Creating Custom Rules

Cybersecurity

Networks Security Day 3

Today's Objectives

By the end of class, you will be able to:

- ☐ Interpret Snort rules
- ☐ Write custom Snort rules
- ☐ Design rule signatures based on the type of network traffic.

NetSec in a Professional Context

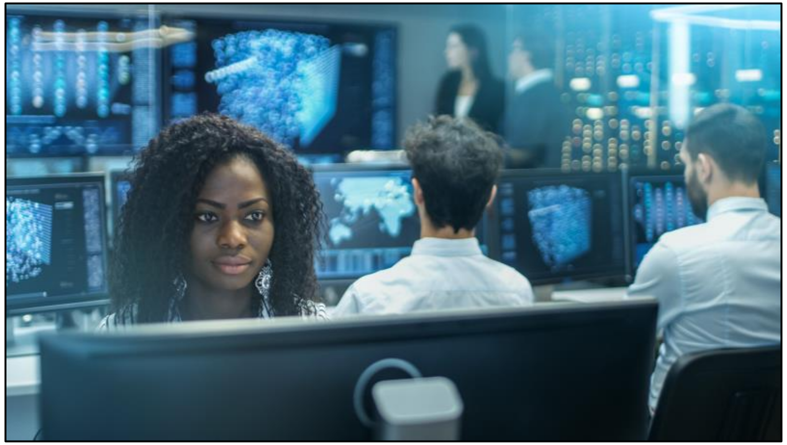
Today's lesson relates to the following career paths:

SOC Analysts

SOC Analysts must be familiar with common types of malicious activity, the kinds of rules they trigger, and how to write and modify existing rules.

Networking Engineering / Administrator:

Networking engineers / administrator are often responsible for implementing and testing new rules on their networks.



Review Topics

✓ **Snort** works similarly to a firewall: It reads incoming packets; scans each one for specific signatures, specified in *rules*; and triggers a rule if a packet contains a signature.

✓ A **signature** is a set of traits that define a type of network traffic.

✓ **Snort** can both detect and block traffic matching certain signatures:

- either simply alert when it spots suspicious traffic, but still allow the packets into/out of the network;
- Or alert and then drop the traffic.

What are the Rules?

Let’s cover the three basics components of a Snort Rule:

Action: Indicates Whether Snort should alert, log, or pass once a signature is identified.

Protocol and Source / Destination: Specifies which source and destination IP addresses and ports to watch, and whether to monitor **`ip`**, **`tcp`**, or **`udp`** traffic.

Rule Options: Rule options allow administrators to tweak rules. Common options include:

- **`Content`** Configures Snort to search packets for specific data
- **`react: block`** Configures Snort to drop packets matching a given rule
- **`resp: rst_all`** Configures Snort to respond to packets matching a given rule with a TCP RST packet, which closes the connection.

Snort Rule Example 1

```
alert tcp !192.168.10.0/24 any -> 192.168.10.1 22
{msg: "Attempt to connect to router's SSH server!";
 react: block;
 sid: 100001;
 rev: 1}
```

This rule sends an alerts whenever a machine outside of 192.168.10.0/24 attempts to connect to port 22 on 192.168.10.1.

Snort Rule Example 1

```
alert tcp !192.168.10.0/24 any -> 192.168.10.1 22
{msg: "Attempt to connect to router's SSH server!";
 react: block;
 sid: 100001;
 rev: 1}
```

Alert tells Snort to generate an alert, rather than ignore, or log the packet.

Snort Rule Example 1

```
alert tcp !192.168.10.0/24 any -> 192.168.10.1 22
{msg: "Attempt to connect to router's SSH server!";
 react: block;
 sid: 100001;
 rev: 1}
```

`!192.168.10/0/24 any -> 192.168.10.1 22` means "traffic coming from any machine not in `192.168.10.0/24` that is heading to port 22 on 192.168.10.1".
The `!` means `"not"`.

Snort Rule Example 1

```
alert tcp !192.168.10.0/24 any -> 192.168.10.1 22
{msg: "Attempt to connect to router's SSH server!";
 react: block;
 sid: 100001;
 rev: 1}
```

The curly braces `{...}` contain the rule options:

`msg`: A string to log when this rule first fires.

`react: block` : Order to block the packet in addition to generating an alert

`sid 100001`: IDs this rule as 100001.

`rev`: Specifies that this is version 1 of the rule

Snort Rule Example 2

```
alert tcp any any -> 192.168.10.0/24 21
{msg: "malware.exe detected!";
 content: “malware.exe”;
 resp: rst_all;
 sid: 100002;
 rev: 1}
```

This rule blocks inbound FTP traffic containing the string `malware.exe`. This rule would be used to prevent attackers from using FTP to upload files containing the names of known malware.

