From a comparison with the normal database, the new trace of system calls has four mismatches, which are highlighted in italics. For a sequence of length  $L$ , the maximum number of mismatches is given by

$$k(L - k) + (k - 1) + (k - 2) + \dots + 1 = k(L - (k + 1)/2) . \tag{13.1}$$

The big advantage of this algorithm is its simplicity with  $O(N)$  complexity, where  $N$  is the length of the trace. Some implementations can analyze at the rate of 1,250 system calls/s [13.24].

Since then, a certain number of novel schemes have been further discovered. Recently the HMM has become a popular tool in anomaly HIDS and has attracted much research activity.

### 13.3 Related Work on HMM-Based Anomaly Intrusion Detection

HMM-based anomaly intrusion detection is a very promising and popular tool. In this part, we introduce the fundamentals of HMM. Several schemes of HMM-based anomaly IDS are described which are based on system calls.

#### 13.3.1 Fundamentals of HMM

##### What Is the HMM?

The HMM is a double stochastic process. An example of a four-state HMM process is shown in Fig. 13.10. The upper layer is a Markov process whose states are not observable. The lower layer is a normal Markov process where emitted outputs can be observed. The observed outputs are probabilistically determined by the upper-layer states. The HMM has four states,  $X = \{X_1, X_2, X_3, X_4\}$ , which are linked to the emitted output,  $O = \{O_1, O_2, O_3, O_4\}$ , via probability transition parameters. The transitions among states and between states and observed outputs are random but can be described in a probabilistic function. HMM can be roughly classified into discrete and continuous depending on whether observations and distinct states are discrete and finite or continuous. As intrusion events are of finite and discrete nature, we limit our discussions to discrete HMM. Warrender et al. [13.22] have pointed out that for a number of machine-learning approaches such as rule induction, HMM

can be used to learn the concise and generalizable representation of the “self” identity of a system program by relying on the program’s run-time system calls. The models learned were shown to be able to accurately detect anomalies caused by attacks on the system programs.

A mathematical description of HMM is given as follows [13.26, 27].

Assume:

- $N$  is the number of hidden states of the HMM.
- $M$  is the number of distinct observation symbols.
- $L$  is the number of observation sequences
- $X$  is the set of hidden states  $X = \{X_1, X_2, \dots, X_N\}$ .
- $V$  is the set of possible observation symbols  $V = \{V_1, V_2, \dots, V_M\}$ .
- $\pi$  is the initial state distribution  $\pi = \{\pi_i\}$ , where  $\pi_i = P(q_1 = X_i)$ ,  $1 \leq i \leq N$ . It is the probability of being in state  $X_i$  at  $t = 1$ .
- $\Lambda$  is the state transition probability matrix  $\Lambda = \{\alpha_{ij}\}$ , where  $\alpha_{ij} = P\{q_{t+1} = X_j, q_t = X_i\}$ ,  $1 \leq i, j \leq N$ . It is the probability of being in state  $X_j$  at time  $t + 1$ , given that the model was in state  $X_i$  at time  $t$ .
- $B$  is the observation probability distribution,  $B = \{\beta_j(k)\}$ , where  $\beta_j(k) = P(v_k \text{ at } t | q_t = X_j)$ ,  $1 \leq j \leq N, 1 \leq k \leq M$ . It is the probability of observing symbol  $v_k$  at time  $t$ , given that the model is in state  $X_j$ .
- $Q$  is the sequence of hidden states,  $Q = \{q_1, q_2, \dots, q_t, \dots, q_T\}$ , where  $q_t$  is the model’s state at time  $t$ .

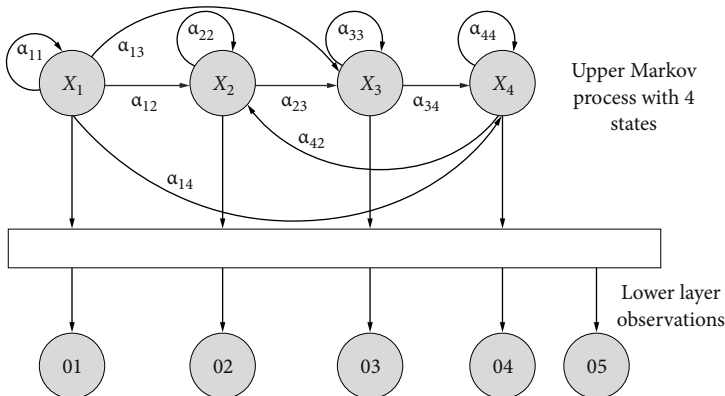


Fig. 13.10 A four-state hidden Markov model (HMM) process

- $O$  is the sequence of observations,  $O = \{O_1, O_2, \dots, O_t, \dots, O_T\}$ , where  $O_t, 1 \leq t \leq T$ . It is the observation symbol observed at time  $t$ .
- $\lambda$  is the whole HMM,  $\lambda = (A, B, \pi)$ .
- $P(O|\lambda)$  is the the probability of the occurrence of observation sequence  $O$ , given the HMM  $\lambda$ .
- $P(O, Q|\lambda)$  is the joint probability of the occurrence of the observation sequence  $O$  for the state sequence  $Q$ , given the HMM  $\lambda$ .

