Working with IDS and IPS

Introduction

Introduction To IDS/IPS

Overview

- In network security monitoring (NSM) operations, the use of `Intrusion Detection Systems (IDS)` and `Intrusion Prevention Systems (IPS)` is paramount.
  - The purpose of these systems is not only to identify potential threats but also to mitigate their impact.
- An `IDS` is a device or application that monitors network or system activities for malicious activities or policy violations and produces reports primarily to a management station.
  - Such a system gives us a clear sense of what's happening within our network, ensuring we have visibility on any potentially harmful actions.
  - It should be noted that an IDS doesn't prevent an intrusion but alerts us when one occurs.
  - The IDS operates in two main modes: signature-based detection and anomaly-based detection.
    - In `signature-based detection`, the IDS recognizes bad patterns, such as malware signatures and previously identified attack patterns.
    - However, this method is limited to only known threats.
    - For this reason, we also implement `anomaly-based detection`, which establishes a baseline of normal behavior and sends an alert when it detects behavior deviating from this baseline.
    - It's a more proactive approach but is susceptible to false positives, hence why we use both methods to balance each other out.
- On the other hand, an `IPS` sits directly behind the firewall and provides an additional layer of protection.
  - It does not just passively monitor the network traffic, but actively prevents any detected potential threats.
  - Such a system doesn't just alert us of intruders, but also actively stops them from entering.
  - An IPS also operates in a similar mode to IDS, offering both signature-based and anomaly-based detection.

        - Once a potential threat is detected, it takes actions such as dropping malicious packets, blocking network traffic, and resetting the connection.
        - The goal is to interrupt or halt activities that are deemed dangerous to our network or against our policy.
- When deploying IDS and IPS, they are typically integrated into the network at different points, each having its optimal place depending on its function and the overall network design.
    - Both IDS and IPS devices are generally positioned behind the firewall, closer to the resources they protect.
    - As they both work by examining network traffic, it makes sense to place them where they can see as much of the relevant traffic as possible, which is typically on the internal network side of the firewall.
    - Intrusion Detection Systems (IDS) passively monitor network traffic, detecting potential threats and generating alerts.
        - By placing them behind the firewall, we can ensure they're analyzing traffic that has already passed the first line of defense, allowing us to focus on potentially more subtle or complex threats that have bypassed the firewall.
    - Intrusion Prevention Systems (IPS), on the other hand, actively intervene to stop detected threats.
        - This means they need to be placed at a point in the network where they can not only see potentially malicious traffic but also have the authority to stop it. This is usually achieved by placing them inline on the network, often directly behind the firewall.
- The deployment may vary based on the network's specific requirements and the kind of traffic we need to monitor.
    - IDS/IPS can also be implemented on the host level, known as Host-based Intrusion Detection Systems (HIDS) and Host-based Intrusion Prevention Systems (HIPS), which monitor the individual host's inbound and outbound traffic for any suspicious activity.
- Please note that the placement of these systems is an integral part of a defense-in-depth strategy, where multiple layers of security measures are used to protect the network.
    - The exact architecture will depend on various factors, including the nature of the network, the sensitivity of the data, and the threat landscape.

IDS/IPS Updates

- Moreover, to ensure these systems perform at their best, we consistently update them with the latest threat signatures and fine-tune their anomaly detection algorithms.

    - This requires a diligent, ongoing effort from our security team, but it's absolutely essential given the continually evolving threat landscape.
- It's also important to mention the role of Security Information and Event Management (SIEM) systems in our network security monitoring operations.
    - By collecting and aggregating logs from IDS and IPS along with other devices in our network, we can correlate events (analyzing the relationships) from different sources and use advanced analytics to detect complex, coordinated attacks.
    - This way, we have a complete, unified view of our network's security, enabling us to respond quickly to threats.

Coming Up

- In this module, we will explore the fundamentals of `Suricata`, `Snort`, and `Zeek`, along with providing insights into signature development and intrusion detection for each of these systems.

One Sentence Summary

- IPS (usually placed directly behind a firewall, a point in the network) prevents threat (e.g. through dropping packets) while IDS (usually placed behind a firewall) only notifies us through alert and they use a combo of signature-based detection and anomaly-based detection.

Suricata

Suricata Fundamentals

Overview

- Regarded as a potent instrument for Network Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), and Network Security Monitoring (NSM), Suricata represents a cornerstone of network security.
    - This open-source powerhouse, managed and developed by the Open Information Security Foundation (OISF), is a testament to the strength of a community-led, non-profit initiative.
- The objective of Suricata is to dissect every iota of network traffic, seeking potential signs of malicious activities.

- Its strength lies in the ability to not only conduct a sweeping evaluation of our network's condition but also delve into the details of individual application-layer transactions.
- The key to Suricata's successful operation lies in an intricately designed set of rules.
- These guidelines direct Suricata's analysis process, identifying potential threats and areas of interest.
- Equipped to perform at high velocities on both off-the-shelf and specifically designed hardware, Suricata's efficiency is second to none.

Suricata Operation Modes

- Suricata operates in four (4) distinct modes:

1. The `Intrusion Detection System (IDS) mode` positions Suricata as a silent observer.
    - In this capacity, Suricata meticulously examines traffic, flagging potential attacks but refraining from any form of intervention.
    - By providing an in-depth view of network activities and accelerating response times, this mode augments network visibility, albeit without offering direct protection.
2. In the `Intrusion Prevention System (IPS) mode`, Suricata adopts a proactive stance.
    - All network traffic must pass through Suricata's stringent checks and is only granted access to the internal network upon Suricata's approval.
    - This mode bolsters security by proactively thwarting attacks before they penetrate our internal network.
    - Deploying Suricata in IPS mode demands an intimate understanding of the network landscape to prevent the inadvertent blocking of legitimate traffic.
    - Furthermore, each rule activation necessitates rigorous testing and validation.
    - While this mode enhances security, the inspection process may introduce latency.
3. The `Intrusion Detection Prevention System (IDPS) mode` brings together the best of both IDS and IPS.
    - While Suricata continues to passively monitor traffic, it possesses the ability to actively transmit RST packets in response to abnormal activities.
    - This mode strikes a balance between active protection and maintaining low latency, crucial for seamless network operations.

4. In its `Network Security Monitoring (NSM) mode`, Suricata transitions into a dedicated logging mechanism, eschewing active or passive traffic analysis or prevention capabilities.
    - It meticulously logs every piece of network information it encounters, providing a valuable wealth of data for retrospective security incident investigations, despite the high volume of data generated.

Suricata Inputs

- Regarding Suricata inputs, there are two main categories:

1. `Offline Input:` This involves reading PCAP files for processing previously captured packets in the `LibPCAP` file format.
    - It is not only advantageous for conducting post-mortem data examination but also instrumental when experimenting with various rule sets and configurations.
2. `Live Input`: Live input can be facilitated via `LibPCAP`, where packets are read directly from network interfaces.
    - However, `LibPCAP` is somewhat hamstrung by its performance limitations and lack of load-balancing capabilities.
    - For inline operations, `NFQ` and `AF_PACKET` options are available.
    - `NFQ`, a Linux-specific inline IPS mode, collaborates with IPTables to divert packets from the kernel space into Suricata for detailed scrutiny.
    - Commonly used inline, `NFQ` necessitates drop rules for Suricata to effectively obstruct packets.
    - Conversely, `AF_PACKET` provides a performance improvement over `LibPCAP` and supports multi-threading.
    - Nevertheless, it's not compatible with older Linux distributions and can't be employed inline if the machine is also tasked with routing packets.

- Please note that there are other, less commonly used or more advanced inputs available.

Suricata Outputs

- Suricata creates multiple outputs, including logs, alerts, and additional network-related data such as DNS requests and network flows.
    - One of the most critical outputs is `EVE`, a JSON formatted log that records a wide range of event types including alerts, HTTP, DNS, TLS metadata, drop, SMTP

metadata, flow, netflow, and more.
    - Tools such as Logstash can easily consume this output, facilitating data analysis.
- We might encounter `Unified2` Suricata output, which is essentially a Snort binary alert format, enabling integration with other software that leverages Unified2.
    - Any Unified2 output can be read using Snort's `u2spewfoo` tool, which is a straightforward and effective method to gain insight into the alert data.

Configuring Suricata & Custom Rules

- Once we've accessed the deployed Suricata instance over SSH, we can get an overview of all the rule files with a simple execution command.

```
areaeric@htb[/htb]$ ls -lah /etc/suricata/rules/
```

```
areaeric@htb[/htb]$ ls -lah /etc/suricata/rules/
total 27M
drwxr-xr-x 2 root root 4.0K Jun 28 12:10 .
drwxr-xr-x 3 root root 4.0K Jul  4 14:44 ..
-rw-r--r-- 1 root root  31K Jun 27 20:55 3coresec.rules
-rw-r--r-- 1 root root 1.9K Jun 15 05:51 app-layer-events.rules
-rw-r--r-- 1 root root 2.1K Jun 27 20:55 botcc.portgrouped.rules
-rw-r--r-- 1 root root  27K Jun 27 20:55 botcc.rules
-rw-r--r-- 1 root root 109K Jun 27 20:55 ciarmy.rules
-rw-r--r-- 1 root root  12K Jun 27 20:55 compromised.rules
-rw-r--r-- 1 root root  21K Jun 15 05:51 decoder-events.rules
-rw-r--r-- 1 root root  468 Jun 15 05:51 dhcp-events.rules
-rw-r--r-- 1 root root 1.2K Jun 15 05:51 dnp3-events.rules
-rw-r--r-- 1 root root 1.2K Jun 15 05:51 dns-events.rules
-rw-r--r-- 1 root root  32K Jun 27 20:55 drop.rules
-rw-r--r-- 1 root root 2.7K Jun 27 20:55 dshield.rules
-rw-r--r-- 1 root root 365K Jun 27 20:55 emerging-activex.rules
-rw-r--r-- 1 root root 613K Jun 27 20:55 emerging-adware_pup.rules
-rw-r--r-- 1 root root 650K Jun 27 20:55 emerging-attack_response.rules
-rw-r--r-- 1 root root  33K Jun 27 20:55 emerging-chat.rules
-rw-r--r-- 1 root root  19K Jun 27 20:55 emerging-coinminer.rules
-rw-r--r-- 1 root root 119K Jun 27 20:55 emerging-current_events.rules
-rw-r--r-- 1 root root 1.7M Jun 27 20:55 emerging-deleted.rules
-rw-r--r-- 1 root root  20K Jun 27 20:55 emerging-dns.rules
-rw-r--r-- 1 root root  62K Jun 27 20:55 emerging-dos.rules
-rw-r--r-- 1 root root 606K Jun 27 20:55 emerging-exploit_kit.rules
-rw-r--r-- 1 root root 1.1M Jun 27 20:55 emerging-exploit.rules
-rw-r--r-- 1 root root  45K Jun 27 20:55 emerging-ftp.rules
-rw-r--r-- 1 root root  37K Jun 27 20:55 emerging-games.rules
-rw-r--r-- 1 root root 572K Jun 27 20:55 emerging-hunting.rules
-rw-r--r-- 1 root root  18K Jun 27 20:55 emerging-icmp_info.rules
-rw-r--r-- 1 root root  11K Jun 27 20:55 emerging-icmp.rules
-rw-r--r-- 1 root root  15K Jun 27 20:55 emerging-imap.rules
-rw-r--r-- 1 root root  11K Jun 27 20:55 emerging-inappropriate.rules
-rw-r--r-- 1 root root 2.9M Jun 27 20:55 emerging-info.rules
-rw-r--r-- 1 root root  48K Jun 27 20:55 emerging-ja3.rules
-rw-r--r-- 1 root root 8.2M Jun 27 20:55 emerging-malware.rules
---SNIP---
```

