Chapter 4

Working with Rules

Solutions in this chapter:

- Introducing Rules
- Understanding the OSSEC HIDS Analysis Process
- Predecoding Events
- Decoding Events
- Understanding Rules
- Working with Real World Examples
- Writing Decoders/Rules for Custom Applications

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

David and his team, under the guidance of Marty, installed the OSSEC HIDS agents on roughly 100 servers and about 50 workstations in the lab environment. Because this was a test environment, Marty deployed only three OSSEC HIDS servers to centralize the events from the deployed agents. He knew that in the corporate environment he would require many more OSSEC HIDS servers. The lab environment was a scaled down clone of the corporate production environment but the same event rate was impossible to simulate given his smaller lab budget. David's team consisted of his three most senior engineers: Ming Tsai, Mark Olyphant, and Sergei Zbruyev. David and Sergei configured five Cisco PIX firewalls, two Juniper Netscreen firewalls, and seven Cisco IOS based routers to send their logs to the first OSSEC HIDS server, named Saturn. Mark handled the OSSEC HIDS agent installations on Windows systems. Ming and Sergei handled the OSSEC HIDS agent installations on all the Linux, UNIX, and BSD systems in the lab environment. Under the guidance of Marty, Mark configured the agents to monitor the mail logs on the Microsoft Exchange server and the FTP logs on the Microsoft FTP servers. All the Windows agents were configured to send their logs to Mars, the second OSSEC HIDS server. Ming and Sergei ensured that the OSSEC HIDS agents were monitoring all the MySQL and PostgreSQL database server logs, Apache HTTPD server logs, Snort IDS logs, Sendmail logs, and Squid proxy server logs on the deployed hosts. All the Linux, UNIX, and BSD agents were configured to send their logs to Venus, the third OSSEC HIDS server.

The configuration in the lab took about two days and, to the surprise of David, there were no issues. He and his team monitored the systems closely for any errors, crashes, or reboots and, in a quick meeting with Marty, he gave his blessing to continue to the next phase.

"They're all yours Marty," said David. "You should have more than enough logs for your proof of concept and rule writing session with Antoine and his team." Marty shook David's hand.

"Thanks for all your help on this one Dave." David shook his head.

"You don't need to thank me!" exclaimed David. "Anything I can do to help prevent another embarrassment like that last breach and ensure my continued employment is no waste of time in my books." Marty thanked David and went to Simran's office to provide an update.

"I'm going to schedule the meeting with Antoine and his team tomorrow to review the rule configuration options with the OSSEC HIDS," said Marty. "I figure I should plan on taking the whole day so we can create a thorough library of alert rules from our lab environment traffic." Marty put his hands in his pockets and looked around the room, hesitant to ask his next question. "So....I can expense lunch for the team right?" asked Marty. Simran looked up from her paperwork at Marty. "Oh come on! I can't be responsible for Antoine's team being malnourished." Marty smiled. "How could I live with myself?" Simran rolled her eyes.

"Fine, have lunch brought in. But," Simran paused, "make sure you save me a salad."

Marty wasn't sure if it was the promise of a free lunch or the topic of discussion that brought so many of Antoine's team to the meeting. "I guess if it was the lunch then they would have just showed up at noon," Marty thought to himself. Antoine had brought seven of his senior incident handlers and three of his senior forensic analysts who were the most familiar with the types of systems used in the lab environment. They were also well versed in the types of information required to perform a proper incident handling exercise. They knew what to look for, where to look for it, and how to preserve it in case it had to be used as evidence.

"Well, Marty," said Antoine "You've got our full attention. I want to make sure that everyone in this room is up to speed by the end of the day." Marty nodded in agreement. He provided an overview of the OSSEC HIDS, its capabilities, how to write predecoders, decoders, and rules. He also discussed where the OSSEC HIDS fits into the current incident handling model. As Marty spoke he noticed that everyone in the room looked interested but he could tell that people were still exhausted from the last breach. Marty made sure to highlight how the OSSSEC HIDS could have helped shave valuable hours off their investigation and perhaps even prevent the breach altogether.

When the overview was complete Marty divided everyone up into three teams.

"Alpha team," Marty pointed to a network diagram showing the installed OSSEC HIDS agents. "You're responsible for configuring rules for the firewall and router logs being sent to Saturn. Bravo team," Marty pointed to another area of the diagram. "You need to handle the Windows system and application events coming into Mars." Marty pointed to the final area of the diagram. "Charlie team, you need to handle the Linux, Unix, BSD system, and application events." Marty took a sip of his coffee. "I want to see e-mails and SMS pages for high-priority alerts. Think real-world incidents everyone," Marty said. "You need to create rules that will make all of our jobs easier in the long run." Marty sat down and noticed an instant message on his laptop from Simran:

Simran> `How's it going?`

Marty> `Good, after lunch we'll get into the syscheck and rootkit discovery.`

Simran> `Excellent, is everyone participating?`

Marty> `Yes, everyone is in their teams and plugging away at the rules.`

Simran> `Good…and my salad?`

Marty> `Operation` *"Salad Grab"* `is still a go!`

Introducing Rules

Now that you are familiar with the installation and configuration of the OSSEC HIDS it's time to move onto the most powerful features of the product. The decoders and rules give the OSSEC HIDS its power. When combined they allow you to configure and tune every alert from the OSSEC HIDS, including those generated for integrity checking alerts, syslog and agent log events, and rootkit detection alerts.

Every OSSEC rule is stored inside the *rules/* directory of your OSSEC HIDS installation. This is typically in */var/ossec/rules/*. Each rule is defined in a separate XML file and is named accordingly. For example, all rules for the Apache HTTP server are located within the *apache_rules.xml* file just as all rules for the Cisco PIX firewall are located within the *pix_rules.xml* file. The default installation of the OSSEC HIDS contains 43 rules files, described in Table 4.1.

Table 4.1 Default OSSEC HIDS Rules

Rule Name	Description
apache_rules.xml	Apache HTTP server rules
arpwatch_rules.xml	Arpwatch rules
attack_rules.xml	Common attack rules
cisco-ios_rules.xml	Cisco IOS firmware rules
courier_rules.xml	Courier mail server rules
firewall_rules.xml	Common firewall rules
ftpd_rules.xml	Rules for the ftpd daemon
hordeimp_rules.xml	Horde Internet Messaging Program rules
ids_rules.xml	Common IDS rules
imapd_rules.xml	Rules for the *imapd* daemon
local_rules.xml	OSSEC HIDS local, user-defined rules
mailscanner_rules.xml	Common mail scanner rules
msauth_rules.xml	Microsoft Authentication rules
ms-exchange_rules.xml	Microsoft Exchange server rules
netscreenfw_rules.xml	Juniper Netscreen firewall rules
ms_ftpd_rules.xml	Microsoft FTP server rules
mysql_rules.xml	MySQL database rules
named_rules.xml	Rules for the *named* daemon
ossec_rules.xml	Common OSSEC HIDS rules
pam_rules.xml	Pluggable Authentication Module (PAM) rules
pix_rules.xml	Cisco PIX firewall rules
policy_rules.xml	Policy specific event rules
postfix_rules.xml	Postfix mail transfer agent rules
postgresql_rules.xml	PostgerSQL database rules

Continued

Table 4.1 Continued. Default OSSEC HIDS Rules

Rule Name	Description
proftpd_rules.xml	ProFTPd FTP server rules
pure-ftpd_rules.xml	Pure-FTPd FTP server rules
racoon_rules.xml	Racoon VPN device rules
rules_config.xml	OSSEC HIDS Rules configuration rules
sendmail_rules.xml	Sendmail mail transfer agent rules
squid_rules.xml	Squid proxy server rules
smbd_rules.xml	Rules for the *smbd* daemon
sonicwall_rules.xml	SonicWall firewall rules
spamd_rules.xml	Rules for the *spamd* spam-deferral daemon
sshd_rules.xml	Secure Shell (SSH) network protocol rules
symantec-av_rules.xml	Symantec Antivirus rules
symantec-ws_rules.xml	Symantec Web Security rules
syslog_rules.xml	Common syslog rules
telnetd_rules.xml	Rules for the telnetd daemon
vpn_concentrator_rules.xml	Cisco VPN Concentrator rules
vpopmail_rules.xml	Rules for the *vpopmail* virtual mail domain application
vsftpd_rules.xml	Rules for the vsftpd FTP server
web_rules.xml	Common web server rules
zeus_rules.xml	Zeus web server rules

Each rule file contains multiple rule definitions for the application or device. This provides you with over 600 rules for anything from ProFTPD logs, to Snort NIDS alerts, to Cisco VPN Concentrator messages, to OSSEC HIDS Integrity Checking and rootkit detection logs.

Coupled with the rules, we have the *decoders*, which are defined within the *decoders.xml* file in the */etc* directory of your OSSEC HIDS installation. This typically is found in */var/ossec/etc/decoders.xml*. Decoders are designed to extract data from the raw events, which enables the OSSEC HIDS to correlate disparate events received from multiple sources.

Every rule has a unique ID that is assigned when it is first created. For each log type, a range of IDs have been assigned to ensure that OSSEC HIDS released decoders are not overwritten by mistake. The OSSEC HIDS team does, however, provide you with a dedicated range of IDs to be used for user created rules. This range is 100,000 to 119,999, inclusive. Table 4.2 shows the current reserved ID assignments.

TIP

Within the OSSEC HIDS world, user-created rules are referred to as *local rules*. Local rules should range from 100,000 to 119,999. If you choose any other ID you might cause a conflict with the official rules from the OSSEC HIDS project. A complete list with all current ranges is available at the OSSEC HIDS Web site: *www.ossec.net/wiki/index.php/Know_How:RuleIDGrouping*.

Table 4.2 Reserved ID Assignments

Rule ID Range	General Category
00000–00999	Reserved for internal OSSEC HIDS rules
01000–01999	General syslog rules
02100–02299	Network File System (NFS) rules
02300–02499	xinetd rules
02500–02699	Access control rules
02700–02729	mail/procmail rules
02800–02829	*smartd* rules
02830–02859	*crond* rules
02860–02899	Mount/Automount rules
03100–03299	Sendmail mail server rules
03300–03499	Postfix mail server rules
03500–03599	*spamd* filter rules
03600–03699	*imapd* mail server rules
03700–03799	Mail scanner rules
03800–03899	Microsoft Exchange mail server rules
03900–03999	Courier mail rules (*imapd/pop3d/pop3-ssl*)
04100–04299	Generic firewall rules
04300–04499	Cisco PIX/FWSM/ASA firewall rules

Continued

Table 4.2 Continued. Reserved ID Assignments

Rule ID Range	General Category
04500–04699	Juniper Netscreen firewall rules
04700–04799	Cisco IOS rules
04800–04899	SonicWall firewall rules
05100–05299	Linux, UNIX, BSD kernel rules
05300–05399	Switch user (*su*) rules
05400–05499	Super user do (*sudo*) rules
05500–05599	Unix pluggable authentication module (*PAM*) rules
05600–05699	*telnetd* rules
05700–05899	*sshd* rules
05900–05999	Add user or user deletion rules
07100–07199	Tripwire rules
07200–07299	*arpwatch* rules
07300–07399	Symantec Antivirus rules
07400–07499	Symantec Web Security rules
09100–09199	Point-to-point tunneling protocol (*PPTP*) rules
09200–09299	Squid syslog rules
09300–09399	Horde IMP rules
09900–09999	*vpopmail* rules
10100–10199	FTS rules
11100–11199	*ftpd* rules
11200–11299	ProFTPD rules
11300–11399	Pure-FTPD rules
11400–11499	vs-FTPD rules
11500–11599	MS-FTP rules
12100–12299	*named* (BIND DNS) rules
13100–13299	Samba (*smbd*) rules
14100–14199	Racoon SSL rules
14200–14299	Cisco VPN Concentrator rules
17100–17399	Policy rules
18100–18499	Windows system rules
20100–20299	IDS rules

Continued

Table 4.2 Continued. Reserved ID Assignments

Rule ID Range	General Category
20300–20499	IDS (Snort specific) rules
30100–30999	Apache HTTP server error log rules
31100–31199	Web access log rules
31200–31299	Zeus web server rules
35000–35999	Squid rules
40100–40499	Attack pattern rules
40500–40599	Privilege escalation rules
40600–40999	Scan pattern rules
50100–50299	MySQL database rules
50500–50799	PostgreSQL database rules
100000–109999	User-defined rules

Every local rule should go in the *local_rules.xml* file located within the *rules/* directory of your OSSEC HIDS installation. This typically is found in */var/ossec/rules/local_rules.xml*.