### Probability Constraints

1. Initial distribution

$$\begin{aligned} \pi_i &\geq 0, \quad 1 \leq i \leq N, \\ \sum_{i=1}^N \pi_i &= 1. \end{aligned} \quad (13.2)$$

2. Transition probability distribution

$$\begin{aligned} \alpha_{ij} &> 0, \quad 1 \leq i, j \leq N, \\ \sum_{j=1}^N \alpha_{ij} &= 1. \end{aligned} \quad (13.3)$$

3. Observation probability distribution  $B$

$$\begin{aligned} \beta_j(k) &\geq 0, \quad 1 \leq j \leq N, \\ 1 \leq k \leq M, \quad \sum_{k=1}^M \beta_j(k) &= 1. \end{aligned} \quad (13.4)$$

### Three General HMM Problems

Most HMM applications can be classified into one or more of the following three general HMM problems [13.26, 27]:

*Problem 1* Given the HMM  $\lambda = (A, B, \pi)$ , estimate the probability of the occurrence of the observation sequence  $O = \{O_1, O_2, \dots, O_t, \dots, O_T\}$ , i.e., compute  $P(O|\lambda)$ .

*Problem 2* Given observation sequence  $O = \{O_1, O_2, \dots, O_t, \dots, O_T\}$  and the HMM  $\lambda = (A, B, \pi)$ , find the state sequence  $Q = \{q_1, q_2, \dots, q_T\}$  such that the joint probability  $P(O, Q|\lambda)$  is maximized.

*Problem 3* Given the observation sequence  $O$ , find the HMM parameters  $\lambda = (A, B, \pi)$  such that  $P(O|\lambda)$  is maximized.

In the context of IDS, problem 1 can be interpreted as given a model and an observation sequence, what is the probability that the observation sequence was produced by the model. If the model is reliable, then a high-probability result means that

the observation sequence tested is intrusion-free. Similarly a low-probability result means that the observation sequence tested is abnormal.

Problem 2 is a decoding problem where we try to find the optimal sequence of hidden states for the given HMM and an observation sequence. Problem 3 is a training issue where we try to find a model that best fits the input sequence of observations. Apparently problem 3 is about training of the HMM given available clean system calls and problem 1 is about testing whether a given trace of system calls is intrusion-free or not.

**Solution to Problem 1 [13.26, 27]: Forward and Backward Procedure** For the given HMM, the joint probability of having the observation sequence  $O$  and the hidden states  $Q$  is given as

$$\begin{aligned} P(O, Q|\lambda) &= \prod_{t=1}^T P(O_t|q_t, \lambda) \\ &= \beta_{q_1}(O_1)\beta_{q_2}(O_2)\dots\beta_{q_T}(O_T) \end{aligned} \quad (13.5)$$

under the assumption that each observation is independent. By conditional probability, we have

$$\begin{aligned} P(O|\lambda) &= \sum_Q P(O, Q|\lambda)P(Q|\lambda) \\ &= \sum_Q \pi_{q_1}\beta_{q_1}(O_1)\alpha_{q_1q_2}\beta_{q_2}(O_2) \\ &\quad \times \alpha_{q_2q_3}\dots\alpha_{q_{(T-1)q_T}}\beta_{q_T}(O_T). \end{aligned} \quad (13.6)$$

It is computationally infeasible to compute the probability using the above formula owing to the exponential combinatory number derived from  $O, Q$ . The complexity is at the order of  $O(2N^T T)$ . A conventional solution is to use the following forward procedure scheme, which is an iterative algorithm.

**Forward Procedure Scheme** Define a forward variable

$$\sigma_t(i) = P(O_1, O_2, \dots, O_t, q_t = X_i|\lambda).$$

The probability  $P(O|\lambda)$  can be computed via the following iterative formula:

1. Initialization

$$\sigma_1(i) = \pi_i\beta_i(O_1), \quad 1 \leq i \leq N. \quad (13.7)$$

2. Iteration

$$\begin{aligned} \sigma_{t+1}(j) &= \left[ \sum_{i=1}^N \sigma_t(i)\alpha_{ij} \right] \beta_j(O_{t+1}), \\ 1 \leq t \leq T-1, \quad 1 \leq j \leq N. \end{aligned} \quad (13.8)$$

## 3. Termination

$$P(O|\lambda) = \sum_{i=1}^N \sigma_T(i). \quad (13.9)$$

The forward procedure scheme is at the order of  $O(N^2T)$  complexity, which is much more efficient. This is a popular approach used in many applications.

**Solution to Problem 3 [13.26, 27]: Baum–Welch Algorithm** Define

$$\begin{aligned} \xi_t(i, j) &= P(q_t = X_i, q_{t+1} = X_j | O, \lambda) \\ &= \frac{P(q_t = X_i, q_{t+1} = X_j, O | \lambda)}{P(O | \lambda)}. \end{aligned} \quad (13.10)$$

Define

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j). \quad (13.11)$$

Then we have following HMM parameter updating formula:  $\bar{\pi}_i$  = expected number of times in state  $i$  at time  $t = 1$ .

$$\begin{aligned} \bar{\pi}_i &= \gamma_1(i), \\ \bar{\alpha}_{ij} &= \frac{\text{exected\_number\_of\_transitions\_from\_state\_i\_to\_state\_j}}{\text{exected\_number\_of\_transitions\_from\_state\_i}}, \\ \bar{\alpha}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \\ \bar{\beta}_j(k) &= \frac{\text{exected\_number\_of\_times\_in\_state\_j\_and\_observed\_symbol\_v\_k}}{\text{exected\_number\_of\_times\_in\_state\_j}} \\ &= \frac{\sum_{t=1}^T \gamma_t(i)}{\sum_{O_t=k}^T \gamma_t(i)}. \end{aligned} \quad (13.12)$$

**Baum–Welch Algorithm Training Procedure**

*Step 1* Initialize HMM parameter  $\lambda_0$ . A common approach is to assign random values.