- The rules can be seen in a straightforward list format and be inspected to understand their functionality, as follows.

```
areaeric@htb[/htb]$ more /etc/suricata/rules/emerging-malware.rules
```

```
# This Ruleset is EmergingThreats Open optimized for suricata-5.0-enhanced.

#alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET MALWARE Psyb0t joining an IRC Channel"; flow:established,to_server; flowbits:
JOIN #mipsel"; reference:url,www.adam.com.au/bogaurd/PSYB0T.pdf; reference:url,doc.emergingthreats.net/2009172; classtype:trojan-act
tadata:created_at 2010_07_30, updated_at 2010_07_30;)

alert tcp $HOME_NET any -> $EXTERNAL_NET 25 (msg:"ET MALWARE SC-KeyLog Keylogger Installed - Sending Initial Email Report"; flow:est
Installation of SC-KeyLog on host "; nocase; content:"<p>You will receive a log report every "; nocase; reference:url,www.soft-centr
url,doc.emergingthreats.net/2002979; classtype:trojan-activity; sid:2002979; rev:4; metadata:created_at 2010_07_30, updated_at 2010_

alert tcp $HOME_NET any -> $EXTERNAL_NET 25 (msg:"ET MALWARE SC-KeyLog Keylogger Installed - Sending Log Email Report"; flow:establi
eyLog log report"; nocase; content:"See attached file"; nocase; content:".log"; nocase; reference:url,www.soft-central.net/keylog.ph
threats.net/2008348; classtype:trojan-activity; sid:2008348; rev:2; metadata:created_at 2010_07_30, updated_at 2010_07_30;)
---SNIP---
```

- Rules might be commented out, meaning they aren't loaded and don't affect the system.
  - This usually happens when a new version of the rule comes into play or if the threat associated with the rule becomes outdated or irrelevant.
- Each rule usually involves specific variables, such as `$HOME_NET` and `$EXTERNAL_NET`.
  - The rule examines traffic from the IP addresses specified in the `$HOME_NET` variable heading towards the IP addresses in the `$EXTERNAL_NET` variable.
- These variables can be defined in the `suricata.yaml` configuration file.

```
areaeric@htb[/htb]$ more /etc/suricata/suricata.yaml
```

- This allows us to customize these variables according to our specific network environment and even define our own variables.
- Finally, to configure Suricata to load signatures from a custom rules file, such as `local.rules` in the `/home/htb-student` directory, we would execute the below.

```
areaeric@htb[/htb]$ sudo vim /etc/suricata/suricata.yaml
```

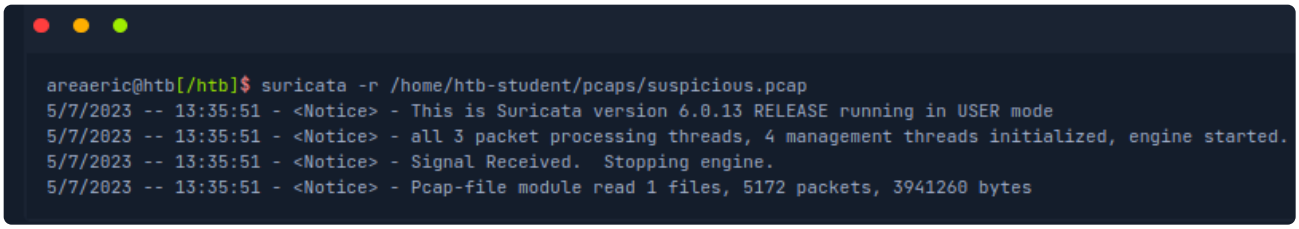
1. Add `/home/htb-student/local.rules` to `rule-files:`
2. Press the `Esc` key
3. Enter `:wq` and then, press the `Enter` key

Hands-on With Suricata Inputs

- With Suricata inputs, we can experiment with both offline and live input:

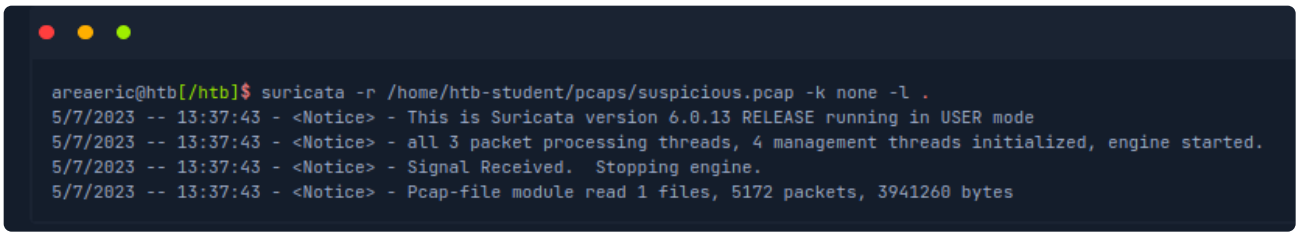
1. For offline input (reading PCAP files - suspicious.pcap in this case), the following command needs to be executed, and Suricata will create various logs (mainly `eve.json`, `fast.log`, and `stats.log`).

```
areaeric@htb[/htb]$ suricata -r /home/htb-student/pcaps/suspicious.pcap
```



- An alternative command can be executed to bypass checksums (`-k` flag) and log in a different directory (`-l` flag).

```
areaeric@htb[/htb]$ suricata -r /home/htb-student/pcaps/suspicious.pcap
-k none -l .
```



2. For live input, we can try Suricata's (Live) `LibPCAP` mode as follows.

```
areaeric@htb[/htb]$ ifconfig
```

```
areaeric@htb[/htb]$ ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.129.205.193  netmask 255.255.0.0  broadcast 10.129.255.255
    inet6 dead:beef::250:56ff:feb9:68dc  prefixlen 64  scopeid 0x0<global>
    inet6 fe80::250:56ff:feb9:68dc  prefixlen 64  scopeid 0x20<link>
    ether 00:50:56:b9:68:dc  txqueuelen 1000  (Ethernet)
    RX packets 281625  bytes 84557478 (84.5 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 62276  bytes 23518127 (23.5 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop  txqueuelen 1000  (Local Loopback)
    RX packets 888  bytes 64466 (64.4 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 888  bytes 64466 (64.4 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
areaeric@htb[/htb]$ sudo suricata --pcap=ens160 -vv
```

```
areaeric@htb[/htb]$ sudo suricata --pcap=ens160 -vv
[sudo] password for htb-student:
5/7/2023 -- 13:44:01 - <Notice> - This is Suricata version 6.0.13 RELEASE running in SYSTEM mode
5/7/2023 -- 13:44:01 - <Info> - CPUs/cores online: 2
5/7/2023 -- 13:44:01 - <Info> - Setting engine mode to IDS mode by default
5/7/2023 -- 13:44:01 - <Info> - Found an MTU of 1500 for 'ens160'
5/7/2023 -- 13:44:01 - <Info> - Found an MTU of 1500 for 'ens160'
5/7/2023 -- 13:44:01 - <Info> - fast output device (regular) initialized: fast.log
5/7/2023 -- 13:44:01 - <Info> - eve-log output device (regular) initialized: eve.json
5/7/2023 -- 13:44:01 - <Info> - stats output device (regular) initialized: stats.log
5/7/2023 -- 13:44:01 - <Info> - Running in live mode, activating unix socket
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for http_uri
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for http_uri
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for http_raw_uri
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for http_raw_uri
5/7/2023 -- 13:44:01 - <Info> - 1 rule files processed. 1 rules successfully loaded, 0 rules failed
5/7/2023 -- 13:44:01 - <Info> - Threshold config parsed: 0 rule(s) found
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for tcp-packet
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for tcp-stream
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for udp-packet
5/7/2023 -- 13:44:01 - <Perf> - using shared mpm ctx' for other-ip
5/7/2023 -- 13:44:01 - <Info> - 1 signatures processed. 0 are IP-only rules, 0 are inspecting packet payload, 1 inspect applica
5/7/2023 -- 13:44:01 - <Perf> - TCP toserver: 1 port groups, 1 unique SGH's, 0 copies
5/7/2023 -- 13:44:01 - <Perf> - TCP toclient: 0 port groups, 0 unique SGH's, 0 copies
5/7/2023 -- 13:44:01 - <Perf> - UDP toserver: 1 port groups, 1 unique SGH's, 0 copies
5/7/2023 -- 13:44:01 - <Perf> - UDP toclient: 0 port groups, 0 unique SGH's, 0 copies
5/7/2023 -- 13:44:01 - <Perf> - OTHER toserver: 0 proto groups, 0 unique SGH's, 0 copies
5/7/2023 -- 13:44:01 - <Perf> - OTHER toclient: 0 proto groups, 0 unique SGH's, 0 copies
5/7/2023 -- 13:44:01 - <Perf> - Unique rule groups: 2
5/7/2023 -- 13:44:01 - <Perf> - Builtin MPM "toserver TCP packet": 0
5/7/2023 -- 13:44:01 - <Perf> - Builtin MPM "toclient TCP packet": 0
5/7/2023 -- 13:44:01 - <Perf> - Builtin MPM "toserver TCP stream": 0
5/7/2023 -- 13:44:01 - <Perf> - Builtin MPM "toclient TCP stream": 0
5/7/2023 -- 13:44:01 - <Perf> - Builtin MPM "toserver UDP packet": 0
5/7/2023 -- 13:44:01 - <Perf> - Builtin MPM "toclient UDP packet": 0
5/7/2023 -- 13:44:01 - <Perf> - Builtin MPM "other IP packet": 0
5/7/2023 -- 13:44:01 - <Perf> - AppLayer MPM "toserver dns_query (dns)": 1
5/7/2023 -- 13:44:01 - <Info> - Using 1 live device(s).
5/7/2023 -- 13:44:01 - <Info> - using interface ens160
```

