

## Working with Snort Rules

Like viruses, most intruder activity has some sort of signature. Information about these signatures is used to create Snort rules. As mentioned in Chapter 1, you can use honey pots to find out what intruders are doing and information about their tools and techniques. In addition to that, there are databases of known vulnerabilities that intruders want to exploit. These known attacks are also used as signatures to find out if someone is trying to exploit them. These signatures may be present in the header parts of a packet or in the payload. Snort's detection system is based on rules. These rules in turn are based on intruder signatures. Snort rules can be used to check various parts of a data packet. Snort 1.x versions can analyze layer 3 and 4 headers but are not able to analyze application layer protocols. Upcoming Snort version 2 is expected to add support of application layer headers as well. Rules are applied in an orderly fashion to all packets depending on their types.

A rule may be used to generate an alert message, log a message, or, in terms of Snort, *pass* the data packet, i.e., drop it silently. The word *pass* here is not equivalent to the traditional meaning of *pass* as used in firewalls and routers. In firewalls and routers, *pass* and *drop* are opposite to each other. Snort rules are written in an easy to understand syntax. Most of the rules are written in a single line. However you can also extend rules to multiple lines by using a backslash character at the end of lines. Rules

are usually placed in a configuration file, typically `snort.conf`. You can also use multiple files by including them in a main configuration file.

This chapter provides information about different types of rules as well as the basic structure of a rule. You will find many examples of common rules for intrusion detection activity at the end of this chapter. After reading this chapter, along with the two preceding chapters, you should have enough information to set up Snort as a basic intrusion detection system.

### 3.1 TCP/IP Network Layers

Before you move to writing rules, let us have a brief discussion about TCP/IP layers. This is important because Snort rules are applied on different protocols in these layers.

TCP/IP is a five layer protocol. These layers interact with each other to make the communication process work. The names of these layers are:

1. The physical layer.
2. The data link layer. In some literature this is also called the network interface layer. The physical and data link layers consist of physical media, the network interface adapter, and the driver for the network interface adapter. Ethernet addresses are assigned in the data link layer.
3. The network layer, which is actually IP (Internet Protocol) layer. This layer is responsible for point-to-point data communication and data integrity. All hosts on this layer are distinguished by IP addresses. In addition to IP protocol, ICMP (Internet Control Message Protocol) is another major protocol in this layer. Information about IP protocol is available in RFC 791 available at <http://www.rfc-editor.org/rfc/rfc791.txt>. Information about ICMP protocol is available at <http://www.rfc-editor.org/rfc/rfc792.txt>.
4. The transport layer, which is actually TCP/UDP layer in the TCP/IP protocol. TCP (Transmission Control Protocol) is used for connection-oriented and reliable data transfer from source to destination. UDP (User Datagram Protocol), on the other hand, is used for connectionless data transfer. There is no assurance that data sent through UDP protocol will actually reach its destination. UDP is used where data loss can be tolerated. Information about UDP protocol is available in RFC 768 at <http://www.rfc-editor.org/rfc/rfc768.txt>. Information about TCP protocol is available in RFC 793 at <http://www.rfc-editor.org/rfc/rfc793.txt>.

5. The application layer consists of applications to provide user interface to the network. Examples of network applications are Telnet, Web browsers, and FTP clients. These applications usually have their own application layer protocol for data communication.

Snort rules operate on network (IP) layer and transport (TCP/UDP) layer protocols. However there are methods to detect anomalies in data link layer and application layer protocols. The second part of each Snort rule shows the protocol and you will learn shortly how to write these rules.

## 3.2 The First Bad Rule

Here is the first (very) bad rule. In fact, this may be the worst rule ever written, but it does a very good job of testing if Snort is working well and is able to generate alerts.

```
alert ip any any -> any any (msg: "IP Packet detected");)
```

You can use this rule at the end of the `snort.conf` file the first time you install Snort. The rule will generate an alert message for *every* captured IP packet. It will soon fill up your disk space if you leave it there! This rule is bad because it does not convey *any* information. What is the point of using a rule on a permanent basis that tells you nothing other than the fact that Snort is working? This should be your first test to make sure that Snort is installed properly. In the next section, you will find information about the different parts of a Snort rule. However for the sake of completeness, the following is a brief explanation of different words used in this rule:

- The word “alert” shows that this rule will generate an alert message when the criteria are met for a captured packet. The criteria are defined by the words that follow.
- The “ip” part shows that this rule will be applied on all IP packets.
- The first “any” is used for source IP address and shows that the rule will be applied to all packets.
- The second “any” is used for the port number. Since port numbers are irrelevant at the IP layer, the rule will be applied to all packets.
- The -> sign shows the direction of the packet.
- The third “any” is used for destination IP address and shows that the rule will be applied to all packets irrespective of destination IP address.
- The fourth “any” is used for destination port. Again it is irrelevant because this rule is for IP packets and port numbers are irrelevant.

- The last part is the rule options and contains a message that will be logged along with the alert.

The next rule isn't quite as bad. It generates alerts for all captured ICMP packets. Again, this rule is useful to find out if Snort is working.

```
alert icmp any any -> any any (msg: "ICMP Packet found");
```

If you want to test the Snort machine, send a ping packet (which is basically ICMP ECHO REQUEST packet on UNIX machines). Again, you can use this rule when you install Snort to make sure that it is working well. As an example, send an ICMP packet to your gateway address or some other host on the network using the following command:

```
ping 192.168.2.1
```

Note that 192.168.2.1 is the IP address of gateway/router or some other host on the same network where the Snort machine is present. This command should be executed on the machine where you installed Snort. The command can be used both on UNIX and Microsoft Windows machines.

---

**TIP** I use a slightly modified version of this rule to continuously monitor multiple Snort sensors just to make sure everybody is up and running. This rule is as follows:

```
alert icmp 192.168.1.4 any -> 192.168.1.1 any (msg: "HEARTBEAT");
```

My Snort sensor IP address is 192.168.1.4 and gateway address is 192.168.1.1. I run the following command through cron daemon on the Linux machine to trigger this rule every 10 minutes.

```
ping -n 1 192.168.1.1
```

The command sends exactly one ICMP packet to the gateway machine. This packet causes an alert entry to be created. If there is no alert every 10 minutes, there is something wrong with the sensor.

---

### 3.3 CIDR

Classless Inter-Domain Routing or CIDR is defined in RFC 1519. It was intended to make better use of available Internet addresses by eliminating different classes (like class A and class B). With the CIDR, you can define any number of bits in the netmask field, which was not possible with class-based networking where the number of bits was fixed. Using CIDR, network addresses are written using the number of bits in the netmask at the end of the IP address. For example, 192.168.1.0/24 defines a network with network address 192.168.1.0 with 24 bits in the netmask. A netmask with 24 bits is

equal to 255.255.255.0. An individual host can be written using all of the netmask bits, i.e., 32. The following rule shows that only those packets that go to a single host with IP address 192.168.2.113 will generate an alert:

```
alert icmp any any -> 192.168.1.113/32 any \
(msg: "Ping with TTL=100"; ttl:100;)
```

All addresses in Snort are written using the CIDR notation, which makes it very convenient to monitor any subset of hosts.

### 3.4 Structure of a Rule

Now that you have seen some rules which are not-so-good but helpful in a way, let us see the structure of a Snort rule. All Snort rules have two logical parts: rule *header* and rule *options*. This is shown in Figure 3-1.



**Figure 3-1** Basic structure of Snort rules.

The rule header contains information about what action a rule takes. It also contains criteria for matching a rule against data packets. The options part usually contains an alert message and information about which part of the packet should be used to generate the alert message. The options part contains additional criteria for matching a rule against data packets. A rule may detect one type or multiple types of intrusion activity. Intelligent rules should be able to apply to multiple intrusion signatures.

The general structure of a Snort rule header is shown in Figure 3-2.



**Figure 3-2** Structure of Snort rule header.

The *action* part of the rule determines the type of action taken when criteria are met and a rule is exactly matched against a data packet. Typical actions are generating an alert or log message or invoking another rule. You will learn more about actions later in this chapter.

The *protocol* part is used to apply the rule on packets for a particular protocol only. This is the first criterion mentioned in the rule. Some examples of protocols used are IP, ICMP, UDP etc.

The *address* parts define source and destination addresses. Addresses may be a single host, multiple hosts or network addresses. You can also use these parts to exclude some addresses from a complete network. More about addresses will be discussed later. Note that there are two address fields in the rule. Source and destination addresses are determined based on direction field. As an example, if the direction field is “->”, the Address on the left side is source and the Address on the right side is destination.

In case of TCP or UDP protocol, the *port* parts determine the source and destination ports of a packet on which the rule is applied. In case of network layer protocols like IP and ICMP, port numbers have no significance.

The *direction* part of the rule actually determines which address and port number is used as source and which as destination.

For example, consider the following rule that generates an alert message whenever it detects an ICMP<sup>1</sup> ping packet (ICMP ECHO REQUEST) with TTL equal to 100, as you have seen in Chapter 2.

```
alert icmp any any -> any any (msg: "Ping with TTL=100"; \
ttl: 100;)
```

The part of the rule before the starting parenthesis is called the rule header. The part of the rule that is enclosed by the parentheses is the options part. The header contains the following parts, in order:

- A rule action. In this rule the action is “alert”, which means that an alert will be generated when conditions are met. Remember that packets are logged by default when an alert is generated. Depending on the action field, the rule options part may contain additional criteria for the rules.
- Protocol. In this rule the protocol is ICMP, which means that the rule will be applied only on ICMP-type packets. In the Snort detection engine, if the protocol of a packet is not ICMP, the rest of the rule is not considered in order to save CPU time. The protocol part plays an important role when you want to apply Snort rules only to packets of a particular type.

1. ICMP or Internet Control Message Protocol is defined in RFC 792. ICMP packets are used to convey different types of information in the network. ICMP ECHO REQUEST is one type of ICMP packet. There are many other types of ICMP packets as defined in the RFC 792. The references at the end of this chapter contains a URL to download the RFC document.

- Source address and source port. In this example both of them are set to “any”, which means that the rule will be applied on all packets coming from any source. Of course port numbers have no relevance to ICMP packets. Port numbers are relevant only when protocol is either TCP or UDP.
- Direction. In this case the direction is set from left to right using the -> symbol. This shows that the address and port number on the left hand side of the symbol are source and those on the right hand side are destination. It also means that the rule will be applied on packets traveling from source to destination. You can also use a <- symbol to reverse the meaning of source and destination address of the packet. Note that a symbol <> can also be used to apply the rule on packets going in either direction.
- Destination address and port address. In this example both are set to “any”, meaning the rule will be applied to all packets irrespective of their destination address. The direction in this rule does not play any role because the rule is applied to all ICMP packets moving in either direction, due to the use of the keyword “any” in both source and destination address parts.

The options part enclosed in parentheses shows that an alert message will be generated containing the text string “Ping with TTL=100” whenever the condition of TTL=100 is met. Note that TTL or *Time To Live* is a field in the IP packet header. Refer to RFC 791 at <http://www.rfc-editor.org/rfc/rfc791.txt> or Appendix C for information on IP packet headers.

## 3.5 Rule Headers

As mentioned earlier, a rule header consists of the section of the rule before starting parentheses and has many parts. Let us take a detailed look at different parts used in the rule header, starting with rule actions.