*Step 2* Update the model parameter  $\lambda$  based on its previous value with the new observed sequence using (13.13).

*Step 3* Compute  $P(O|\lambda_0)$  and  $P(O|\lambda)$ . If  $P(O|\lambda) - P(O|\lambda_0) < \Delta$  (where  $\Delta$  is the convergence threshold), go to step 5.

*Step 4* Else set  $\lambda \rightarrow \lambda_0$ , and go to step 2.

*Step 5* Stop.

The time and space complexities of the Baum–Welch algorithm are  $O(N(1 + T(M + N)))$  and  $O(N(N + M + TN))$ , respectively [13.27, 28].

**Scaling**

In using the Baum–Welch algorithm for HMM parameter estimation, we may encounter very small numbers and very large numbers in the probability calculation and estimation of the HMM parameters. This will cause a value-underflow problem where intermediate values will be wrongly set to zeros. Also very large numbers can be wrongly capped by the computer precision range. To address this issue, the following scaling with normalization and the logarithm can be deployed [13.27, 29]:

## 1. Initialization with normalization

$$\begin{aligned} \tilde{\sigma}_1(i) &= \pi_i \beta_i(O_1), \quad 1 \leq i \leq N, \\ c_1 &= \frac{1}{\sum_{i=1}^N \tilde{\sigma}_1(i)}, \\ c_1 \tilde{\sigma}_1(i) &\rightarrow \tilde{\sigma}_1(i), \end{aligned} \quad (13.14)$$

## 2. Iteration

$$\begin{aligned} \tilde{\sigma}_{t+1}(j) &= \left[ \sum_{i=1}^N \tilde{\sigma}_t(i) \alpha_{ij} \right] \beta_j(O_{t+1}), \\ &1 \leq t \leq T-1, \quad 1 \leq j \leq N, \\ c_{t+1} &= \frac{1}{\sum_{i=2}^N \tilde{\sigma}_{t+1}(i)}, \\ c_{t+1} \tilde{\sigma}_{t+1}(i) &\rightarrow \tilde{\sigma}_{t+1}(i). \end{aligned} \quad (13.15)$$

## 3. Probability calculation with logarithm scaling

$$\log(P(O|\lambda)) = - \sum_{t=1}^T \log(c_t). \quad (13.16)$$

**13.3.2 How to Apply the HMM to HIDS?**

The raw system calls are of textual type such as read, open, close. We need to convert them into HMM symbol notation. The first step is to determine

the size of the HMM, i.e., the number of hidden states  $M$ , and the set of possible HMM observation symbols  $V = \{v_1, v_2, \dots, v_M\}$ . There are no well-established rules on selecting the size of the HMM. A practical rule is to select  $M$  equal to the number of system calls supported by the operating system. As the actual number of distinct system calls used by a program is often a certain portion of the full set of possible system calls, a common practice is to select  $M$  as the actual number of distinct system calls found in the training data, and  $V$  as the set of actual system calls used [13.14, 18, 19, 27]. This will reduce the computational cost significantly in HMM training and system call sequence testing. Because training data have been collected over a long period of normal operation, it is reasonable that the data collected have covered most of the program's operational activities.

HMM is a powerful tool in modeling and analyzing complicated stochastic process. For example, in weather forecasting, we may observe that a cohort of many frogs singing may lead to a rainy tomorrow. However, this conclusion is based on a probabilistic sense. Also there exists no direct link between the observed frogs singing and tomorrow's rainy state. Experience tells us that there is a hidden link. HMM can be used in such scenarios. HMM has been widely used for protein sequence analysis and speech recognition [13.30–32]. In an effort to find the best modeling method for normal program behavior using system calls, Warrender et al. [13.22] investigated various modeling techniques through extensive experiments. They used the normal database method [13.16, 24], a frequency-based method, data mining, and the HMM to construct detection models from the same normal traces of system calls. Their experimental results have shown that the HMM method can generate the most accurate results on average, although the training cost of the HMM method is very high. Therefore, it is essential to reduce this training cost before it is feasible for practical applications.

