

The OSSEC Host Intrusion Detection System

1 Overview

This exercise provides hands-on experience with the OSSEC host-based intrusion detection system (IDS). This IDS is commonly used and serves as the core of commercial IDS products¹. Like most IDS products, it applies a set of rules to identify attacks on computers. And as with many host-based IDS systems, OSSEC relies to a large extent on logs messages captured by the underlying operating system.

The lab includes the following objectives:

- Configure OSSEC agents for client computers, i.e., those whose activity will be monitored by the OSSEC server.
- Generate log-based events and observe resulting alerts generated by OSSEC.
- Observe the effects of “active responses” to system events, e.g., disabling traffic from an offending source.
- Configure a client agent to alert on changes to the ports that the computer is listening to.
- Define a rule to alert on web server access to a particular URL.
- Explore limitations and complications associated rule-based IDS.
- Consider **system** security attack surface trade-offs related to introducing 5MB of privileged code, some of which consumes whatever an attacker feeds your computers.

1.1 Background

This lab assumes the student has some introduction to IDS systems and some familiarity with Unix logging.

2 Lab Environment

This lab runs in the Labtainer framework, available at <http://nps.edu/web/c3o/labtainers>. That site includes links to a pre-built virtual machine that has Labtainers installed, however Labtainers can be run on any Linux host that supports Docker containers.

From your labtainer-student directory start the lab using:

```
labtainer ossec
```

A link to this lab manual will be displayed.

¹<https://wazuh.com/>

3 Lab topology

In addition to an OSSEC server, this lab includes a workstation and a web server.

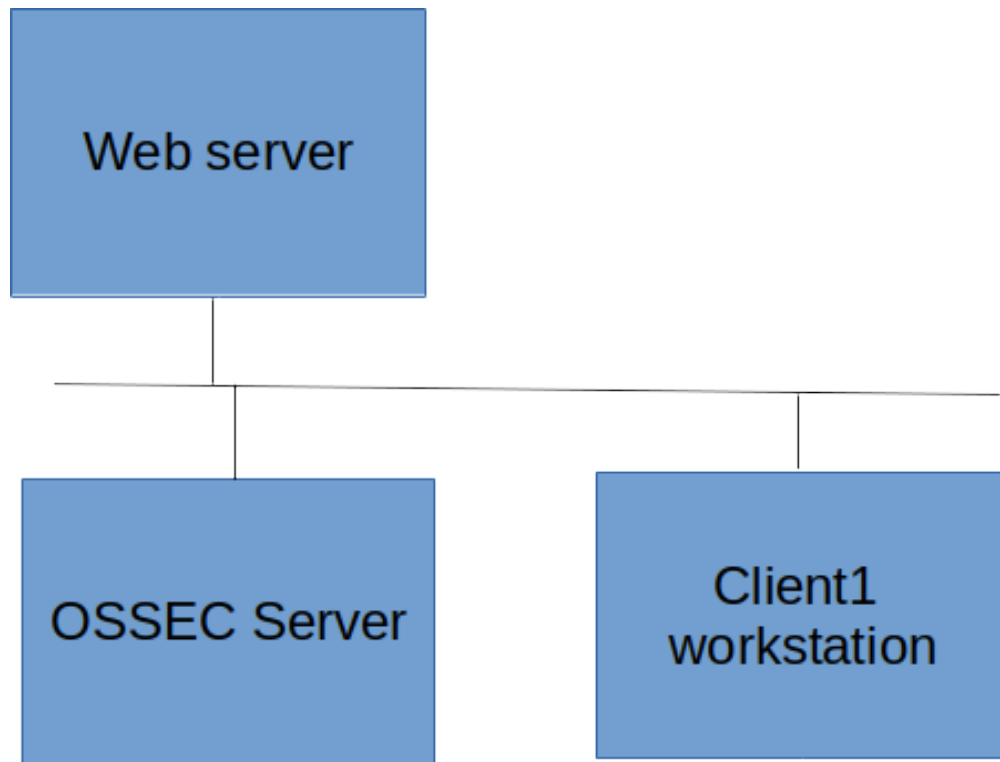


Figure 1: OSSEC Lab Topology

4 OSSEC Operation

Details on OSSEC can be found at <https://www.ossec.net/docs/index.html>. The OSSEC server receives log entries from monitored computers via OSSEC agents that run on each monitored computer. A computer will not be monitored unless it has an agent installed and configured to communicate with the OSSEC server. This communication requires:

- The client agent is registered on the OSSEC server and a cryptographic key is generated for the client. Both of these steps occur using the `manage_agents` command on the server.
- The key is imported into the agent on the client, using the `manage_agents` command on the client.
- The IP address of the server is defined in the client `/var/ossec/etc/ossec.conf` file.
- The `ossec` service on the client and the server are each restarted.