### 3.5.1 Rule Actions

The action is the first part of a Snort rule. It shows what action will be taken when rule conditions are met. An action is taken only when all of the conditions mentioned in a rule are true. There are five predefined actions. However, you can also define your own actions as needed. As a precaution, keep in mind that Snort versions 1.x and 2.x apply rules in different ways. In Snort 1.x, if multiple rules match a given packet, only the first one is applied. After applying the first rule, no further action is taken on the packet. However in Snort version 2, all rules are applied before generating an alert message. The most severe alert message is then generated.

### 3.5.1.1 Pass

This action tells Snort to ignore the packet. This action plays an important role in speeding up Snort operation in cases where you don't want to apply checks on certain packets. For example, if you have a vulnerability assessment host on your own network that you use to find possible security holes in your network, you may want Snort to ignore any attacks from that host. The pass rule plays an important part in such a case.

### 3.5.1.2 Log

The log action is used to log a packet. Packets can be logged in different ways, as discussed later in this book. For example, a message can be logged to log files or in a database. Packets can be logged with different levels of detail depending on the command line arguments and configuration file. To find available command line arguments with your version of Snort, use “`snort -?`” command.

### 3.5.1.3 Alert

The alert action is used to send an alert message when rule conditions are true for a particular packet. An alert can be sent in multiple ways. For example, you can send an alert to a file or to a console. The functional difference between Log and Alert actions is that Alert actions send an alert message and then log the packet. The Log action only logs the packet.

### 3.5.1.4 Activate

The activate action is used to create an alert and then to activate another rule for checking more conditions. Dynamic rules, as explained next, are used for this purpose. The activate action is used when you need further testing of a captured packet.

### 3.5.1.5 Dynamic

Dynamic action rules are invoked by other rules using the “activate” action. In normal circumstances, they are not applied on a packet. A dynamic rule can be activated only by an “activate” action defined in another rule.

### 3.5.1.6 User Defined Actions

In addition to these actions, you can define your own actions. These rule actions can be used for different purposes, such as:

- Sending messages to syslog. Syslog is system logger daemon and creates log file in `/var/log` directory. Location of these files can be changed using `/etc/syslog.conf` file. For more information, use “`man syslog`” and “`man syslog.conf`” commands on a UNIX system. Syslog may be compared to the event logger on Microsoft Windows systems.

- Sending SNMP traps. SNMP traps are sent to a network management system like HP OpenView or Open NMS at <http://www.opennms.org>.
- Taking multiple actions on a packet. As you have seen earlier in the structure of Snort rules, a rule only takes one action. User defined rules can be used to take multiple actions. For example, a user defined rule can be used to send an SNMP trap as well as to log the alert data to the syslog daemon.
- Logging data to XML files.

Logging messages into a database. Snort is able to log messages to MySQL, PostgreSQL, Oracle and Microsoft SQL server.

These new action types are defined in the configuration file `snort.conf`. A new action is defined in the following general structure:

```
ruletype action_name
{
    action definition
}
```

The `ruletype` keyword is followed by the action name. Two braces enclose the actual definition of the action, just like a function in C programming. For example, an action named `smb_db_alert` that is used to send SMB pop-up window alert messages to hosts listed in `workstation.list` file and to MySQL database named “snort” is defined below:

```
ruletype smb_db_alert
{
    type alert
    output alert_smb: workstation.list
    output database: log, mysql, user=rr password=rr \
        dbname=snort host=localhost
}
```

These types of rules will be discussed in the next chapter in detail. Usually they are related to configuration of output plug-ins.

### 3.5.2 Protocols

Protocol is the second part of a Snort rule. The protocol part of a Snort rule shows on which type of packet the rule will be applied. Currently Snort understands the following protocols:

- IP
- ICMP

- TCP
- UDP

If the protocol is IP, Snort checks the link layer header to determine the packet type. If any other type of protocol is used, Snort uses the IP header to determine the protocol type. Different packet headers are discussed in Appendix C.

The protocols only play a role in specifying criteria in the header part of the rule. The options part of the rule can have additional criteria unrelated to the specified protocol. For example, consider the following rule where the protocol is ICMP.

```
alert icmp any any -> any any (msg: "Ping with TTL=100"; \
  ttl: 100;)
```

The options part checks the TTL (Time To Live) value, which is not part of the ICMP header. TTL is part of IP header instead. This means that the options part can check parameters in other protocol fields as well. Header fields for common protocols and their explanation is found in Appendix C.

### 3.5.3 Address

There are two address parts in a Snort rule. These addresses are used to check the source from which the packet originated and the destination of the packet. The address may be a single IP address or a network address. You can use *any* keyword to apply a rule on all addresses. The address is followed by a slash character and number of bits in the netmask. For example, an address 192.168.2.0/24 represents C class network 192.168.2.0 with 24 bits in the network mask. A network mask with 24 bits is 255.255.255.0. Keep the following in mind about number of bits in the netmask:

- If the netmask consists of 24 bits, it is a C class network.
- If the netmask consists of 16 bits, it is a B class network.
- If the netmask consists of 8 bits, it is an A class network.
- For a single host, use 32 bits in the netmask field.

You can also use any number of bits in the address part allowed by Classless Inter-Domain Routing or CIDR. Refer to RFC 791 at <http://www.rfc-editor.org/rfc/rfc791.txt> for structure of IP addresses and netmasks and to RFC 1519 at <http://www.rfc-editor.org/rfc/rfc1519.txt> for more information on CIDR.

As mentioned earlier, there are two address fields in the Snort rule. One of them is the source address and the other one is the destination address. The direction part of the

rule determines which address is source and which one is destination. Refer to the explanation of the direction part to find more information about how this selection is made.

Following are some examples of how addresses are mentioned in Snort rules:

- An address 192.168.1.3/32 defines a single host with IP address 192.168.1.3.
- An address 192.168.1.0/24 defines a class C network with addresses ranging from 192.168.1.0 to 192.168.1.255. There are 24 bits in the netmask, which is equal to 255.255.255.0.
- An address 152.168.0.0/16 defines a class B network with addresses ranging from 152.168.0.0 to 152.168.255.255. There are 16 bits in the netmask, which is equal to 255.255.0.0.
- An address 10.0.0.0/8 defines a class A network with addresses ranging from 10.0.0.0 to 10.255.255.255. There are 8 bits in the netmask, which is equal to 255.0.0.0.
- An address 192.168.1.16/28 defines an address range of 192.168.1.16 to 192.168.1.31. There are 28 bits in the netmask field, which is equal to 255.255.255.240, and the network consists of 16 addresses. You can place only 14 hosts in this type of network because two of the total 16 addresses are used up in defining the network address and the broadcast address. Note that the first address in each network is always the network address and the last address is the broadcast address. For this network 192.168.1.16 is the network address and 192.168.1.31 is the broadcast address.

For example, if you want to generate alerts for all TCP packets with TTL=100 going to web server 192.168.1.10 at port 80 from any source, you can use the following rule:

```
alert tcp any any -> 192.168.1.10/32 80 (msg: "TTL=100"; \
  ttl: 100;)
```

This rule is just an example to provide information about how IP addresses are used in Snort rules.

### 3.5.3.1 Address Exclusion

Snort provides a mechanism to exclude addresses by the use of the negation symbol !, an exclamation point. This symbol is used with the address to direct Snort not to test packets coming from or going to that address. For example, the following rule is applied to all packets except those that originate from class C network 192.168.2.0.

```
alert icmp ![192.168.2.0/24] any -> any any \  
  (msg: "Ping with TTL=100"; ttl: 100;)
```

This rule is useful, for instance, when you want to test packets that don't originate from your home network (which means you trust everyone in your home network!).

### 3.5.3.2 Address Lists

You can also specify list of addresses in a Snort rule. For example, if your home network consists of two C class IP networks 192.168.2.0 and 192.168.8.0 and you want to apply the above rule to all addresses but hosts in these two, you can use the following modified rule where the two addresses are separated by a comma.

```
alert icmp ![192.168.2.0/24,192.168.8.0/24] any -> any \  
  any (msg: "Ping with TTL=100"; ttl: 100;)
```

Note that a square bracket is used with the negation symbol. You don't need to use brackets if you are not using the negation symbol.

### 3.5.4 Port Number

The port number is used to apply a rule on packets that originate from or go to a particular port or a range of ports. For example, you can use source port number 23 to apply a rule to those packets that originate from a Telnet server. You can use the keyword *any* to apply the rule on all packets irrespective of the port number. Port number is meaningful only for TCP and UDP protocols. If you have selected IP or ICMP as the protocol in the rule, port number does not play any role. The following rule is applied to all packets that originate from a Telnet server in 192.168.2.0/24, which is a class C network and contains the word "confidential":

```
alert tcp 192.168.2.0/24 23 -> any any \  
  (content: "confidential"; msg: "Detected confidential";)
```

The same rule can be applied to traffic either going to or originating from any Telnet server in the network by modifying the direction to either side as shown below:

```
alert tcp 192.168.2.0/24 23 <> any any \  
  (content: "confidential"; msg: "Detected confidential";)
```

Port numbers are useful when you want to apply a rule only for a particular type of data packet. For example, if a vulnerability is related to only a HTTP (Hyper Text Transfer Protocol) web server, you can use port 80 in the rule to detect anybody trying to exploit it. This way Snort will apply that rule only to web server traffic and not to any other TCP packets. Writing good rules always improves the performance of IDS.

### 3.5.4.1 Port Ranges

You can also use a range of ports instead of only one port in the port field. Use a colon to separate starting and ending port numbers. For example, the following rule will create an alert for all UDP traffic coming from ports 1024 to 2048 from all hosts.

```
alert udp any 1024:2048 -> any any (msg: "UDP ports");
```

### 3.5.4.2 Upper and Lower Boundaries

While listing port numbers, you can also use only the starting port number or the ending port number in the range. For example, a range specified as :1024 includes all port numbers up to and including port 1024. A port range specified as 1000: will include all ports numbers including and above port 1000.

### 3.5.4.3 Negation Symbol

As with addresses, you can also use the negation symbol with port numbers to exclude a port or a range of ports from the scope of the Snort rule. The following rule logs all UDP traffic except for source port number 53.

```
log udp any !53 -> any any log udp
```

You can't use comma character in the port field to specify multiple ports. For example, specifying 53,54 is not allowed. However you can use 53:54 to specify a port range.

### 3.5.4.4 Well-Known Port Numbers

Well-known port numbers are used for commonly used applications. Some of these port numbers and their applications are listed in Table 3-1.

**Table 3-1** Well-Known Port Numbers

Port Number	Description
20	FTP data
21	FTP
22	SSH or Secure shell
23	Telnet
25	SMTP, used for e-mail server like Sendmail
37	NTP (Network Time Protocol) used for synchronizing time on network hosts
53	DNS server
67	BootP/DHCP client
68	BootP/DHCP server
69	TFTP
80	HTTP, used for all web servers

**Table 3-1** Well-Known Port Numbers (continued)

Port Number	Description
110	POP3, used for e-mail clients like Microsoft Outlook
161	SNMP
162	SNMP traps
443	HTTPS or Secure HTTP
514	Syslog
3306	MySQL

You can also look into `/etc/services` file on the UNIX platform to see more port numbers. Refer to RFC 1700 for a detailed list at <http://www.rfc-editor.org/rfc/rfc1700.txt>. The Internet Corporation for Assigned Names and Numbers (ICANN) now keeps track of all port numbers and names. You can find more information at <http://www.icann.org>.

