Chapter 19

Intrusion detection systems

A network administrator tries hard to prevent possible break-ins. He regularly monitors the status of all network devices and reviews dozens of system log lines. In short, he dedicates a huge amount of time to checking the network status. To perform these tasks manually requires a lot of energy and oftentimes an administrator still won't notice the evidence that a hacker was in the system. An experienced hacker covers his tracks instantly, so an administrator has little or no chance to discover a possible attack. The administrator's reaction time is too long due to the impossibility of checking all potential suspicious signals in real time.

In this chapter we will focus on the tools supporting the administrator's work in detecting unauthorized access to the network.

What is an IDS?

An intrusion detection system, or IDS, is like a burglar alarm system for a computer.

Intrusion detection systems can broadly be divided into disk and network systems. We will provide some examples of applications belonging to both these groups. While they work very differently, their goal is much the same.

Among the disk systems are those that check modifications, access rights, or the file content on the computer hard disk. They work locally and encompass only one computer, and they use the network mainly to send reports. Examples of these systems are the AIDE and Tripwire applications.

IDS network systems, on the other hand, focus on monitoring the data flow in the packets circulating in a protected network. Therefore these applications can provide almost instant discovery of possible cracking attempts, as long as they are configured properly. Because of their clearly defined rules, these systems can reject suspected packets and inform the administrator about them.

Another useful function of intrusion detection network systems is the ability to register and notify suspicious activities in the network. Here, a fundamental element is a database storing all intercepted, incorrect, or suspicious packets along with detailed information about them. By reviewing this archive, we are able to check the data that interest us and to detect possible links between an attack and earlier packets. The databases have the advantage that they don't store useless correct packets and an unnecessary quantity of data. They therefore outperform the system logs, which have to be searched laboriously to give us the intended results.

From the point of view of security the exploitation of IDS systems also creates the possibility of leaving a trap for the hacker. We can mislead the hacker by sending him false data. It will prevent him getting into the real system and will allow evidence of his activity to accumulate. Some IDS systems are able to simulate any operating system, which can constitute a big challenge for a potential hacker.

Therefore network IDS systems help the administrator in analyzing system logs, which is a clear advantage due to the time savings.

Snort

To take full advantage of the Snort application, it is necessary to download the pcap library if it is not already present in the system. We download it from:

```
http://www.tcpdump.org
```

It is located in the "Current tar files" section. After downloading the file libcap-current.tar.gz we compile and install the software:

```
[root@localhost]# tar -xzf libpcap-current.tar.gz
[root@localhost]# cd libpcap-current
[root@localhost]# ./configure
[root@localhost]# make
[root@localhost]# make install
```

We have thus installed the pcap library, and we should now install the Snort application.

Installation of Snort

The first step is to download the application from the project homepage, which is available under the address:

```
http://www.snort.org
```

While at this site, we would suggest looking at the online program documentation. To download the software it is necessary to go to the download section and to choose the packet appropriate for us. Then we perform a series of commands, already known to us, to compile and install the application.

```
[root@localhost]# ./configure
[root@localhost]# make
[root@localhost]# make install
```

If before the installation we would like to change the Snort parameters, it is necessary to have a look at the INSTALL file, located inside the archive.

Configuration of Snort

For convenience we will now create a directory, in which we will store the program logs. Let's go to, for example, the /var directory and create the snort folder there.

```
[root@localhost]# cd /var
[root@localhost]# mkdir snort
```

```
[root@localhost]# snort -l /var/snort
```

Having given the above command to start up Snort will cause it to store all log files in the directory /var/snort, created a short time ago. Snort handles a very large number of parameters, some of which are described below:

-**A** Sets the alarm mode: fast, full, console, none.
-**b** Logs packets in the tcpdump format. This option is faster and more convenient to use.
-**c <rules>** Uses a rules file with the path <rules> instead of the one given by default
-**C** Prints overloads only with character data, without hexadecimal values.
-**D** Start up Snort in the background.
-**i <if>** Listens for the packets on the given network device (e.g., eth0).
-**l <folder>** Logs the events to the folder <folder>.
-**N** Switches off logging; the alarms are still active.
-**s** Sends alarm message to the system logs, and not to the console.
-**T** Tests the current Snort configuration.
-**y** Includes year in dates in alarms and logs.
-**?** Shows information on all available options.