### Several Efficient HMM-Based IDS Schemes

To reduce the computational training cost, an improved estimation of HMM parameters from a multiple-observation scheme was proposed by Davis et al. [13.33]. A weighting average is used to combine sub-HMM which have been individually trained by multiple sequences. Gotoh

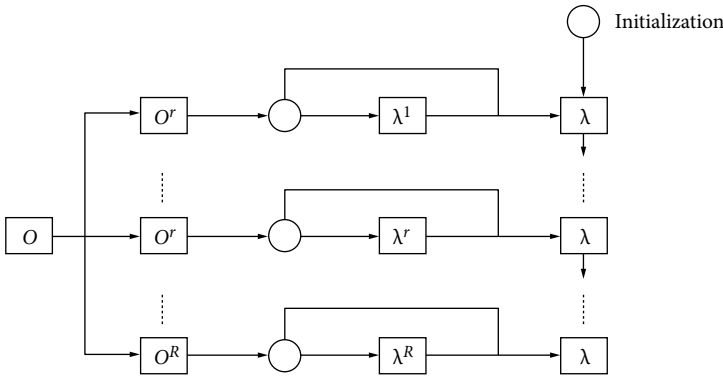
et al. [13.32] discussed alternatives to the usual expectation maximization algorithm for estimation of HMM parameters. They proposed two efficient HMM training approaches, incremental maximum-likelihood estimation and incremental maximum a posteriori probability estimation. It has been experimentally verified that the training of the incremental algorithms is substantially faster than with the conventional method and suffers no loss of recognition performance. There are also other similar approaches to using multiple observations for HMM training [13.34].

Hoang and Hu [13.18] proposed a scheme that can integrate multiple-observation training and incremental HMM training. Hu et al. [13.14] presented an extended work by proposing a simple data preprocessing method designed to reduce redundant training data for HMM training. This method attempts to reduce subsequences that are used for multiple individual sub-HMM. It calculates the maximum number of subsequences by different subsequence partitions. Since correlated subsequences provide redundant information, elimination of this redundant information does not affect HMM training significantly but can reduce the number of sub-HMM to be trained. The details of the simple data preprocessing HMM scheme (SDPHMMS) are described as follow [13.14].

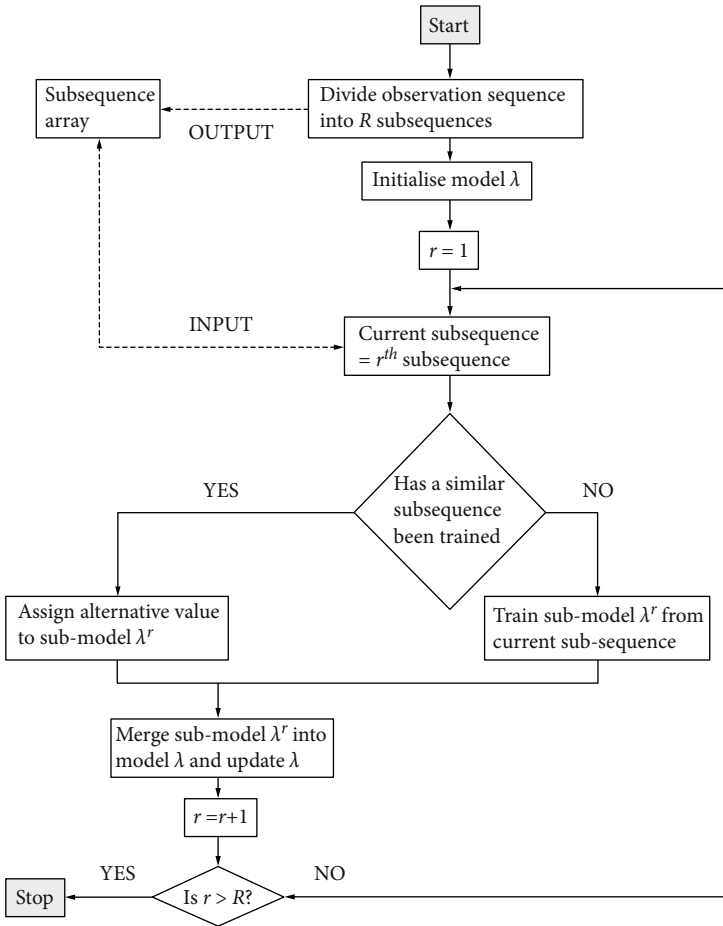
The SDPHMMS architecture is shown in Fig. 13.11.

As illustrated in Fig. 13.11, a long training data set is partitioned into a number of subsequences. Then each subsequence is used to train a sub-HMM, and the trained sub-HMM is incrementally merged into the final HMM using the weighting-average algorithm proposed in Hoang and Hu [13.18]. Compared with the method of Davis et al. [13.33], the incremental HMM training approach proposed by Hoang and Hu [13.18] incrementally merges the submodel into the final model rather than merging all submodels after they have been completely trained. In the SDPHMMS, highly similar subsequences can be removed without their participating in the training process. Hence, it can effectively reduce the number of submodels during the training process. Figure 13.12 shows the operational flow chart of the SDPHMMS.

**Similarity Calculation [13.14]** A subsequence is generated by an operational condition of the program. Therefore, there indeed exists a correla-



**Fig. 13.11** The simple data preprocessing HMM scheme (SDPHMMS) [13.14]



**Fig. 13.12** Operational flow chart of the SDPHMMS [13.14]

tion between operation condition similarity and subsequence similarity. In the extreme case, the same operational condition will generate the same

subsequence. Therefore, it is reasonable to identify similar operational conditions of a running program through identifying subsequences. For identifying

a pair of subsequence, there are many standard correlation methods, such as the correlation matrix [13.35]. To identify similar subsequences, a correlation threshold needs to be determined first. In general, the higher the threshold, the higher the correlation of the two subsequences involved will be. A higher threshold will lead to fewer similar subsequences and hence result in lower cost savings, and vice versa.