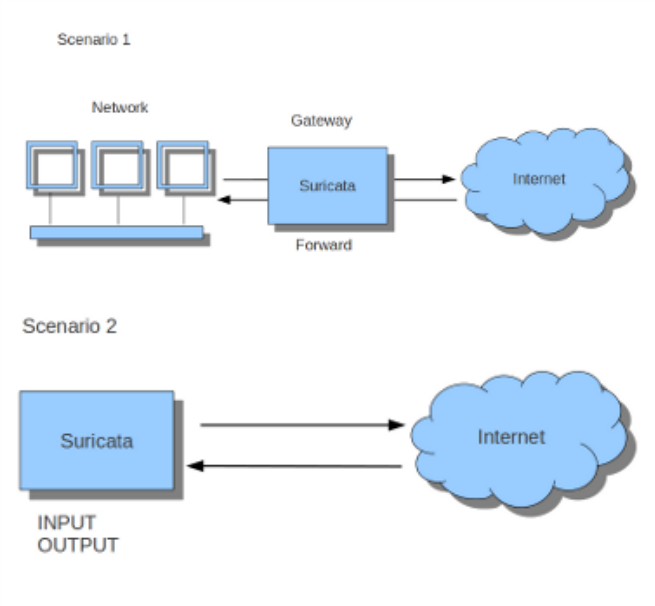
3. For Suricata in Inline (`NFQ`) mode, the following command should be executed first.

```
areaeric@htb[/htb]$ sudo iptables -I FORWARD -j NFQUEUE
```

E.g. Illustration

First of all, it is important to know which traffic you would like to send to Suricata. There are two choices:

1. Traffic that passes your computer
2. Traffic that is generated by your computer.



If Suricata is running on a gateway and is meant to protect the computers behind that gateway you are dealing with the first scenario: *forward_ing* .

If Suricata has to protect the computer it is running on, you are dealing with the second scenario: *host* (see drawing 2).

These two ways of using Suricata can also be combined.

The easiest rule in case of the gateway-scenario to send traffic to Suricata is:

```
sudo iptables -I FORWARD -j NFQUEUE
```

- Then, we should be able to execute the following.

```
areaeric@htb[/htb]$ sudo suricata -q 0
```

- Moreover, to try Suricata in `IDS` mode with `AF_PACKET` input, execute one of the below.

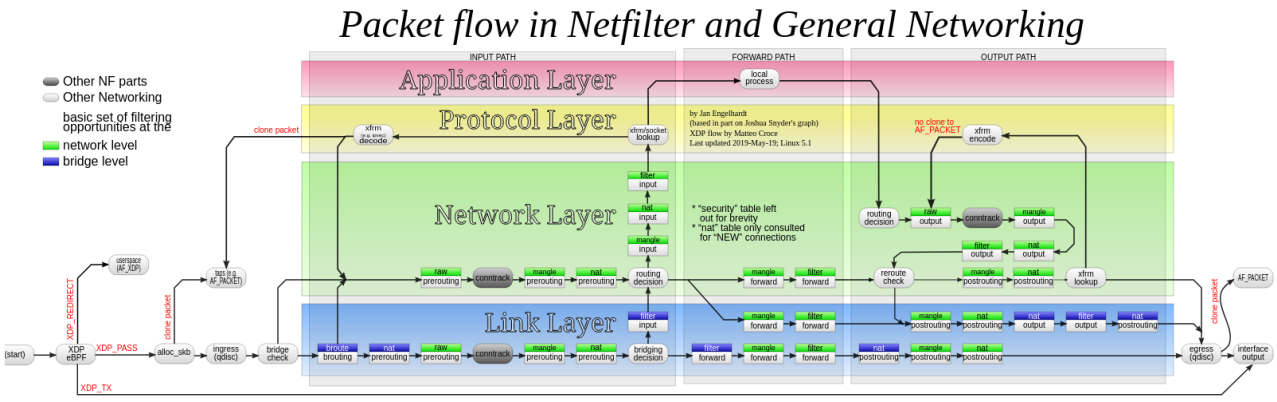
```
areaeric@htb[/htb]$ sudo suricata -i ens160
```

```
areaeric@htb[/htb]$ sudo suricata -i ens160
5/7/2023 -- 13:53:35 - <Notice> - This is Suricata version 6.0.13 RELEASE running in SYSTEM mode
5/7/2023 -- 13:53:35 - <Notice> - all 1 packet processing threads, 4 management threads initialized, engine started.
```

```
areaeric@htb[/htb]$ sudo suricata --af-packet=ens160
```

```
areaeric@htb[/htb]$ sudo suricata --af-packet=ens160
5/7/2023 -- 13:54:34 - <Notice> - This is Suricata version 6.0.13 RELEASE running in SYSTEM mode
5/7/2023 -- 13:54:34 - <Notice> - all 1 packet processing threads, 4 management threads initialized, engine started.
```

E.g. References:



15.2. Setting up IPS at Layer 2

15.2.1. AF_PACKET IPS mode

AF_PACKET capture method is supporting a IPS/Tap mode. In this mode, you just need the interfaces to be up. Suricata will take care of copying the packets from one interface to the other. No `iptables` or `nftables` configuration is necessary.

You need to dedicate two network interfaces for this mode. The configuration is made via configuration variable available in the description of an AF_PACKET interface.

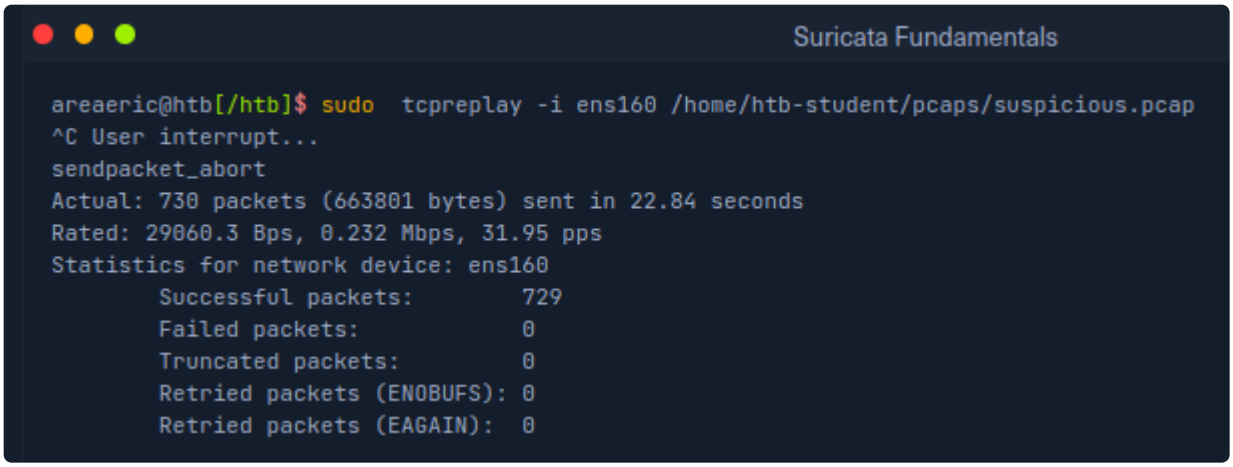
For example, the following configuration will create a Suricata acting as IPS between interface `eth0` and `eth1`:

```
af-packet:
  - interface: eth0
    threads: 1
    defrag: no
    cluster-type: cluster_flow
    cluster-id: 98
    copy-mode: ips
    copy-iface: eth1
    buffer-size: 64535
    use-mmap: yes
  - interface: eth1
    threads: 1
    cluster-id: 97
    defrag: no
    cluster-type: cluster_flow
    copy-mode: ips
    copy-iface: eth0
    buffer-size: 64535
    use-mmap: yes
```

-> AF_Packet copies packet from one interface to another.

- To observe Suricata dealing with "live" traffic, let's establish an additional SSH connection and utilize `tcpreplay` to replay network traffic from a PCAP file (`suspicious.pcap` in this case).

```
areaeric@htb[/htb]$ sudo  tcpreplay -i ens160 /home/htb-
student/pcaps/suspicious.pcap
```



- Then, feel free to terminate both tcpreplay and Suricata. The logs from the observed (replayed) traffic will be available at `/var/log/suricata`

Note: The -i option helps Suricata choose the best input option.
- In the case of Linux, the best input option is AF_PACKET. If pcap mode is needed, the --pcap option is recommended.
- Configuration of (Live) LibPCAP can be achieved via the suricata.yaml file, including settings for buffer size, BPF or tcpdump filters, checksum validation, threads, promiscuous mode, snap length, etc.