TIP

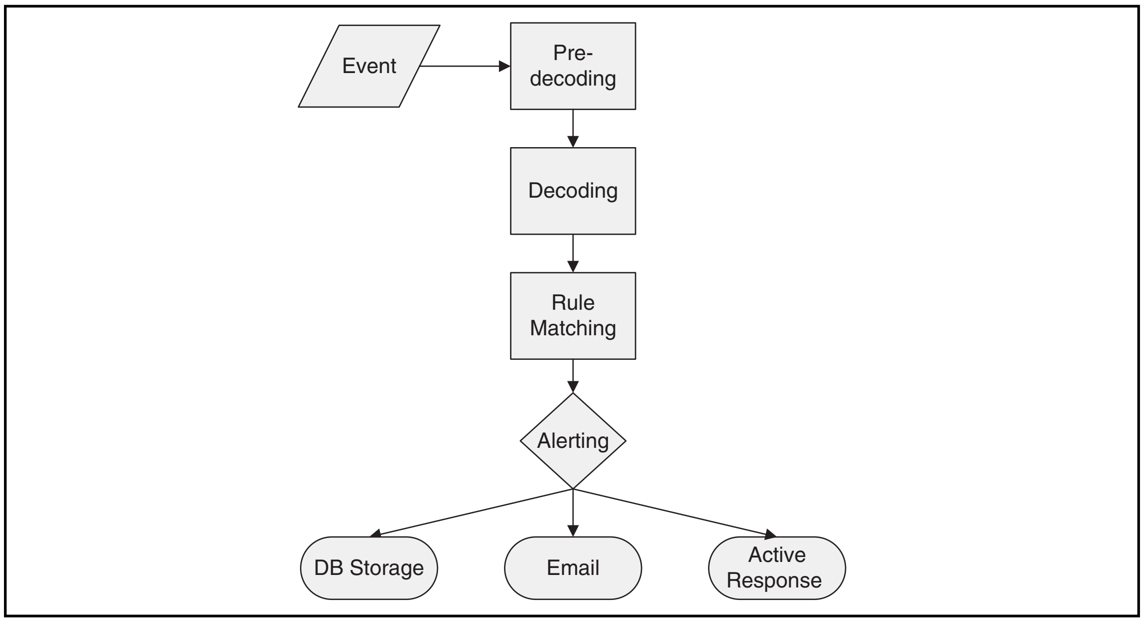
The other rules files should not be modified because they could impact how the core OSSEC HIDS rules function.

During the upgrade process, the scripts overwrite all rules files, except the *local_rules.xml* file. If you need to tweak or tune a specific rule that is shipped with the OSSEC HIDS, the *local_rules.xml* can be used to override how the standard rule functions. This is discussed later in this chapter.

Understanding the OSSEC HIDS Analysis Process

The first thing to understand is how the OSSEC HIDS handles every event received. Figure 4.1 shows this event flow.

Figure 4.1 Event Flow Diagram



As soon as an event is received, the OSSEC HIDS tries to decode and extract any relevant information from it. Decoding or normalization of the event is done in two parts, called predecoding and decoding. The fields used to decode these events are shown in Table 4.3.

Table 4.3 Decoded Fields

Field	Description
log	The message section of the event.
full_log	The entire event.
location	Where the log came from.
hostname	Hostname of the event source.
program_name	Program name. This is taken from the syslog header of the event.
srcip	The source IP address within the event.
dstip	The destination IP address within the event.
srcport	The source port within the event.

Continued

Table 4.3 Continued. Decoded Fields

Field	Description
dstport	The destination port within the event.
protocol	The protocol within the event.
action	The action taken within the event.
srcuser	The originating user within the event.
dstuser	The destination user within the event.
id	Any ID decoded as the ID from the event.
status	The decoded status within the event.
command	The command being called within the event.
url	The URL within the event.
data	Any additional data that you want to extract from the payload of the event.
systemname	The system name within the event.

After this data is extracted, the rule-matching engine is called to verify if an alert should be created.

Predecoding Events

The process of predecoding is very simple and is meant to extract only static information from well-known fields of an event. It generally is used with log messages that follow widely used protocols, like Syslog or the Apple System Log (ASL) formats. The information extracted during this phase is time, date, hostname, program name, and the log message.

The following log indicates that the syslog daemon has stopped. The predecoder can inspect this log and extract important information from it as seen in Table 4.4.

```
Apr 13 13:00:01 linux_server syslogd: stopped
```

Table 4.4 Predecoded Example (syslogd)

Field	Description
hostname	linux_server
program_name	syslogd
log	stopped
time/date	13:00:01, Apr 13

As you can see, only the static information from the log is extracted. Dealing with additional fields, like usernames or source IP addresses, are performed later in the *decoding* part of the process.

Another example of predecoding can be seen in the following log generated by the *sshd* daemon:

```
Apr 14 17:32:06 linux_server sshd[1025]: Accepted password for dcid from
192.168.2.180 port 1618 ssh2
```

Table 4.5 shows how this log would be predecoded.

Table 4.5 Predecoded Example (sshd)

Field	Description
hostname	linux_server
program_name	sshd
log	Accepted password for dcid from 192.168.2.180 port 1618 ssh2
time/date	Apr 14 17:32:06

The last example, a failed authentication log from an ASL-enabled system, is shown here:

```
[Time 2006.12.28 15:53:55 UTC] [Facility auth] [Sender sshd] [PID 483]
[Message error: PAM: Authentication failure for username from 192.168.0.2]
[Level 3] [UID -2] [GID -2] [Host mymac]
```

Table 4.6 shows how this log would be predecoded.

Table 4.6 Predecoded Example (ASL sshd)

Field	Description
hostname	mymac
program_name	sshd
log	error: PAM: Authentication failure for username from 192.168.0.2
time/date	Dec 28, 2006 15:53:55

When you compare a Syslog message and an ASL message, they look very different. Within the OSSEC HIDS, however, they would be normalized in a way that the same rule can be written to match both message formats. Another benefit is that you are able to write rules looking for each of these fields separately, like looking for any log from *hostname X* or any log with *program_name Y*.

Decoding Events

Decoding is the next step in the process, following predecoding. The goal of decoding is to extract nonstatic information from the events that we can use in our rules later in the process. Generally we want to extract IP address information, usernames, and similar data.

Let's revisit our previously discussed *sshd* message:

```
Apr 14 17:32:06 linux_server sshd[1025]: Accepted password for dcid from
192.168.2.180 port 1618 ssh2
```

The message is predecoded as previously explained in Table 4.5, but we could user our decoder to extract the fields as shown in Table 4.7 if the log message contained the information.

Table 4.7 Decoded Example (sshd)

Field	Description
srcip	192.168.2.180
user	dcid

The decoders, unlike the predecoders, are not static, and can vary from event to event. All decoders are user-configured in the *decoder.xml* file in the etc/ directory of your OSSEC HIDS installation. This typically is located in */var/ossec/etc/decoder.xml*. There are several *decoder* options available and these are explained in Table 4.8.

Table 4.8 Available Decoder Options

Field	Description
program_name	Executes the decoder if the *program_name* matches the syslog program name.
prematch	Executes the decoder if *prematch* matches any portion of the *log* field.

Continued

Table 4.8 Continued. Available Decoder Options

Field	Description
regex	Regular expression to specify where each field is.
offset	Attribute of regex. It can be *after_prematch* or *after_parent*. It essentially tells the regex where to start computing the expression.
order	Order within the regular expression. It can be all the fields in the normalized event (*srcip*, *user*, *dstip*, *dstport*, etc.)
parent	Parent decoder that must be matched for this decoder to be called.

To start, each decoder is delimited by a *<decoder></decoder>* tag, where the name of the decoder is specified. Within the *<decoder></decoder>* tag the fields explained in Table 4.8 can be used.

```
<decoder name="decoder-test"></decoder>
```

We will now walk through several decoder examples to help you fully understand the decoder process.

Decoder Example: sshd Message

In this example, we have a decoder to extract the Source IP address and username from a the following *sshd* log message:

```
Apr 14 17:32:06 linux_server sshd[1025]: Accepted password for dcid from
192.168.2.180 port 1618 ssh2
```

To extract the information we must define our decoder. The first step is to create the decoder, named sshd-test. The *name* attribute is mandatory so that your decoder can be properly identified.

```
<decoder name="sshd-test">
</decoder>
```

Within the decoder we need to define the <program_name></program_name> tag so that this decoder is called only if the program_name matches sshd in the syslog header.

```
<program_name>sshd</program_name>
```

To extract the user and port information from the message we must use the *<regex></regex>* tag. The *\S+* will match any sequence of consecutive nonwhite space characters (i.e., spaces and tabs).

The two capture groups (in parenthesis) will match the username and IP address part of the message, respectively.

```
<regex>^Accepted \S+ for (\S+) from (\S+) port </regex>
```

We must also specify the order to tell the OSSEC HIDS the field that we are parsing out of the message. In this case, the order is a username followed by a source IP address.

```
<order>user, srcip</order>
```

Our completed decoder is:

```
<decoder name="sshd-test">
  <program_name>sshd</program_name>
  <regex>^Accepted \S+ for (\S+) from (\S+) port </regex>
  <order>user, srcip</order>
</decoder>
```

Decoder Example: vsftpd Message

In addition to the program_name, we can use the <prematch> conditional tag to specify when the decoder should be called. It can be used with the program_name to have two conditionals or by itself on non-syslog messages (only syslog can have a program_name). The following is an example to match a vsftpd message.

The following log is a vsftpd login message that we want to decode:

```
Sun Jun 4 22:08:39 2006 [pid 21611] [dcid] OK LOGIN: Client "192.168.1.1"
```

To extract the information we must define our decoder. The first step is to create the decoder, named **vsftpd**. The *name* attribute is mandatory so that your decoder can be properly identified.

```
<decoder name="vsftpd">
</decoder>
```

Within the decoder we need to define the <*prematch*></*prematch*> tag so when the prematch matches what is in the log, the regex is going to be called to extract the user and source IP address.

