This lab may incorporate AI tools to support your learning.

## Activity overview

Previously, you learned about packet analysis and the basic syntax and components of intrusion detection systems (IDS) signatures and rules. You also learned how to examine network traffic using Wireshark and how to use the Linux command line to perform network system administration and network troubleshooting.

In this lab, activity you'll explore more about Suricata alerts and logs, including the general process of rule creation.

The Suricata tool monitors network interfaces and applies rules to the packets that pass through the network. Suricata's configurations and rules are similar to those of other IDS and intrusion prevention systems (IPS).

Source and destination networks can be specified in the Suricata configuration. Custom rules can be written to specify actions with precision provided.

You'll examine a rule and practice using Suricata to trigger alerts on network traffic, find alerts in logs, and write rules. Lastly, you'll use `jq`, a tool for JSON data, to help you understand some of the alerts and logs generated by Suricata.

Let's get started with Suricata!

**Note:** The terms **rules** and **signatures** are used interchangeably in this lab activity.

## Scenario

In this scenario, you're a security analyst who must monitor traffic on your employer's network. You've been assigned to configure Suricata and use its logs and alerts.

Here's the job: As the lab user, **Pete**, you'll operate custom rules in Suricata. **Sonnet**, your colleague, has left you a file containing a rule that she wrote. You'll need to add the rule to the `local.rules` file. **Finally**, you'll examine the generated alerts and logs. Suricata provides in the standard `eve.json` log file.

Sonnet's comments at the top of the rule file are strictly, you've been supplied with a rule to `ping` the VM's `eth0` network interface. Pete thinks it's a good idea.

Let's define the rule you'll be working with in this lab activity:

- The `sample` array: This is a packet capture file that contains an example of network traffic, which you'll use to test your rule. Suricata will also use this file to generate alerts and logs.
- The `custom.rules` file contains a custom rule where the lab activity starts. You'll utilize rules in this file and run them against the replayed traffic stored in `sample.pcap` file.
- The `fast.log` file will contain the alerts that Suricata generates. The `fast.log` file is simply where the lab starts. Each line you see is a log, and, in all of rules, against the `sample.pcap` file. The `fast.log` file is located in `/var/log/suricata/`. The rule's value of this condition is part of the rule's `sid` field. The `fast.log` file will be located in the `/var/log/suricata/` directory after Suricata runs the `fast.log` file is located in `/var/log/suricata/` directory after Suricata runs the `fast.log` file. Suricata's response or these logging tools can be used to troubleshoot and generate helpful reports related to quality assurance.
- The `eve.json` file is the main, standard, and default log file newly generated by Suricata. It contains detailed information about alerts triggered, as well as other network activity and events. The `eve.json` file is located in `/var/log/suricata/` directory. Analysts only, and can also be located in the `/var/log/suricata/` directory.

When you create a new rule, you'll need to test the rule to confirm whether or not it worked as expected. You can use the `fast.log` file to quickly compare the number of alerts generated each time you run Suricata to test a signature against the `sample.pcap` file.

Pete's to-do list:

**Note:** The lab starts with your user account, called `student`, already logged in as a `root` shell. This means you can start with the tasks as soon as you click the **Start lab** button.

**Disclaimer:** For optimal performance and compatibility, it is recommended to use either **Google Chrome** or **Mozilla Firefox** browsers while accessing this lab.

## Start your lab

You'll need to open the lab before you can access the materials. To do this, click the green "Start lab" button at the top of the screen.

Start lab

After you click the **Start Lab**, a pop-up, or a dialog, where you will be performing further steps in the lab, You should have a dialog like this:

When you have completed all the tasks, refer to the final your lab section that follows the tasks for information on how to end your lab.

## Task 1. Examine a custom rule in Suricata

The `/home/student/` directory contains a custom `rules` file that defines the network traffic alerts, which Suricata captures.

In this task, you'll explore the composition of the Suricata rule defined in the `custom.rules` file.

- Use the `cat` command to display the rule in the `custom.rules` file:

```
cat /home/student/suricata/rules/custom.rules
```

**Note:** The `/suricata` command can also be used to read the content you page at a time, making it useful for reading lengthy output.

The command returns the rule as the output in the shell:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"GET on WIN";
flow:established,to_server; content:"GET"; http_method; sid:1245; rev:3;)
```

The rule consists of three components: **action**, **header**, and **rule options**.

Let's examine each component in more detail:

### Action

**Action** is the first part of the signature. It determines the syntax to trigger a rule's condition are met.

Action offer various network intrusion detection system (NIDS) rule languages, but some common actions are `alert`, `drop`, `pass`, and `reject`.

In this example, `alert` is the action, which is the action. The `alert` keyword tells Suricata to generate an alert when a rule's condition. The lab will inspect the rule's syntax and send an alert in case it matches.

Said that the `alert` action is the generic action, and it's best practice to use an action only occurs when Suricata runs in IPS mode.

The `pass` action allows the traffic to pass through the network interface. The action rule can be used to override other rules, for example to a drop rule rule to pass traffic from a specific IP address. The `drop` action drops the packet and generates an alert. Lastly, `reject` sends a packet rejection or reset packet for protocols or actions where rules from the address to pass.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"GET on WIN";
flow:established,to_server; content:"GET"; http_method; sid:1245; rev:3;)
```

The `->` arrow indicates the direction of the traffic to be inspected.