However, care must be taken when there is low threshold because it can produce more cost savings at the price of losing more useful information, which leads to the degradation of the intrusion detection rate performance. A balance needs to be struck and this balance point can be found experimentally. Intuitively, the same program operating in similar conditions will generate similar system call sequences. Therefore, the volume of redundant information can be huge because IDS training data are normally collected over a long period.

### 13.4 Emerging HIDS Architectures

Although significant advances have been made in developing individual intrusion detection engines, it seems that it is infeasible to cover a very broad feature spectrum owing to wide varieties of potential attacks. There is a trend to design new schemes that can address this issue. In this part, we introduce emerging HIDS architectures along this path. Finally, we propose a general HIDS framework that can be useful for designing new HIDS schemes which can deal with various aspects of attacks.

#### 13.4.1 Data Mining Approach Combining Macro-Level and Micro-Level Activities

Most HIDS focus either on macro-level or micro-level activities. Lee et al. [13.20] established a theoretic framework which can fuse meta data from different sources. In [13.17], a concrete application of combining macro-level and micro-level activities (CMLML) was provided. User behavior is modeled by using data mining, and the frequent-episode algorithms are used to build the user's normal profiles. Next we use this sample to illustrate its operational procedures.

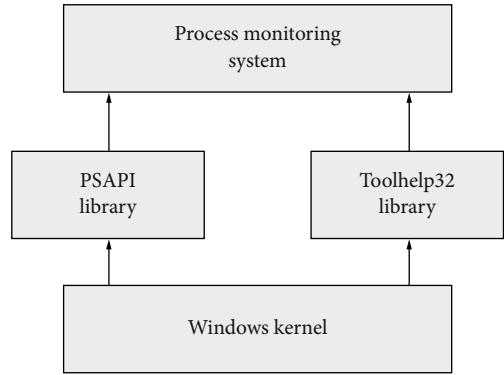


Fig. 13.13 Process data collection scheme

#### Operational Procedures of the CMLML Scheme [13.17]

**Collect the User Process Data** The sample IDS system has been built on the Windows NT platform using the Win32 library to monitor user and program activities. As shown in Fig. 13.13, the *Process Status Helper* (PSAPI.DLL) and the *Tool Help Library* are used to retrieve running process information. When a new session is started, the HIDS begins to obtain session information such as login user name and session start time. Then it collects information on all running processes associated with the user's session every 5 s.

Table 13.8 shows some critical information collected from running processes. The process

Table 13.8 Information collected by the monitoring process

Process attributes	Description
ProcessID	Identifier of the process
ProcessName	Name of the process
StartTime	Date and time the process started
ExitTime	Date and time the process ended
HandleCount	Number of handles of the process
ThreadCount	Number of running threads of the process
MemoryUsed	Include information about the memory used by the process such as current memory used, peak memory used, and virtual memory used
I/O information	Include information about input/output operations such as read, write, and other count and transferred data

**Table 13.9** Process data sample [13.17]

Session ID	Process ID	Process name	Start time
57	884	msimn.exe	2002-10-19 16:17:02
57	748	iexplore.exe	2002-10-19 16:17:12
57	720	winword.exe	2002-10-19 16:19:20
57	156	smss.exe	2002-10-19 16:07:35
57	204	winlogon.exe	2002-10-19 16:07:48
57	232	services.exe	2002-10-19 16:07:50
57	244	lsass.exe	2002-10-19 16:07:51

**Table 13.10** Resource usage for winword.exe

Process attribute	Minimum	Maximum
ThreadCount	2	4
WorkingSetSize (byte)	6,582,272	12,955,648
PeakWorkingSetSize (byte)	6,582,272	12,955,648
PagefileUsage (byte)	2,830,336	4,730,880
PeakPagefileUsage (byte)	2,830,336	4,755,456
ReadOperationCount	28	266
WriteOperationCount	2	6,646
OtherOperationCount	1,363	9,330
ReadTransferCount (byte)	7,513	64,129
WriteTransferCount (byte)	162	210,906
OtherTransferCount (byte)	21,312	122,506

identifier is unique and is assigned by the operating system. Other process information such as memory and input/output (I/O) information represents system resources consumed by user processes. Sample data collected are shown in Tables 13.9 and 13.10.

**Building the User Profile** During the training process, process data from 30 user login sessions are collected. Running processes in the system are grouped into two types: system processes and user processes. The system processes such as “winlogon.exe” and “services.exe” are processes which are generated automatically by the system. These system processes provide basic services to the user processes and user-working environment.

**Macro-Level Profile – User Activity** At the user program-activity level, consumption of system resources by user processes is being monitored. Table 13.9 shows some process information collected by the monitoring system. We applied the *frequent-episode algorithms* [13.36] on a collected data set to find the normal usage patterns of a given user at the program level. For example, *Alice* is a secretary and

she usually uses programs such as a e-mail client, a Web browser, and a word processor, and her application usage pattern is *Alice(mail:0.95, browser:0.80, word:0.80)*. This means that with 95% probability *Alice* will use an e-mail client in her working session and with 80% probability she will use both a Web browser and a word processor.

### Micro-Level Profile – Process Activity

We can also establish a user profile at the process-activity level. The most critical process information is the number of threads running concurrently, which is called the ThreadCount of a process. The more threads a process has, the more system resources it uses. For instance, if a user initiates multiple runs of a Web browser, each browser window needs a separate thread. Other information such as handle count, memory usage, and I/O information is also used to construct the micro-level activity profile.