In this example, the pattern will match if the line starts with (^) one or more word characters (\w+), followed by a space then one or more word characters (\w+), one or more white-space characters (\s+), one or more digits (\d+), and a space. Next, is one or more nonwhite-space characters (\S+), a space then the *pid* section where the *pid* has one or more digits (\d+).

```
<prematch>^\w+ \w+\s+\d+ \S+ \d+ [pid \d+] </prematch>
```

The *offset* option must be used within the <regex></regex> tag to make sure our regular expression is compared only after what was read from this prematch. This saves a lot of time when computing the regular expressions. In this example, the expression matches a line beginning with a word in braces followed by *OK LOGIN: Client* and then an IP address in dotted quad notation.

```
<regex offset="after_prematch">^[(\w+)] OK LOGIN: Client "(\d+.\d+.\d+.\d+)"$</regex>
```

We need to also specify the order to tell the OSSEC HIDS what each field is that we are parsing out of the message. In this case the order is a username followed by a source IP address.

```
<order>user, srcip</order>
```

Our completed decoder is:

```
<decoder name="vsftpd">
  <prematch>^\w+ \w+\s+\d+ \S+ \d+ [pid \d+] </prematch>
  <regex offset="after_prematch">^[(\w+)] OK LOGIN: Client
"(\d+.\d+.\d+.\d+)"$</regex>
  <order>user, srcip</order>
</decoder>
```

Tools & Traps…

Using Offsets

Offsets should be used whenever you can. They specify where in the string OSSEC should start evaluating the regular expression. For example, imagine you have a log:

```
Sun Jun 4 22:08:39 2007 test: Login USER dcid
```

and you write a simple <prematch> looking for the date and time format:

```
<prematch>\w+ \w+ \d+ \S+ \d\d\d\d </prematch>
```

When the <prematch> finishes evaluating, it will read everything up to "Sun Jun 4 22:08:39 2007 ", so there is no point in reading it all again. Because of that, in our regex, we use:

```
<regex offset="after_prematch">^test: Login USER (\w+)</regex>
```

so that it starts reading at "test: Login USER dcid" instead of the whole log.

If your decoder has a "parent" decoder (explained later), you can use "after_\parent" to make sure your decoder does not read everything that the parent already did.

Using the *<parent>* Option

The next step of *decoding* is to use the <*parent*></*parent*> option to create trees of *decoders*, where each extracts parts of the event. Imagine a case where you need to extract the username and source IP address from both a successful login event and a failed login event. Instead of checking each every time, a *parent decoder* can be written that looks for a specific *program_name* or part of the log. This allows you to create *subdecoders* to extract the data you need. Using the following sshd log messages:

```
Apr 14 17:32:06 linux_server sshd[1025]: Accepted password for dcid from
192.168.2.180 port 1618 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1799 ssh2
```

We could write two decoders, but the *prematches* and *program_name* would be checked multiple times, slowing OSSEC down. With a parent decoder we can look for *sshd* within the program name of the log messages:

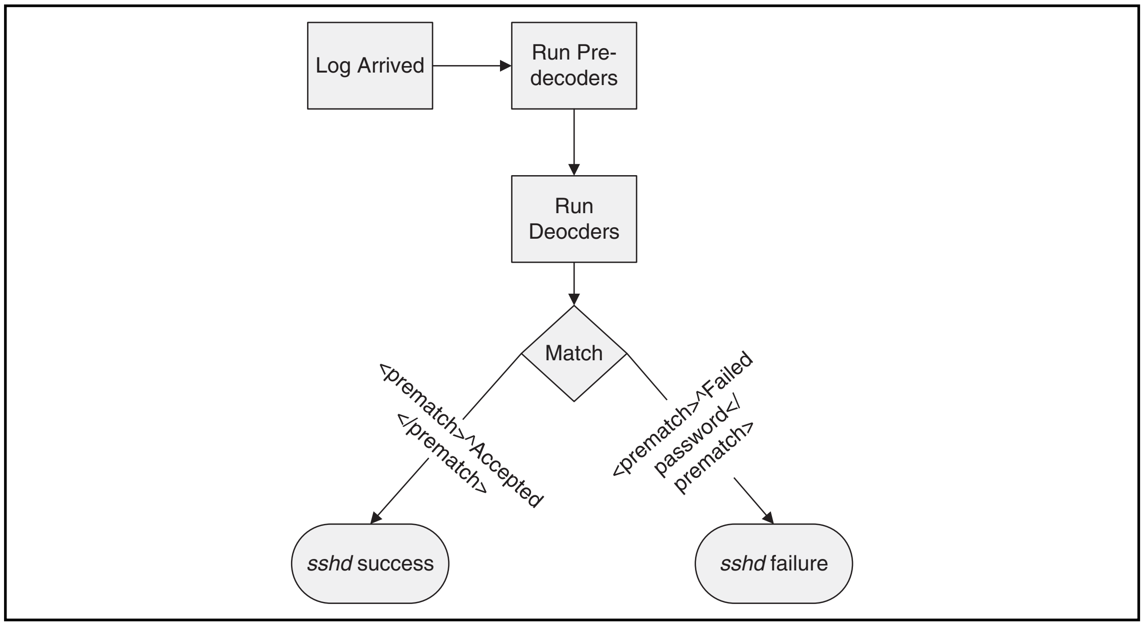
```
<decoder name="sshd">
  <program_name>^sshd</program_name>
</decoder>
```

Now that we have a parent decoder created we can create a separate decoder for each log message. One decoder can be created for the successful authentication and another for the failed authentication message.

```
<decoder name="sshd-success">
  <parent>sshd</parent>
  <prematch>^Accepted</prematch>
  <regex offset="after_prematch">^ password for (\S+) from (\S+) port </regex>
  <order>user, srcip</order>
</decoder>
<decoder name="ssh-failed">
  <parent>sshd</parent>
  <prematch>^Failed password </prematch>
  <regex offset="after_prematch">^for invalid user \S+ from (\S+) </regex>
  <order>srcip</order>
</decoder>
```

Note how on each one we have a different <*prematch*></*prematch*> value specified. If a message contains the string **Accepted** the regex for success is going to be extracted. Conversely if the message contains the string **Failed**, the regex for the failed decoder is called. Figure 4.2 shows how the event would flow, if subject to the preceding <*prematch*></*prematch*> configuration.

Figure 4.2 <prematch> Event Flow


Log Arrived
Run Pre-decoders
Run Deocders
Match
<prematch>^Accepted </prematch>
<prematch>^Failed password</prematch>
sshd success
sshd failure

Decoder Example: Cisco PIX Message

In this example we need to create a decoder for the following Cisco PIX message:

```
%PIX-2-106001: Inbound TCP connection denied from 165.139.46.7/3854 to
165.189.27.70/139 flags
```

Because we might have to write *decoders* for different PIX IDs at a later time we should first create a *parent decoder* for all PIX messages:

```
<decoder name="pix">
  <prematch>^%PIX-</prematch>
</decoder>
```

Any additional PIX decoders we create should reference this parent decoder for consistency. We should also use the *after_parent* option to make sure we are not reading the message more times then we should:

```
<decoder name="pix-fw1">
  <parent>pix</parent>
  <prematch offset="after_parent">^2-106001</prematch>
  <regex offset="after_parent">^(\S+): \w+ (\w+) \S+ (\S+) from </regex>
  <regex>(\S+)/(\S+) to (\S+)/(\S+)</regex>
  <order>id, protocol, action, srcip, srcport, dstip, dstport</order>
</decoder>
```

Our regular expression is a bit more complex, but it extracts everything from the event, including protocol, action, source and destination IP addresses, and ports.

Decoder Example: Cisco IOS ACL Message

In this example we need to extract the action, source IP address, source port, destination IP address, and destination port for use in Cisco IOS access control list (ACL) rules.

WARNING

The following example does not need to be created within the *decoder.xml* file since it is already present. The example is presented so that you might understand how to create your own advanced decoders.

The following log is a Cisco IOS ACL message that we want to decode:

```
%SEC-6-IPACCESSLOGP: list 102 denied tcp 10.0.6.56(3067) -> 172.36.4.7(139),
  1 packet
%SEC-6-IPACCESSLOGP: list 199 denied tcp 10.0.61.108(1477) -> 10.0.127.20(44
5), 1 packet
```

To extract the information we must define our decoder. The first step is to create the decoder, named **cisco-ios-acl**. The *name* attribute is mandatory so that your decoder can be properly identified.

```
<decoder name="cisco-ios-acl">
</decoder>
```

The Cisco IOS ACL decoder relies on the already created Cisco IOS decoder named *cisco-ios*. As such we must defined the <*parent*></*parent*> tag to reference the other decoder.

```
<parent>cisco-ios</parent>
```

Because this is an ACL rule we can identify this *type* of message as a *firewall* message using the <*type*></*type*> tag. The <*type*></*type*> tag can be syslog, firewall, ids, web-log, squid, windows, or ossec, and are defined in the *rules_config.xml* file.

```
<type>firewall</type>
```

Within the decoder we need to define the <*prematch*></*prematch*> tag so when the prematch matches what is in the log, the regex is going to be called to extract the user and source IP address.

```
<prematch>^%SEC-6-IPACCESSLOGP: </prematch>
```

The *offset* option must be used within the <regex></regex> tag to make sure our regular expression is compared only after what was read from the earlier prematch. This saves a lot of time when computing the regular expressions. The regular expression used matches any string beginning with *list*, followed by multiple nonwhite-space and two more space-separated words. The two words are captured in groups *denied* and *tcp*.

```
<regex offset="after_prematch">^list \S+ (\w+) (\w+) </regex>
```

Anther regular expression must be created to locate the source IP address (*srcip*), source port (*srcport*), destination IP address (*dstip*), and destination port (*dstport*) so another *<regex> </regex>* tag must be used.

```
<regex>(\S+)\((\d+)\) -> (\S+)\((\d+)\),</regex>
```

We also need to specify the order to tell the OSSEC HIDS what each field is that we are parsing out of the message. In this case the order is an action, protocol, source IP address, source port, destination IP address, and destination port.

```
<order>action, protocol, srcip, srcport, dstip, dstport</order>
```

Our completed decoder appears as follows:

```
<decoder name="cisco-ios-acl">
  <parent>cisco-ios</parent>
  <type>firewall</type>
  <prematch>^%SEC-6-IPACCESSLOGP: </prematch>
  <regex offset="after_prematch">^list \S+ (\w+) (\w+) </regex>
  <regex>(\S+)\((\d+)\) -> (\S+)\((\d+)\),</regex>
  <order>action, protocol, srcip, srcport, dstip, dstport</order>
</decoder>
```

Understanding Rules

The OSSEC HIDS evaluates the rules to see if the received event should be escalated to an alert or if it should be part of a composite alert (with multiple events). As we mentioned earlier, the rules are stored in XML files located within the *rules/* directory of your OSSEC HIDS installation directory. These typically are found in the */var/ossec/rules/* directory.

There are two types of OSSEC HIDS rules: atomic and composite. Atomic rules are based on single events, without any correlation. For example, if you see an authentication failure, you can alert on that unique event. Composite rules are based on multiple events. For example, if you want an alert after 10 authentication failures, from the same source IP address, then a composite rule would be required.