Hands-on With Suricata Outputs

- Suricata records a variety of data into logs that reside in the `/var/log/suricata` directory by default.
  - For us to access and manipulate these logs, we require root-level access.
  - Among these logs, we find the `eve.json`, `fast.log`, and `stats.log` files, which provide invaluable insight into the network activity. Let's delve into each:

1. `eve.json`: This file is Suricata's recommended output and contains JSON objects, each carrying diverse information such as timestamps, flow_id, event_type, and more.
   - Try inspecting the content of `old_eve.json` residing at `/var/log/suricata` as follows.

```
areaeric@htb[/htb]$ less /var/log/suricata/old_eve.json
```

```
areaeric@htb[/htb]$ less /var/log/suricata/old_eve.json
{"timestamp":"2023-07-06T08:34:24.526482+0000","event_type":"stats","stats":{"uptime":8,"capture":{"kernel_packets":4,"kernel_d
---SNIP---
```

- If we wish to filter out only alert events, for example, we can utilize the `jq` command-line JSON processor as follows.

```
areaeric@htb[/htb]$ cat /var/log/suricata/old_eve.json | jq -c 'select(.event_type == "alert")'
```

```
areaeric@htb[/htb]$ cat /var/log/suricata/old_eve.json | jq -c 'select(.event_type == "alert")'
{"timestamp":"2023-07-06T08:34:35.003163+0000","flow_id":1959965318909019,"in_iface":"ens160","event_type":"alert","src_ip":"10
```

- If we wish to identify the earliest DNS event, for example, we can utilize the `jq` command-line JSON processor as follows.

```
areaeric@htb[/htb]$ cat /var/log/suricata/old_eve.json | jq -c 'select(.event_type == "dns")' | head -1 | jq .
```

```
areaeric@htb[/htb]$ cat /var/log/suricata/old_eve.json | jq -c 'select(.event_type == "dns")' | head -1 | jq .
{
 "timestamp": "2023-07-06T08:34:35.003163+0000",
 "flow_id": 1959965318909019,
 "in_iface": "ens160",
  "event_type": "dns",
  "src_ip": "10.9.24.101",
 "src_port": 51833,
 "dest_ip": "10.9.24.1",
 "dest_port": 53,
 "proto": "UDP",
 "dns": {
  "type": "query",
  "id": 6430,
     "rrname": "adv.epostoday.uk",
     "rrtype": "A",
     "tx_id": 0,
    "opcode": 0
 }
}
```

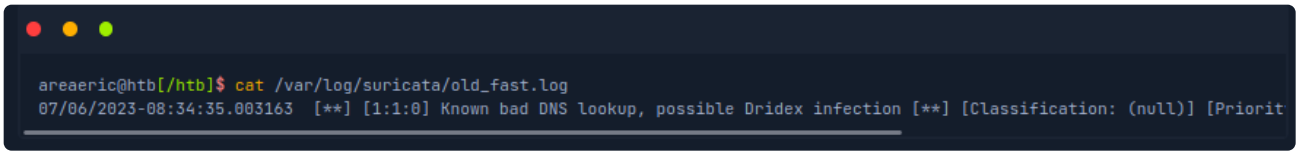
- We can also use similar commands to filter for other event types like TLS and SSH.

`flow_id`: This is a unique identifier assigned by Suricata to each network flow.
- A flow, in Suricata terms, is defined as a set of IP packets passing through a network interface in a specific direction and between a given pair of source and destination endpoints.
- Each of these flows gets a unique flow_id.
- This identifier helps us track and correlate various events related to the same network flow in the EVE JSON log.
- Using `flow_id`, we can associate different pieces of information related to the same flow, such as alerts, network transactions, and packets, providing a cohesive view of what is happening on a specific communication channel.

`pcap_cnt`: This is a counter that Suricata increments for each packet it processes from the network traffic or from a PCAP file (in offline mode).
- `pcap_cnt` allows us to trace a packet back to its original order in the PCAP file or network stream.
- This is beneficial in understanding the sequence of network events as they occurred.
- It can help to precisely pinpoint when an alert was triggered in relation to other packets, which can provide valuable context in an investigation.

2. `fast.log`: This is a text-based log format that records alerts only and is enabled by default.
    - Try inspecting the content of `old_fast.log` residing at `/var/log/suricata` as follows.

```
areaeric@htb[/htb]$ cat /var/log/suricata/old_fast.log
```



3. `stats.log`: This is a human-readable statistics log, which can be particularly useful while debugging Suricata deployments.
- Try inspecting the content of `old_stats.log` residing at `/var/log/suricata` as follows.

```
areaeric@htb[/htb]$ cat /var/log/suricata/old_stats.log
```

```
areaeric@htb[/htb]$ cat /var/log/suricata/old_stats.log
------------------------------------------------------------------------------------
Date: 7/6/2023 -- 08:34:24 (uptime: 0d, 00h 00m 08s)
------------------------------------------------------------------------------------
Counter                                       | TM Name                   | Value
------------------------------------------------------------------------------------
capture.kernel_packets                        | Total                     | 4
decoder.pkts                                  | Total                     | 3
decoder.bytes                                 | Total                     | 212
decoder.ipv6                                  | Total                     | 1
decoder.ethernet                              | Total                     | 3
decoder.icmpv6                                | Total                     | 1
decoder.avg_pkt_size                          | Total                     | 70
decoder.max_pkt_size                          | Total                     | 110
flow.icmpv6                                   | Total                     | 1
flow.wrk.spare_sync_avg                       | Total                     | 100
flow.wrk.spare_sync                           | Total                     | 1
flow.mgr.full_hash_pass                       | Total                     | 1
flow.spare                                    | Total                     | 9900
tcp.memuse                                    | Total                     | 606208
tcp.reassembly_memuse                         | Total                     | 98304
flow.memuse                                   | Total                     | 7394304
------------------------------------------------------------------------------------
---SNIP---
```

- For those of us who want a more focused output strategy, there's an option to deactivate the comprehensive `EVE` output and activate particular outputs instead.
  - Take `http-log`, for instance.
  - By activating this, every time Suricata is run and encounters HTTP events, a fresh `http.log` file will be generated.

Hands-on With Suricata Outputs - File Extraction

- Suricata, has an underused yet highly potent feature - file extraction.
  - This feature allows us to capture and store files transferred over a number of different protocols, providing invaluable data for threat hunting, forensics, or simply data analysis.
- Here's how we go about enabling file extraction in Suricata.
- We start by making changes to the Suricata configuration file (`suricata.yaml`). In this file, we'll find a section named `file-store`.
  - This is where we tell Suricata how to handle the files it extracts.
  - Specifically, we need to set `version` to `2`, enabled to `yes`, and the `force-filestore` option also to `yes`.

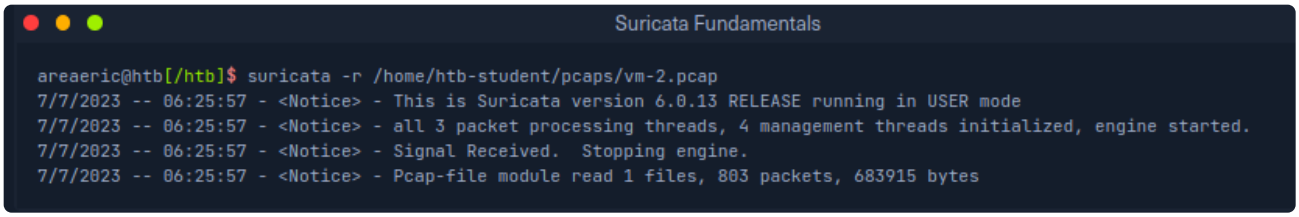
- The resulting section should look something like this.

- In the same `file-store` section, we define where Suricata stores the extracted files.
  - We set the `dir` option to the directory of our choice.
- As a quick exercise, let's enable file extraction and run Suricata on the `/home/htb-student/pcaps/vm-2.pcap` file from www.netresec.com.
- In accordance with the guidelines put forth in Suricata's documentation, file extraction isn't an automatic process that occurs without our explicit instructions.
  - It's fundamentally crucial for us to craft a specific rule that instructs Suricata when and what kind of files it should extract.
- The simplest rule we can add to our `local.rules` file to experiment with file extraction is the following.

```
alert http any any -> any any (msg:"FILE store all"; filestore; sid:2; rev:1;)
```

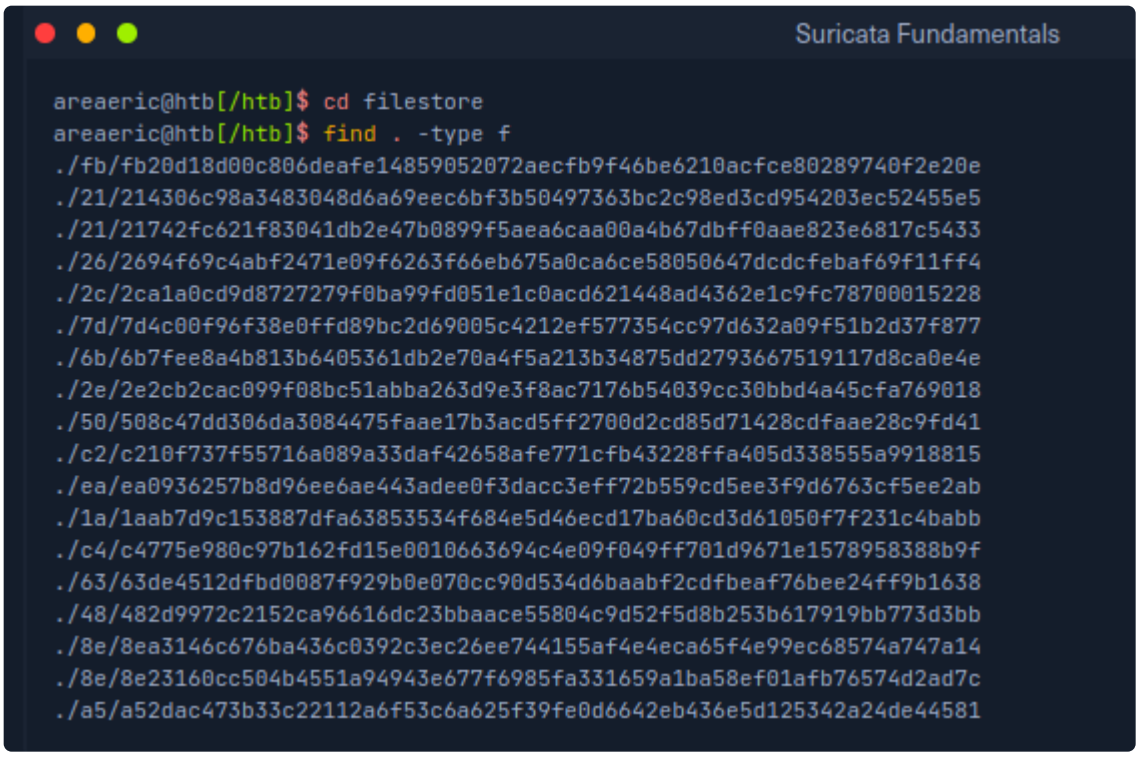
- If we configured Suricata correctly, multiple files will be stored inside the `filestore` directory.
- Let's run Suricata on the `/home/htb-student/pcaps/vm-2.pcap` file.

```
areaeric@htb[/htb]$ suricata -r /home/htb-student/pcaps/vm-2.pcap
```