In general, each user profile contains two sub-profiles:

1. Macro-level profile: the list of user applications with frequency of use and normal start time. This is the user-activity profile.
2. Micro-level profile: the system resources usage pattern of processes associated with each user.

### Intrusion Detection Process

The user profile consisting of the macro-level profile and the micro-level profile is established during the off-line training phase using the procedures described above. Because training data are collected over a long period, the statistics of relevant parameters such as their lower and upper bounds can be determined. In the system resources usage table, each process has an entry and each parameter has its own normal range. For example, *Alice*'s Web browser process has a thread count between 6 and 10, memory usage between 12 and 15 MB, and data transferred (read) between 5 and 10 MB. Table 13.10 shows the system resource table entry for “winword.exe” (a word processor application). During the operation of the IDS, it retrieves statistics about the user activity and associated processes using the same procedures described above for building the user profile. Then

these statistics are compared with the prestored user profile including the macro-level component and the micro-level component. A similarity score can be calculated. A simple similarity score can be given as

$$similarity\_score = 1 - \frac{no\_abnormal\_parameter}{total\_no\_of\_parameters} \quad (13.17)$$

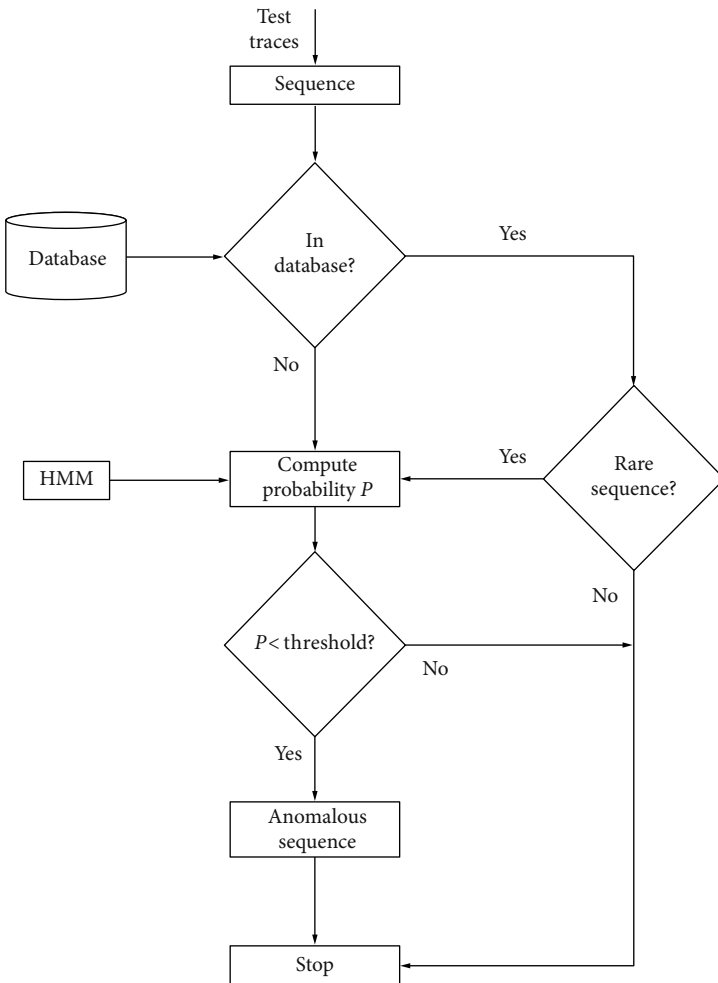
A sample experimental result is shown in Tables 13.11 and 13.12. There are nine parameters to be assessed. The similarity score is 0.44, which is a strong indication of abnormal behavior. Many other similarity measures can be defined. One natural choice could be to use separate similarity scores

for user-level activity and micro-level activity, then combine them using weightings to generate an overall similarity score. Obviously the higher the threshold, the lower the false acceptance rate will be, but at the cost of missing genuine attacks, and vice versa.

The experimental results demonstrate that user anomalies and changes in the user's normal working patterns can be detected effectively.

### 13.4.2 Two-Layer Approach

Hoang et al. [13.19] proposed a two-layered HIDS scheme as shown in Fig. 13.14.



**Fig. 13.14** Two-layered anomaly intrusion detection scheme

**Table 13.11** Alice’s normal/abnormal behaviors

Start time	Application	Normal/abnormal
9.00	E-mail client	Normal (9.00–9.30)
9.20	Web browser	Normal (9.00–10.00)
10.00	Word processor	Normal (9.30–10.30)
9.10	C++ compiler	Abnormal (application not in the list)
9.20	FTP program	Abnormal (application not in the list)
16.05	E-mail client	Abnormal (not valid time pattern)

**Table 13.12** ThreadCount normal/abnormal ranges

Processes	Normal range	Current usage	Normal/abnormal
msimn.exe (e-mail client)	6–9	8	Normal
winword.exe (word processor)	2–4	10	Abnormal
iexplore.exe (Eeb browser)	1–17	25	Abnormal

The test procedures of a sequence are divided into two steps:

1. The sequence of system calls is compared with those in the normal database to find a mismatch or a rare sequence indicated by low occurring frequency.