Atomic Rules

We will begin by explaining how atomic rules work and then expand on your knowledge of rules by working with the advanced composite rules.

Writing a Rule

Each rule, or grouping of rules, must be defined within a *<group></group>* element. Your attribute *name* must contain the rules you want to be a part of this group. In the following example we have indicated that our group contains the **syslog** and **sshd** rules.

```
<group name="syslog,sshd,">
</group>
```

NOTE

A trailing comma in the group name definition is required if additional rules will append groups into the rule.

A group can contain as many rules as you require. The rules are defined using the *<rule></rule>* element and must have at least two attributes, the **id** and the **level**. The **id** is a unique identifier for that signature and the **level** is the severity of the alert. In the following example, we have created two rules, each with a different rule id and level.

```
<group name="syslog,sshd,">
<rule id="100120" level="5">
  </rule>
  <rule id="100121" level="6">
  </rule>
</group>
```

NOTE

User-defined rules should range from 100,000 to 119,999. If you choose any other ID, it might collide with the official ones from the OSSEC HIDS project.

Tools & Traps…

OSSEC HIDS Severities

The OSSEC HIDS severities (levels) range from 0 to 15, with 0 being the lowest and 15 the highest. When the rules are written, they are stored using a hierarchical model, where the rules with higher severity are evaluated first. The only exception is the severity 0, which is evaluated before all other severities.

The severities can be changed on a case-by-case basis but the default OSSEC HIDS severities are defined as follows:

Level 0: Ignored, no action taken
Primarily used to avoid false positives. These rules are scanned before all the others and include events with no security relevance.

Level 2: System low priority notification
System notification or status messages that have no security relevance.

Level 3: Successful/authorized events
Successful login attempts, firewall allow events, etc.

Level 4: System low priority errors
Errors related to bad configurations or unused devices/applications. They have no security relevance and are usually caused by default installations or software testing.

Level 5: User-generated errors
Missed passwords, denied actions, etc. These messages typically have no security relevance.

Level 6: Low relevance attacks
Indicate a worm or a virus that provide no threat to the system such as a Windows worm attacking a Linux server. They also include frequently triggered IDS events and common error events.

Level 9: Error from invalid source
Include attempts to login as an unknown user or from an invalid source. The message might have security relevance especially if repeated. They also include errors regarding the *admin* or *root* account.

Level 10: Multiple user generated errors
Include multiple bad passwords, multiple failed logins, etc. They might indicate an attack, or it might be just that a user forgot his or her credentials.

Continued

Level 12: High-importance event
Include *error* or *warning* messages from the system, kernel, etc. They might indicate an attack against a specific application.

Level 13: Unusual error (high importance)
Common attack patterns such as a buffer overflow attempt, a larger than normal syslog message, or a larger than normal URL string.

Level 14: High importance security event.
Typically the result of the correlation of multiple attack rules and indicative of an attack.

Level 15: Attack successful
Very small chance of false positive. Immediate attention is necessary.

You can define additional subgroups within the parent group using the <*group*></*group*> tag. This subgroup can reference any of the predefined OSSEC HIDS groups listed in Table 4.9. A description is also required in the <*description*></*description*> tag so that you can differentiate between rules as you create them.

```
<group name="syslog,sshd,">
  <rule id="100120" level="5">
    <group>authentication_success</group>
    <description>SSHD testing authentication success</description>
  </rule>
  <rule id="100121" level="6">
    <description>SSHD rule testing 2</description>
  </rule>
</group>
```

Table 4.9 OSSEC HIDS Groups

Group Type	Group Name	Description
Reconnaissance	connection_attempt	Connection attempt
	web_scan	Web scan
	recon	Generic scan
Authentication Control	authentication_success	Success
	authentication_failed	Failure
	invalid_login	Invalid
	login_denied	Login denied
	authentication_failures	Multiple failures

Continued

Table 4.9 Continued. OSSEC HIDS Groups

Group Type	Group Name	Description
	adduser	User account added
	account_changed	User account changed or removed
Attack/Misuse	automatic_attack	Worm (nontargeted attack)
	exploit_attempt	Exploit pattern
	invalid_access	Invalid access
	spam	Spam
	multiple_spam	Multiple spam messages
	sql_injection	SQL injection
	attack	Generic attack
	rootcheck	Rootkit detection
	virus	Virus detected
Access Control	access_denied	Access denied
	access_allowed	Access allowed
	unknown_resource	Access to nonexistent resource
	firewall_drop	Firewall drop
	multiple_drops	Multiple firewall drops
	client_misconfig	Client misconfiguration
	client_error	Client error
Network Control	new_host	New host detected
	ip_spoof	Possible ARP spoofing
System Monitor	service_start	Service start
	service_availability	Service availability at risk
	system_error	System error
	system_shutdown	Shutdown
	logs_cleared	Logs cleared
	invalid_request	Invalid request
	promisc	Interface switched to promiscuous mode
	policy_changed	Policy changed
	config_changed	Configuration changed

Continued

Table 4.9 Continued. OSSEC HIDS Groups

Group Type	Group Name	Description
	syscheck	Integrity checking
	low_diskspace	Low disk space
	time_changed	Time changed
Policy Violation	login_time	Login time
	login_day	Login day

Another important tag is the *<decoded_as></decoded_as> tag.* This explicitly states that the rule will be evaluated only if the specified decoder decoded it. As an example, using the *<decoded_as></decoded_as>* tag, we can explicitly state that the rule will execute only if the event is decoded by the *sshd* decoder. As you can see, we also have created a *<description></description>* tag detailing that this rule will log every decoded *sshd* message.

```
<rule id="100123" level="5">
  <decoded_as>sshd</decoded_as>
  <description>Logging every decoded sshd message</description>
</rule>
```

The previous rule is not necessarily a very useful rule. We can, however, expand it a bit further by using the *<match></match>* tag to evaluate parts of the log. For example, if we are interested in capturing failed attempts to log into a server due to incorrect SSH passwords, a rule could be created to match for certain strings in the event. For example, take a look at the following Linux *sshd* failed password log:

```
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1799 ssh2
```

Using the *<match></match>* tag we can search through the log and use the *Failed password* part of the message as a key to detect all events of this type. As you can see in the following example we have also updated the *<description></description>* tag to describe our new rule.

```
<rule id="100124" level="5">
  <decoded_as>sshd</decoded_as>
  <match>^Failed password</match>
  <description>Failed SSHD password attempt</description>
</rule>
```

If you want to alert on the successful password attempts and log with a lower severity, we would need three additional rules. Since these rules are looking at the same data, we can create a rule tree for the *sshd* logs. The rule tree organizes our rules and increases the speed at which they are parsed.

For this to work properly, you must use the *<if_sid></if_sid>* tag. This tag adds a rule to the tree under the signature specified. In the following example we use the *<if_sid></if_sid>* tag to indicate that this rule is a child of the parent *100123* rule in the tree. We also use the *<match></match>* tag to search through the log and use the *Accepted password* part of the message as a key to detect all events of this type. The *<group></group>* tag is used to associate this rule with the internal OSSEC HIDS *authentication_success* group. Finally, a meaningful description is added to identify this rule using the *<description></description>* tag.

```
<rule id="100125" level="3">
  <if_sid>100123</if_sid>
  <match>^Accepted password</match>
  <group>authentication_success</group>
  <description>Successful SSHD password attempt</description>
</rule>
```

The following example shows our entire tree for this group. The first *Logging every decoded sshd message* rule (100123), our *Failed SSHD password attempt* rule (100124), and finally our *Successful SSHD password attempt* rule (100125) are shown. Please note that rules *100124* and *100125* have been modified to be children of the 100123 rule using the *<if_sid></if_sid>* tag and have been associated with the *authentication_failure* and *authentication_success* OSSEC HIDS groups, respectively, using the *<group></group>* tag.

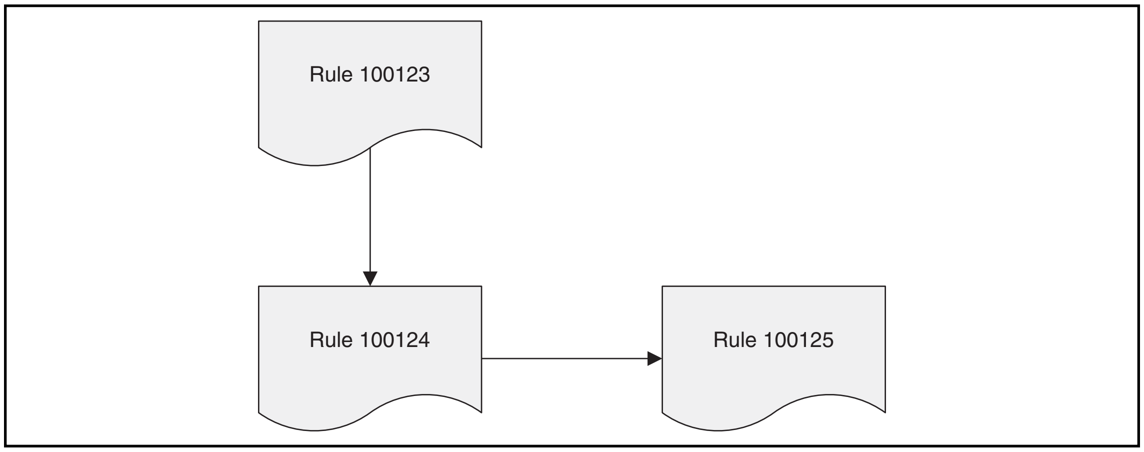
```
<group name="syslog,sshd,">
  <rule id="100123" level="2">
    <decoded_as>sshd</decoded_as>
    <description>Logging every decoded sshd message</description>
  </rule>

  <rule id="100124" level="7">
    <if_sid>100123</if_sid>
    <match>^Failed password</match>
    <group>authentication_failure</group>
    <description>Failed SSHD password attempt</description>
  </rule>

  <rule id="100125" level="3">
    <if_sid>100123</if_sid>
    <match>^Accepted password</match>
    <group>authentication_success</group>
    <description>Successful SSHD password attempt</description>
  </rule>
</group>
```

Figure 4.3 shows the rule hierarchy of the preceding group.

Figure 4.3 SSH Rule Hierarchy



If the following three logs were recorded by the OSSEC HIDS:

```
Apr 14 17:32:06 linux_server sshd[1025]: Accepted password for dcid from
192.168.2.180 port 1618 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user
lcid from 192.168.2.180 port 1799 ssh2
Apr 14 17:33:12 linux_server sshd[1231]: SSHD debug: key shared.
```

then the rules that we had just created would ensure that the events were recorded as alerts and written to the */var/ossec/logs/alerts.log* log as follows:

```
** Alert 1199140087.788: - syslog,sshd,authentication_success,
2007 Apr 14 17:32:06 linux_server->/var/log/auth.log
Rule: 100125 (level 3) -> '>Successful SSHD password attempt'
Src IP: 192.168.2.180
User: dcid
Apr 14 17:32:06 linux_server sshd[1025]: Accepted password for dcid from
192.168.2.180 port 1618 ssh2
** Alert 1199140089.788: - syslog,sshd,authentication_failure,
2007 Apr 14 17:33:09 linux_server->/var/log/auth.log
Rule: 100124 (level 7) -> 'Failed SSHD password attempt'
```

```
Src IP: 192.168.2.180
User: lcid
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1799 ssh2
```

Note

Because we did not create a rule to look for the *SSHD debug* log, an alert was not generated.