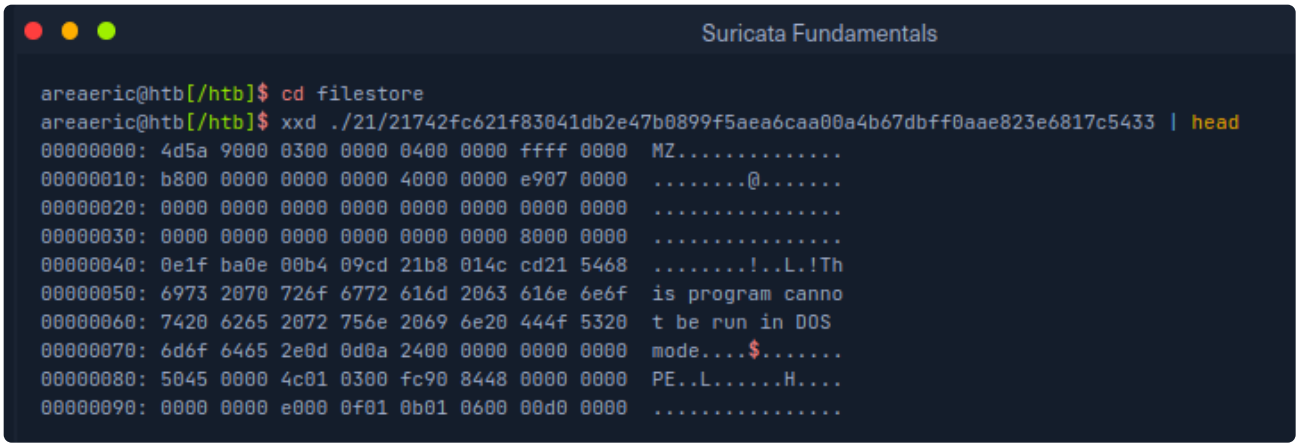
- We will notice that `eve.json`, `fast.log`, `stats.log`, and `suricata.log` were created, alongside a new directory called `filestore`. `filestore`'s content in terms of the files it contains can be inspected as follows.

```
areaeric@htb[/htb]$ cd filestore
areaeric@htb[/htb]$ find . -type f
```



- Again in accordance with the guidelines put forth in Suricata's documentation the `file-store` module uses its own log directory (default: `filestore` in the default logging directory) and logs files using the SHA256 of the contents as the filename.
  - Each file is then placed in a directory named 00 to ff where the directory shares the first 2 characters of the filename.
  - For example, if the SHA256 hex string of an extracted file starts with `f9bc6d...` the file we be placed in the directory `filestore/f9`.
- If we wanted to inspect, for example, the `/21/21742fc621f83041db2e47b0899f5aea6caa00a4b67dbff0aae823e6817c5433` file inside the `filestore` directory, we could use the `xxd` tool as follows.

```
areaeric@htb[/htb]$ cd filestore
areaeric@htb[/htb]$ xxd
./21/21742fc621f83041db2e47b0899f5aea6caa00a4b67dbff0aae823e6817c5433 |
head
```



- In this case, the file was a Windows executable based on the file's header. More about the MS-DOS EXE format can be found in following resource MZ.

Live Rule Reloading Feature & Updating Suricata Rulesets

- Live rule reloading is a crucial feature in Suricata that allows us to update our ruleset without interrupting ongoing traffic inspection.
  - This feature provides continuous monitoring and minimizes the chances of missing any malicious activity.
- To enable live rule reloading in Suricata, we need to configure our Suricata configuration file (`suricata.yaml`).
  - In the `suricata.yaml` file, we should locate the `detect-engine` section and set the value of the `reload` parameter to `true`.
  - It looks something like this:

```
detect-engine:
  - reload: true
```

- Proceed to execute the following `kill` command, which will signal the Suricata process (determined by `$(pidof suricata)`) to refresh its rule set without necessitating a complete restart.

```
areaeric@htb[/htb]$ sudo kill -usr2 $(pidof suricata)
```

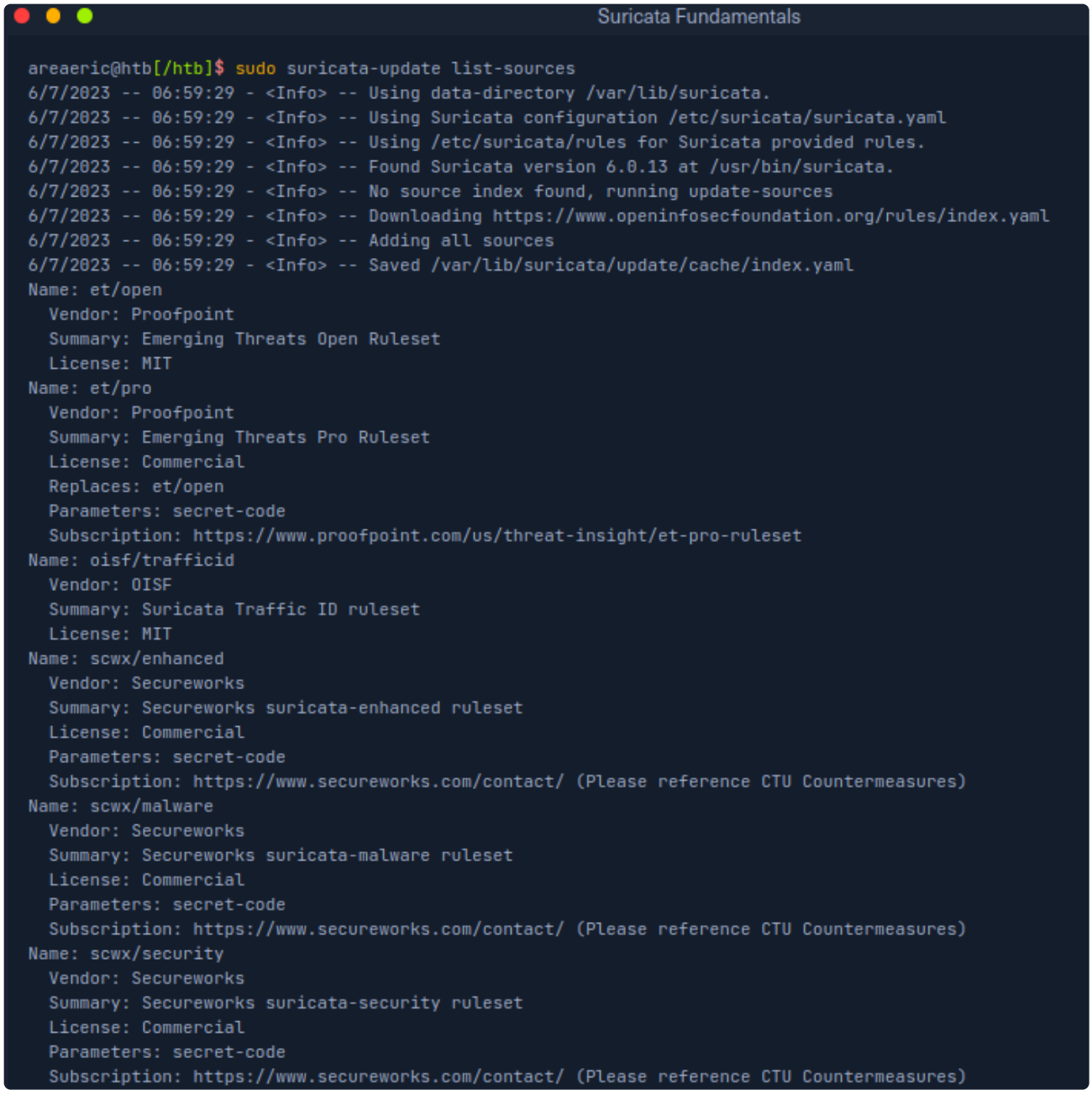
- This modification tells Suricata to check for changes in the ruleset periodically and apply them without needing to restart the service.

- Updating Suricata's ruleset can be performed using the `suricata-update` tool.
  - We can perform a simple update to the Suricata ruleset using the following command.

```
areaeric@htb[/htb]$ sudo suricata-update
```

- As displayed in the example above, the output indicates that the `suricata-update` command has successfully retrieved the rules by establishing a connection with `https://rules.emergingthreats.net/open/`.
  - Subsequently, the command saves the newly obtained rules to the `/var/lib/suricata/rules/` directory.
- Moving forward, let's execute the command provided below to generate a comprehensive list of all ruleset providers.

```
areaeric@htb[/htb]$ sudo suricata-update list-sources
```



- Let's make a note from the output above of a specific ruleset name from which we want Suricata to fetch rulesets.
  - The `et/open` rulesets serve as an excellent choice for demonstration purposes.
- Next, let's proceed with executing the following command to retrieve and enable the `et/open` rulesets within our Suricata rules.

```
areaeric@htb[/htb]$ sudo suricata-update enable-source et/open
6/7/2023 -- 07:02:08 - <Info> -- Using data-directory /var/lib/suricata.
6/7/2023 -- 07:02:08 - <Info> -- Using Suricata configuration
/etc/suricata/suricata.yaml
6/7/2023 -- 07:02:08 - <Info> -- Using /etc/suricata/rules for Suricata
provided rules.
```

```
6/7/2023 -- 07:02:08 - <Info> -- Found Suricata version 6.0.13 at
/usr/bin/suricata.
6/7/2023 -- 07:02:08 - <Info> -- Creating directory
/var/lib/suricata/update/sources
6/7/2023 -- 07:02:08 - <Info> -- Source et/open enabled
```

- Lastly, let's reissue the `suricata-update` command to load the newly acquired ruleset.

```
areaeric@htb[/htb]$ sudo suricata-update
```