Snort Rule Example 2

```
alert tcp any any -> 192.168.10.0/24 21
{msg: "malware.exe detected!";
 content: “malware.exe”;
 resp: rst_all;
 sid: 100002;
 rev: 1}
```

`any any` indicates traffic from any source port and IP address

The `content` option allows administrators to specify a string to search for in packets.

`resp: rst_all` option tells Snort to close connections to machines that attempted to transfer data containing malware.exe

Filtering Traffic with Snort Rules

In Cyberscore website launch the **Snort Signatures, IDS Tuning, and Blocking** lab, and follow the instructions from **Create an ICMP Drop Rule**

.

Suggested Time:
1:00

Filtering Traffic Review

Key Steps:

01 **Verify IDS Functionality**: A simple "sanity check" to verify that SNORBY works properly. (Completed in the previous class)

02 **Create an ICMP Drop Rule**: Write a Snort rule that *drops* incoming ICMP (ping) packets.

03 **Generate ICMP Traffic:** Used the Kali machine to ping the SNORBY machine and verify that the new rule is in effect.

04 **Drop Network Traffic:** Writing a rule to block anyone on the subnet from sending HTTP traffic. This prevents users on the subnet from browsing the web.

Review Snort Rules

The **Drop ICMP** rule *rejects* all ICMP traffic between 192.168.11.30 (the SNORBY machine) and 192.168.11.20 (the Kali Machine).

```
reject icmp 192.168.11.30 any <> 192.168.11.20 any
(msg:"Blocking ICMP Packet from 192.168.11.20";
 sid:1000001;
 rev:1;)
```

Review Snort Rules

The **Drop All ICMP** rule rejects all ICMP traffic between *all* machines on the 192.168.11.0/24 subnet.

```
reject icmp 192.168.11.0/24 any <> 192.168.11.0/24 any
(msg:"Blocking ICMP Packet from Host on Local Domain";
 sid:1000001;
 rev:1;)
```

Review Snort Rules

The **Drop Outgoing HTTP** rule prevents machines on 192.168.11.0/24 subnet from communicating with *any* HTTP server.

Note: the `react:block` line in this rule causes the firewall to reject HTTP traffic, rather than only alerting.

```
alert tcp 192.168.11.0/24 80 -> any 80
(msg: "Outgoing HTTP connection";
 react: block;
 sid:1000008;)
```

Review Snort Rules

The **Drop Incoming HTTP** rule prevents machines *outside* the subnet from communicating *within* the subnet. In other words, this rule prevents HTTP servers on the subnet from sending data to the public Internet.

Note: the `react:rst_all` causes the firewall to respond to HTTP request with TCP RST (connection closed) packets, instead of requested data.

```
alert tcp any any -> 192.168.11.0/24 80
(flags: S;
 resp: rst_all;
 msg: "Incoming HTTP Traffic blocked";
 sid:1000009;)
```

Review Custom Rules

Block / Alert on SMB Scans

```
alert tcp !192.168.11.0/24 any -> 192.168.11.0/24 445
{msg: "Blocking attempted SMB connection!";
 resp: rst_all;
 sid: 1000010;
  rev: 1}
```

Review Custom Rules

Drop Outgoing HTTP

```
alert tcp 192.168.11.0/24 443 -> any 443
(msg: "Outgoing HTTP connection";
 react: block;
 sid:1000011;)
```

Review Custom Rules

Block / Alert on NetBIOS Connections

```
alert udp 192.168.11.0/24 any -> any 137
(msg: "Blocking NetBIOS connection attempt!";
 react: block;
 sid:1000012;)
```

Testing Signatures with PCAPs

We should always verify the final implementation of rules in live environments, ***but*** it can be difficult to generate fake traffic to test complex rules.

Testing with PCAP files

We wouldn't want to send live malware onto our network, so instead, network engineers test rules using packet captures of the traffic they want to detect or block.

01 Find or generate a PCAP containing the traffic you want to detect.

02 Write a rule to detect that traffic.

03 Use Snort to read the PCAP *instead of live traffic*.

Benefits of using PCAP files

Using PCAP files to customize and test rules have the following advantages:

- Engineers can test complex rules reliably, without generating live network traffic.
- Engineers can develop rules that detect potentially dangerous traffic (malware, DoS traffic, etc.) without subjecting the network to those conditions.
- Engineers can test *new* rules against *old* captures therefore recognizing if attacks they only just started looking for have ever happened in the past.

Customizing Snort Rules

Customizing Snort Rules

Snort rules works by **identifying the source and destination IP addresses and ports of incoming and outgoing traffic.**

Network engineers are typically not told which ports and addresses to block when tasked with implementing snort rules.

- Instead, they're told what kind of traffic to reject.
- Therefore, they must determine the IP addresses and ports themselves.

For example: Assume that an organization's corporate network lives on 192.168.11.0/24.