### 3.5.5 Direction

The direction field determines the source and destination addresses and port numbers in a rule. The following rules apply to the direction field:

- A `->` symbol shows that address and port numbers on the left hand side of the direction field are the source of the packet while the address and port number on the right hand side of the field are the destination.
- A `<-` symbol in the direction field shows that the packet is traveling from the address and port number on the right hand side of the symbol to the address and port number on the left hand side.
- A `<>` symbol shows that the rule will be applied to packets traveling on either direction. This symbol is useful when you want to monitor data packets for both client and server. For example, using this symbol, you can monitor all traffic coming from and going to a POP or Telnet server.

## 3.6 Rule Options

Rule options follow the rule header and are enclosed inside a pair of parentheses. There may be one option or many and the options are separated with a semicolon. If you use multiple options, these options form a logical AND. The action in the rule header is invoked only when all criteria in the options are true. You have already used options like `msg` and `ttl` in previous rule examples. All options are defined by keywords. Some rule options also contain arguments. In general, an option may have two parts: a keyword

and an argument. Arguments are separated from the option keyword by a colon. Consider the following rule options that you have already seen:

```
msg: "Detected confidential";
```

In this option *msg* is the keyword and “*Detected confidential*” is the argument to this keyword.

The remainder of this section describes keywords used in the options part of Snort rules.

### 3.6.1 The ack Keyword

The TCP header contains an Acknowledgement Number field which is 32 bits long. The field shows the next sequence number the sender of the TCP packet is expecting to receive. This field is significant only when the ACK flag in the TCP header is set. Refer to Appendix C and RFC 793 for more information about the TCP header.

Tools like nmap (<http://www.nmap.org>) use this feature of the TCP header to ping a machine. For example, among other techniques used by nmap, it can send a TCP packet to port 80 with ACK flag set and sequence number 0. Since this packet is not acceptable by the receiving side according to TCP rules, it sends back a RST packet. When nmap receives this RST packet, it learns that the host is alive. This method works on hosts that don't respond to ICMP ECHO REQUEST ping packets.

To detect this type of TCP ping, you can have a rule like the following that sends an alert message:

```
alert tcp any any -> 192.168.1.0/24 any (flags: A; \
  ack: 0; msg: "TCP ping detected");
```

This rule shows that an alert message will be generated when you receive a TCP packet with the A flag set and the acknowledgement contains a value of 0. Other TCP flags are listed in Table 3-2. The destination of this packet must be a host in network 192.168.1.0/24. You can use any value with the ACK keyword in a rule, however it is added to Snort only to detect this type of attack. Generally when the A flag is set, the ACK value is not zero.

### 3.6.2 The classtype Keyword

Rules can be assigned classifications and priority numbers to group and distinguish them. To fully understand the classtype keyword, first look at the file `classification.config` which is included in the `snort.conf` file using the *include* keyword. Each line in the `classification.config` file has the following syntax:

```
config classification: name,description,priority
```

The *name* is a name used for the classification. The name is used with the *classtype* keyword in Snort rules. The *description* is a short description of the class type. *Priority* is a number that shows the default priority of the classification, which can be modified using a *priority* keyword inside the rule options. You can also place these lines in `snort.conf` file as well. An example of this configuration parameter is as follows:

```
config classification: DoS,Denial of Service Attack,2
```

In the above line the classification is DoS and the priority is 2. In Chapter 6, you will see that classifications are used in ACID,<sup>2</sup> which is a web-based tool to analyze Snort alert data. Now let us use this classification in a rule. The following rule uses default priority with the classification DoS:

```
alert udp any any -> 192.168.1.0/24 6838 (msg:"DoS"; \
content: "server"; classtype:DoS;)
```

The following is the same rule but we override the default priority used for the classification.

```
alert udp any any -> 192.168.1.0/24 6838 (msg:"DoS"; \
content: "server"; classtype:DoS; priority:1)
```

Using classifications and priorities for rules and alerts, you can distinguish between high- and low-risk alerts. This feature is very useful when you want to escalate high-risk alerts or want to pay attention to them first.

---

**NOTE** Low priority numbers show high priority alerts.

---

If you look at the ACID browser window, as discussed in Chapter 6, you will see the classification screens as shown in Figure 3-3. The second column in the middle part of the screen displays different classifications for captured data.

Other tools also use the classification keyword to prioritize intrusion detection data. A typical `classification.config` file is shown below. This file is distributed with the Snort 1.9.0. You can add your own classifications to this file and use them in your own rules.

2. ACID stands for Analysis Control for Intrusion Detection. It provides a web-based user interface to analyze data generated by Snort.

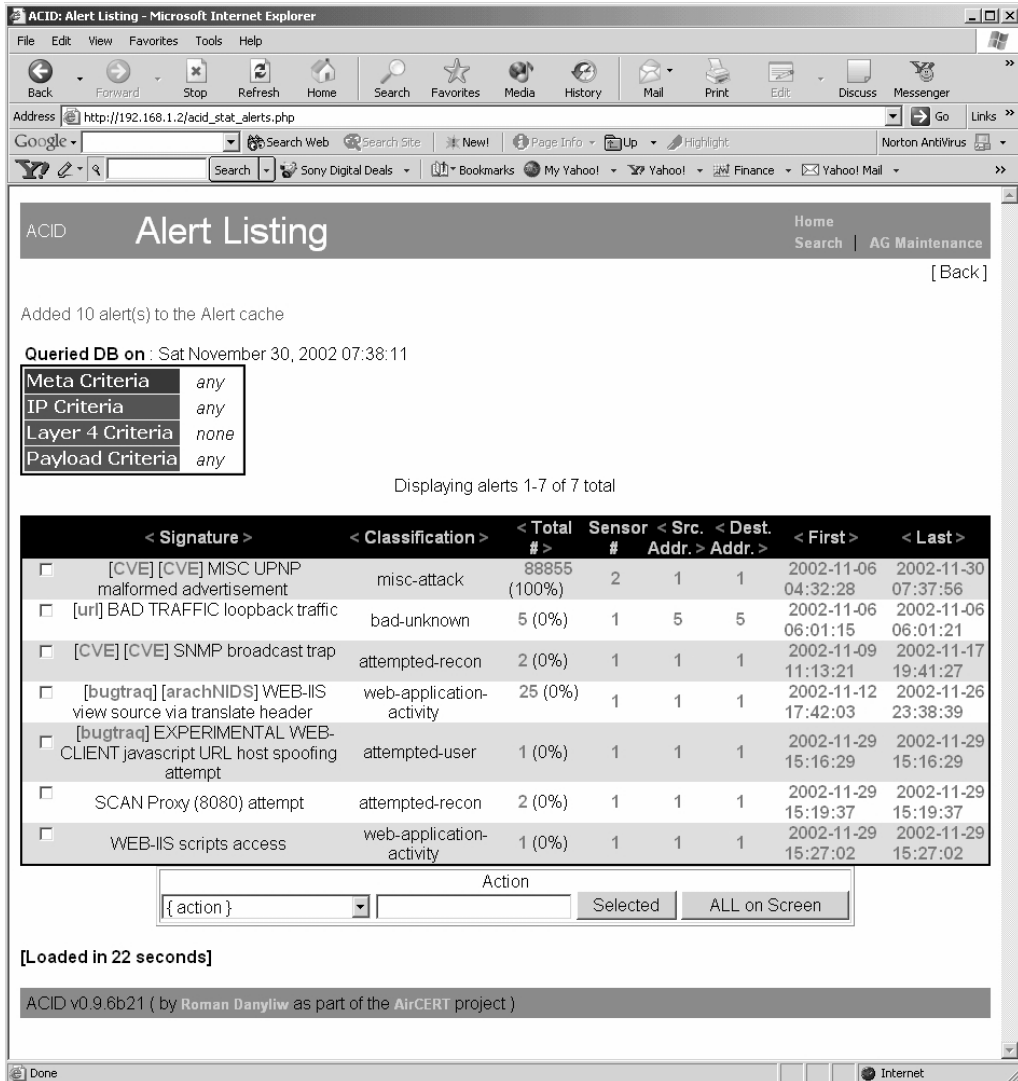


Figure 3-3 Use of the classification keyword in displaying Short alerts inside ACID window.

```
# $Id: classification.config,v 1.10 2002/08/11 23:37:18 cazz Exp $
# The following includes information for prioritizing rules
#
# Each classification includes a shortname, a description, and a
# default
# priority for that classification.
```

```
#
# This allows alerts to be classified and prioritized. You can specify
# what priority each classification has. Any rule can override the
# default
# priority for that rule.
#
# Here are a few example rules:
#
# alert TCP any any -> any 80 (msg: "EXPLOIT ntpdx overflow";
#     dsize: > 128; classtype:attempted-admin; priority:10;
#
# alert TCP any any -> any 25 (msg:"SMTP expn root"; flags:A+; \
#     content:"expn root"; nocase; classtype:attempted-recon;)
#
# The first rule will set its type to "attempted-admin" and override
# the default priority for that type to 10.
#
# The second rule set its type to "attempted-recon" and set its
# priority to the default for that type.
#
#
# config classification:shortname,short description,priority
#
config classification: not-suspicious,Not Suspicious Traffic,3
config classification: unknown,Unknown Traffic,3
config classification: bad-unknown,Potentially Bad Traffic, 2
config classification: attempted-recon,Attempted Information Leak,2
config classification: successful-recon-limited,Information Leak,2
config classification: successful-recon-largescale,Large Scale
    Information Leak,2
config classification: attempted-dos,Attempted Denial of Service,2
config classification: successful-dos,Denial of Service,2
config classification: attempted-user,Attempted User Privilege Gain,1
config classification: unsuccessful-user,Unsuccessful User Privilege
    Gain,1
config classification: successful-user,Successful User Privilege Gain,1
config classification: attempted-admin,Attempted Administrator
    Privilege Gain,1
config classification: successful-admin,Successful Administrator
    Privilege Gain,1

# NEW CLASSIFICATIONS
config classification: rpc-portmap-decode,Decode of an RPC Query,2
config classification: shellcode-detect,Executable code was detected,1
```

```
config classification: string-detect,A suspicious string was detected,3
config classification: suspicious-filename-detect,A suspicious filename
was detected,2
config classification: suspicious-login,An attempted login using a
suspicious username was detected,2
config classification: system-call-detect,A system call was detected,2
config classification: tcp-connection,A TCP connection was detected,4
config classification: trojan-activity,A Network Trojan was detected, 1
config classification: unusual-client-port-connection,A client was
using an unusual port,2
config classification: network-scan,Detection of a Network Scan,3
config classification: denial-of-service,Detection of a Denial of
Service Attack,2
config classification: non-standard-protocol,Detection of a non-
standard protocol or event,2
config classification: protocol-command-decode,Generic Protocol Command
Decode,3
config classification: web-application-activity,access to a potentially
vulnerable web application,2
config classification: web-application-attack,Web Application Attack,1
config classification: misc-activity,Misc activity,3
config classification: misc-attack,Misc Attack,2
config classification: icmp-event,Generic ICMP event,3
config classification: kickass-porn,SCORE! Get the lotion!,1
config classification: policy-violation,Potential Corporate Privacy
Violation,1
config classification: default-login-attempt,Attempt to login by a
default username and password,2
```

### 3.6.3 The content Keyword

One important feature of Snort is its ability to find a data pattern inside a packet. The pattern may be presented in the form of an ASCII string or as binary data in the form of hexadecimal characters. Like viruses, intruders also have signatures and the content keyword is used to find these signatures in the packet. Since Snort version 1.x does not support application layer protocols, this keyword, in conjunction with the offset keyword, can also be used to look into the application layer header.