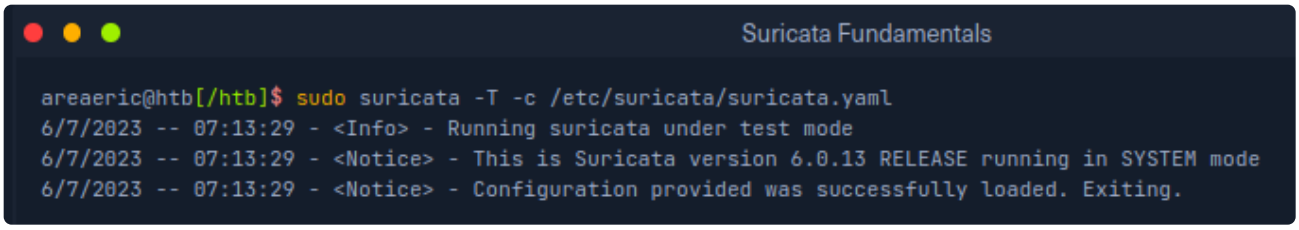
- A Suricata service restart may also be required.

```
areaeric@htb[/htb]$ sudo systemctl restart suricata
```

Validating Suricata's Configuration

- Validation of Suricata's configuration is also an essential part of maintaining the robustness of our IDS/IPS setup.
  - To validate the configuration, we can use the `-T` option provided by the Suricata command.
  - This command runs a test to check if the configuration file is valid and all files referenced in the configuration are accessible.
- Here is how we can do this.

```
areaeric@htb[/htb]$ sudo suricata -T -c /etc/suricata/suricata.yaml
```



Suricata Documentation

- Suricata is an incredibly versatile tool with extensive functionality.

    - We highly recommend exploring Suricata's documentation to gain a deeper understanding of its capabilities.

Suricata Key Features

- Key features that bolster Suricata's effectiveness include:
    - Deep packet inspection and packet capture logging
    - Anomaly detection and Network Security Monitoring
    - Intrusion Detection and Prevention, with a hybrid mode available
    - Lua scripting
    - Geographic IP identification (GeoIP)
    - Full IPv4 and IPv6 support
    - IP reputation
    - File extraction
    - Advanced protocol inspection
    - Multitenancy

Note: Suricata can also be used to detect "non-standard/anomalous" traffic.
- We can leverage strategies outlined in Suricata's Protocol Anomalies Detection page.
- This approach enhances our visibility into unusual or non-compliant behavior within our network, thus augmenting our security posture.

One sentence summary

- Suricata is a tool that can be used as IDS/IPS and more, with the main ability of giving it input (network traffic, which can be live or in terms of packets) which tranforms it to some outputs (E.g. alerts, log files and more) that can be useful for us.

Suricata Rule Development Part 1

Overview

- At its core, a rule in Suricata serves as a directive, instructing the engine to actively watch for certain markers in the network traffic.
    - When such specific markers appear, we will receive a notification.
- Suricata rules are not exclusively focused on the detection of nefarious activities or potentially harmful traffic.

    - In many instances, rules can be designed to furnish network defenders or blue team members with critical insights or contextual data regarding ongoing network activity.
- The specificity or generality of the rules is in our hands.
    - Striking a balance is paramount to, say, identify variations of a certain malware strain while evading false positives.
- The development of these rules often leverages crucial information provided by the infosec communities and threat intelligence.
    - However, it's worth noting that each rule we deploy consumes a portion of the host's CPU and memory resources.
    - Hence, Suricata provides specific guidelines for writing effective rules.

Suricata Rule Anatomy

- A sample Suricata rule can be found below. Let's break it down.

```
action protocol from_ip port -> to_ip port (msg:"Known malicious
behavior, possible X malware infection"; content:"some thing";
content:"some other thing"; sid:10000001; rev:1;)
```

- **Header** (`action protocol from_ip port -> to_ip port` part): The `header` section of a rule encapsulates the intended action of the rule, along with the protocol where the rule is expected to be applied.
    - Additionally, it includes IP addresses, port information, and an arrow indicating traffic directionality.
    - `action` instructs Suricata on what steps to take if the contents match.
        - This could range from generating an alert (`alert`), logging the traffic without an alert (`log`), ignoring the packet (`pass`), dropping the packet in IPS mode (`drop`), or sending TCP RST packets (`reject`).
    - `protocol` can vary, including `tcp`, `udp`, `icmp`, `ip`, `http`, `tls`, `smb`, or `dns`.
    - Traffic directionality is declared using `rule host variables` (such as `$HOME_NET`, `$EXTERNAL_NET`, etc. that we saw inside `suricata.yaml`) and `rule direction`.
        - The direction arrow between the two IP-Port pairs informs Suricata about the traffic flow.
        - Examples:
            - Outbound: `$HOME_NET any -> $EXTERNAL_NET 9090`
            - Inbound: `$EXTERNAL_NET any -> $HOME_NET 8443`

        - Bidirectional: `$EXTERNAL_NET any <> $HOME_NET any`
    - `Rule ports` define the ports at which the traffic for this rule will be evaluated.
        - Examples:
            - `alert tcp $HOME_NET any -> $EXTERNAL_NET 9443`
            - `alert tcp $HOME_NET any -> $EXTERNAL_NET $UNCOMMON_PORTS`
                - $UNCOMMON_PORTS can be defined inside `suricata.yaml`
            - `alert tcp $HOME_NET any -> $EXTERNAL_NET [8443,8080,7001:7002,!8443]`
- **Rule message & content** (`(msg:"Known malicious behavior, possible X malware infection"; content:"some thing"; content:"some other thing";` part): The `rule message & content` section contains the message we wish to be displayed to the analysts or ourselves when an activity we want to be notified about is detected. `content` are the segments of the traffic that we deem essential for such detections.
    - `Rule message (msg)` is an arbitrary text displayed when the rule is triggered.
        - Ideally, the rule messages we create should contain details about malware architecture, family, and action.
        - `flow` identifies the originator and responder.
            - Always remember, when crafting rules, to have the engine monitor "established" tcp sessions.
                - Examples:
                    - `alert tcp any any -> 192.168.1.0/24 22 (msg:"SSH connection attempt"; flow:to_server; sid:1001;)`
                    - `alert udp 10.0.0.0/24 any -> any 53 (msg:"DNS query"; flow:from_client; sid:1002;)`
                    - `alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"Potential HTTP-based attack"; flow:established,to_server; sid:1003;)`
        - `dsize` matches using the payload size of the packet. It relies on TCP segment length, not the total packet length.
            - Example: `alert http any any -> any any (msg:"Large HTTP response"; dsize:>10000; content:"HTTP/1.1 200 OK"; sid:2003;)`
- `Rule content` comprises unique values that help identify specific network traffic or activities.
    - Suricata matches these unique content values in packets for detection.
    - Example: `content:"User-Agent|3a 20|Go-http-client/1.1|0d 0a|Accept-Encoding|3a 20|gzip";`

    - `|0d 0a|`: This represents the hexadecimal representation of the characters "\r\n", which signifies the end of a line in HTTP headers.
    - `|0d 0a|`: This represents the hexadecimal representation of the characters "\r\n", which signifies the end of a line in HTTP headers.
- By using `Rule Buffers`, we don't have to search the entire packet for every content match.
    - This saves time and resources. More details can be found here: https://suricata.readthedocs.io/en/latest/rules/http-keywords.html
    - Example: `alert http any any -> any any (http.accept; content:"image/gif"; sid:1;)`
        - `http.accept`: Sticky buffer to match on the HTTP Accept header.
            - Only contains the header value.
            - The `\r\n` after the header are not part of the buffer.
- `Rule options` act as additional modifiers to aid detection, helping Suricata locate the exact location of contents.
    - `nocase` ensures rules are not bypassed through case changes.
        - Example: `alert tcp any any -> any any (msg:"Detect HTTP traffic with user agent Mozilla"; content:"User-Agent: Mozilla"; nocase; sid:8001;)`
    - `offset` informs Suricata about the start position inside the packet for matching.
        - Example: `alert tcp any any -> any any (msg:"Detect specific protocol command"; content:"|01 02 03|"; offset:0; depth:5; sid:3003;)`
            - This rule triggers an alert when Suricata detects a specific protocol command represented by the byte sequence `|01 02 03|` in the TCP payload.
                - The `offset:0` keyword sets the content match to start from the beginning of the payload, and `depth:5` specifies a length of five bytes to be considered for matching.
    - `distance` tells Suricata to look for the specified content `n` bytes relative to the previous match.
        - Example: `alert tcp any any -> any any (msg:"Detect suspicious URL path"; content:"/admin"; offset:4; depth:10; distance:20; within:50; sid:3001;)`
            - This rule triggers an alert when Suricata detects the string `/admin` in the TCP payload, starting from the fifth byte (`offset:4`) and considering a length of ten bytes (`depth:10`).

        - The `distance:20` keyword specifies that subsequent matches of `/admin` should not occur within the next 20 bytes after a previous match.
        - The `within:50` keyword ensures that the content match occurs within the next 50 bytes after a previous match.
- **Rule metadata** (`sid:10000001; rev:1;` part):
    - `reference` provides us with a lead, a trail that takes us back to the original source of information that inspired the creation of the rule.
    - `sid` (signature ID). The unique quality of this numeric identifier makes it essential for the rule writer to manage and distinguish between rules.
    - `revision` offers insights into the rule's version. It serves as an indicator of the evolution of the rule over time, highlighting modifications and enhancements made.
- Having discussed the crux of Suricata rules, it's now time to shed light on a powerful tool in rule development: `PCRE` or `Pearl Compatible Regular Expression`.
    - Utilizing PCRE can be a game-changer when crafting rules. To employ PCRE, we use the `pcre` statement, which is then followed by a regular expression. \
    - Keep in mind that the PCRE should be encased in leading and trailing forward slashes, with any flags positioned after the last slash.
- Also, note that anchors are positioned after and before the encasing slashes, and certain characters demand escaping with a backslash.
    - A piece of advice from the trenches - steer clear from authoring a rule that relies solely on PCRE.
- Example: `alert http any any -> $HOME_NET any (msg: "ATTACK [PTsecurity] Apache Continuum <= v1.4.2 CMD Injection"; content: "POST"; http_method; content: "/continuum/saveInstallation.action"; offset: 0; depth: 34; http_uri; content: "installation.varValue="; nocase; http_client_body; pcre: !"/^\$?[\sa-z\\_0-9.-]*(\&|$)/iRP"; flow: to_server, established;sid: 10000048; rev: 1;)`
    - Firstly, the rule triggers on HTTP traffic (`alert http`) from any source and destination to any port on the home network (`any any -> $HOME_NET any`).
    - The msg field gives a human-readable description of what the alert is for, namely `ATTACK [PTsecurity] Apache Continuum <= v1.4.2 CMD Injection`.
    - Next, the rule checks for the `POST` string in the HTTP method using the `content` and `http_method` keywords.
        - The rule will match if the HTTP method used is a POST request.
    - The `content` keyword is then used with `http_uri` to match the URI `/continuum/saveInstallation.action`, starting at `offset 0` and going to a `depth` of `34` bytes.