- An engineer for this organization might be told to "*block all SSH traffic to corporate machines from outside the corporate network*".
- It is then the engineer's job to figure out that they must drop traffic to port 22 from machines that are *not* in the range 192.168.11.0/24.

Creating Custom Snort Rules

Network admins can use the following process to customize a Snort rule, when given a specific request:

01 Identify Behavior (Should the rule reject, block, etc.)

02 Identify Source IP Addresses to Block

03 Identify Source/Destination Ports to Block

04 Identify "Special" Signatures to Include

Scenario for the next example

We’ll use the following example to create a snort rule based on the signature.

Assume that an organization's corporate network lives on `192.168.11.0/24`.

An engineer for this organization might be told to "*block all SSH traffic to corporate machines from outside the corporate network*".

It is then the engineer's job to figure out that they must drop traffic to port 22 from machines that are *not* in the range `192.168.11.0/24`.

Create an SSH rule

Step	**Action**	**Snort Syntax**
1. Identify Behavior	This rule should alert on and block TCP traffic to SSH ports.	Rule should start with alert tcp and include react: block
2. Identify Source IP Addresses / Ports to Block	The rule should block traffic to port 22 on any machine in the corporate intranet from any outside machine.	`!192.168.11.0/24 any > ...`
3. Identify Destination IP Addresses/Ports to Block	Again, this rule should block traffic *to* SSH servers in the corporate network	`... -> 192.168.11/0/24 22`
4. Identify "Special" Signatures to Include	This rule does not require scanning packets for malware or other "embedded" signatures, so no `content` or similar rule modifications are required.	

Create an SSH rule

When put together, those steps produce the following rule:

```
alert tcp !192.168.11.0/24 any -> 192.168.11.0/24 22 (
  msg:"Blocking foreign request to open SSH connection.";
  react: block;
  sid:2000001;
  rev: 1;)
```

Testing Signatures with PCAPs

To test your signature with a PCAP, run the following command in your lab environment terminal:

```
sudo snort -l . -c /etc/nsm/onion-dmz-eth0/snort.conf -r
/home/student/capture131.pcap
```

`sudo snort`	Run Snort with admin privileges
`-l .`	Save logs/alerts to the current directory.
`-c /etc/nsm/onion-dmz-eth0/snort.conf`:	Use the configuration file in `/etc/nsm/onion-dmz-eth0/snort.conf`
`-r /home/student/capture131.pcap`	Use the traffic capture `capture131.pcap` to test the rules specified in `/etc/nsm/onion-dmz-eth0/snort.conf`

Your Turn: Creating and Testing Signatures

In this activity, you will act as a network administrator tasked with writing Snort rules to block malicious activity detected by your SOC analyst.

Instruction Sent via Slack

Suggested Time:
1:00

Creating and Testing Signatures Review

Rule 1: Detect FTP Connections

```
alert tcp any any -> any 21
(msg: "Established FTP connections ";
 flags:S;
 sid:10000;)
```

This rule blocks traffic from *any* local machine to port 21 on *any* remote machine.

The flags:S rule option specifies that the rule should only apply to TCP packets with the SYN (S) flag set.

Creating and Testing Signatures Review

Rule 2: Block Traffic to/from Known C2 Server

```
alert ip any any <> 172.16.200.2 any
(msg:" KNOWN C2 Server ";
 sid:10001;)
```

This rule simply blocks traffic from *any* machine on the local subnet to 172.16.200.2.

Creating and Testing Signatures Review

Rule 3: Block Traffic Containing `nc.exe`

```
alert tcp any any <> any any
(msg: "Looking for nc.exe ";
 content:"nc.exe";
 nocase; sid:10002;)
```

This rule blocks *any* packet to/from the local subnet containing the string `nc.exe`.

Creating and Testing Signatures Review

Rule 4: Alert on UDP Traffic to/from Local Subnet

```
alert udp any any <> any any
(msg: "UDP Traffic";
sid:10004;)
```

This rule alerts on *any* UDP traffic to/from the local subnet, because attackers often use DNS tunneling through ports *other* than the standard 53.

Creating and Testing Signatures Review

Rule 5: Block Outbound DNS Traffic to Foreign Machines

```
alert udp any any <> any 53
(msg: "Outbound DNS Requests";
sid:10005;)
```

This rule blocks traffic to UDP port 53 on foreign machines.

Creating and Testing Signatures Review

These rules can be tests against a PCAP file using the following command:

```
sudo snort -l . -c /etc/nsm/onion-dmz-eth0/snort.conf -r
/home/student/capture131.pcap
```

Today’s Objectives

By the end of class, you will be able to:

- 🟩 Interpet Snort rules
- 🟩 Write custom Snort rules
- 🟩 Create rule signatures based on the type of network traffic.