The following rule detects a pattern “GET” in the data part of all TCP packets that are leaving 192.168.1.0 network and going to an address that is not part of that network. The GET keyword is used in many HTTP related attacks; however, this rule is only using it to help you understand how the content keyword works.

```
alert tcp 192.168.1.0/24 any -> ![192.168.1.0/24] any \
(content: "GET"; msg: "GET matched");
```

The following rule does the same thing but the pattern is listed in hexadecimal.

```
alert tcp 192.168.1.0/24 any -> ![192.168.1.0/24] any \
(content: "|47 45 54|"; msg: "GET matched");
```

Hexadecimal number 47 is equal to ASCII character G, 45 is equal to E, and 54 is equal to T. You can also match both ASCII strings and binary patterns in hexadecimal form inside one rule. Just enclose the hexadecimal characters inside a pair of bar symbols: ||.

When using the content keyword, keep the following in mind:

- Content matching is a computationally expensive process and you should be careful of using too many rules for content matching.
- If you provide content as an ASCII string, you should escape the double quote, colon and bar symbols.
- You can use multiple content keywords in one rule to find multiple signatures in the data packet.
- Content matching is case sensitive.

There are three other keywords that are used with the content keyword. These keywords add additional criteria while finding a pattern inside a packet. These are:

- The offset keyword
- The depth keyword
- The nocase keyword

These keywords are discussed later in this chapter. The first two keywords are used to confine the search within a certain range of the data packet. The nocase keyword is used to make the search case-insensitive.

### 3.6.4 The offset Keyword

The offset keyword is used in combination with the content keyword. Using this keyword, you can start your search at a certain offset from the start of the data part of the packet. Use a number as argument to this keyword. The following rule starts searching for the word “HTTP” after 4 bytes from the start of the data.

```
alert tcp 192.168.1.0/24 any -> any any \
(content: "HTTP"; offset: 4; msg: "HTTP matched");
```

You can use the depth keyword to define the point after which Snort should stop searching the pattern in the data packets.

### 3.6.5 The depth Keyword

The depth keyword is also used in combination with the content keyword to specify an upper limit to the pattern matching. Using the depth keyword, you can specify an offset from the start of the data part. Data after that offset is not searched for pattern matching. If you use both offset and depth keywords with the content keyword, you can specify the range of data within which pattern matching should be done. The following rule tries to find the word “HTTP” between characters 4 and 40 of the data part of the TCP packet.

```
alert tcp 192.168.1.0/24 any -> any any (content: \
    "HTTP"; offset: 4; depth: 40; msg: "HTTP matched");
```

This keyword is very important since you can use it to limit searching inside the packet. For example, information about HTTP GET requests is found in the start of the packet. There is no need to search the entire packet for such strings. Since many packets you capture are very long in size, it wastes a lot of time to search for these strings in the entire packet. The same is true for many other Snort signatures.

### 3.6.6 The content-list Keyword

The content-list keyword is used with a file name. The file name, which is used as an argument to this keyword, is a text file that contains a list of strings to be searched inside a packet. Each string is located on a separate line of the file. For example, a file named “porn” may contain the following three lines:

```
“porn”
“hardcore”
“under 18”
```

The following rule will search these strings in the data portion of all packets matching the rule criteria.

```
alert ip any any -> 192.168.1.0/24 any (content-list: \
    "porn"; msg: "Porn word matched");
```

You can also use the negation sign ! with the file name if you want to generate an alert for a packet where no strings match.

### 3.6.7 The dsize Keyword

The dsize keyword is used to find the length of the data part of a packet. Many attacks use buffer overflow vulnerabilities by sending large size packets. Using this keyword, you can find out if a packet contains data of a length larger than, smaller than, or

equal to a certain number. The following rule generates an alert if the data size of an IP packet is larger than 6000 bytes.

```
alert ip any any -> 192.168.1.0/24 any (dsize: > 6000; \
  msg: "Large size IP packet detected");
```

### 3.6.8 The flags Keyword

The flags keyword is used to find out which flag bits are set inside the TCP header of a packet. Each flag can be used as an argument to flags keyword in Snort rules. A detailed description of the TCP flag bits is present in RFC 793 at <http://www.rfc-editor.org/rfc/rfc793.txt>. These flag bits are used by many security related tools for different purposes including port scanning tools like nmap (<http://www.nmap.org>). Snort supports checking of these flags listed in Table 3-2.

**Table 3-2** TCP flag bits

Flag	Argument character used in Snort rules
FIN or Finish Flag	F
SYN or Sync Flag	S
RST or Reset Flag	R
PSH or Push Flag	P
ACK or Acknowledge Flag	A
URG or Urgent Flag	U
Reserved Bit 1	1
Reserved Bit 2	2
No Flag set	0

You can also use `!`, `+`, and `*` symbols just like IP header flag bits (discussed under the fragbits keyword) for AND, OR and NOT logical operations on flag bits being tested. The following rule detects any scan attempt using SYN-FIN TCP packets.

```
alert tcp any any -> 192.168.1.0/24 any (flags: SF; \
  msg: "SYNC-FIN packet detected");
```

Note that ! symbol is used for NOT, + is used for AND, and \* is used for OR operation.

### 3.6.9 The fragbits Keyword

The IP header contains three flag bits that are used for fragmentation and re-assembly of IP packets. These bits are listed below:

- Reserved Bit (RB), which is reserved for future use.
- Don't Fragment Bit (DF). If this bit is set, it shows that the IP packet should not be fragmented.
- More Fragments Bit (MF). If this bit is set, it shows that more fragments of this IP packet are on the way. If this bit is not set, it shows that this is the last fragment (or the only fragment) of the IP packet. The sending host fragments IP packets into smaller packets depending on the maximum size packet that can be transmitted through a communication medium. For example, the Maximum Transfer Units or MTU defines the maximum length of a packet on the Ethernet networks. This bit is used at the destination host to reassemble IP fragments.

For more information on Flag bits refer to RFC 791 at <http://www.rfc-editor.org/rfc/rfc791.txt>. Sometimes these bits are used by hackers for attacks and to find out information related to your network. For example, the DF bit can be used to find the minimum and maximum MTU for a path from source to destination. Using the fragbits keyword, you can find out if a packet contains these bits set or cleared. The following rule is used to detect if the DF bit is set in an ICMP packet.

```
alert icmp any any -> 192.168.1.0/24 any (fragbits: D; \
  msg: "Don't Fragment bit set");
```

In this rule, D is used for DF bit. You can use R for reserved bit and M for MF bit. You can also use the negation symbol ! in the rule. The following rule detects if the DF bit is not set, although this rule is of little use.

```
alert icmp any any -> 192.168.1.0/24 any (fragbits: !D; \
  msg: "Don't Fragment bit not set");
```

The AND and OR logical operators can also be used to check multiple bits. The + symbol specifies all bits be matched (AND operation) while the \* symbol specifies any of the specified bits be matched (OR operation).

### 3.6.10 The icmp\_id Keyword

The `icmp_id` option is used to detect a particular ID used with ICMP packet. Refer to Appendix C for ICMP header information. The general format for using this keyword is as follows:

```
icmp_id: <ICMP_id_number>
```

An ICMP identified field is found in ICMP ECHO REQUEST and ICMP ECHO REPLY messages as discussed in RFC 792. This field is used to match ECHO REQUEST and ECHO REPLY messages. Usually when you use the ping command, both of these types of ICMP packets are exchanged between sending and receiving hosts. The sending host sends ECHO REQUEST packets and the destination host replies with ECHO REPLY-type ICMP packets. This field is useful for discovering which packet is the reply to a particular request. The following rule checks if the ICMP ID field in the ICMP header is equal to 100. It generates an alert if this criterion is met.

```
alert icmp any any -> any any (icmp_id: 100; \
  msg: "ICMP ID=100");
```

### 3.6.11 The icmp\_seq Keyword

The `icmp_seq` option is similar to the `icmp_id` keyword. The general format for using this keyword is as follows:

```
icmp_seq: <ICMP_seq_number>
```

The sequence number is also a field in the ICMP header and is also useful in matching ICMP ECHO REQUEST and ECHO REPLY matches as mentioned in RFC 792. The keyword helps to find a particular sequence number. However, the practical use of this keyword is very limited. The following rule checks a sequence number of 100 and generates an alert:

```
alert icmp any any -> any any (icmp_seq: 100; \
  msg: "ICMP Sequence=100");
```

### 3.6.12 The itype Keyword

The ICMP header comes after the IP header and contains a type field. Appendix C explains the IP header and the different codes that are used in the type field. A detailed discussion is found in RFC 792 at <http://www.rfc-editor.org/rfc/rfc792.txt>. The `itype` keyword is used to detect attacks that use the type field in the ICMP packet header. The argument to this field is a number and the general format is as follows:

```
itype: "ICMP_type_number"
```

The type field in the ICMP header of a data packet is used to determine the type of the ICMP packet. Table 3-3 lists different ICMP types and values of the type field in the ICMP header.

**Table 3-3** ICMP type field values

Value	Type of ICMP Packet
0	Echo reply
3	Destination unreachable
4	Source quench
5	Redirect
8	Echo request
11	Time exceed
12	Parameter problem
13	Timestamp request
14	Timestamp reply
15	Information request
16	Information reply

For example, if you want to generate an alert for each source quench message, use the following rule:

```
alert icmp any any -> any any (itype: 4; \
  msg: "ICMP Source Quench Message received";)
```

The ICMP code field is used to further classify ICMP packets.

### 3.6.13 The `icode` Keyword

In ICMP packets, the ICMP header comes after the IP header. It contains a code field, as shown in Appendix C and RFC 792 at <http://www.rfc-editor.org/rfc/rfc792.txt>. The `icode` keyword is used to detect the code field in the ICMP packet header. The argument to this field is a number and the general format is as follows:

```
icode: "ICMP_codee_number"
```

The type field in the ICMP header shows the type of ICMP message. The code field is used to explain the type in detail. For example, if the type field value is 5, the ICMP packet type is “ICMP redirect” packet. There may be many reasons for the generation of an ICMP redirect packet. These reasons are defined by the code field as listed below:

- If code field is 0, it is a network redirect ICMP packet.
- If code field is 1, it is a host redirect packet.
- If code is 2, the redirect is due to the type of service and network.
- If code is 2, the redirect is due to type of service and host.

The `icode` keyword in Snort rule options is used to find the code field value in the ICMP header. The following rule generates an alert for host redirect ICMP packets.

```
alert icmp any any -> any any (itype: 5; \  
    icode: 1; msg: "ICMP ID=100");
```

Both `itype` and `icode` keywords are used. Using the `icode` keyword alone will not do the job because other ICMP types may also use the same code value.

### 3.6.14 The `id` Keyword

The `id` keyword is used to match the fragment ID field of the IP packet header. Its purpose is to detect attacks that use a fixed ID number in the IP header of a packet. Its format is as follows:

```
id: "id_number"
```

If the value of the `id` field in the IP packet header is zero, it shows that this is the last fragment of an IP packet (if the packet was fragmented). The value 0 also shows that it is the only fragment if the packet was not fragmented. The `id` keyword in the Snort rule can be used to determine the last fragment in an IP packet.

### 3.6.15 The `ipopts` Keyword

A basic IPv4 header is 20 bytes long as described in Appendix C. You can add options to this IP header at the end. The length of the options part may be up to 40 bytes. IP options are used for different purposes, including:

- Record Route (`rr`)
- Time Stamps (`ts`)

- Loose Source Routing (lsrr)
- Strict Source Routing (ssrr)

For a complete list of IP options see RFC 791 at <http://www.rfc-editor.org/rfc/rfc791.txt>. In Snort rules, the most commonly used options are listed above. These options can be used by some hackers to find information about your network. For example, loose and strict source routing can help a hacker discover if a particular network path exists or not.