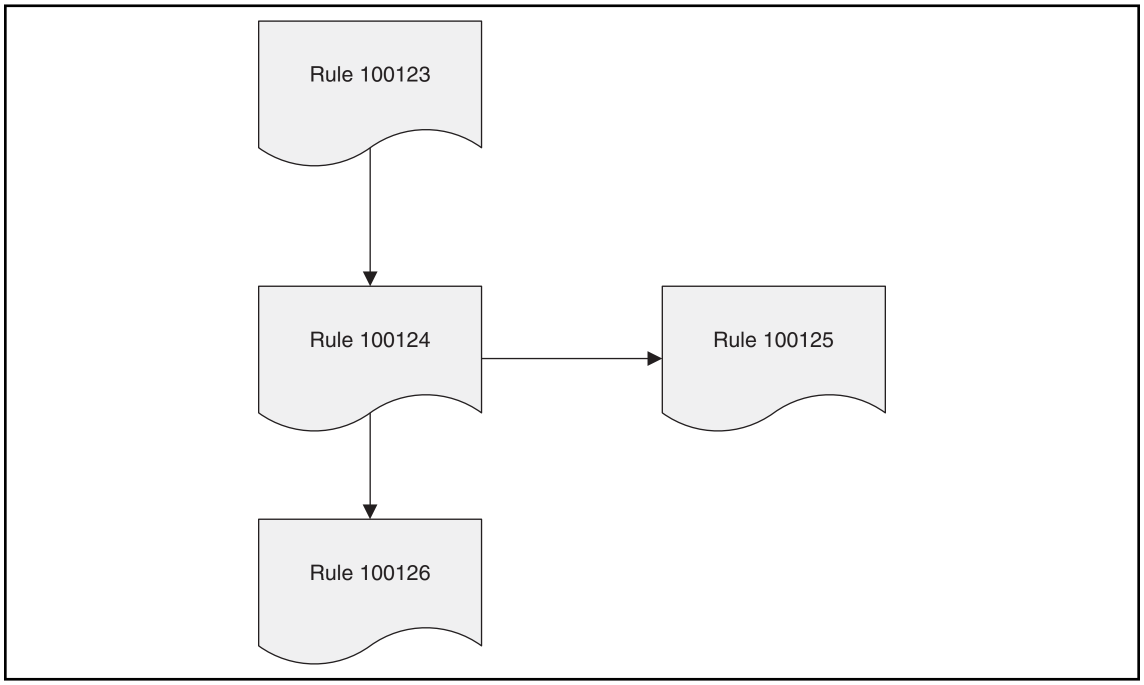
We can use the <*hostname*></*hostname*> and <*srcip*></*srcip*> tags to create more granular rules. Imagine that our company, *fakeinc.com,* has a very important server whose hostname is *main_sys*. Because this server contains highly sensitive customer information we want to record a high severity alert (12) for every authentication failure from outside of the corporate 192.168.2.0/24 network.

Our new rule uses the <*if_sid*></*if_sid*> tag to define the rule as a child of rule 100124. Because rule 100124 already looks for *Failed password* events, we do not need to parse them again. The new conditions that we want to use to increase the severity of the event are the hostname of the server being accessed and source IP address from which the users are attempting to authenticate. We can define these two conditions using the <*hostname*></*hostname*> and <*srcip*></*srcip*> tags, respectively.

```
<rule id="100126" level="12">
  <if_sid>100124</if_sid>
  <group>authentication_failure</group>
  <hostname>main_sys</hostname>
  <srcip>!192.168.2.0/24</srcip>
  <description>Severe SSHD password failure.</description>
</rule>
```

The new rule hierarchy is shown in Figure 4.4.

Figure 4.4 SSH Rule Hierarchy



The benefit of this hierarchical tree structure is that our parsing engine operates more efficiently. With more than 600 rules by default, only eight or nine rules are checked for every log instead of all 600. This structure is the main difference between the *linear* and *tree-based* analysis used by the OSSEC HIDS.

Using the <*time*></*time*> conditional atomic rule tag, we can alert on every successful login outside business hours and increase the severity of the alert. A full list of available atomic rule conditional options are shown in Table 4.10. We use the <*if_sid*>*100125*</*if_sid*> tag to make this rule depend on the 100125 rule. This rule will be checked only for *sshd* messages that already matched the successful login rule.

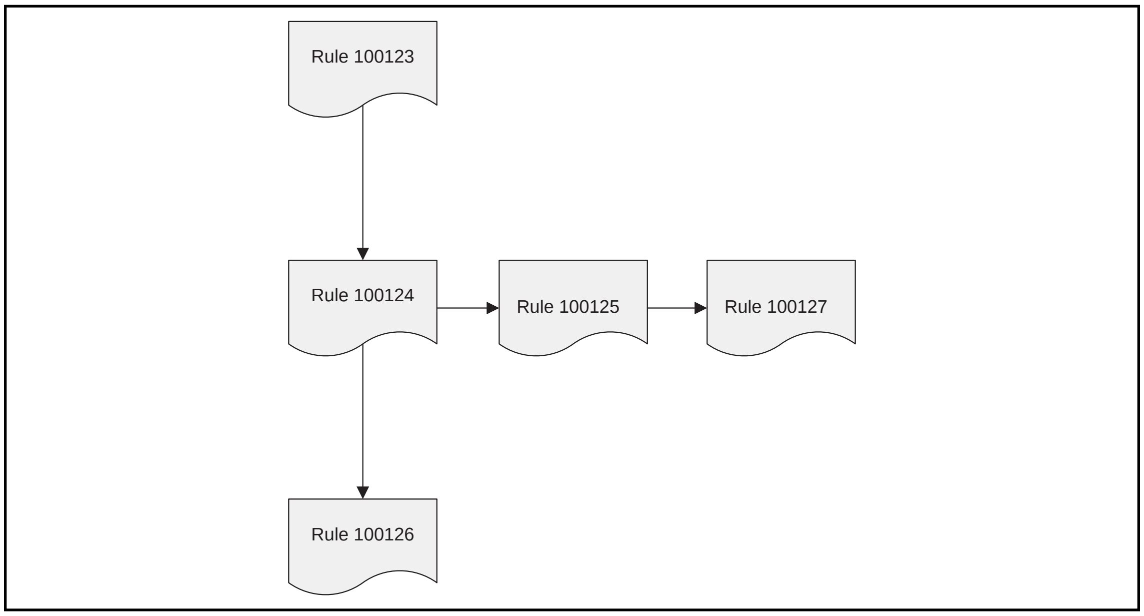
```
<rule id="100127" level="10">
  <if_sid>100125</if_sid>
  <time>6 pm - 8:30 am</time>
  <description>Login outside business hours.</description>
  <group>policy_violation</group>
</rule>
```

Table 4.10 Atomic Rule Conditional Options

Option	Value	Description
match	Any pattern	Any string to match against the event (log).
regex	Any regular expression	Any regular expression to match against the event(log).
decoded_as	Any string	Any prematched string. For more information see the *decoders* section of this chapter.
srcip	Any source IP address	Any IP address that is decoded as the source IP address. Use "!" to negate the IP address.
dstip	Any destination IP address	Any IP address that is decoded as the destination IP address. Use "!" to negate the IP address.
srcport	Any source port	Any source port (match format).
dstport	Any destination port	Any destination port (match format).
user	Any username	Any username that is decoded as a username.
program_name	Any program name	Any program name that is decoded from the syslog process name.
hostname	Any system hostname	Any hostname that is decoded as a syslog hostname.
time	Any time range in the format hh:mm - hh:mm or hh:mm am - hh:mm pm	The time range that the event must fall within for the rule to alert.
weekday	Any week day (sunday, monday, weekends, weekday, etc)	Day of the week that the event must fall on for the rule to alert.
id	Any ID	Any ID that is decoded from the event.
url	Any URL	Any URL that is decoded from the event.

The final rule hierarchy is shown in Figure 4.5.

Figure 4.5 SSH Rule Hierarchy with Policy Violation



Notes from the Underground…

After-Hours Attacks

When is the best time to attack a network? Is it during the day when the organization has the majority of their staff working and analyzing potential intrusions into their network? Perhaps it's better to wait for everyone to go home and attack outside of regular business hours.

Typically, an attacker would want to breach your network when there is a lesser chance of being detected. This provides them a certain level of anonymity to perform their covert action while your security staff is fast asleep. With no analysts in the office to track and action the associated alerts(provided alerts are generated) until the next morning, the attacker gains some room in which to work.

Using the *<time></time>* tag in your OSSEC HIDS rules allows you to increase the severity of alerts if they happen outside of regular operational hours. Take the following scenario as an example:

Abdalah works a typical Monday to Friday 9 A.M. to 5 P.M. job. Because Abdalah's job as a corporate accountant has him dealing with sensitive accounting records for the company, he is not permitted to bring his work home with him. He does, however, have a corporate laptop so that he can log into the company VPN and check e-mail and *perform other duties. All the corporate accounting information is stored on a central server that he, and other accountants, can collaboratively work on.*

Taking this into consideration, there should never be a reason for Abdul to log into the accounting server outside of regular business hours. With this information you could create your own rule to create a high severity alert if Abdalah logs into the server between 5:30 P.M. and 8:30 A.M. The following example shows one way of accomplishing this by using our already created *SSH* rules.

```
<group name="syslog,sshd,">
  <rule id="100123" level="2">
    <decoded_as>sshd</decoded_as>
    <description>Logging every decoded sshd message</description>
  </rule>

  <rule id="100124" level="7">
    <if_sid>100123</if_sid>
    <match>^Failed password</match>
    <group>authentication_failure</group>
    <description>Failed SSHD password attempt</description>
  </rule>

  <rule id="100125" level="3">
    <if_sid>100123</if_sid>
    <match>ÐAccepted password</match>
    <group>authentication_success</group>
    <description>Successful SSHD password attempt</description>
  </rule>

  <rule id="100130" level="12">
    <if_sid>100125</if_sid>
    <time>5:30 pm - 8:30 am</time>
    <description>Accounting access outside of regular business
hours.</description>
    <user>abdalahg035</user>
    <group>policy_violation</group>
    <hostname>accounting01</hostname>
  </rule>
</group>
```

As you can see, we have created a rule that relies on the successful SSH authentication rule. It will alert only if the event occurs between the hours of 5:30 P.M. and 8:30 A.M. and is a successful authentication by *abdalahg035* to the *accounting01* server. This rule could be made even more granular with additional atomic rule attributes as defined in Table 4.11.

Another way of writing the 100127 rule would be to include it under every rule in the *successful_login* group replacing the *<if_sid></if_sid>* tag with the *<if_group></if_group>* tag. Extending the rule in this way allows us to create an alert if **any** logins, reported from any device or agent, occur outside of regular business hours. A full listing of the available tree-based options is shown in Table 4.12.

```
<rule id="100127" level="10">
  <if_group>successful_login</if_group>
  <time>6 pm - 8:30 am</time>
  <description>Login outside business hours.</description>
  <group>policy_violation</group>
</rule>
```

NOTE

Only one alert is generated per event. So, for our rule looking for logins outside business hours, only the 100127 rule is going alert if the time is between 6 P.M. and 8:30 A.M. The same applies for rule 100126.