Once those steps are complete, the server will begin to monitor the client based on log entries sent from the client to the server. The set of client logs to be monitored are defined in the `ossec.conf` file on the client. That file has a broad initial set of log files defined, though some of the log names may require modification as we will see in this lab.

What the server does with received log messages is primarily defined in a set of *rules* located in the server `/var/ossec/rules` directory. Server actions include generating alerts and causing active responses, e.g., directing a client to temporarily disable network traffic from an offending source.

Log messages are parsed and categorized based on *decoding* rules defined in

```
/var/ossec/etc/decoder.xml
```

OSSEC includes decoders for most common log formats. The decoders assign identifiers to different types of log messages, and these identifiers may then be named in rules. For example, a decoder may assign selected messages generated by a web server as being of type `web-accesslog`. A rule might then define an alert to be generated if it finds a message of type `web-accesslog` to contain a character string indicative of an SQL injection attack.

5 Tasks

5.1 Configure OSSEC to monitor the client1 workstation

The configuration files for each agent in the lab have been preconfigured to identify the server IP address. So you only need get keys into each agent to get them talking with the server. Most OSSEC operations require use of `sudo`, so you might as well just “`sudo su`”.

- On the server, run the `/var/ossec/bin/manage_agents` command to define an agent for the `client1` computer and to export a key for that client. (Just run the command, you’ll figure it out.)
- Copy the key that was generated on the server.
- On the `client1` computer, run the `manage_agents` command and import the key by pasting it when prompted.
- Use `systemctl restart ossec` on the client and the server.

5.2 Cause and observe alerts

Out of the box, OSSEC monitors many different security relevant events, some of which get reported as *alerts* in the `/var/ossec/logs/alerts/alerts.log` file in the server. You’ll be looking at that file a bit, so tail it in a new terminal:

- At the Labtainers terminal (labtainer-student), create a new terminal for the OSS computer:

```
moreterm.py ossec ossec
```

- On the new terminal:

```
sudo su
tail -f /var/ossec/logs/alerts/alert.log
```

Once you are monitoring alerts, create one. Go to the `client1` computer, which may still be in a `sudo` shell, i.e., with the root `#` prompt. Type `exit` at that prompt, or `sudo su` if not yet in a `sudo` shell. Switch back and forth from a `sudo su` shell. Note the alerts in the OSSEC `alerts.log`. Those of you with Unix experience may recognize the alerts as being little more than system log messages, which they are.

5.3 Add the Web server and test log monitoring

Use the `manage_agents` command on the server to add the web server agent. Then use the `manage_agent` command on the web server to import the generated key.² Restart the `ossec` service on the OSSEC server and on the web server, noting that the service name on the web server is `ossec-hids`.

You should have seen alerts in the alert log for each of these actions. Now go to the `client1` computer and `ssh` to the web server, providing a bogus password. Note the alerts. Again, these are triggered by standard Unix log messages, in this case generated by the SSH daemon.

5.4 Active responses

Repeat your failed attempt to `ssh` from `client1` to the web server. Use `ctl-C` to break out the first failed password attempt to speed things up (you are looking to run the `ssh` command multiple times). And keep repeating it until the `ssh` command just hangs and you see an alert such as:

```
** Alert 1619194381.6291: mail - syslog,sshd,authentication_failures,
2021 Apr 23 16:13:01 (webserver) 172.0.0.4->/var/log/secure
Rule: 5720 (level 10) -> 'Multiple SSHD authentication failures.'
Src IP: 172.0.0.3
```

The rule defining this alert assigns its importance as “level 10”, and any level over 6 will trigger an active response. You can see the active response definitions within the `ossec.conf` file on the server. The response is to alter the `iptables` on the web server to block all traffic from `client1`. You can observe this using `iptables -L` on the web server, assuming you are quick enough³. The network blocking is set for 10 minutes by default, but we’ve changed that to 1 minute for this lab.

5.5 Monitor changes to command output

So far we’ve looked at OSSEC monitoring of log file entries. The IDS also lets you monitor the output of defined commands. In this section, you will configure OSSEC to generate alerts if there are any changes to the network ports listened to by the web server. The first step is to tell OSSEC on the web server about the command to monitor. In this example, you will monitor output of the `netstat` command. Enter this command on the web server:

```
netstat -tan |grep LISTEN|grep -v 127.0.0.1
```

The output shows which network ports are currently being listened to by the web server. Your goal is to generate alerts when that output changes. Edit the web server `ossec.conf` file. Note the different `localfile` definitions. Add a new `localfile` entry at the end of the file, just above the last line.

```
<localfile>
  <log_format>full_command</log_format>
  <command>netstat -tan |grep LISTEN|grep -v 127.0.0.1</command>
  <frequency>5</frequency>
</localfile>
```

²Note the lack of an `s` on the end of this command on the web server. Command syntax varies between the Ubuntu configuration of OSS (on the client) and the CentOS configuration on the web server.