Using Snort rules, you can detect such attempts with the `ipopts` keyword. The following rule detects any attempt made using Loose Source Routing:

```
alert ip any any -> any any (ipopts: lsrr; \
    msg: "Loose source routing attempt");
```

You can also use a `logto` keyword to log the messages to a file. However, you can't specify multiple IP options keywords in one rule.

### 3.6.16 The `ip_proto` Keyword

The `ip_proto` keyword uses IP Proto plug-in to determine protocol number in the IP header. The keyword requires a protocol number as argument. You can also use a name for the protocol if it can be resolved using `/etc/protocols` file. Sample entries in this file look like the following:

```
ax.25    93      AX.25      # AX.25 Frames
ipip     94      IPIP       # Yet Another IP encapsulation
micp     95      MICP       # Mobile Internetworking
Control Pro.
scc-sp   96      SCC-SP     # Semaphore Communications
Sec. Pro.
etherip  97      ETHERIP    # Ethernet-within-IP
Encapsulation
encap    98      ENCAP     # Yet Another IP encapsulation
#        99      # any private encryption
scheme
gmtp     100     GMTP      # GMTP
ifmp     101     IFMP      # Ipsilon Flow Management
Protocol
pnni     102     PNNI      # PNNI over IP
```

The following rule checks if IPIP protocol is being used by data packets:

```
alert ip any any -> any any (ip_proto: ipip; \
    msg: "IP-IP tunneling detected");
```

The next rule is the same except that it uses protocol number instead of name (more efficient).

```
alert ip any any -> any any (ip_proto: 94; \
  msg: "IP-IP tunneling detected");
```

Protocol numbers are defined in RFC 1700 at <http://www.rfc-editor.org/rfc/rfc1700.txt>. The latest numbers can be found from the ICANN web site at <http://www.icann.org> or at IANA web site <http://www.iana.org>.

### 3.6.17 The logto Keyword

The logto keyword is used to log packets to a special file. The general syntax is as follows:

```
logto:logto_log
```

Consider the following rule:

```
alert icmp any any -> any any (logto:logto_log; ttl: 100;)
```

This rule will log all ICMP packets having TTL value equal to 100 to file logto\_log. A typical logged packet in this file is as follows:

```
[root@conformix]# cat logto_log
07/03-03:57:56.496845 192.168.1.101 -> 192.168.1.2
ICMP TTL:100 TOS:0x0 ID:33822 IpLen:20 DgmLen:60
Type:8 Code:0 ID:768 Seq:9217 ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
qrstuvwxyzabcdefghijklmnop
```

```
====+
```

```
[root@conformix]#
```

Information logged in the above example is as follows:

- Data and time the packet was logged.
- Source IP address is 192.168.1.101.
- Destination IP address is 192.168.1.2.
- Protocol used in the packet is ICMP.
- The TTL (Time To Live) field value in the IP header is 100.
- The TOS (Type Of Service) field value in IP header is 0. This value shows that this is a normal packet. For details of other TOS values, refer to RFC 791.

- IP packet ID is 33822.
- Length of IP header is 20 bytes.
- Length of the packet is 60 bytes.
- ICMP type field value is 8.
- ICMP code value is 0.
- ICMP ID value is 768.
- ICMP Sequence field value is 9217.
- The ECHO part shows that this is an ICMP ECHO packet.
- The remaining part of the log shows the data that follows the ICMP header.

There are a few things to remember when you use this option:

- Don't use the full path with the file name. The file will automatically be created in the log directory which is `/var/log/snort` by default.
- Don't use a space character after the colon character used with `logto` keyword. If you use a space character, it is considered part of the file name. If you use a space character for clarity, enclose the file name in double quotation marks.

### 3.6.18 The `msg` Keyword

The `msg` keyword in the rule options is used to add a text string to logs and alerts. You can add a message inside double quotations after this keyword. The `msg` keyword is a common and useful keyword and is part of most of the rules. The general form for using this keyword is as follows:

```
msg: "Your message text here";
```

If you want to use some special character inside the message, you can escape them by a backslash character.

### 3.6.19 The `nocase` Keyword

The `nocase` keyword is used in combination with the `content` keyword. It has no arguments. Its only purpose is to make a case insensitive search of a pattern within the data part of a packet.

### 3.6.20 The `priority` Keyword

The `priority` keyword assigns a priority to a rule. Priority is a number argument to this keyword. Number 1 is the highest priority. The keyword is often used with the `classtype` keyword. The following rule has a priority 10:

```
alert ip any any -> any any (ipopts: lsrr; \  
    msg: "Loose source routing attempt"; priority: 10;)
```

The `priority` keyword can be used to differentiate high priority and low priority alerts.

### 3.6.21 The `react` Keyword

The `react` keyword is used with a rule to terminate a session to block some sites or services. Not all options with this keyword are operational. The following rule will block all HTTP connections originating from your home network 192.168.1.0/24. To block the HTTP access, it will send a TCP FIN and/or FIN packet to both sending and receiving hosts every time it detects a packet that matches these criteria. The rule causes a connection to be closed.

```
alert tcp 192.168.1.0/24 any -> any 80 (msg: "Outgoing \  
    HTTP connection"; react: block;)
```

In the above rule, *block* is the basic modifier. You can also use the *warn* modifier to send a visual notice to the source. You can also use the additional modifier *msg* which will include the *msg* string in the visual notification on the browser. The following is an example of this additional modifier.

```
alert tcp 192.168.1.0/24 any -> any 80 (msg: "Outgoing \  
    HTTP connection"; react: warn, msg;)
```

In order to use the `react` keyword, you should compile Snort with `--enable-flexresp` command line option in the configure script. For a discussion of the compilation process, refer to Chapter 2.

The `react` should be the last keyword in the options field. The `warn` modifier still does not work properly in the version of Snort I am using.

### 3.6.22 The `reference` Keyword

The `reference` keyword can add a reference to information present on other systems available on the Internet. It does not play any role in the detection mechanism itself and you can safely ignore it as far as writing Snort rules is concerned. There are many reference systems available, such as CVE and Bugtraq. These systems keep additional information about known attacks. By using this keyword, you can link to this additional information in the alert message. For example, look at the following rule in the `misc.rules` file distributed with Snort:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1900 \  
(msg:"MISC UPNP malformed advertisement"; \  
content:"NOTIFY * "; nocase; classtype:misc-attack; \  
reference:cve,CAN-2001-0876; reference:cve, \  
CAN-2001-0877; sid:1384; rev:2;)
```

This rule generates the following entry in `/var/log/snort/alert` file:

```
[**] [1:1384:2] MISC UPNP malformed advertisement [**]  
[Classification: Misc Attack] [Priority: 2]  
12/01-15:25:21.792758 192.168.1.1:1901 -> 239.255.255.250:1900  
UDP TTL:150 TOS:0x0 ID:9 IpLen:20 DgmLen:341  
Len: 321  
[Xref => cve CAN-2001-0877][Xref => cve CAN-2001-0876]
```

The last line of this alert shows a reference where more information about this alert can be found. The `reference.config` file plays an important role because it contains the actual URL to reach a particular reference. For example, the following line in `reference.config` file will reach the actual URL using the last line of the alert message.

```
config reference: cve http://cve.mitre.org/cgi-bin/  
cvename.cgi?name=
```

When you add `CAN-2001-0876` at the end of this URL, you will reach the web site containing information about this alert. So the actual URL for information about this alert is `http://cve.mitre.org/cgi-bin/cvename.cgi?name= CAN-2001-0876`.

Multiple references can be placed in a rule. References are also used by tools like ACID<sup>3</sup> to provide additional information about a particular vulnerability. The same log message, when displayed in an ACID window, will look like Figure 3-4. In this figure, the URL is already inserted under the “Triggered Signature” heading. You can click on it to go to the CVE web site for more information.

### 3.6.23 The `resp` Keyword

The `resp` keyword is a very important keyword. It can be used to knock down hacker activity by sending response packets to the host that originates a packet matching the rule. The keyword is also known as Flexible Response or simply FlexResp and is based on the FlexResp plug-in. The plug-in should be compiled into Snort, as explained in Chapter 2, using the command line option (`--with-flexresp`) in the

3. ACID is discussed in Chapter 6.

ACID: Alert - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address [http://192.168.1.2/acid\\_gry\\_alert.php?submit=%230-%285-16290%29&sort\\_order=](http://192.168.1.2/acid_gry_alert.php?submit=%230-%285-16290%29&sort_order=)

Google Search Web Search Site New! Page Info Up Highlight Norton AntiVirus

Y! Search Bookmarks My Yahoo! Yahoo! Finance Yahoo! Mail

ACID **Alert** Home Search AG Maintenance [Back]

Queried DB on : Sun December 01, 2002 16:06:14

Meta Criteria	any
IP Criteria	any
Layer 4 Criteria	none
Payload Criteria	any

Added 10 alert(s) to the Alert cache

**Alert #1**  
[First] >> Next #1-(5-16177)

Meta	<b>ID #</b>	5 - 16290
	<b>Time</b>	2002-12-01 16:05:39
	<b>Triggered Signature</b>	[CVE] [CVE] MISC UPNP malformed advertisement
Sensor	<b>name</b>	192.168.1.2
	<b>interface</b>	eth0
Alert Group	<b>filter</b>	none
	<b>Alert Group</b>	none

IP	<b>source addr</b>	192.168.1.1
	<b>dest addr</b>	239.255.255.250
	<b>Ver</b>	4
FQDN	<b>Hdr Len</b>	5
	<b>TOS</b>	0
Options	<b>length</b>	341
	<b>ID</b>	9
Alert Group	<b>flags</b>	0
	<b>offset</b>	0
Alert Group	<b>TTL</b>	150
	<b>chksum</b>	29163
Alert Group	<b>Source Name</b>	Unable to resolve address
	<b>Dest. Name</b>	Unable to resolve address
Alert Group	<b>Options</b>	none
	<b>Alert Group</b>	none

UDP	<b>source port</b>	1901
	<b>dest port</b>	1900
Alert Group	<b>length</b>	321
	<b>Alert Group</b>	none

Length = 313

Figure 3-4 Use of reference keyword in ACID window.

configure script. The following rule will send a TCP Reset packet to the sender whenever an attempt to reach TCP port 8080 on the local network is made.

```
alert tcp any any -> 192.168.1.0/24 8080 (resp: rst_snd;)
```

You can send multiple response packets to either sender or receiver by specifying multiple responses to the resp keyword. The arguments are separated by a comma. The list of arguments that can be used with this keyword is found in Table 3-4.

**Table 3-4** Arguments to resp keyword

Argument	Description
rst_snd	Sends a TCP Reset packet to the sender of the packet
rst_rcv	Sends a TCP Reset packet to the receiver of the packet
rst_all	Sends a TCP Reset packet to both sender and receiver
icmp_net	Sends an ICMP Network Unreachable packet to sender
icmp_host	Sends an ICMP Host Unreachable packet to sender
icmp_port	Sends an ICMP Port Unreachable packet to sender
icmp_all	Sends all of the above mentioned packets to sender

### 3.6.24 The rev Keyword

The rev keyword is added to Snort rule options to show a revision number for the rule. If you are updating rules, you can use this keyword to distinguish among different revision. Output modules can also use this number to identify the revision number. The following rule shows that the revision number is 2 for this rule:

```
alert ip any any -> any any (ipopts: lsrc; \
    msg: "Loose source routing attempt"; rev: 2;)
```

For more information, refer to the sid keyword, which is related to the rev keyword.

### 3.6.25 The rpc Keyword

The rpc keyword is used to detect RPC based requests. The keyword accepts three numbers as arguments:

- Application number

- Procedure number
- Version number

These arguments are separated by a comma. You can also use an asterisk to match all numbers in a particular location of the arguments. The following rule detects RPC requests for TPC number 10000, all procedures and version number 3.