        - This specifies the targeted endpoint, which in this case is the `saveInstallation` action of the Apache Continuum application.
    - Following this, another content keyword searches for `installation.varValue=` in the HTTP client body, case insensitively (`nocase`).
        - This string may be part of the command injection payload that the attacker is trying to deliver.
    - Next, we see a `pcre` keyword, which is used to implement Perl Compatible Regular Expressions.
        - `^` marks the start of the line.
        - `\$?` checks for an optional dollar sign at the start.
        - `[\sa-z\\_0-9.-]*` matches zero or more (`*`) of the characters in the set. The set includes:
            - `\s` a space
            - `a-z` any lowercase letter
            - `\\` a backslash
            - `_` an underscore
            - `0-9` any digit
            - `.` a period
            - `-` a hyphen
            - `(\&|$)` checks for either an ampersand or the end of the line.
            - `/iRP` at the end indicates this is an inverted match (meaning the rule triggers when the match does not occur), case insensitive (`i`), and relative to the buffer position (`RP`).
    - Finally, the `flow` keyword is used to specify that the rule triggers on `established`, inbound traffic towards the server.
- For those who seek to broaden their understanding of Suricata rules and delve deeper into rule development, the following resource serves as a comprehensive guide: https://docs.suricata.io/en/latest/rules/index.html.

IDS/IPS Rule Development Approaches

- When it comes to creating rules for Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), there's an art and a science behind it.
    - It requires a comprehensive understanding of network protocols, malware behaviors, system vulnerabilities, and the threat landscape in general.
- A key strategy that we employ while crafting these rules involves the detection of specific elements within network traffic that are unique to malware.

    - This is often referred to as signature-based detection, and it's the classic approach that most IDS/IPS rely on.
    - Signatures can range from simple patterns in packet payloads, such as the detection of a specific command or a distinctive string associated with a particular malware, to complex patterns that match a series of packets or packet characteristics.
    - Signature-based detection is highly effective when dealing with known threats as it can identify these threats with high precision, however, it struggles to detect novel threats for which no signature exists yet.
- Another approach focuses on identifying specific behaviors that are characteristic to malware.
    - This is typically referred to as anomaly-based or behavior-based detection.
    - For instance, a certain HTTP response size constantly appearing within a threshold, or a specific beaconing interval might be indicative of a malware communication pattern.
    - Other behaviors can include unusually high volumes of data transfers and uncommon ports being used.
    - The advantage of this approach is its ability to potentially identify zero-day attacks or novel threats that would not be detected by signature-based systems.
    - However, it also tends to have higher false-positive rates due to the dynamic nature of network behaviors.
- A third approach that we utilize in crafting IDS/IPS rules is stateful protocol analysis.
    - This technique involves understanding and tracking the state of network protocols and comparing observed behaviors to the expected state transitions of these protocols.
    - By keeping track of the state of each connection, we can identify deviations from expected behavior which might suggest a malicious activity.

Suricata Rule Development Example 1: Detecting PowerShell Empire

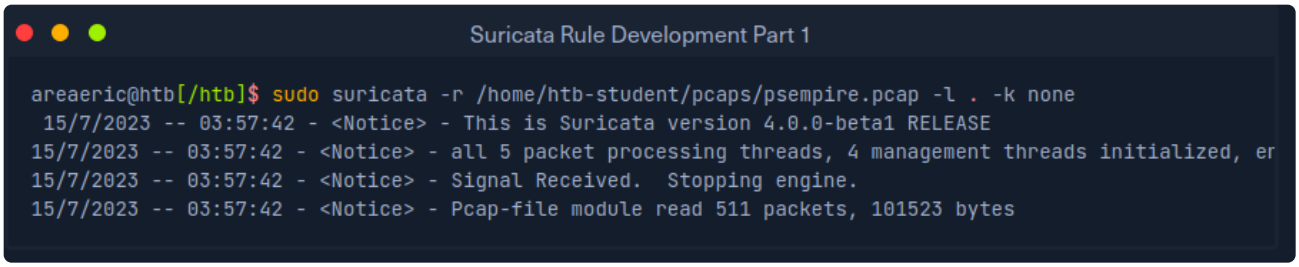
```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ET MALWARE Possible
PowerShell Empire Activity Outbound"; flow:established,to_server;
content:"GET"; http_method; content:"/"; http_uri; depth:1;
pcre:"/^(?:login\/process|admin\/get|news)\.php$/RU";
content:"session="; http_cookie; pcre:"/^(?:[A-Z0-9+/]{4})*(?:[A-Z0-9+/]
{2}==|[A-Z0-9+/]{3}=|[A-Z0-9+/]{4})$/CRi"; content:"Mozilla|2f|5.0|20
28|Windows|20|NT|20|6.1"; http_user_agent; http_start;
content:".php|20|HTTP|2f|1.1|0d 0a|Cookie|3a 20|session="; fast_pattern;
```

```
http_header_names; content:!"Referer"; content:!"Cache";
content:!"Accept"; sid:2027512; rev:1;)
```

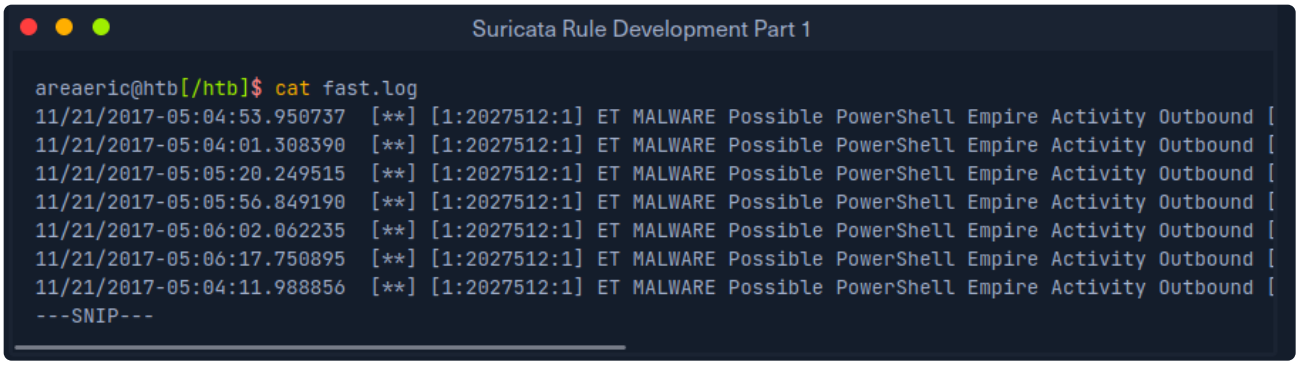
- The Suricata rule above is designed to detect possible outbound activity from PowerShell Empire, a common post-exploitation framework used by attackers.
  - Let's break down the important parts of this rule to understand its workings.
  - `alert`: This is the rule action, indicating that Suricata should generate an alert whenever the conditions specified in the rule options are met.
  - `http`: This is the rule protocol. It specifies that the rule applies to HTTP traffic.
  - `$HOME_NET any -> $EXTERNAL_NET any`: These are the source and destination IP address specifications.
    - The rule will be triggered when HTTP traffic originates from any port (any) on a host within the `$HOME_NET` (internal network) and is destined to any port (any) on a host in the `$EXTERNAL_NET` (external network).
  - `msg:"ET MALWARE Possible PowerShell Empire Activity Outbound"`: This is the message that will be included in the alert to describe what the rule is looking for.
  - `flow:established,to_server`: This specifies the direction of the traffic. The rule is looking for established connections where data is flowing to the server.
  - `content:"GET"; http_method;`: This matches the HTTP GET method in the HTTP request.
  - `content:"/"; http_uri; depth:1;`: This matches the root directory ("/") in the URI.
  - `pcre:"/^(?:login\/process|admin\/get|news)\.php$/RU";`: This Perl Compatible Regular Expression (PCRE) is looking for URIs that end with login/process.php, admin/get.php, or news.php.
    - `PowerShell Empire` is an open-source Command and Control (C2) framework. Its agent can be explored via the following repository.https://github.com/EmpireProject/Empire/blob/master/data/agent/agent.ps1#L78
    - Examine the `psempire.pcap` file which is located in the `/home/htb-student/pcaps` directory of this section's target using Wireshark to pinpoint the related requests.
- - `content:"session="; http_cookie;`: This is looking for the string "session=" in the HTTP cookie.
  - `pcre:"/^(?:[A-Z0-9+/]{4})*(?:[A-Z0-9+/]{2}==|[A-Z0-9+/]{3}=|[A-Z0-9+/]{4})$/CRi";`: This PCRE is checking for base64-encoded data in the Cookie.