³`iptables` is a Linux function for filtering network traffic. See the `iptables2` lab and/or user “man iptables” to learn more about their use

The `log_format` entry tells OSSEC you are defining a command that it is to periodically run. The `command` entry is the command you want it to run. And the `frequency` is how often, in seconds, that you want to run the command. After you restart the `ossec-hids` service, OSSEC will start to periodically run that command and send the output to the server.

Now, on the server, we need a rule to monitor that output. There already is such a rule, which you can see in the file at

```
/var/ossec/rules/ossec_rules.xml
```

In that file, find rule 533, which is reproduced below. The rule format definitions can be found in the OSSEC web pages.

```
<rule id="533" level="7">
  <if_sid>530</if_sid>
  <pcre2>ossec: output: 'netstat -tan</pcre2>
  <check_diff />
  <description>Listened ports status (netstat) changed
    (new port opened or closed).</description>
</rule>
```

This rule can be read as follows: The id is an arbitrary number that identifies the rule. The level is 7, which is high enough to generate an alert. Rule classifications are characterized at: <https://www.ossec.net/docs/docs/manual/rules-decoders/rule-levels.html>.

The `if_sid` entry reflects the OSSEC rule chaining strategy. It says to consider this rule only if the event already matched rule id 530, which is a rule that identifies output from monitored commands. The `pcre2` entry identifies which command output to evaluate, in this case, output from the `netstat` command defined to run on the web server⁴. The `check_diff` entry tells OSSEC to generate alerts when the monitored messages change.

Close the file and restart the `ossec` service. Then go to the web server and use the `netcat` command to listen to some arbitrary port, e.g.,

```
nc -l 22345
```

You should see a corresponding alert. Then stop netcat using `ctrl-c`. Note another alert, this time because the port was not longer being listened to. Recall the frequency of our command output generation is every 5 seconds. In a real deployment, you may wish to reduce the frequency so that the web server service can be updated and restarted without generating alerts. On the other hand, the lower the frequency, the more time rouge software has to listen to a port without being detected.

5.6 Monitor web resource access

In this section, we'll create rules to monitor access to a specific web resource based on web log entries. The first step is to make sure our web logs are forwarded to the server.

⁴`pcre2` is a "regular expression" syntax. This simple example just does a string match. Use "man pcre" to learn more, and checkout regexone.com for a tutorial.

5.6.1 Log locations

Recall that log locations are defined in the client's `ossec.conf` file. Open that file on the web server and find the entry for the web server, which is `apache`. You will see two entries, one for the access log and one for the error log. Neither match our installation, which puts the logs in `/var/log/httpd`. Alter the 2 entries to reflect the log locations. Then restart the `ossec-hids` service.

You can confirm the agent is processing the expected logs by viewing the `/var/ossec/logs/ossec.log` file.

5.6.2 Rules testing

OSSEC provides a tool to help create and test new rules, and we'll use this tool to help understand the structure of the rules chains. At the client1 workstation, issue the following web request from the command line⁵:

```
curl 172.0.0.4
```

At the web server, tail the access.log:

```
tail -f /var/log/httpd/access.log
```

At the OSSEC server start the `ossec-logtest` program in verbose mode:

```
/var/ossec/bin/ossec-logtest -v
```

This program will consume a log entry provided as standard input, and it will display its processing steps and any alerts that would have been generated had the lab entry been real. Copy the log entry from the web server's access.log and paste it into the server window where the logtest program is running. Observe the output.

The first phase simply repeats the log entry. The second phase reflects the results of decoding per the `/var/ossec/etc/decoder.xml` file. In this example, we see the decoder has decoded this as a `web-accesslog`, and it identifies a set of values defined for that log type, including the success of the GET command, which is 200 (successful). The decoder assigns this decoding a *type* of `web-log`, (unfortunately not reflected in the tools output.) Phase 3 is the rules processing.

OSSEC rules processing for this example can be summarized as: Start at the lowest numbered rule and find the first match. The "rule 4" in this example was found in the `rules.config.xml` file. It then looks at rules having a category of `web-log`, which includes rules defined in the `web.rules.xml` file, where it finds a match with rule ID 31100. It then searches for rules that are a "child" of rule 31100, i.e., those with an `if_sid` of 31100. Again, it searches in order until it finds a match. In this case the match is rule ID 31108, which represents itself as "Ignored URLs (simple queries)". It then looks for a child of 31108 that matches the log entry. We see it tried rule 31103 and rule 31509, but did not find a match and thus halted with rule 31108 as the best match. Since the rule has a level of 0, it is ignored and no alert is generated.