```
alert ip any any -> 192.168.1.0/24 any (rpc: 10000,*,3; \
    msg: "RPC request to local network");)
```

### 3.6.26 The sameip Keyword

The sameip keyword is used to check if source and destination IP addresses are the same in an IP packet. It has no arguments. Some people try to spoof IP packets to get information or attack a server. The following rule can be used to detect these attempts.

```
alert ip any any -> 192.168.1.0/24 any (msg: "Same IP"; \
    sameip;)
```

### 3.6.27 The seq Keyword

The seq keyword in Snort rule options can be used to test the sequence number of a TCP packet. The argument to this keyword is a sequence number. The general format is as follows:

```
seq: "sequence_number";
```

Sequence numbers are a part of the TCP header. More explanation of sequence number is found in Appendix C where the TCP header is discussed.

### 3.6.28 The flow<sup>4</sup> Keyword

The flow keyword is used to apply a rule on TCP sessions to packets flowing in a particular direction. You can use options with the keyword to determine direction. The following options can be used with this keyword determine direction:

- to\_client
- to\_server
- from\_client
- from\_server

4. This is available in Snort 1.9 and above.

These options may be confusing the first time you look at them. Just keep in mind that options starting with “to” are used for responses and options starting with “from” are used for requests.

Other options are also available which are used to apply the rule to different states of a TCP connection.

- The *stateless* option is used to apply the rule without considering the state of a TCP session.
- The *established* option is used to apply the rule to established TCP sessions only.
- The *no\_stream* option enables rules to be applied to packets that are not built from a stream.
- The *stream\_only* option is used to apply the rules to only those packets that are built from a stream.

TCP streams are handled by the stream4 preprocessor discussed in the next chapter. TCP streams are also discussed in RFC 793. A TCP session is established and finished with a defined sequence of TCP packet exchanges as defined in RFC 793. The *stateless* and *established* options are related to TCP session state.

### 3.6.29 The session Keyword

The session keyword can be used to dump all data from a TCP session. It can dump all session data or just printable characters. The following rule dumps all printable data from POP3 sessions:

```
log tcp any any -> 192.168.1.0/24 110 (session: printable;)
```

If you use “all” as argument to this keyword, everything will be dumped. Use the logto keyword to log the traffic to a particular file.

A TCP session is a sequence of data packets exchanged between two hosts. The session is usually initiated and closed by the client using the three-way handshake method discussed in RFC 793. For example, when your e-mail client software starts collecting e-mail from a POP3 server, it first starts the communication by exchanging TCP packets. The mail is then downloaded. After downloading the e-mail, the client closes the connection. All communication taking place during this process is a TCP session.

### 3.6.30 The sid Keyword

The sid keyword is used to add a “Snort ID” to rules. Output modules or log scanners can use SID to identify rules. Authors have reserved SID ranges for rules as shown below:

- Range 0-99 is reserved for future use.
- Range 100-1,000,000 is reserved for rules that come with Snort distribution.
- All numbers above 1,000,000 can be used for local rules.

Refer to the list of rules that came with your Snort distribution for examples. The only argument to this keyword is a number. The following rule adds SID equal to 1000001.

```
alert ip any any -> any any (ipopts: lsrr; \
  msg: "Loose source routing attempt"; sid: 1000001;)
```

Using SID, tools like ACID can display the actual rule that generated a particular alert.

### 3.6.31 The tag Keyword

The tag keyword is another very important keyword that can be used for logging additional data from/to the intruder host when a rule is triggered. The additional data can then be analyzed later on for detailed intruder activity. The general syntax of the keyword is as follows:

```
tag: <type>, <count>, <metric>[, direction]
```

The arguments are explained in Table 3-5.

**Table 3-5** Arguments used with tag keyword

Argument	Description
Type	You can use either “session” or “host” as the type argument. Using session, packets are logged from the particular session that triggered the rule. Using host, all packets from the host are logged.
Count	This indicates either the number of packets logged or the number of seconds during which packets will be logged. The distinction between the two is made by the metric argument.
Metric	You can use either “packets” or “seconds” as mentioned above.
Direction	This argument is optional. You can use either “src” to log packets from source or “dst” to log packets from the destination.

The following rule logs 100 packets on the session after it is triggered.

```
alert tcp 192.168.2.0/24 23 -> any any \  
  (content: "boota"; msg: "Detected boota"; \  
  tag: session, 100, packets;)
```

### 3.6.32 The tos Keyword

The tos keyword is used to detect a specific value in the Type of Service (TOS) field of the IP header. The format for using this keyword is as follows:

```
tos: 1;
```

For more information on the TOS field, refer to RFC 791 and Appendix C, where the IP packet header is discussed.

### 3.6.33 The ttl Keyword

The ttl keyword is used to detect Time to Live value in the IP header of the packet. The keyword has a value which should be an exact match to determine the TTL value. This keyword can be used with all types of protocols built on the IP protocol, including ICMP, UDP and TCP. The general format of the keyword is as follows:

```
ttl: 100;
```

The traceroute utility uses TTL values to find the next hop in the path. The traceroute sends UDP packets with increasing TTL values. The TTL value is decremented at every hop. When it reaches zero, the router generates an ICMP packet to the source. Using this ICMP packet, the utility finds the IP address of the router. For example, to find the fifth hop router, the traceroute utility will send UDP packets with TTL value set to 5. When the packet reaches the router at the fifth hop, its value becomes zero and an ICMP packet is generated.

Using the ttl keyword, you can find out if someone is trying to traceroute through your network. The only problem is that the keyword needs an exact match of the TTL value.

For more information on the TTL field, refer to RFC 791 and Appendix C where the IP packet header is discussed.

### 3.6.34 The uricontent Keyword

The uricontent keyword is similar to the content keyword except that it is used to look for a string only in the URI part of a packet.

## 3.7 The Snort Configuration File

Snort uses a configuration file at startup time. A sample configuration file `snort.conf` is included in the Snort distribution. You can use any name for the configuration file, however `snort.conf` is the conventional name. You use the `-c` command line switch to specify the name of the configuration file. The following command uses `/opt/snort/snort.conf` as the configuration file.

```
/opt/snort/snort -c /opt/snort/snort.conf
```

You can also save the configuration file in your home directory as `.snortrc`, but specifying it on the command line is the most widely used method. There are other advantages to using the configuration file name as a command line argument to Snort. For example, it is possible to invoke multiple Snort instances on different network interfaces with different configuration. This file contains six basic sections:

- Variable definitions, where you define different variables. These variables are used in Snort rules as well as for other purposes, like specifying the location of rule files.
- Config parameters. These parameters specify different Snort configuration options. Some of them can also be used on the command line.
- Preprocessor configuration. Preprocessors are used to perform certain actions before a packet is operated by the main Snort detection engine.
- Output module configuration. Output modules control how Snort data will be logged.
- Defining new action types. If the predefined action types are not sufficient for your environment, you can define custom action types in the Snort configuration file.
- Rules configuration and include files. Although you can add any rules in the main `snort.conf` file, the convention is to use separate files for rules. These files are then included inside the main configuration file using the *include* keyword. This keyword will be discussed later in this chapter.

Although the out-of-the-box configuration file works, you need to modify it to adapt it to your environment. A sample configuration file is presented later on.

### 3.7.1 Using Variables in Rules

In the configuration file, you can use variables. This is a very convenient way of creating rules. For example, you can define a variable `HOME_NET` in the configuration file.

```
var HOME_NET 192.168.1.0/24
```

Later on you can use this variable `HOME_NET` in your rules:

```
alert ip any any -> $HOME_NET any (ipopts: lsrr; \  
  msg: "Loose source routing attempt"; sid: 1000001;)
```

As you can see, using variables makes it very convenient to adapt the configuration file and rules to any environment. For example, you don't need to modify all rules when you copy rules from one network to another; you just need to modify a single variable.

### 3.7.1.1 Using a List of Networks in Variables

You can also define variables that contain multiple items. Consider that you have multiple networks in the company. Your intrusion detection system is right behind the company firewall connecting to the Internet. You can define a variable as a list of all of these networks. The following variable shows that `HOME_NETWORK` consists of two networks, `192.168.1.0/24` and `192.168.10.0/24`.

```
var HOME_NET [192.168.1.0/24,192.168.10.0/24]
```

All networks in the variable name are separated by a comma.

### 3.7.1.2 Using Interface Names in Variables

You can also use interface names in defining variables. The following two statements define `HOME_NET` and `EXTERNAL_NET` variables on a Linux machine.

```
var HOME_NET $eth0_ADDRESS  
var EXTERNAL_NET $eth1_ADDRESS
```

The `HOME_NET` variable uses the IP address and network mask value assigned to interface `eth0` and `EXTERNAL_NET` uses the IP address and network mask assigned to network interface `eth1`. This arrangement is more convenient since you can change IP addresses on the interfaces without modifying rules or even variables themselves.

### 3.7.1.3 Using the any Keyword

The `any` keyword can also be a variable. It matches to everything, just as it does in rules (such as addresses and port numbers). For example, if you want to test packets regardless of their source, you can define a variable like the following for `EXTERNAL_NET`.

```
var EXTERNAL_NET any
```

There are many variables defined in the `snort.conf` file that come with the Snort distribution. While installing Snort, you need to modify these variables according to your network.

### 3.7.2 The config Directives

The config directives in the `snort.conf` file allow a user to configure many general settings for Snort. Examples include the location of log files, the order of applying rules and so on. These directives can be used to replace many command line options as well. The general format of applying a config directive is as follows:

```
config directive_name[: value]
```

Table 3-6 shows a list of directives used in the `snort.conf` file.

**Table 3-6** Snort config directives

Directive	Description
order	Changes the order in which rules are applied. It is equivalent to the <code>-o</code> command line option.
alertfile	Used to set the name of the alert file. Alert file is created in log directory (see <code>logdir</code> directive).
classification	Builds classification for rules. See explanation of the <code>classtype</code> keyword used in rules.
decode_arp	Equivalent to <code>-a</code> command line option. It turns ON arp decoding.
dump_chars_only	Equivalent <code>-C</code> command line option.
dump_payload	Equivalent to <code>-d</code> command line option. It is used to dump the data part of the packet.
decode_data_link	Equivalent to <code>-e</code> command line option. Using this directive you can decode data link layer headers (Ethernet header, for example).
bpf_file	Equivalent to <code>-F</code> command line option.
set_gid	Equivalent to <code>-g</code> command line option. Using this directive you can set the group ID under which Snort runs. For example, you can use “ <code>config set_gid: mygroup</code> ”
daemon	Equivalent to <code>-D</code> command line option. It invokes Snort as daemon instead of foreground process.
reference_net	Equivalent to <code>-h</code> command line option. It sets the home network address.
interface	Equivalent to <code>-i</code> command line option. It sets the interface for Snort.
alert_with_interface_name	Equivalent to <code>-T</code> command line option. This directive is used to append the interface name to the alert message. This is sometimes useful if you are monitoring multiple interfaces on the same sensor.
logdir	Equivalent to <code>-l</code> command line option. It sets the directory where Snort logs data. The default location of the log directory is <code>/var/log/snort</code> .

**Table 3-6** Snort config directives (continued)

Directive	Description
umask	Equivalent to <code>-m</code> command line option. Using this option you can set the UMASK while running Snort.
pkt_count	Equivalent to <code>-n</code> command line option. Using this directive you can exit from Snort after a defined number of packets.
nolog	Equivalent to <code>-N</code> command line option. Logging is disabled except alerts. Remember, alerts are really both alerts and logs.
obfuscate	Equivalent to <code>-O</code> command line option. It is used to obfuscate IP addresses so that you are able to send the logs for analysis to someone without disclosing the identity of your network.
no_promisc	Equivalent to <code>-p</code> command line option and is used to disable promiscuous mode.
quiet	Equivalent to <code>-q</code> command line option. This will disable banner information at Snort startup time and prevent statistical information from being displayed.
chroot	Equivalent to <code>-t</code> command line option. It is used to change root directory for Snort to a specific directory.
checksum_mode	Used to checksum for particular types of packets. It takes arguments such as none, noip, notcp, noicmp, noudp, and all.
set_uid	Equivalent to <code>-u</code> command line option and is used to set user ID for the Snort process.
utc	Equivalent to <code>-U</code> command line option and is used to use UTC instead of local time in alerts and logs.
verbose	Equivalent to <code>-v</code> command line option. It is used to log messages to standard output in addition to standard logging.
dump_payload_verbose	Equivalent to <code>-X</code> command line option. This dumps the received raw packet on the standard output.
show_year	Equivalent to <code>-y</code> command line option and is used to display year in the timestamp.
stateful	Used to set assurance mode for stream4 preprocessor. Preprocessors are discussed in detail in Chapter 4.

You have already seen how the classification directive is used in the `classification.config` file. As another example, the following line is used to start Snort in the daemon mode.

```
config daemon
```

You can also use `-D` command line option to start Snort in the daemon mode.

### 3.7.3 Preprocessor Configuration

Preprocessors or input plug-ins operate on received packets before Snort rules are applied to them. The preprocessor configuration is the second major part of the configuration file. This section provides basic information about adding or removing Snort preprocessors. Detailed information about each preprocessor is found in the next chapter.

The general format of configuring a preprocessor is as follows:

```
preprocessor <preprocessor_name>[: <configuration_options>]
```

The first part of the line is the keyword *preprocessor*. The name of the preprocessor follows this keyword. If the preprocessor can accept some options or arguments, you can list these options after a colon character at the end of the name of preprocessor, which is optional.

The following is an example of a line in the configuration file for IP defragmentation preprocessor frag2.

```
preprocessor frag2
```

The following is an example of a stream4 preprocessor with an argument to detect port scans. The stream4 preprocessor has many other arguments as well, as described in Chapter 4.

```
preprocessor stream4: detect_scans
```

Both frag2 and stream4 are predefined preprocessors. You can also write your own preprocessors if you are a programmer. Guidelines for writing preprocessors are provided with the Snort source code.

### 3.7.4 Output Module Configuration

Output modules, also called output plug-ins, manipulate output from Snort rules. For example, if you want to log information to a database or send SNMP traps, you need output modules. The following is the general format for specifying an output module in the configuration file.

```
output <output_module_name>[: <configuration_options>]
```

For example, if you want to store log messages to a MySQL database, you can configure an output module that contains the database name, database server address, user name and password.

```
output database: alert, mysql, user=rr password=boota \  
  dbname=snort host=localhost
```

There may be additional steps to make the output module work properly. In the case of MySQL database, you need to setup a database, create tables, create user, set permissions and so on. More information on configuring output modules is found in Chapter 4.

### 3.7.5 Defining New Action Types

You already know that the first part of each Snort rule is the action item. Snort has predefined action types; however, you can also define your own action types in the configuration file. A new action type may use multiple output modules. The following action type creates alert messages that are logged into the database as well as in a file in the tcpdump format.

```
ruletype dump_database
{
  type alert
  output database: alert, mysql, user=rr dbname=snort \
    host=localhost
  output log_tcpdump: tcpdump_log_file
}
```

This new action type can be used in rules just like other action types.

```
dump_database icmp any any -> 192.168.1.0/24 any \
  (fragbits: D; msg: "Don't Fragment bit set");
```

When a packet matches the criteria in this rule, the alert will be logged to the database as well as to the `tcpdump_log_file`.

### 3.7.6 Rules Configuration

The rules configuration is usually the last part of the configuration file. You can create as many rules as you like using variables already defined in the configuration file. All of the previous discussion in this chapter was about writing new rules. The rules configuration is the place in the configuration file where you can put your rules. However the convention is to put all Snort rules in different text files. You can include these text files in the `snort.conf` file using the “include” keyword. Snort comes with many predefined rule files. The names of these rule files end with `.rule`. You have already seen in the last chapter how to put these rule files in the proper place during the installation process.

### 3.7.7 Include Files

You can include other files inside the main configuration file using the *include* keyword. You can think of including a file as equivalent to inserting the contents of the

included file into the main configuration file at the point where it is included. In fact, most of the predefined rules that come with the Snort distribution are found in include files. All files in the Snort distribution whose name ends with `.rules` contain rules and they are included in the `snort.conf` file. These rule files are included in the main `snort.conf` file using the “include” keyword. The following is an example of including `myrules.rules` file in the main configuration file.

```
include myrules.rules
```

It is not necessary that the name of the rules file must end with `.rule`. You can use a name of your choice for your rule file.

### 3.7.8 Sample `snort.conf` File

The following is a sample configuration file for Snort. All lines starting with the `#` character are comment lines. Whenever you modify the configuration file, you have to restart Snort for the changes to take effect.

```
# Variable Definitions
var HOME_NET 192.168.1.0/24
var EXTERNAL_NET any
var HTTP_SERVERS $HOME_NET
var DNS_SERVERS $HOME_NET
var RULE_PATH ./

# preprocessors
preprocessor frag2
preprocessor stream4: detect_scans
preprocessor stream4_reassemble
preprocessor http_decode: 80 -unicode -cginull
preprocessor unidecode: 80 -unicode -cginull
preprocessor bo: -nobrute
preprocessor telnet_decode
preprocessor portscan: $HOME_NET 4 3 portscan.log
preprocessor arpspoof

# output modules
output alert_syslog: LOG_AUTH LOG_ALERT
output log_tcpdump: snort.log
output database: log, mysql, user=rr password=boota \
    dbname=snort host=localhost
output xml: log, file=/var/log/snortxml

# Rules and include files
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
```

```
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/smtp.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/sql.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/myrules.rules
```

### 3.8 Order of Rules Based upon Action

The five types of the rules can be categorized into three basic types.

1. Alert rules
2. Pass rules
3. Log rules

When a packet is received by Snort, it is checked in this order. Each packet has to go through all Alert rule checks before it is allowed to pass. This scheme is the most secure since no packet passes through without being checked against all alert types. However most of the packets are normal traffic and do not show any intruder activity. Testing all of the packets against all alert rules requires a lot of processing power. Snort provides a way to change this testing order to one which is more efficient, but more dangerous.

1. Pass rules
2. Alert rules
3. Log rules

You must be careful when choosing this order because just one badly written pass rule may allow many alert packets to pass through without being checked. If you really know what you are doing, you can use the `-o` command line switch to disable the default order and enable the new order of applying rules. You can also use “config order” in the configuration file for this purpose. Again, this is dangerous and you have been warned twice now! If you are sure of what you are doing, add this line in the `snort.conf` file:

```
config order
```

If you define your own rule types, they are checked last in the sequence. For example, if you have defined a rule type `snmp_alerts`, the order of rule application will be:

```
Alert -> Pass -> Log ->snmp_alerts
```

## 3.9 Automatically Updating Snort Rules

There are multiple tools available to update Snort signatures. When using any of these tools you must be careful because you may accidentally modify or delete your customized rules. I shall discuss two methods of updating rules.

### 3.9.1 The Simple Method

This method consists of a simple shell script. It requires that you have `wget` program installed on your system. The `wget` program is used to retrieve any file using HTTP protocol. In essence, it is just like a web browser, but it retrieves one file from a command line argument.

```
#!/bin/sh
# Place of storing your Snort rules. Change these variables
# according to your installation.

RULESDIR=/etc/snort
RULESDIRBAK=/etc/snort/bak

# Path to wget program. Modify for your system if needed.
WGETPATH=/usr/bin

# URI for Snort rules
RULESURI=http://www.snort.org/downloads/snortrules.tar.gz

# Get and untar rules.
cd /tmp
rm -rf rules
$WGETPATH/wget $RULESURI
```

```
tar -zxf snortrules.tar.gz
rm -f snortrules.tar.gz

# Make a backup copy of existing rules
mv $RULESDIR/*.rules $RULESDIRBAK

# Copy new rules to the location
mv /tmp/rules/*.rules $RULESDIR
```

Let us explore how this script works. The following lines simply set some variables.

```
RULESDIR=/etc/snort
RULESDIRBAK=/etc/snort/bak
WGETPATH=/usr/bin
RULESURI=http://www.snort.org/downloads/snortrules.tar.gz
```

The following three lines are used to go to `/tmp` directory, remove any existing directory `/tmp/rules` and download the `snortrules.tar.gz` file from the URI specified by the `$RULESURI` variable.

```
cd /tmp
rm -rf rules
$WGETPATH/wget $RULESURI
```

After downloading, you extract the rules files from `snortrules.tar.gz` file and then delete it using the following two lines. The files extracted are placed in `/tmp/rules` directory.

```
tar -zxf snortrules.tar.gz
rm -f snortrules.tar.gz
```

The following line makes a backup copy of existing rules files, just in case you need the old copy later on.

```
mv $RULESDIR/*.rules $RULESDIRBAK
```

The last line in the script moves new rules from `/tmp/rules` directory to the actual rules directory `/etc/snort` where Snort can read them.

```
mv /tmp/rules/*.rules $RULESDIR
```

Make sure to restart Snort after running this script. If you have a start script like the one described in Chapter 2, you can add a line at the end of the shell script to restart Snort.

```
/etc/init.d/snortd restart
```

You may also restart Snort using the command line.

### 3.9.2 The Sophisticated and Complex Method

This section provides information about the use of Oinkmaster found at <http://www.algonet.se/~nitzer/oinkmaster/>. Oinkmaster is a tool to update Snort rule files. It is written in Perl, so you must have Perl installed on your Snort machine to make this tool work. It can be configured to download new rule files from the Internet, find out what rules need to be updated and then updates them. If you have modified some standard rules according to your own requirements, you can configure Oinkmaster not to update these customized rules. At the time of writing this book, version 0.6 of this tool is available. By now updated versions may be available. Oinkmaster is a Perl script and uses a configuration file to update the rules.

It is recommended that you use a temporary directory the first time you use this Perl script. I have used `/tmp/rules` directory. When you use the following command, it will download all rules, untar them and save all files in `/tmp/rules` directory.