Table 4.11 Atomic Rule Values

Option	Value	Description
level	Any number (0 to 15)	Specifies the severity level of the rule.
id	Any number (100 to 999999)	Specifies the unique ID of the rule.
maxsize	Any number (1 to 99999)	Specifies the maximum size of the log event.

Table 4.12 Tree-Based Options

Option	Value	Description
if_sid	Any rule ID	Adds this rule as a child rule of the rules that match the specified signature ID.
if_group	Any group ID	Adds this rule as a child rule of the rules that match the specified group.

Continued

Table 4.12 Continued. Tree-Based Options

Option	Value	Description
if_level	Any level	Adds this rule as a child rule of the rules that match the specified severity level.
description	Any string	A description of the rule.
info	Any string	Extra information about the rule.
cve	Any CVE number	Any Common Vulnerabilities and Exposures (CVE) number that you would like associated with the rule.
options	alert_by_email no_email_alert no_log	Additional rule options to indicate if the alert should generate an e-mail, *alert_by_email*, should not generate an email, *no_email_alert*, or should not log anything at all, *no_log*.

Composite Rules

So far we have covered the options for single events. If we want to correlate multiple events, there are a few other options that we need to understand with composite rules. Composite rules are supposed to match the current event with those already received by the OSSEC HIDS. To have our rule maintain state, our rule needs two additional options. The *frequency* option specifies how many times an event/pattern must occur before the rule generates an alert. The *timeframe* option tells the OSSEC HIDS how far back, in seconds, it should look for the events. All composite rules have the following structure:

```
<rule id="100130" level="10" frequency="x" timeframe="y">
</rule>
```

For example, you could create a composite rule that creates a higher severity alert after five failed passwords within a period of 10 minutes. Using the *<if_matched_sid></if_matched_sid>* tag you can indicate which rule needs to be seen within the desired frequency and timeframe for your new rule to create an alert. In the following example, we have used the *level* option to set the severity level to **10**. The *frequency* option is configured to alert when five of the events are seen by the OSSEC HIDS and the *timeframe* option is used to specify our time window as **600** seconds (10 minutes).

The *<if_match_sid></if_match_sid>* tag is used to define which rule we want our composite rule to watch. In this case, we want to look for single failed password alerts, which is OSSEC HIDS rule **100124**. A meaningful description is also added using the *<description></description>* tag.

```
<rule id="100130" level="10" frequency="5" timeframe="600">
  <if_matched_sid>100124</if_matched_sid>
  <description>5 Failed passwords within 10 minutes</description>
</rule>
```

If we send the following logs to our OSSEC HIDS server:

```
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1799 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1729 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1749 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1729 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1819 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1909 ssh2
```

we would see the following alert generated by our composite rule:

```
# tail /var/ossec/logs/alerts/alerts.log
** Alert 1199697806.0: - syslog, sshd,authentication_failures,
2008 Jan 01 05:23:26 linux_server->/var/log/auth.log
Rule: 100130 (level 10) -> '5 Failed passwords within 10 minutes.'
Src IP: 192.168.2.180
User: lcid
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user lcid
from 192.168.2.180 port 1799 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user
lcid from 192.168.2.180 port 1729 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user
lcid from 192.168.2.180 port 1749 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user
lcid from 192.168.2.180 port 1729 ssh2
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for invalid user
lcid from 192.168.2.180 port 1819 ssh2
```

NOTE

The logs that triggered the alert are shown within the composite alert.

There are several additional tags that we can use to create more granular composite rules. These rules, as shown in Table 4.13, allow you to specify that certain parts of the event must be the same. This allows us to tune our composite rules and reduce false positives.

Table 4.13 Composite Rule Tags

Tag	Description
same_source_ip	Specifies that the *source IP address* must be the same.
same_dest_ip	Specifies that the *destination IP address* must be the same.
same_src_port	Specifies that the *source port* must be the same.
same_dst_port	Specifies that the *destination port* must be the same.
same_location	Specifies that the location (*hostname* or *agent name*) must be the same.
same_user	Specifies that the decoded *username* must be the same.
same_id	Specifies that the decoded *id* must be the same.

Looking back at our composite rule to alert on multiple failed passwords, we can improve it to restrict it to the same source IP address using the *<same_source_ip />* tag. This reduces false positives across our network environment. If you have deployed the OSSEC HIDS on a large network, often you will see multiple failed passwords during the course of a normal day. Restricting the composite rule to a single source IP address ensures that your rule will alert only if the same user unsuccessfully tries to authenticate five times in a row.

```
<rule id="100130" level="10" frequency="5" timeframe="600">
  <if_matched_sid>100124</if_matched_sid>
  <same_source_ip />
  <description>5 Failed passwords within 10 minutes</description>
</rule>
```

If you wanted your composite rule to alert on every authentication failure, instead of a specific OSSEC HIDS rule ID, you could replace the *<if_matched_sid></if_matched_sid>* tag with the *<if_matched_group></if_matched_group>* tag. This allows you to specify a category, such as *authentication_failure*, to search for authentication failures across your entire infrastructure.

```
<rule id="100130" level="10" frequency="5" timeframe="600">
  <if_matched_group>authentication_failure</if_matched_group>
  <same_source_ip />
  <description>5 Failed passwords within 10 minutes</description>
</rule>
```

In addition to *<if_matched_sid></if_matched_sid>* and *<if_matched_group></if_matched_group>* tags, we can also use the *<if_matched_regex></if_matched_regex>* tag to specify a regular expression to search through events as they are received.

```
<rule id="100130" level="10" frequency="5" timeframe="600">
<if_matched_regex>^Failed password</if_matched_regex>
<same_source_ip />
  <description>5 Failed passwords within 10 minutes</description>
</rule>
```

Working with Real World Examples

Because the OSSEC HIDS comes with more than 600 default rules, you might find that you do not need to create additional rules for your environment. You might, however, find that a certain rule doesn't fit your particular environment, so you must modify or extend the existing rule sets.

The first thing to note is that you should modify only the *local_rules.xml* file and not any of the other rules files. As stated earlier, if you modify these core OSSEC HIDS rules your changes will be lost when you upgrade. Also, please remember to only use OSSEC HIDS rule IDs 100,000 and above because these are reserved for user-defined rules.

Let's look at a few cases on how to use the rules.

Increasing the Severity Level of a Rule

Depending on your environment and alerting requirements you might decided that a particular OSSEC HIDS rule ID should have a higher severity level than the default. In the following example, it has been determined that invalid SSH logins are very important for your security team to monitor. You can change the severity of rule 5710, located within the *sshd_rules.xml* file by copying the rule definition into your *local_rules.xml* file.

Edit *the /var/ossec/rules/sshd_rules.xml* file and copy rule 5710:

```
# vi /var/ossec/rules/sshd_rules.xml
```

```
<rule id="5710" level="5">
  <if_sid>5700</if_sid>
  <match>illegal user|invalid user</match>
  <description>Attempt to login using a non-existent user</description>
  <group>invalid_login,</group>
</rule>
```

Paste rule 5710 to your *local_rules.xml* file and change the *level* tag from **5** to **10**. Also, add the *overwrite="yes"* option to the rule. Using the *overwrite* option instructs the OSSEC HIDS rule engine to use the local rule definition instead of the one found in the */var/ossec/rules/* directory.

```
# vi /var/ossec/rules/local_rules.xml
<rule id="5710" level="10" overwrite="yes">
  <if_sid>5700</if_sid>
  <match>illegal user|invalid user</match>
  <description>Attempt to login using a non-existent user</description>
  <group>invalid_login,</group>
  </rule>
```

Tuning Rule Frequency

Sometimes the frequency of the standard OSSEC HIDS rules is too small or large for your environment. To change the frequency of a rule, you can copy the standard rule to your *local_ rules.xml* file and use the same *overwrite* attribute. In this example, we will increase the frequency of the rule that looks for *sshd* brute force attempts (5712) from **6** events to **15** events.

Edit *the /var/ossec/rules/sshd_rules.xml* file and copy rule 5712:

```
# vi /var/ossec/rules/sshd_rules.xml
<rule id="5712" level="10" frequency="6" timeframe="120" ignore="60">
  <if_matched_sid>5710</if_matched_sid>
    <description>SSHD brute force trying to get access to </description>
    <description>the system.</description>
    <group>authentication_failures,</group>
  </rule>
```

Paste rule 5712 to your *local_rules.xml* file and change the *frequency* tag from **6** to **15**. Also, add the *overwrite="yes"* option to the rule.

```
# vi /var/ossec/rules/local_rules.xml
  <rule id="5712" level="10" frequency="15" timeframe="120" ignore="60"
overwrite="yes">
    <if_matched_sid>5710</if_matched_sid>
    <description>SSHD brute force trying to get access to </description>
    <description>the system.</description>
    <group>authentication_failures,</group>
  </rule>
```

Ignoring Rules

Sometimes a rule might be too noisy for your environment and might generate more alerts than required. You can ignore the rule completely or tune the existing rule to alert only if a particular patter is found within the event. For example, an OSSEC HIDS user recently sent the following alert to the OSSEC HIDS mailing list asking for help on how to tune it out:

```
OSSEC HIDS Notification.
2007 Nov 08 12:00:46
```

```
Received From: neo->/var/log/messages
Rule: 1002 fired (level 7) -> "Unknown problem somewhere in the system."
Portion of the log(s):

Nov 8 12:00:45 neo ntop[11016]: **WARNING** RRD:
rrd_update(/usr/local/var/ntop/rrd/interfaces/eth0/matrix/196.35.43.xxx/196.35.xx.
xxx/pkts.rrd) error: illegal attempt to update using time 1194516045 when last
update time is 1194516045 (minimum one second step)
```

The first action is to identify how you want to adjust the rule. One solution is to disable the rule completely by using the *<if_sid></if_sid>* tag and changing the severity to **0** (ignored). Whenever rule 1002 matches an event, rule 100301 will force the event to be ignored:

```
<rule id="100301" level="0">
  <if_sid>1002</if_sid>
  <description>Ignoring rule 1002.</description>
</rule>
```

However, completely disabling a rule is not always the best option. Using the previous example, it might be better to just ignore this specific *ntop* error that is causing the log. This ensures that other events continue to match against rule 1002. Using the *<program_name></program_name>* tag we can specify the application that is generating the event. Also, the *<match></match>* tag can be used to identify the common **illegal attempt to update** message within this log:

```
<rule id="100302" level="0">
  <if_sid>1002</if_sid>
  <program_name>ntop</program_name>
  <match>illegal attempt to update</match>
  <description>Ignoring rule 1002.</description>
</rule>
```