For our example purposes, that rule chain is fine because we don't care if that particular web page is accessed. Our focus will be on access to the `172.0.0.4/plan.html` page, (those of you who've performed the snort lab will recognize these web pages). Go back to the client workstation and use `curl` to retrieve the plan.html page. Look at the web server log entry and consider how we can identify log entries reflecting access to the plan.html page. Obviously we can just look for that string. So we'll create a new rule that looks for that string.

⁵There are several ways of making web requests, including `curl`, `wget` and browsers. For most of this lab, please use `curl`. You'll see why toward the end.

In order for our new rule to even be considered, it will have to be a child of rule 31108. Though we may want to make the rule a child of 31100, we cannot because the OSSEC rules algorithm will always match the 31108 first. (OSSEC prescribes that you not modify the existing OSSEC rules and follow the convention of putting new rules in the `local_rules.xml` file with rule ID > 100000). Note this is a non-trivial constraint because rules are searched in order of rule ID.

Open the `local_rules.xml` file and add this rule:

```
<rule id="140234" level="7">
  <if_sid>31108</if_sid>
  <url_pcre2>plan.html</url_pcre2>
  <description>Accessed the business plan.</description>
</rule>
```

Then re-run the `ossec-logtest` (use `ctrl-c` to break out of a previous use of the program and then start the program again.)

Once you've tested the rule and see that it works, restart the `ossec` service on the server and issue the `curl` request from the workstation again to confirm the alert appears in the `alerts.log`.

5.6.3 Event coverage

The new rule generates an alert when the `plan.html` resource is successfully accessed. But does it always? Alter your query to: `172.0.0.4/plan.html?`. Do you see an alert? The question mark causes the rule parsing to no longer consider this log entry a simple query. Run the log entry in the `logtest` program and view the processing. Note the selected rule is 31100, which we've seen before. To handle non-simple queries, we'd like a rule that is a child of 31100. OSSEC lets us make a rule that is a child to multiple other rules by including a comma-separated list of IDs in the `if_sid` field. Make that change to the new rule and test it.

5.6.4 Failed attempts

What about when someone tries and fails to access the plan? In other words, could we alert if it appears that someone is using brute force to try to find the business plan? At the client, try to retrieve this resource: `172.0.0.4/plan9.html`. Notice how no alert is generated. Go back to the `logtest` and view the processing of the new access log entry.

Create a new rule to generate an alert that reports the following when a failed attempt is made to access any resources with the string *plan* in the URL.

```
Attempt to access the business plan.
```

To complete this lab, you must test your new rule along with the previous rule and demonstrate the following by using `curl` at the client1:

- A query for `plan.html` or `plan.html?` generates the "Accessed the business plan" alert.
- A query for `plan9.html` generates the "Attempt to access the business plan" alert.
- A query for `about.html` does not generate any alert.

5.7 Completeness (i.e., rat holes)

Now that we have our web content rules in place, try one more experiment. Instead of using `curl`, use `wget` on `client1` to retrieve the resources in the list of queries above, and observe the alerts. What is missing? Use the `logtest` program to observe the rules processing on the log entry that failed to create the anticipated alert. You need not correct the problem for this lab. But do make note of this property of rules-based bolt-on security mechanisms: **You never know when you are done.** This is often true when trying to protect systems using IDS, anti-virus, and rules-based “mandatory access controls”.

5.8 Effects on system security

On the `client1` workstation, go to the `/var/ossec/bin` directory and view the directory content. Use `ps` to see which `ossec` program run as root:

```
ps aux | grep ossec
```

The `ossec-logcollector` program is 167KB, without libraries, which can be seen using the `ldd` program:

```
ldd ossec-logcollector
```

Then go to the server computer and view its binaries and note which ones run as root.

While OSSEC does separate some processing to the non-root `ossec` user, much processing is still performed by root, and this includes collection of logs whose content may be determined by arbitrary external entities.

Any system contemplating use of an IDS should weigh the trade-offs associated with introducing large amounts of privileged code into their systems.

5.8.1 Abuse of active responses

Another system security consideration is the potential for abuse of active responses. For example, how might an attacker cause denial of service, preventing `client1` from accessing the web server, just by causing some non-privileged program to run on `client1`? Or consider the case of an attacker having non-privileged access to `client1` who would like ten minutes of unobserved interaction with `client1`? If `client1` were the source of numerous failed `ssh` attempts to the OSSEC server itself, what would be the result?

6 Submission

After finishing the lab, go to the terminal on your Linux system that was used to start the lab and type:

```
stoplab
```

When you stop the lab, the system will display a path to the zipped lab results on your Linux system. Provide that file to your instructor, e.g., via the Sakai site.

This lab was developed for the Labtainer framework by the Naval Postgraduate School, Center for Cybersecurity and Cyber Operations under sponsorship from the National Science Foundation Award Number 1932950. This work is in the public domain, and cannot be copyrighted.