### Header

```
alert http $HOME_NET any -> $EXTERNAL_NET any
(msg:"GET on WIN";
flow:established,to_server; content:"GET";
http_method; sid:1245; rev:3;)
```

**Note:** The `#` symbol indicates the start of a comment. Variables and comments are discussed in a later section.

So far, we know that this signature triggers an alert when it detects any http traffic leaving the home network and going to the external network.

### Rule options

```
alert http $HOME_NET any -> $EXTERNAL_NET
any (msg:"GET on WIN";
flow:established,to_server; content:"GET";
http_method; sid:1245; rev:3;)
```

The many available **rule options** allow you to customize signatures with additional parameters. Configuring rule options helps security analysts to define rules with precision. Rule options are enclosed in parentheses `()`, and multiple rule options are typically enclosed in a pair of parentheses and separated by semicolons.

Let's review rule options that are defined in our example:

- The `msg:` option provides the alert text. In this case, the alert will print out the text `"GET on WIN"`, which describes why the alert was triggered.
- The `flow:established,to_server` option determines that packets from the client to the server are part of an established TCP connection. Suricata will ignore the alert, corresponding to the rule if it's not part of a TCP stream.
- The `content:"GET"` option tells Suricata to look for the word `GET` in the content of the packet. Suricata will trigger the alert.
- The `http_method` option instructs Suricata to inspect only the HTTP method field.
- The `sid:` option indicates the signature's unique value (sid) is used to identify the signature's version. Here, the revision version is 3.

To summarize, this signature triggers an alert whenever Suricata observes the text `GET` as the HTTP method in all HTTP packets from the home network going to the external network.

Click **Check my progress** to verify that you have completed this task correctly.

Examine a custom rule in Suricata

Check my progress

## Task 2. Trigger a custom rule in Suricata

Now that you know how rules work, let's create a rule that triggers an alert. Let's test and trigger this rule and examine the alert logs that Suricata generates.

1. Cat the file in the `/var/log/suricata` folder:

```
ls -l /var/log/suricata
```

Note that before running Suricata, there are no files in the `/var/log/suricata` directory.

2. Run `suricata -c /etc/suricata/suricata.yaml -vvvv -r sample.pcap` files:

```
sudo suricata -c /etc/suricata/suricata.yaml -vvvv -r sample.pcap
```

This command starts the Suricata application and processes the `sample.pcap` file. Suricata uses the `-c` option to specify a configuration file, and the `-vvvv` option prints more statistics and processing details.

**Note:** In this lab setup `sudo` is required to process packet capture files with `-r` option, although it may not be required in a production environment.

Now you'll further examine the options in the command:

- The `-c /etc/suricata/suricata.yaml` option specifies an input file to read, relatively, in this case the `sample.pcap` file.
- The `-r /suricata /var/log/suricata` option instructs Suricata to use the rules defined in the `/var/log/suricata/` directory.
- The `-vvvv` option instructs Suricata to display all debugging output.

As a refresher, `/etc/suricata` are a way to separate configuration files and rules. Because you are using relatively with Pete's existing packet capture file, you don't need to specify a packet capture file with `-r` option.

Suricata adds a new alert file in the `/var/log/suricata/fast.log` file when all the processes are run and the rules are `-r`.

3. Cat the file in the `/var/log/suricata` folder again:

```
ls -l /var/log/suricata
```

Note that after running Suricata, there are now five files in the `/var/log/suricata` directory, including the `fast.log` and `eve.json` files. Suricata generates these files when it runs.

4. Use the `cat` command to display the `fast.log` file generated by Suricata:

```
cat /var/log/suricata/fast.log
```

The output returns alert entries in the log:

```
01/06/2023:15:00:36.404000  [**] [1:1245:3] GET on WIN [**]
[Classification: (null)] [Priority: 3] {TCP} 172.17.0.2:49152 ->
142.250.1.139:80
01/06/2023:15:00:36.404000  [**] [1:1245:3] GET on WIN [**]
[Classification: (null)] [Priority: 3] {TCP} 172.17.0.2:49152 ->
142.250.1.139:80
```

Each line is similar to the `fast.log` file corresponds to an alert generated by Suricata when it processes a packet that meets the conditions of an alert generating rule. Suricata's alert line includes the message that identifies the rule that triggered the alert, as well as the timestamp, destination, and direction of the traffic.