Ignoring IP Addresses

A common problem with security scanners is that they tend to generate an exceptional amount of alerts within the OSSEC HIDS. If you want to ignore a specific IP address completely, such as an internal vulnerability scanner, you can create a new rule to change the severity level to **0** for events with the source IP addresses specified using the *<srcip></srcip>* tags. You can also use the *<if_level></if_level>* tag, which tests all the alerts above the specified severity. In the following example, every alert with *level* if **4** or above, and a source IP address of 192.168.2.1 or 192.168.2.2, will be ignored.

```
<rule id="100303" level="0">
  <if_level>4</if_level>
  <srcip>192.168.2.1</srcip>
  <srcip>192.168.2.2</srcip>
  <description>Ignoring rule any level above 4 from ip X.</description>
</rule>
```

Correlating Multiple Snort Alerts

The OSSEC HIDS is widely used to monitor Snort IDS alerts. You can correlate multiple Snort events using an atomic and a composite rule. Let's say you want to see a higher severity alert if the Snort IDs 1:1002, 1:1003, and 1:1004 are seen from the same source IP address. The first thing you need to do is to create a local rule looking for them. Using the *<if_sid></if_sid>* tag you can specify that you want to use the OSSEC HIDS rule that parses IDS events. You can specify that the events must be decoded as Snort events by using the *<decoded_as></decoded_as>* tag. To look for specific Snort IDs you can use the *<id></id>* tag and the pipe (|) character to logically *OR* our IDs.

```
<rule id="100415" level="6">
  <if_sid>20101</if_sid>
  <decoded_as>snort</decoded_as>
  <id>1:1002|1:1003|1:1004</id>
  <description>Watched snort ids</description>
</rule>
```

With your atomic rule created, you must create a composite rule looking for **5** of these events within **180** seconds. Using the *frequency* option you can specify that **5** events are required to create an alert from this rule. The *timeframe* option allows you to specify the **180** second time window to search within. The *<if_matched_sid></if_matched_sid>* tag uses your previously created atomic rule and the *<same_source_ip />* tag specifies that you want to create the alert only if the events are seen from the same source IP address.

```
<rule id="100416" frequency="5" level="10" timeframe="180">
  <if_matched_sid>100415</if_matched_sid>
  <same_source_ip />
  <description>Multiple snort alerts with the watched ids</description>
</rule>
```

Ignoring Identity Change Events

Sometimes the amount of file changes from the *syscheck* daemon might be more than you require in your environment or perhaps you want to create some additional alerts from them by ignoring or correlating some events. Because the default syscheck alerts are under the *syscheck* group, our basic rule always contains the following:

```
<rule id="100501" level="7">
  <if_group>syscheck</if_group>
</rule>
```

As you can see from the following alert, the filename that changed is always contained in single quotes:

```
** Alert 1199273862.1538: mail - ossec,syscheck,
2008 Jan 02 07:37:42 copacabana->syscheck
Rule: 550 (level 7) -> 'Integrity checksum changed.'
Src IP: (none)
User: (none)
Integrity checksum changed for: '/usr/bin/groups'
Size changed from '1931' to '1934'
Old md5sum was: 'b37f687b322e9fe7b0ee50408dde8770'
New md5sum is : 'd8b83bcbdf9f4f0474f6ad04fb0683da'
Old sha1sum was: 'caff65849b5547e5bc2bb665b97a7c5e12e16e9f'
New sha1sum is : 'b7683926118cf253a51fc29a33e61987a3a85fd9'
```

If you want to create a rule to ignore any change on the groups binary, you have to create a new rule with the *level* option set to **0** to ignore the event. Using the *<match></match>* tag you can exclude the */usr/bin/groups* file by specifying the **for: '/usr/bin/groups'** match string from the event.

```
<rule id="100501" level="0">
  <if_group>syscheck</if_group>
  <match>for: '/usr/bin/groups'</match>
  <description>Ignoring /bin/groups change.</description>
</rule>
```

NOTE

You do not have to escape (\) the single quote characters ('') within the *<match></match>* tag.

If you want to ignore any change on the **C:\temp** directory of a Windows system, you could change your *<match></match>* tag to look for the **for: 'C:\temp** match string from the event.

```
<rule id="100501" level="0">
  <if_group>syscheck</if_group>
  <match>for: 'C:\temp</match>
  <description>Ignoring C:temp changes.</description>
</rule>
```

You can also go further and alert with a higher severity on any Windows executable file changes from your Windows systems by using the *<hostname></hostname>* tag and matching on them using the *<regex></regex>* tag to look for a string followed by the **.exe** file extension.

```
<rule id="100501" level="10">
  <if_group>syscheck</if_group>
  <regex>for: '\S+.exe'</regex>
  <hostname>winxp1|win20031</hostname>
  <description>Alerting on .exe changes from winxp1 and win20031.</description>
</rule>
```

TIP

Remember, you can logically *OR* multiple values within the rule tags using the pipe (|) character.

Writing Decoders/Rules for Custom Applications

The OSSEC HIDS comes with hundreds of default decoders and rules. Often, however, you will have custom applications and custom events to parse that do not have prebuilt rules and decoders. To solve that problem you must write the rules and decoders yourself.

Deciding What Information to Extract

The first thing that we need to do is to collect sample logs from your application. Remember, the more log samples you have, the more accurate your decoders and rules will be.

TIP

Some product vendors provide listings of all the possible events that their application can generate. It is strongly suggested that you consult your vendor documentation as part of your research phase.

In our example we have an application that is able to generate the following four events:

```
2007/28/Dec 17:45:10 Fakeinc: Failed password for user test, IP: 1.2.3.4 .
2007/28/Dec 17:45:21 Fakeinc: Accepted password for user test2, IP: 1.2.3.3 .
```

```
2007/28/Dec 17:45:35 Fakeinc: Application is shutting down: Internal error.
2007/28/Dec 17:45:47 Fakeinc: DEBUG: Received OK.
```

We need to categorize them, select the information we want to extract, and determine their severity before we start with the rules and decoders. Table 4.14 shows the information you might want to extract for each of the preceding log messages.

With the events categorized, we can start creating the decoders.

Table 4.14 Information to Extract

Log Message	Information to Extract
2007/28/Dec 17:45:10 Fakeinc: Failed password for user test, IP: 1.2.3.4.	Extract source IP (*srcip*) and user (*user*) information. Define as an *Authentication Failure (level 7) event.* Alert on multiple failed passwords *(level 10)*.
2007/28/Dec 17:45:21 Fakeinc: Accepted password for user test2, IP: 1.2.3.3.	Extract source IP (*srcip*) and user (*user*) information. Define as an *Authentication Success (level 3)* event.
2007/28/Dec 17:45:35 Fakeinc: Application is shutting down: Internal error.	Nothing to extract. Define as an *Internal Error -> Alert ASAP (level 10)*.
2007/28/Dec 17:45:47 Fakeinc: DEBUG: Received OK.	Nothing to extract. Define as a *Debug Message (level 0)*.

Creating the Decoders

Because the messages are well formatted, and we only need the source IP address and username from two of the messages, a single decoder can work for us.

We will start by creating a new decoder in the *decoder.xml* file called *fakeinc_custom*. We will also define a *<prematch></prematch>* tag looking for the date, time, and *fakeinc* tag in the log:

```
<decoder name="fakeinc_custom">
  <prematch>^\S+ \d\d:\d\d:\d\d Fakeinc: </prematch>
</decoder>
```

Once the *prematch* is established, we must create the regular expression to extract the IP address and user name. We also need to set the *offset* so that the part of the message that was already checked during the *prematch* is not read again by using the *after_prematch* option:

```
<regex offset="after_prematch">^\w+ password for user (\w+), IP: (\S+) </regex>
```

We also need to specify the order to tell the OSSEC HIDS what each field is that we are parsing out of the message. In this case the order is a username and a source IP address.

```
<order>user, srcip</order>
```

Our completed decoder is:

```
<!-- Final decoder for fakeinc custom application -->
<decoder name="fakeinc_custom">
  <prematch>^\S+ \d\d:\d\d:\d\d Fakeinc: </prematch>
  <regex offset="after_prematch">^\w+ password for user (\w+), IP:
(\S+) </regex>
  <order>user, srcip</order>
</decoder>
```

Now that we have our decoders created in the *decoder.xml* file we can proceed to the creation of the rules.

Creating the Rules

The first rule we must create is a parent rule that looks for every event that was decoded by the *fakeinc_custom* decoder. This saves a lot of processing power compared to having a custom rule for each event.

Using the *<rule></rule>* tag we can create a new rule in the user defined rule ID range.

```
<rule id="100102" level="0">
</rule>
```

Using the *<decoded_as></decoded_as>* tag we can specify the decoder name that we're looking for.

```
<decoded_as>fakeinc_custom</decoded_as>
```

We can also use the *<description></description>* tag to add a meaningful description to the rule.

```
<description>Parent rule for FakeInc custom</description>
```

Our completed rule is:

```
<rule id="100102" level="0">
  <decoded_as>fakeinc_custom</decoded_as>
  <description>Parent rule for FakeInc custom</description>
</rule>
```

Our next rules will all be dependent on the parent rule, *100102*, meaning that they will be called only if it matches. Using the *<if_sid></if_sid>* tag we can specify that this rule is a subrule of the parent rule specified.

We can use the following template for all subrules to ensure we reference the parent rule properly:

```
<rule id="10010x" level="y">
  <if_sid>100102</if_sid>
  <match>XX</match>
  <description>Fakeinc Custom: XX</description>
</rule>
```

Each subrule that we create must have a distinct rule ID. It is generally a good idea to use sequential subrule ID numbers to make it easier for someone else to understand your rule structure. We will create our first rule to look for failed login attempts by using the *<match></match>* tag. Within the tag we will look for the string **Failed**.

```
<rule id="100103" level="7">
  <if_sid>100102</if_sid>
  <match>^Failed</match>
  <description>Fakeinc Custom: Failed password</description>
</rule>
```

We will create our second rule to look for successful login attempts by using the *<match></match>* tag. Within the tag we will look for the string **Accepted**.

```
<rule id="100104" level="3">
  <if_sid>100102</if_sid>
  <match>^Accepted</match>
  <description>Fakeinc Custom: Accepted password</description>
</rule>
```

We will create our third rule to look for internal errors by using the *<match></match>* tag. Within the tag we will look for the string **Internal error**.

```
<rule id="100105" level="10">
  <if_sid>100102</if_sid>
  <match>Internal error</match>
  <description>Fakeinc Custom: Internal error</description>
</rule>
```

We do not need a rule for the *DEBUG* message, because we simply want to ignore it because it represents little value to us from a security or operations perspective. To finish, our last rule will be a composite one looking for multiple failed passwords:

```
<rule id="100106" level="10">
  <if_matched_sid>100103</if_matched_sid>
  <same_source_ip />
  <description>Fakeinc Custom: Multiple Failed passwords</description>
</rule>
```

Notice how we use *<if_matched_sid></if_matched_sid>* instead of *<if_sid>*</if_sid>. This is because this is a composite rule. A composite rule is a rule that looks for multiple events that match another existing rule. In the previous example we are looking for multiple events that match the 100103 rule ID. If you recall this rule was created to catch failed login attempts. We also need to make use of the *<same_source_ip />* tag to make sure that this rule will be triggered only if the events matched in rule ID 100103 are from the same source IP address.