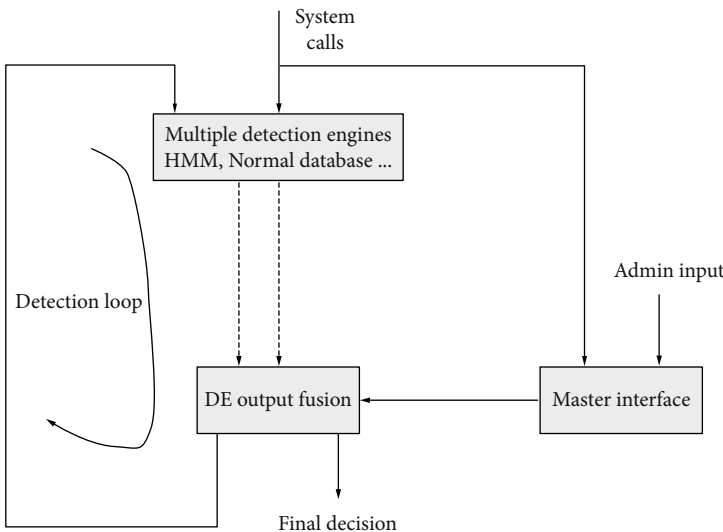
2. If the sequence is rare and/or a mismatch, it is then input into the HMM to compute the corresponding probability. If the probability required to produce the sequence is smaller than a pre-defined probability threshold, it is considered as an anomalous sequence. Use of the HMM as an additional analysis of the mismatch sequences can help to reduce the false alarms.

### 13.4.3 New Multi-Detection-Engine Architecture

Although the HMM can reduce false alarms significantly, it is still far from having practical applications. To address this issue, an enhanced HIDS architecture is proposed in this chapter as shown in Fig. 13.15.

In this architecture, multiple detection engines can be deployed and in many ways. In general, each detection engine has its own advantages and disadvantages. They tend to reveal different aspects of an attack. A good combination is expected to outperform what has been achieved by an individual detection engine. The following are several suggested combination schemes:

1. First, apply multiple detection engines to the system calls to be tested. This will generate multiple detection outputs from these detection engines. The detection outputs can be either at raw



**Fig. 13.15** A new multi-detection-engine architecture. DE detection engine

signal level (e.g., probability values) or at decision level (yes or no). Then we can use various data fusion technologies, such as majority vote or weightings, to fuse these outputs.

2. We can also use an output of a detection engine as an input into another engine to form a closed loop. A Bayesian network appears to be useful and can help develop very advanced intrusion detection engines.

## 13.5 Conclusions

In this chapter, the fundamentals of HIDS were introduced and popular HIDS were discussed. Several implementation details were provided and emerging HIDS technologies were discussed. A new framework has been proposed to integrate multiple detection engines. Several schemes under this framework have been suggested. We believe that in addition to designing new individual detection engines and improving existing detection engines, more effort is needed to develop new architectures/schemes such that various advantages of individual detection engines can be fused effectively. These will be good research topics in the future.

**Acknowledgements** The author appreciates discussion of the new multi-detection-engine architecture with X. Yu of RMIT University, Australia, and A. Nicholson of Monash University, Australia. The work was supported by Australia Research Council Discovery Grant DP0985838.

## References

- 13.1. A.S. Tanenbaum, A.S. Woodhull: *Operating Systems: Design and Implementation*, 3rd edn. (Pearson, NJ, USA 2006)
- 13.2. J.M. Garrido: *Principles of modern operating systems* (Jones and Barlett, MA, USA 2008)
- 13.3. A.S. Tanenbaum: *Computer Networks*, 3rd edn. (Prentice-Hall, NJ, USA 1996)
- 13.4. W.R. Stevens: *TCP/IP Illustrated: the protocols* (Addison Wesley Longman, MA, USA 1994)
- 13.5. J. Joshi, P. Krishnamurthy: Network Security. In: *Information Assurance: Dependability and Security in Networked Systems*, ed. by Y. Qian (Elsevier, Amsterdam, The Netherlands 2008), Chap. 2
- 13.6. B. Schneier: *Applied Cryptography, Protocols, Algorithms, and Source Code in C* (Wiley, NJ, USA 1996)
- 13.7. Y. Wang, J. Hu, D. Philips: A fingerprint orientation model based on 2D Fourier expansion (FOMFE) and its application to singular-point detection and fingerprint indexing, *IEEE Trans. Pattern Anal. Mach. Intell.* **29**(4), 13 (2007)
- 13.8. K. Xi, J. Hu: Introduction to bio-cryptography. In: *Springer Handbook on Communication and Information Security*, ed. by P. Stavroulakis (Springer, Berlin, Germany 2009), Chap. 6
- 13.9. J. Hu, P. Bertok, Z. Tari: Taxonomy and framework for integrating dependability and security. In: *Information Assurance: Dependability and Security in Networked Systems*, ed. by Y. Qian (Elsevier, Berlin, Germany 2008), Chap. 6
- 13.10. P.E. Proctor: *The Practical Intrusion Detection Handbook* (Prentice Hall PTR, NJ, USA 2001)
- 13.11. CNN.com: Worm strikes down Windows 200 systems (2005), available from: <http://www.cnn.com/2005/TECH/internet/08/16/computer:worm/> (last accessed November 25, 2008)
- 13.12. Sophos: Breaking news: worm attacks CNN, ABC, The Financial Times, and The New York Times (2005), [http://www.sophos.com/pressoffice/news/articles/2005/08/va\\_breakingnews.html](http://www.sophos.com/pressoffice/news/articles/2005/08/va_breakingnews.html) (last accessed November 25, 2008)
- 13.13. D. Denning: An intrusion detection model, *IEEE Symposium on Security and Privacy* (IEEE, NJ, USA 1986) pp. 118–131
- 13.14. J. Hu, Q. Dong, X. Yu, H.H. Chen: A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection, *IEEE Netw.* **23**(1), 42–47 (2009)
- 13.15. R.R. Kompella, S. Singh, G. Varghese: On scalable attack decision in the network, *IEEE/ACM Trans. Netw.* **15**(1), 14–25 (2007)
- 13.16. S.A. Hofmeyr, S. Forrest, A. Somayaji: Intrusion detection using sequences of system calls, *J. Comput. Secur.* **6**(3), 151–180 (1998)
- 13.17. D. Hoang, J. Hu, P. Bertok: Intrusion detection based on data mining, 5th Int. Conference on Enterprise Information Systems (Angers 1998) pp. 341–346
- 13.18. X.D. Hoang, J. Hu: An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls, *IEEE Int. Conference on Networks (ICON 2004)* (Singapore 2004) pp. 470–474
- 13.19. X.D. Hoang, J. Hu, P. Bertok: A multi-layer model for anomaly intrusion detection using program sequences of system calls, 11th IEEE Int. Conference on Network (ICON 2003) (Sydney 2003) pp. 531–536
- 13.20. W. Lee, S.I. Stolfo: A framework for constructing features and models for intrusion detection systems, *ACM Trans. Inf. Syst. Secur.* **3**(4), 227–261 (2000)
- 13.21. W. Lee, S.J. Stolfo: Data mining approaches for intrusion detection, *Proc. 7th USENIX Security Symposium* (San Antonio 1998)