Click **Check my progress** to verify that you have completed this task correctly.

Trigger a custom rule in Suricata

Check my progress

## Task 3. Examine eve.json output

In this task, you must examine the additional output that Suricata generates in the `eve.json` file.

As previously mentioned, this file is located in the `/var/log/suricata` directory.

The `eve.json` file is the standard and default log file that Suricata uses to store data that the `fast.log` file. This data is stored in a JSON format, which makes it much more useful for people and programs to read and parse.

1. Use the `cat` command to display the entire `eve.json` file:

```
cat /var/log/suricata/eve.json
```

The output returns the raw content of the file. You'll notice that there's a lot of data generated that is not easy to understand in this format.

2. Use the `jq` command to display the entire in an improved format:

```
jq . /var/log/suricata/eve.json | less
```

**Note:** You can use the Ctrl+Shift+Alt+F keys to move forward or backward through the output. Alternatively, you can use the `/` keyword to search for a specific term. Use the `q` keyword to exit the output. Alternatively, you can use the `/` keyword to search for a specific term in the command prompt.

3. Press `q` to exit the `less` command and to return to the command line prompt.

Note how much easier it is to read the output now as opposed to the `cat` command output.

**Note:** The `jq` tool is very useful for processing JSON data, however, a full explanation of its capabilities is beyond the scope of this lab.

What is the value of the severity property for the first alert reported by the jq command?

- 1
- 2
- 3
- 4

Submit

**Answer:** The value of the severity property for the first alert is 3.

4. Use the `jq` command to extract specific event data from the `eve.json` file:

```
jq -c "[.timestamp, .flow_id, alert.signature, proto, dest_ip]"
/var/log/suricata/eve.json
```

**Note:** The `-c` command option prevents the fields specified in the list to the square brackets `[]` from being printed on multiple lines. The `timestamp` field is a string representation of the date and time, `flow_id` is the ID of a specific network flow, `alert.signature` is the `proto` protocol, `proto` and the destination IP address `dest_ip`.

What is the destination IP address listed for the last event in the `eve.json` file?

- 142.250.1.139
- 142.250.1.132
- 192.168.0.1
- 172.17.0.2/24.0.1

Submit

**Answer:** The destination IP address is 142.250.1.139.

What is the alert signature for the first alert entry in the `eve.json` file?

- BAD CHECK ACKNOWT
- DROP CMPT ON WIN0NETT
- ACK ATTACK
- Fake CMPT ON WIN0NETT

Submit

**Answer:** The first event in the `eve.json` file flags "GET on WIN0" as the alert signature.

The following is an example of the output of the command phrase. The `flow_id` is the long numeric field highlighted. Longer is a quick two minutes.

```
{"ts":"2023-01-06 15:00:36.404000+0000","dest_ip":"
142.250.1.139","flow_id":
"139611271251 4e7b0000","alert.signature":"GET on WIN0",
"proto":"TCP"}
```

5. Use the `jq` command to display all event logs matching to specific `flow_id` from the `eve.json` file. The `flow_id` value is a 16-digit number and will vary for each event. Replace `139611271251 4e7b0000` with your `flow_id` value obtained in the previous command.

```
jq "select(.flow_id==139611271251 4e7b0000)" /var/log/suricata/eve.json
```

**Note:** A network flow refers to a sequence of packets between a source and destination that share common characteristics such as IP addresses, protocols, and port numbers. Network flows are used to monitor network traffic and analyze patterns of network traffic to identify and analyze threats. Suricata assigns a unique `flow_id` to each network flow. All logs from a network flow share the same `flow_id`, which allows you to filter and analyze all events associated with a specific network flow using the same unique `flow_id`.

Click **Check my progress** to verify that you have completed this task correctly.

Examine eve.json output

Check my progress

## Conclusion

Great work!

You've completed the activity and should be able to use Suricata to trigger alerts on network traffic.

You now have practical experience in running Suricata to:

- Create custom rules and use them in Suricata,
- Monitor traffic captured in a packet capture file, and
- Examine the `fast.log` and `eve.json` logs.

You've developed a key skill in your journey towards becoming a security analyst.

## End your lab

Before you end the lab, make sure you're satisfied that you've completed all the tasks, and that they are correct.

1. Click **End Lab**. A pop-up box will appear. Click **Submit** to confirm that you're done. Waiting for a short time will ensure your progress is fully recorded. You can wait for the lab to time out, or you can click **End Lab** to end the lab immediately.

2. Another pop-up box will ask you to rate the lab and provide feedback comments. You can complete that if you choose to.

3. Close the browser tab containing the lab to return to your course.

4. Refresh the browser tab for the course to mark the lab as complete.