Our completed rule is shown here:

```
<rule id="100102" level="0">
  <decoded_as>fakeinc_custom</decoded_as>
  <description>Parent rule for FakeInc custom</description>
</rule>
<rule id="100103" level="7">
  <if_sid>100102</if_sid>
  <match>^Failed</match>
  <description>Fakeinc Custom: Failed password</description>
</rule>
<rule id="100104" level="3">
  <if_sid>100102</if_sid>
  <match>^Accepted</match>
  <description>Fakeinc Custom: Accepted password</description>
</rule>
<rule id="100105" level="10">
  <if_sid>100102</if_sid>
  <match>Internal error</match>
  <description>Fakeinc Custom: Internal error</description>
</rule>
<rule id="100106" level="10">
  <if_matched_sid>100103</if_matched_sid>
  <same_source_ip />
  <description>Fakeinc Custom: Multiple Failed passwords</description>
</rule>
```

Monitoring the Log File

Now that we have our decoders and rules created we need to ensure that the application server, where the OSSESC HIDS agent is installed, is configured to monitor the applications log file. For example, if the log file is located in */var/log/fakeinc.log*, the following must be added to the agents *ossec.conf*:

```
<localfile>
  <location>/var/log/fakeinc.log</location>
  <log_format>syslog</log_format>
</localfile>
```

NOTE

Always restart the OSSEC HIDS after changing the *ossec.conf* configuration.

Summary

OSSEC HIDS rules are located within the *rules* directory of your OSSEC HIDS installation as an individual XML file. Each XML file is named similar to the type of event for which it checks. For example, all rules for the Cisco PIX firewall are located within the *pix_rules.xml* file. The OSSEC HIDS installation provides you with 43 rules for various applications and devices. Every rule has a unique ID that is assigned when it is first created. Each log type has a specific range of IDs assigned to ensure that OSSEC HIDS released decoders are not overwritten by mistake. The OSSEC HIDS team provides you with a dedicated range of IDs (100,000 to 119,999) to be used for user-created rules. During the upgrade process, the scripts overwrite all rules files, except the *local_rules.xml* file. If you need to tweak or tune a specific rule that is shipped with the OSSEC HIDS, the *local_rules.xml* can be used to override how the standard rule functions.

The OSSEC HIDS tries to decode and extract any relevant information from received events. The event is decoded in two parts, called predecoding and decoding. The process of predecoding is meant to extract only static information from well-known fields of an event. Information such as time, dates, hostnames, program names, and the log message is extracted during predecoding. The goal of decoding is to extract nonstatic information from the events that we can use in our rules later in the process. IP address information, usernames, URLs, and port information are some of the common fields that can be decoded from the event. All decoders are user configured in the decoder.xml file in the etc directory of your OSSEC HIDS installation.

After *predecoded* and *decoded* information is extracted, the rule-matching engine is called to verify if an alert should be created. The OSSEC HIDS evaluates the rules to see if the received event should be escalated to an alert or if it should be a part of a composite alert (with multiple events). There are two types of OSSEC HIDS rules; Atomic and Composite. Atomic rules are based on single events, without any correlation. For example, if you see an authentication failure, you can alert on that unique event. Composite rules are based on multiple events.

Because the OSSEC HIDS comes with more than 600 default rules, you might find that you do not need to create additional rules for your environment. You might, however, find that a certain rule doesn't fit your particular environment, so you must modify or extend the existing rule sets. You should modify the *local_rules.xml* file only to adjust how the default rules operate, because changes to the OSSEC HIDS supplied rules will be overwritten on upgrades. It is quite easy to increase the severity, change the frequency, and so on of existing rules by creating your own rules in the *local_rules.xml* file. You can also ignore specific IP addresses, rules, users, hosts, and such by tuning the rules for your environment.

The OSSEC HIDS comes with hundreds of default decoders and rules, but they might not be able to process the logs from your custom application or device. To create custom decoders and rules for your application you must first research the types of logs that your application or device can create. Often, product vendors provide listings of all the possible events that their application can generate. It is strongly suggested that you consult your vendor documentation as part of your research phase.

Solutions Fast Track

Introducing Rules

☑ Every rule is stored, in XML format, within the *rules* directory of your OSSEC HIDS installation.

☑ Rules contain decoders designed to extract data from the raw events, which allows the OSSEC HIDS to correlate disparate events from multiple sources.

☑ Every local rule should go in the *local_rules.xml* file located within the *rules* directory of your OSSEC HIDS installation.

Understanding the OSSEC HIDS Analysis Process

☑ As soon as an event is received, the OSSEC HIDS will try to decode and extract any relevant information from it.

☑ Decoding of the event is performed in two parts, called predecoding and decoding.

☑ After predecoded and decoded information is extracted, the rule-matching engine is called to verify if an alert should be created.

Predecoding Events

☑ The process of predecoding is meant to extract only static information from well-known fields of an event.

☑ Log messages that follow widely used protocols, like Syslog or the Apple System Log (ASL) formats, are processed during the predecoding phase.

☑ The information extracted during this phase is *time*, *dates*, *hostnames*, *program names*, and the *log message*.

Decoding Events

☑ The goal of decoding is to extract nonstatic information from the events that we can use in our rules later in the process.

- ☑ IP address information, usernames, URLs, and port information are some of the common fields that can be *decoded* from the event.
- ☑ All decoders are user configured in the *decoder.xml* file in the *etc* directory of your OSSEC HIDS installation.

Understanding Rules

- ☑ There are two types of OSSEC HIDS rules: Atomic, which are based on single events without any correlation; and Composite, which are based on multiple events.
- ☑ Each rule, or grouping of rules, must be defined within a <group></group> element.
- ☑ Composite rules provide you with tags to specify that certain parts of the received event must be the same.

Working with Real World Examples

- ☑ Certain rules might not fit your environment so you can modify or extend the existing rule sets to better suit your needs.
- ☑ You should modify the *local_rules.xml* file only to adjust how the default rules operate because changes to the OSSEC HIDS supplied rules will be overwritten on upgrades.
- ☑ Any user rule created to replace a preexisting OSSEC HIDS rule must contain the *overwrite="yes"* option within the rule.

Writing Decoders/Rules for Custom Applications

- ☑ The OSSEC HIDS comes with hundreds of default decoders and rules, but they might not be able to process the logs from your custom application or device.
- ☑ In order to create custom decoders and rules for your application you must first research the types of logs that your application or device can create.
- ☑ Some product vendors provide listings of all the possible events that their application can generate. It is strongly suggested that you consult your vendor documentation as part of your research phase.

Frequently Asked Questions

Q: Where are my rules stored?

A: All the rules are stored in the *rules* directory where you installed the OSSEC HIDS. This typically is located at */var/ossec/rules/*. Each rule is defined in a separate XML file and is named accordingly.

Q: Where should I put the rules that I create?

A: Any user-created rule should go into the *local_rules.xml* file within the *rules* directory of your OSSEC HIDS installation. This ensures that the standard package of rules functions as they are intended.

Q: When I upgrade my OSSEC HIDS will my local rules be overwritten?

A: During the upgrade process, the scripts overwrite all rules files, except the *local_rules.xml* file. If you need to tweak or tune a specific rule that is shipped with the OSSEC HIDS, the *local_rules.xml* can be used to override how the standard rule functions.

Q: How does an event flow through the OSSEC HIDS?

A: An event, received by the OSSEC HIDS, is subject to several processes. The first is the predecoding process, which extracts static information from the event. This is followed by the decoding process, which extracts nonstatic information from the event. The decoded event is then matched against the OSSEC HIDS rules and, if the rule determines an alert should be generated, an alert in the form of an e-mail message, active response, or log is created.

Q: What are the decoded fields that I can match against?

A: Fields such as *location*, *hostname*, *program_name*, *srcip*, *dstip*, *srcport*, *dstport*, *protocol*, for example, can all be decoded from the event.

Q: What information can I extract in the predecoding phase?

A: The information extracted during this phase is *time*, *dates*, *hostnames*, *program names*, and the *log message*.

Q: Can I decode usernames as well?

A: You can, but not during the predecoding phase. The decoding of user names is performed during the decoding phase.

Q: How can the OSSEC HIDS take logs from different log types and predecode them the same way?

A: Within the OSSEC HIDS every event is normalized in a way that the same rule can be written to match multiple message formats.

Q: What is a nonstatic field?

A: A nonstatic field is a field that is common across most logs but is not necessarily located in the same place in the event. For example, a successful authentication event from a Linux server would not have the *user* field in the same place in the message that a Nortel VPN Gateway device would.

Q: Where do I configure my decoders?

A: All decoders are user-configured in the *decoder.xml* file in the *etc* directory of your OSSEC HIDS installation. This typically is located in */var/ossec/etc/decoder.xml*.

Q: What tag do I use to identify a decoder in the *decoder.xml* file?

A: Each decoder is delimited by a *<decoder></decoder>* tag, where the name of the decoder is specified.

Q: What is the difference between Atomic and Composite rules?

A: There are two types of OSSEC HIDS rules: Atomic, which are based on single events without any correlation; and Composite, which are based on multiple events.

Q: What rule IDs should I use for my rules?

A: User-defined rules should range from 100,000 to 119,999. If you choose any other ID, it might collide with the official ones from the OSSEC HIDS project.

Q: How many rules can I use within a group?

A: There is no limit to the number of rules that you can include within a group. Keep in mind, however, that the purpose of the group is to combine *like* rules to make processing of events more efficient.

Q: How can I increase the severity of a preexisting OSSEC HIDS rule?

A: The recommended way to increase the severity of a preexisting OSSEC HIDS rule is to create an identical rule in your *local_rules.xml* file and change the *level* value to the required severity level.

Q: I have a couple of really noisy servers that I don't want the OSSEC HIDS to create alerts for. How can I hide these systems from my OSSEC HIDS?

A: You can ignore servers in your environment by creating a rule in your *local_rules.xml* file with a *level* of **0**. This tells the OSSEC HIDS to ignore any event that matches the contents of the rule. Within the rule you can use the *<srcip></srcip>* tag to list the source IP addresses to ignore.

Q: I don't control my organization's Snort IDS sensor so I cannot tune out individual Snort IDs. Can the OSSEC HIDS ignore specific Snort IDs if I don't want them to create alerts?

A: You can create a rule in your *local_rules.xml* file to ignore Snort IDs using the *<if_sid></if_sid>* tag to select the rule that parses Snort IDS events, the *<decoded_as></decoded_as>* tag to check logs decoded as Snort events, and the *<id></id>* tag to specify as many Snort IDs as you would like to ignore.

Q: Why doesn't the OSSEC HIDS have decoders or rulesfor "Application [XYZ]"?

A: The OSSEC HIDS comes with hundreds of default decoders and rules but they might not be able to process the logs from your custom application or device. Because the OSSEC HIDS team cannot possibly create rules for every device or application we do provide you with the ability to create your own decoders and rules so that you can process the events.

Q: How do I know what information to extract from my application or device logs?

A: This information is entirely up to the capabilities of your application or device and your needs. The logs that your application or device can generate will dictate what kind of information you can bring into the OSSEC HIDS. Once you know the format and possible event types you can decide how much, or how little, information from the events you need.

Q: I have my decoders and rules created for my applications log but I'm not seeing any events from my OSSEC HIDS agent. What have I missed?

A: You must ensure that you have configured your OSSEC HIDS agent to monitor the log file for which you have created the decoders and rules. Also remember to restart your OSSEC HIDS after making changes to the *ossec.conf* file on the agent.