Use

We will now assume that we want to intercept incoming packets in much the same way as the tcpdump application. We give the command snort -dev -l /var/snort so the program will write all the results to the console as well as to a file. After the startup we should see packets appearing regularly on the console screen.

```
[root@xsys snort]# snort -dev -l /var/snort
Running in packet logging mode
Log directory = /var/snort

Initializing Network Interface eth0

        --== Initializing Snort ==--
Initializing Output Plugins!
Decoding 'ANY' on interface eth0

--== Initialization Complete ==--
```

```
-*> Snort! <*-
Version 2.8.3.2 (Build 22)
By Martin Roesch (roesch@sourcefi re.com, www.snort.org)
12/09-20:43:42.241451 < l/l len: 0 l/l type: 0x200 0:1:0:0:0:0
pkt type:0x0 proto: 0x800 len:0x94
80.55.190.134:22 -> 213.77.167.139:32786 TCP TTL:59 TOS:0x10 ID:51743 IpLen:20 D
gmLen:132 DF
***AP*** Seq: 0x2DAEA3D7 Ack: 0x63C6157B Win: 0x2400 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1634201148 2668815
C8 54 FC 26 66 A3 71 E6 CB C4 3A C6 30 B7 DC 24 .T.&f.q...:.0..$
9B 78 14 66 E3 0E 6D 49 54 FD 5F BF 4C 79 78 4F .x.f..mIT._.LyxO
43 51 03 99 F9 16 4B 14 7B D4 2F AD E0 36 92 6B CQ....K.{./..6.k
7D 8B 79 9B 44 77 BF 61 BA 7A A4 F7 D1 A1 1A 1F }.y.Dw.a.z......
DE 12 99 14 1D B4 A5 2F BE DA BF F6 3C 4A 83 49 ......./....<J.I
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
12/09-20:43:42.242406 > l/l len: 0 l/l type: 0x200 0:1:0:0:0:0
pkt type:0x4 proto: 0x800 len:0x44
213.77.167.139:32786 -> 80.55.190.134:22 TCP TTL:64 TOS:0x10 ID:30904 IpLen:20 D
gmLen:52 DF
***A**** Seq: 0x63C6157B Ack: 0x2DAEA427 Win: 0x3E96 TcpLen: 32
TCP Options (3) => NOP NOP TS: 2725994 1634201148
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

We can see that starting up the program caused several packets to be intercepted immediately. Let's have a look at the basic structure of the above report. Information about the mode in which we started up Snort is displayed at the beginning. The next is the point of saving the intercepted packets and information about the successful initialization.

Now we will analyze the report structure more closely. At the beginning we see the date: 12/09-20:43:42.242406. It is structured in sequence from the month and day on which the packet was logged, right down to the exact hundredth of a second. Next comes the type of the packet, the content length, and any information that may have been included in its header (here lines 1 to 5, counting from the beginning of the packet). In the third line we see which IP address the packet has come from and which address it was directed to; we also have the name of the protocol used – in this case TCP. TTL is the packet time to live, which tells us how long it can be sent before it is stopped in the network. Bypassing other information, we reach the actual packet content. In the columns on the left hand side the information is shown regarding the data part of the packet, written in the hexadecimal system, whereas the same information is shown as ASCII characters in the column on the right.

Let's now try to interrupt Snort by pressing Ctrl + C. We will see the analysis summary:

```
===================================================
Snort received 353 packets
    Analyzed: 353(100.000%)
    Dropped: 0(0.000%)
===================================================
Breakdown by protocol:
        TCP: 339         (96.034%)
        UDP: 0           (0.000%)
        ICMP: 14         (3.966%)
        ARP: 0           (0.000%)
        EAPOL: 0         (0.000%)
        IPv6: 0          (0.000%)
        IPX: 0           (0.000%)
        OTHER: 0         (0.000%)
        DISCARD: 0       (0.000%)
===================================================
Action Stats:
ALERTS: 0
LOGGED: 353
PASSED: 0
===================================================
Snort exiting
```

The summary is divided into three sections. In the first section is the number of packets Snort has received, how many it has analyzed, and how many it has rejected. The next section gives information on the number of packets belonging to specific protocols. At the very end are activity statistics. We can see that nothing suspicious has been detected, which is shown by the number of alerts being zero. While 353 packets were logged, none was rejected.

Now we will have a look at how Snort stores the logs. For this purpose let's go to the directory /var/snort, which we created earlier, and list its contents:

```
[root@xsys xanatos]# cd /var/snort
[root@xsys snort]# ls -al
razem 72
drwxr-xr-x   9 root root 4096 dec  9 20:48 .
drwxr-xr-x  24 root root 4096 dec  8 21:14 ..
drwx------   2 root root 4096 dec  9 20:46 213.23.153.84
drwx------   2 root root 4096 dec  9 20:47 213.77.152.215
drwx------   2 root root 4096 dec  9 20:47 213.77.167.139
drwx------   2 root root 4096 dec  9 20:47 213.77.197.63
drwx------   2 root root 4096 dec  9 20:46 213.77.203.163
drwx------   2 root root 4096 dec  9 20:44 213.77.206.242
drwx------   2 root root 4096 dec  9 20:48 62.29.170.148
[root@xsys snort]#
```

The packets are usefully grouped by the IP address of the packet exchange.

Let's check, for example, the last folder:

```
[root@xsys snort]# cd 62.29.170.148
[root@xsys 62.29.170.148]# ls -al
razem 24
drwx------  2 root root 4096 dec  9 20:48 .
drwxr-xr-x  9 root root 4096 dec  9 20:48 ..
-rw-------  1 root root  742 dec  9 20:48 TCP:2121-445

[root@xsys 62.29.170.148]# cat TCP\:2121-445
12/09-20:48:00.449005 < l/l len: 0 l/l type: 0x200 0:1:0:0:0:0
pkt type:0x0 proto: 0x800 len:0x40
62.29.170.148:2121 -> 213.77.167.139:445 TCP TTL:118 TOS:0x0 ID:9047 IpLen:20 DgmLen:48
DF
******S* Seq: 0x4820ABBC  Ack: 0x0  Win: 0x2238  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

12/09-20:48:04.642365 < l/l len: 0 l/l type: 0x200 0:1:0:0:0:0
pkt type:0x0 proto: 0x800 len:0x40
62.29.170.148:2121 -> 213.77.167.139:445 TCP TTL:118 TOS:0x0 ID:9651 IpLen:20 DgmLen:48
DF
******S* Seq: 0x4820ABBC  Ack: 0x0  Win: 0x2238  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

Analyzing the results, the location of individual packets is easy to see. The filename contains information on the protocol used for the communication (TCP), while the two numbers after the colon are the numbers of the ports used by the network when transferring the packets. It can be concluded from this that the information exchange between the two computers took place between ports 2121 and 445.