- 13.22. C. Warrender, S. Forrest, B. Perlmutter: Detecting intrusions using system calls: alternative data models, IEEE Computer Society Symposium on Research in Security and Privacy (1999) pp. 257–286
- 13.23. J.L. Gauvain, C.H. Lee: Bayesian learning of Gaussian mixture densities for hidden Markov models, Proc. DARPA Speech and Natural Language Workshop (1991)
- 13.24. S. Forrest: A sense of self for Unix processes, IEEE Symposium on Computer Security and Privacy (1996)
- 13.25. X.H. Dau: E-Commerce Security Enhancement and Anomaly Intrusion Detection Using Machine Learning Techniques. Ph.D. Thesis (RMIT University, Melbourne 2006)
- 13.26. L.R. Rabiner: A tutorial on hidden Markov model and selected applications in speech recognition, Proc. IEEE 77(2), 257–286 (1989)
- 13.27. X.H. Dau: *Intrusion detection*, School of Computer Science and IT (RMIT University, Melbourne 2007)
- 13.28. J. Langford: *Optimizing hidden Markov model learning*, Technical Report (Toyota Technological Institute at Chicago, Chicago 2007)
- 13.29. R. Dugad, U.B. Desai: A tutorial on hidden Markov models, Technical Report No: SPANN-96.1, Indian Institute of Technology, Bombay (1996)
- 13.30. J.L. Gauvain, C.H. Lee: MAP estimation of continuous density HMM: Theory and Applications, Proceedings of the DARPA Speech and Natural Language Workshop (1992)
- 13.31. J.L. Gauvain, C.H. Lee: A posteriori estimation for multivariate Gaussian mixture observations of Markov chains, IEEE Trans. Speech Audio Process. 1(2), 291–298 (1994)
- 13.32. Y. Gotoh, M.M. Hochberg, H.F. Silverman: Efficient training algorithm for HMM's using incremental estimation, IEEE Trans. Speech Audio Process. 6(6), 539–548 (1998)
- 13.33. R.I.A. Davis, B.C. Lovell, T. Caelli: Improved estimation of hidden Markov model parameters from multiple observation sequences, 16th Int. Conference on Pattern Recognition (2002) pp. 168–171
- 13.34. X. Li, M. Parizean, R. Plamondon: Training hidden Markov models with multiple observations—A combinatorial method, IEEE Trans. Pattern Anal. Mach. Int. 22(4), 371–377 (2000)
- 13.35. R.J. Rummel: *Understanding correlation* (Department of Political Science University of Hawaii, Honolulu 1976)
- 13.36. H. Mannila, H. Toivonen, I. Verkamo: *Discovery of frequent episodes in event sequences*, Data Mining and Knowledge Discovery, Vol. 1 (Springer, MA, USA 1997)

## The Author



Jiankun Hu obtained his master degree from the Department of Computer Science and Software Engineering of Monash University, Australia, and his PhD degree from Control Engineering, Harbin Institute of Technology, China. He was awarded an Alexander von Humboldt Fellowship while working at Ruhr University, Germany. He is currently an associate professor at the School of Computer Science and IT, RMIT University, Australia. He leads the networking cluster within the discipline of distributed systems and networks. His current research interests are in network security, with emphasis on biometric security, mobile template protection, and anomaly intrusion detection. These research activities have been funded by three Australia Research Council grants. His research work has been published in top international journals.

Jiankun Hu  
School of Computer Science and IT  
RMIT University  
Melbourne 3001, Australia  
jiankun.hu@rmit.edu.au