```
[rr@conformix]$ ./oinkmaster.pl -o /tmp/rules/
Downloading rules archive from http://www.snort.org/dl/signatures/
snortrules.tar.gz...
12:27:09 URL:http://www.snort.org/dl/signatures/snortrules.tar.gz [79487/79487]
-> "/tmp/oinkmaster.9875/snortrules.tar.gz" [1]
Archive successfully downloaded, unpacking... tar: rules/attack-responses.rules:
time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/classification.config: time stamp 2002-07-14 13:10:24 is 348194 s in
the future
tar: rules/sid-msg.map: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/x11.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/web-misc.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/web-iis.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/web-frontpage.rules: time stamp 2002-07-14 13:10:24 is 348194 s in
the future
tar: rules/web-coldfusion.rules: time stamp 2002-07-14 13:10:24 is 348194 s in
the future
tar: rules/web-cgi.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/web-attacks.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/virus.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/tftp.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/telnet.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/sql.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/smtp.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/shellcode.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/scan.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
```

```
tar: rules/rservices.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/rpc.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/porn.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/policy.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/netbios.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/misc.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/local.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/info.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/icmp.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/icmp-info.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/ftp.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/finger.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/exploit.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/dos.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/dns.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/ddos.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules/bad-traffic.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/backdoor.rules: time stamp 2002-07-14 13:10:24 is 348194 s in the
future
tar: rules/snort.conf: time stamp 2002-07-14 13:10:24 is 348194 s in the future
tar: rules: time stamp 2002-07-14 13:10:24 is 348194 s in the future
done.
Disabling rules according to ./oinkmaster.conf... 0 rules disabled.
Comparing new files to the old ones... done.
```

```
[***] Results from Oinkmaster started Wed Jul 10 12:25:37 2002 [***]
```

```
[*] Rules added/removed/modified: [*]
```

```
[+++]           Added:           [+++]
```

```
-> File "tftp.rules":
    alert udp any any -> any 69 (msg:"TFTP GET shadow"; content: "|0001|";
offset:0; depth:2; content:"shadow"; nocase; classtype:successful-admin;
sid:1442; rev:1;)
    alert udp any any -> any 69 (msg:"TFTP GET passwd"; content: "|0001|";
offset:0; depth:2; content:"passwd"; nocase; classtype:successful-admin;
sid:1443; rev:1;)
    alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP parent directory";
content:".."; reference:arachnids,137; reference:cve,CVE-1999-0183;
classtype:bad-unknown; sid:519; rev:1;)
```

```
[///]           Modified active:   [///]
```

```
-> File "tftp.rules":
```

```

Old: alert udp $EXTERNAL_NET any -> $HOME_NET 64 (msg:"TFTP Put";
content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183;
reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)
New: alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Put";
content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183;
reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)

```

```

[*] Non-rule lines added/removed: [*]
None.

```

```

[*] Added files: [*]
None.

```

The tool gives you a detailed report of actions taken during the update process. You can test this by deleting and modifying some rules and running the tool again. The following is a partial output seen when Oinkmaster adds and updates some rules.

Comparing new files to the old ones... done.

```

[***] Results from Oinkmaster started Wed Jul 10 12:25:37 2002 [***]

```

```

[*] Rules added/removed/modified: [*]

```

```

[+++]           Added:           [+++]

```

```

-> File "tftp.rules":
  alert udp any any -> any 69 (msg:"TFTP GET shadow"; content:"|0001|";
offset:0; depth:2; content:"shadow"; nocase; classtype:successful-admin;
sid:1442; rev:1;)
  alert udp any any -> any 69 (msg:"TFTP GET passwd"; content:"|0001|";
offset:0; depth:2; content:"passwd"; nocase; classtype:successful-admin;
sid:1443; rev:1;)
  alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP parent directory";
content:".."; reference:arachnids,137; reference:cve,CVE-1999-0183;
classtype:bad-unknown; sid:519; rev:1;)

```

```

[////]           Modified active:           [////]

```

```

-> File "tftp.rules":
  Old: alert udp $EXTERNAL_NET any -> $HOME_NET 64 (msg:"TFTP Put";
content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183;
reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)
  New: alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Put";
content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183;
reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)

```

```

[*] Non-rule lines added/removed: [*]
None.

```

```

[*] Added files: [*]
None.

```

The script uses a configuration file where many options can be configured. Specifically you can configure the following in the configuration file `oinkmaster.conf`:

- URL of the location from where it downloads the Snort rules. By default this URL is `http://www.snort.org/downloads/signatures/snortrules.tar.gz` or `http://www.snort.org/downloads/snortrules.tar.gz`. This is configured using the `url` keyword in the configuration file.
- Files to be updated. By default files ending with `.rules`, `.config`, `.conf`, `.txt` and `.map` are updated and all other files are ignored. This is done using the `update_files` keyword.
- Files to be skipped when updating rules. This is done using the `skipfile` keyword. You can use as many `skipfiles` lines as you like. This option is useful when you have customized rules in some files. When you skip these files, your customized rules will not be overwritten during the update process.
- You can disable certain rules permanently using the `disableid` keyword in the configuration file. The tool will not update these rules during the update.

Please use the `README` and `INSTALL` files that come with the tool. You can use this tool from a cron script to periodically update your rule set.

### 3.10 Default Snort Rules and Classes

Snort comes with a rich set of rules. These rules are divided into different files. Each file represents one class of rules. In the source code distribution of Snort, these files are present under the `rules` directory in the source code tree. The following is a list of the rule files in Snort 1.9.0 distribution:

```
attack-responses.rules
backdoor.rules
bad-traffic.rules
chat.rules
ddos.rules
deleted.rules
dns.rules
dos.rules
experimental.rules
exploit.rules
finger.rules
ftp.rules
icmp-info.rules
icmp.rules
```

```
imap.rules
info.rules
local.rules
Makefile
Makefile.am
Makefile.in
misc.rules
multimedia.rules
mysql.rules
netbios.rules
nntp.rules
oracle.rules
other-ids.rules
p2p.rules
policy.rules
pop3.rules
porn.rules
rpc.rules
rservices.rules
scan.rules
shellcode.rules
smtp.rules
snmp.rules
sql.rules
telnet.rules
tftp.rules
virus.rules
web-attacks.rules
web-cgi.rules
web-client.rules
web-coldfusion.rules
web-frontpage.rules
web-iis.rules
web-misc.rules
web-php.rules
x11.rules
```

For example, all rules related to X-Windows attacks are combined in `x11.rules` file.

```
# (C) Copyright 2001,2002, Martin Roesch, Brian Caswell, et al.
# All rights reserved.
# $Id: x11.rules,v 1.12 2002/08/18 20:28:43 cazz Exp $
#-----
# X11 RULES
#-----
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 6000 (msg:"X11 MIT Magic
  Cookie detected"; flow:established
; content: "MIT-MAGIC-COOKIE-1"; reference:arachnids,396;
  classtype:attempted-user; sid:1225; rev:3;
)
alert tcp $EXTERNAL_NET any -> $HOME_NET 6000 (msg:"X11 xopen";
  flow:established; content: "|6c00 0b
00 0000 0000 0000 0000|"; reference:arachnids,395; classtype:unknown;
  sid:1226; rev:2;)
```

Similarly, each file contains rules specific to a particular class. The `dns.rules` file contains all rules related to attacks on DNS servers, the `telnet.rules` file contains all rules related to attacks on the telnet port, and so on.

### 3.10.1 The local.rules File

The `local.rules` file has no rules. This is meant to be used by Snort administrator for customized rules. However, you can use any file name for your own customized rules and include it in the main `snort.conf` file.

## 3.11 Sample Default Rules

You have learned the structure of Snort rules and how to write your own rules. This section lists some predefined rules that come with Snort. All of the rules in this section are taken from the `telnet.rules` file. Let us discuss each of these to give you an idea about rules that are used in production systems.

### 3.11.1 Checking su Attempts from a Telnet Session

The first rule generates an alert when a user tries to `su` to root through a telnet session. The rule is as shown below:

```
alert tcp $TELNET_SERVERS 23 -> $EXTERNAL_NET any (msg:"TELNET
  Attempted SU from wrong group"; flow:
from_server,established; content:"to su root"; nocase;
  classtype:attempted-admin; sid:715; rev:6;)
```

There are a number of things to note about this rule. The rule generates an alert and applies to TCP packets. Major points are listed below:

- The variable `$TELNET_SERVERS` is defined in `snort.conf` file and shows a list of Telnet servers.
- Port number 23 is used in the rule, which means that the rule will be applied to TCP traffic going from port 23. The rule checks only response from Telnet servers, not the requests.

- The variable `$EXTERNAL_NET` is defined in the `snort.conf` file and shows all addresses which are outside the private network. The rule will apply to those telnet sessions which originate from outside of the private network. If someone from the internal network starts a Telnet session, the rule will not detect that traffic.
- The `flow` keyword is used to apply this rule only to an established connection and traffic flowing from the server.
- The `content` keyword shows that an alert will be generated when a packet contains “to su root”.
- The `nocase` keyword allows the rule to ignore case of letters while matching the content.
- The `classtype` keyword is used to assign a class to the rule. The `attempted-admin` class is defined with a default priority in `classification.config` file.
- The rule ID is 715.
- The `rev` keyword is used to show version of the rule.

### 3.11.2 Checking for Incorrect Login on Telnet Sessions

The following rule is similar to the rule for checking su attempts. It checks incorrect login attempts on the Telnet server port.

```
alert tcp $TELNET_SERVERS 23 -> $EXTERNAL_NET any (msg:"TELNET login
  incorrect"; content:"Login inco
  rrect"; flow:from_server,established; reference:arachnids,127;
  classtype:bad-unknown; sid:718; rev:6;)
```

There is one additional keyword used in this rule which is “reference: arachnids, 127”. This is a reference to a web site where you can find more information about this vulnerability. The URLs for external web sites are placed in the `reference.config` file in the Snort distribution. Using the information in `reference.config`, the URL for more information about this rule is <http://www.whitehats.com/info/IDS=127>. 127 is the ID used for searching the database at the arachnids web site.

## 3.12 Writing Good Rules

There is a large list of predefined rules that are part of Snort distribution. Looking at these rules gives you a fairly good idea of how to write good rules. Although it is not mandatory, you should use the following parts in the options for each rule:

- A message part using the `msg` keyword.
- Rule classification, using the `classification` keyword.

- Use a number to identify a rule with the help of the sid keyword.
- If the vulnerability is known, always use a reference to a URL where more information can be found using the reference keyword.
- Always use the rev keyword in rules to keep a record of different rule versions.

In addition, you should always try to write rules that are generalized and are able to detect multiple variations of an attack. Usually bad guys use the same tools with little modifications for different purposes. Good rules can and should be able to detect these variations.

### 3.13 References

1. Classless Inter-Domain Routing or CIDR. RFC 1519 at <http://www.rfc-editor.org/rfc/rfc1519.txt>
2. Transmission Control Protocol RFC 793 at <http://www.rfc-editor.org/rfc/rfc793.txt>
3. User Datagram Protocol RFC 768 at <http://www.rfc-editor.org/rfc/rfc768.txt>
4. The nmap at it web site <http://www.nmap.org>
5. The Internet Protocol RFC 791 at <http://www.rfc-editor.org/rfc/rfc791.txt>
6. The Internet Control Message Protocol at <http://www.rfc-editor.org/rfc/rfc792.txt>
7. Assigned Numbers RFC 1700 at <http://www.rfc-editor.org/rfc/rfc1700.txt>
8. Oinkmaster at <http://www.algonet.se/~nitzer/oinkmaster/>
9. Open NMS at <http://www.opennms.org>
10. Internet Corporation for Assigned Names and Numbers (ICANN) at <http://www.icann.org>
11. The arachnids web site at <http://www.whitehats.com/info/IDS>
12. The securityfocus mailing list archive at <http://online.securityfocus.com/archive/1>