The rules of Snort

A very useful option is the possibility to define the rules by which Snort will intercept the packets. The user receives the ability to reject packets containing specific content, which can be a very useful feature. The current rule files can be found on the page:

```
http://www.snort.org/snort-rules
```

The method of creating rules is discussed in detail in the program documentation. Most of Snort's rules regarding the interception of packets would fit in one line of text. The following example makes this clear:

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; msg:"access attempt to
mountd";)
```

Although this may look complicated, it is not. The rules can be broken down quite simply: The part before the brackets is what is known as the rule header. The rule options are contained in the brackets, separated by semicolons. The terms "content" and "msg" are parameters defining packet detection. Content determines that the packet is considered suspicious after a specific data sequence has been detected in its content section. Then the message defined in the msg parameter is displayed. The parameter structure is therefore simple:

```
PARAMETER_NAME:"parametere content";
```

Most Snort rules can be defined in this way. The rule options can be divided into four groups:

- **meta-data** – These options make information about the rule available, but are not important during packet detection.
- **payload** – Determines data inside the packet that allows to consider it as suspicious. These can be combined, so that each option is able to affect another.
- **non-payload** – Searches through other data inside the packet.
- **post-detection** – Parameters from this group are specific options that activate when a rule has been broken and its function has been triggered.

We can use many parameters, but the most useful are described below:

Msg – We use this to determine the text sent to the console and to the log file, when a rule has been broken. As its content we enter the message to be displayed, the structure of which is as follows: msg: "text of our message."

Reference – This parameter allows us to attach a reference to external attack identification systems. At present, it provides individual links to several of these systems. This plugin is used with output plugins to obtain the address to the additional information called by the alarm.

We invite the reader to have a look at the page where more information about systems supporting the indexing of Snort alarms can be obtained.

Here is its address:

```
http://www.snort.org/snort-db/
```

The systems served by the reference parameter are:

```
System  URL
bugtraq          http://www.securityfocus.com/bid/
cve              http://cve.mitre.org/cgi-bin/cvename.cgi?name=
nessus           http://cgi.nessus.org/plugins/dump.php3?id=
mcafee           http://vil.nai.com/vil/dispVirus.asp?virus_k=
url              http://
```

So this parameter will return the full output URL address to the report. The syntax is as follows:

```
reference: <system_identifier>,<id>; [reference: <system_identifier>,<id>; ]
```

Using square brackets it is possible to insert any number of references to one rule. The field <system_identifier> determines the name presented in the above table (e.g. bugtraq), while <id> its identification number.

For example:

```
reference: bugtraq,3416
```

This will cause the full address with additional information to be displayed in the report:

```
http://www.securityfocus.com/bid/3416
```

We see, therefore, that this is a very useful option.

Sid – This parameter is used to identify the program rules. Each of them can have its unique sid, enabling us to easily identify the rule content by the output plugins. This option should be used together with the rev parameter, described above. The format of the sid identification number is as follows:

```
sid: <identifier>;
```

<identifier> This is the number assigned to the rule. To maintain cohesion and legibility, it is only possible locally to use numbers greater than 1,000,000. Numbers less than 100 are reserved by Snort for future use, whereas the rules related to Snort and its distribution are in the range 100 – 1,000,000.

Rev – Revision options together with the sid rule identifiers will allow the signatures and descriptions to present information in a more legible way. This option should be used together with the sid parameter:

```
rev: <revision_identifier>;
```

Classtype – This is one of the most useful and versatile options of the Snort rules. This keyword helps in assigning alarms to several predefined attack classes. The user can determine which validity a specific rule class has in case of detection. Use:

```
classtype: <class_name>;
```

The rule classifications are placed and defined in the file classification.config, located in the Snort directory. This file has the following structure:

```
config classification: <class_name>,<class_description>,<default_validity>
```

The standard types are delivered together with the application. Their use doesn't require reviewing the configuration file. These predefined settings are assigned as standard to three default validity priorities. Priority 1 is the most

important classification type in the default rule set, whereas 3 is the least important.

Types

Type	*Description*	*Priority*
attempted-admin	Attempted administrator privilege gain	High
attempted-user	Attempted user privilege gain	High
shellcode-detect	Executable code was detected	High
successful-admin	Successful administrator privilege gain	High
successful-user	Successful user privilege gain	High
trojan-activity	A network Trojan was detected	High
unsuccessful-user	Unsuccessful user privilege gain	High
web-application-attack	Web application attack	High
attempted-dos	Attempted denial of service	Medium
attempted-recon	Attempted information leak	Medium
bad-unknown	Potentially bad traffic	Medium
denial-of-service	Detection of a denial of service attack	Medium
misc-attack	Miscellaneous attack	Medium
non-standard-protocol	Detection of a non-standard protocol or event	Medium
rpc-portmap-decode	Decode of an RPC query	Medium
successful-dos	Denial of service	Medium
successful-recon-largescale	Large scale information leak	Medium
successful-recon-limited	Information leak	Medium
suspicious-filename-detect	A suspicious filename was detected	Medium
suspicious-login	An attempted login using a suspicious username was detected	Medium
system-call-detect	A system call was detected	Medium
unusual-client-port-connection	A client was using an unusual port	Medium
web-application-activity	Access to a potentially vulnerable web application	Medium

icmp-event	Generic ICMP event	Low
misc-activity	Miscellaneous activity	Low
network-scan	Detection of a network scan	Low
not-suspicious	Not suspicious traffic	Low
protocol-command-decode	Generic protocol command decode	Low
string-detect	A suspicious string was detected	Low
unknown	Unknown traffic	Low

Priority – The priority option assigns an event importance level to a Snort rule. The classtype rule assigns a default priority level. However, it can be overwritten by a user value using the priority option.

Use format:

```
priority <importance_level>;
```

To test Snort, we will create an example rule (**/CD/Chapter19/Listings/rules.txt**):

```
alert tcp any any -> any any (msg: "any packet";)
```

This rule intercepts any packets of the TCP protocol on any port of our computer. We marked it with the alert "any packet." We will now put it in the file principles.txt, to be able to use it while starting up Snort. We will now start up Snort, entering the localization of the rule file as one of the parameters. We will therefore perform the command below:

```
[root@xsys snort]# snort -dev -l /var/snort –K ascii -c rules.txt
```

Let's interrupt the functioning of the program by pressing CTRL + C as soon as several packets have been received. Then we go to the directory /var/snort, where we store the logs. We will list it. The alert file should appear there.

Let's check its content:

```
[root@localhost]# ls
213.23.153.84   213.77.197.63   213.77.219.48   213.77.240.48   83.132.45.114
213.77.152.215  213.77.203.163  213.77.226.69   213.77.255.145  alert
213.77.167.139  213.77.206.242  213.77.228.109  62.29.170.148   rules.txt
[root@localhost]# cat alert
[**] [1:0:0] any packet [**]
[Priority: 0]
12/27-20:26:00.585898 < l/l len: 0 l/l type: 0x200 0:1:0:0:0:0
pkt type:0x0 proto: 0x800 len:0x40
213.77.228.109:3520 -> 213.77.167.139:445 TCP TTL:122 TOS:0x0 ID:49132 IpLen:20
DgmLen:48 DF
******S* Seq: 0x3179O4F6  Ack: 0x0  Win: 0x2238  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:0:0] any packet [**]
[Priority: 0]
12/27-20:26:03.491263 < l/l len: 0 l/l type: 0x200 0:1:0:0:0:0
pkt type:0x0 proto: 0x800 len:0x40
213.77.228.109:3520 -> 213.77.167.139:445 TCP TTL:122 TOS:0x0 ID:49843 IpLen:20
DgmLen:48 DF
******S* Seq: 0x3179O4F6  Ack: 0x0  Win: 0x2238  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:0:0] any packet [**]
[Priority: 0]
12/27-20:26:07.250697 < l/l len: 0 l/l type: 0x200 0:1:0:0:0:0
pkt type:0x0 proto: 0x800 len:0x174
80.55.190.130:22 -> 213.77.167.139:32774 TCP TTL:59 TOS:0x10 ID:29646 IpLen:20
DgmLen:356 DF
***AP*** Seq: 0xD9323E34  Ack: 0x80C0B5CA  Win: 0x2080  TcpLen: 32
TCP Options (3) => NOP NOP TS: 890650283 1770923
```

By defining the example rules, Snort stopped packets (in this case we followed all connections) and displayed their content together with the name of the applicable rule.

Snort's strength lies in the huge rules library. In order to make the IDS system fully functional, we should obtain the current set of rules by visiting this web site:

```
http://www.snort.org/snort-rules
```

Portsentry

Portsentry is a tiny program, whose task is to monitor ports in our system that are open and in use. It is able to detect an unauthorized packet

transmission on a suspicious port that is normally not used for communication, and to notify the administrator about it. It is also able to block further transmission immediately after detection of the suspicious activity. This is a very useful function. Before we discuss the other characteristics of the application we will focus on installation and configuration.

Installation of Portsentry

To download the current version of the program we go to:

```
http://sourceforge.net/projects/sentrytools
```

We download the portsentry-1.2.tar.gz package (or a newer version if available) and proceed with configuration, using the following series of commands:

```
[xanatos@localhost download]$ tar -xzf portsentry-1.2.tar.gz
[xanatos@localhost download]$ cd portsentry_beta/
```

Before we go any further, we can change the default localization of the configuration file by editing the file portsentry_config.h. The next step is to prepare the file portsentry.conf. Let's have a look at it before we begin to modify it. We take into consideration only the active lines, meaning these not preceded by the # character. Configuration consists of the options set as follows:

```
OPTION_NAME=option_value
```

Now we will have a look at the options TCP_PORTS and UDP_PORTS. In these parameters the numbers of ports observed by Portsentry are entered. They are separated by commas. We'll leave the default settings. The next options are not important for us at this stage.

So let's edit the next file, which is portsentry.ignore. It contains the IP addresses of the computers that will be excluded from scanning. We should

leave the address 127.0.0.1, the local return address of our computer, as well as the address of the external interface, e.g., 192.168.0.1.

Now, it's time to compile and install our application:

```
[xanatos@localhost download]$ make linux
[xanatos@localhost download]$ make install
```

Important: If the reader finds an error during the process compilation that points to line 1585 of the file portsentry.c, it is necessary to modify the line 1585 so that it contains:

```
printf ("Copyright 1997-2003 Craig H. Rowland <craigrowland at users dot
sourceforget dot net>\n");
```

The error here is the new line character, which can be found in the word "dot"; we simply delete this. As a result we obtain:

```
printf ("Copyright 1997-2003 Craig H. Rowland <craigrowland at users dot sourceforget dot
net>\n");
```

This is omitted in version 1.2 of the application.

Using Portsentry

We start up the program with the command /usr/local/psionic/portsentry/portsentry. However, to speed up access to the application we can create a link:

```
[root@localhost]# ln -s /usr/local/psionic/portsentry/portsentry /usr/bin/portsentry
```

Here is a list of the basic options of the program:
-**tcp** – activates scanning of the TCP ports, which we defined earlier,
-**udp** – starts up scanning of the UDP ports, defined earlier,
-**atcp** – scans ports in the advanced mode (TCP),
-**audp** – scans the UDP ports in the advanced mode,
-**stcp** – reacts to the scanning of the TCP ports of types SYN, NULL, etc.,
-**sudp** – reacts to the scanning of the UDP ports of types SYN, NULL,etc.

After the application is started up with the appropriate options, its will run in the background. It is recommended that the program call be added to the system start scripts.

If Portsentry detects and prevent access to any of the ports specified earlier, if will inform us about it putting the relevant information in an appropriate file /usr/local/psionic/portsentry/portsentry.blocked.xxx. In this case xxx means the type of the monitoring performed.

As we can see, the IDS tools, after proper configuration, can prove to be quite a challenge for the attacker. We invite the reader to study this topic further, because it can contribute considerably to increasing the security of the protected system.