        - A plethora of articles examining `PowerShell Empire` exist, here is one noting that the cookies utilized by PowerShell Empire adhere to the Base64 encoding standard. https://www.keysight.com/blogs/tech/nwvs/2021/06/16/empire-c2-networking-into-the-dark-side
    - `content:"Mozilla|2f|5.0|20 28|Windows|20|NT|20|6.1"; http_user_agent; http_start;`: This matches a specific User-Agent string that includes "Mozilla/5.0 (Windows NT 6.1".
        - https://github.com/EmpireProject/Empire/blob/master/data/agent/agent.ps1#L78
    - `content:".php|20|HTTP|2f|1.1|0d 0a|Cookie|3a 20|session="; fast_pattern; http_header_names;`: This matches a pattern in the HTTP headers that starts with ".php HTTP/1.1\r\nCookie: session=".
    - `content:!"Referer"; content:!"Cache"; content:!"Accept";`: These are negative content matches.
        - The rule will only trigger if the HTTP headers do not contain "Referer", "Cache", and "Accept".
- This Suricata rule triggers an alert when it detects an established HTTP GET request from our network to an external network, with a specific pattern in the URI, cookie, and user-agent fields, and excluding certain headers.
- The above rule is already incorporated in the `local.rules` file found in the `/home/htb-student` directory of this section's target.
    - To test it, first, you need to uncomment the rule. Then, execute Suricata on the `psempire.pcap` file, which is located in the `/home/htb-student/pcaps` directory.

```
areaeric@htb[/htb]$ sudo suricata -r /home/htb-
student/pcaps/psempire.pcap -l . -k none
```




```
areaeric@htb[/htb]$ cat fast.log
```



- The `local.rules` file contains another rule for detecting `PowerShell Empire`, located directly below the rule we just examined.
  - Invest some time in scrutinizing both the `psempire.pcap` file using `Wireshark` and this newly found rule to comprehend how it works.

Suricata Rule Development Example 2: Detecting Covenant

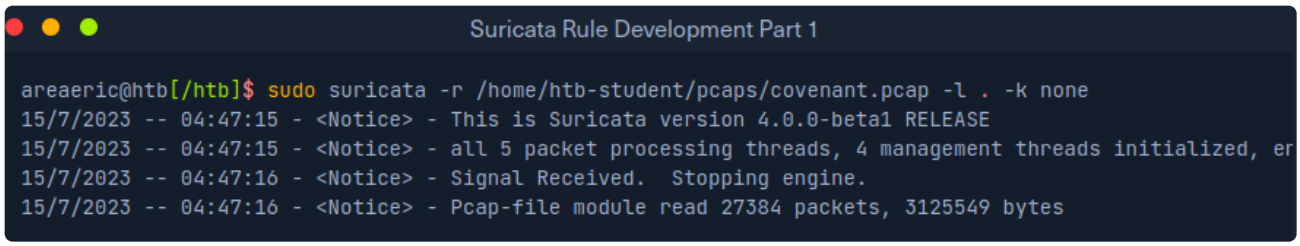
```
alert tcp any any -> $HOME_NET any (msg:"detected by body"; content:"
<title>Hello World!</title>"; detection_filter: track by_src, count 4 ,
seconds 10; priority:1; sid:3000011;)
```

Rule source: Signature-based IDS for Encrypted C2 Traffic Detection - Eduardo Macedo

- The (inefficient) Suricata rule above is designed to detect certain variations of Covenant, another common post-exploitation framework used by attackers.
  - Let's break down the important parts of this rule to understand its workings.
  - `alert`: This is the rule action. When the conditions in the rule options are met, Suricata will generate an alert.
  - `tcp`: This is the rule protocol. The rule applies to TCP traffic.
  - `any any -> $HOME_NET any`: These are the source and destination IP address and port specifications.
    - The rule is watching for TCP traffic that originates from any IP and any port (`any any`) and is destined for any port (`any`) on a host in the `$HOME_NET` (internal network).
  - `content:"<title>Hello World!</title>";`: This instructs Suricata to look for the string `<title>Hello World!</title>` in the TCP payload.
    - `Covenant` is an open-source Command and Control (C2) framework. Its underpinnings can be explored via the following

repository.https://github.com/cobbr/Covenant/blob/master/Covenant/Data/Profiles/DefaultHttpProfile.yaml#L35
    - Examine the `covenant.pcap` file which is located in the `/home/htb-student/pcaps` directory of this section's target using Wireshark to pinpoint the related requests.
- `detection_filter: track by_src, count 4, seconds 10;`: This is a post-detection filter.
    - It specifies that the rule should track the source IP address (`by_src`) and will only trigger an alert if this same detection happens at least 4 times (`count 4`) within a 10-second window (`seconds 10`).
        - Examine the `covenant.pcap` file which is located in the `/home/htb-student/pcaps` directory of this section's target using Wireshark to pinpoint the related requests.
- This Suricata rule is designed to generate a high-priority alert if it detects at least four instances of TCP traffic within ten seconds that contain the string `<title>Hello World!</title>` in the payload, originating from the same source IP and headed towards any host on our internal network.
- The above rule is already incorporated in the `local.rules` file found in the `/home/htb-student` directory of this section's target.
    - To test it, first, you need to uncomment the rule.
    - Then, execute Suricata on the `covenant.pcap` file, which is located in the `/home/htb-student/pcaps` directory.

```
areaeric@htb[/htb]$ sudo suricata -r /home/htb-
student/pcaps/covenant.pcap -l . -k none
```




```
areaeric@htb[/htb]$ cat fast.log
```

- The `local.rules` file contains three (3) other rules for detecting `Covenant`, located directly below the rule we just examined.
  - Invest some time in scrutinizing https://github.com/cobbr/Covenant/blob/master/Covenant/Data/Profiles/DefaultHttpProfile.yaml, the `covenant.pcap` file using `Wireshark`, and these newly found rule to comprehend how they work.
  - These rules may yield false-positive results, and hence for optimal performance, it's advisable to integrate them with other detection rules.

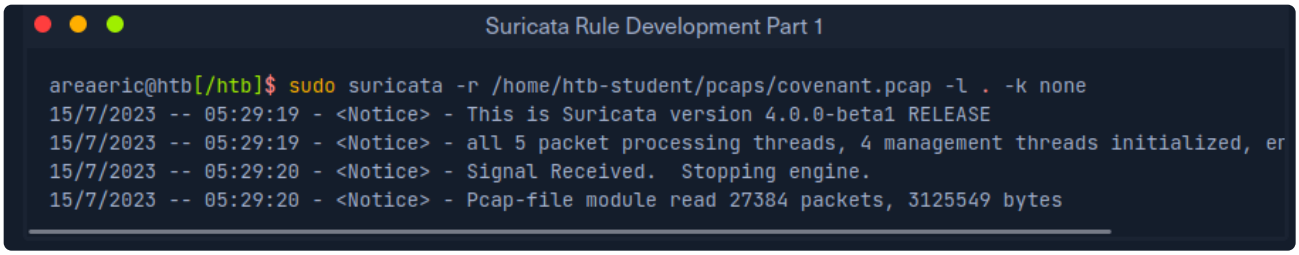
Suricata Rule Development Example 3: Detecting Covenant (Using Analytics)

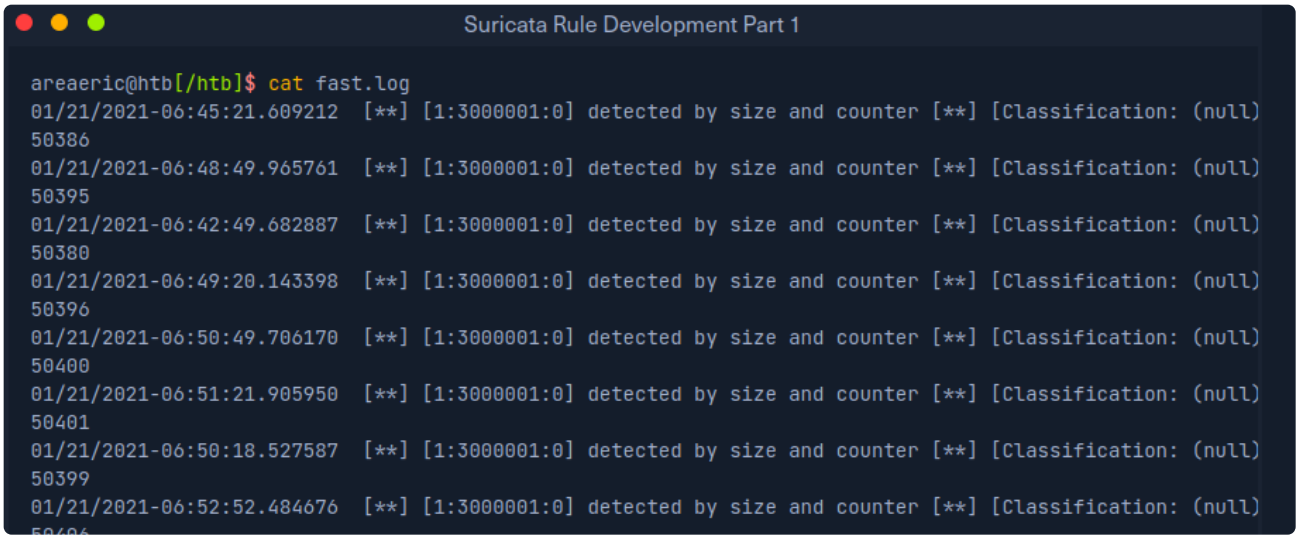
```
alert tcp $HOME_NET any -> any any (msg:"detected by size and counter";
dsize:312; detection_filter: track by_src, count 3 , seconds 10;
priority:1; sid:3000001;)
```

- The `local.rules` file also contains the above rule for detecting `Covenant`.
  - Let's break down the important parts of this rule to understand its workings.
    - `dsize:312;`: This instructs Suricata to look for TCP traffic with a data payload size of exactly 312 bytes.
    - `detection_filter: track by_src, count 3 , seconds 10;`: This is a post-detection filter.
      - It says that the rule should keep track of the source IP address (`by_src`), and it will only trigger an alert if it detects the same rule hit at least 3 times (`count 3`) within a 10-second window (`seconds 10`).
- This Suricata rule is designed to generate a high-priority alert if it detects at least three instances of TCP traffic within ten seconds that each contain a data payload of exactly 312 bytes (TCP payload size), all originating from the same source IP within our network and headed anywhere.

- The above rule is already incorporated in the `local.rules` file found in the `/home/htb-student` directory of this section's target.
  - To test it, first, you need to uncomment the rule.
  - Then, execute Suricata on the `covenant.pcap` file, which is located in the `/home/htb-student/pcaps` directory.

```
areaeric@htb[/htb]$ sudo suricata -r /home/htb-
student/pcaps/covenant.pcap -l . -k none
```





- Invest some time in scrutinizing both the `covenant.pcap` file using `Wireshark` and this newly found rule to comprehend how it works.

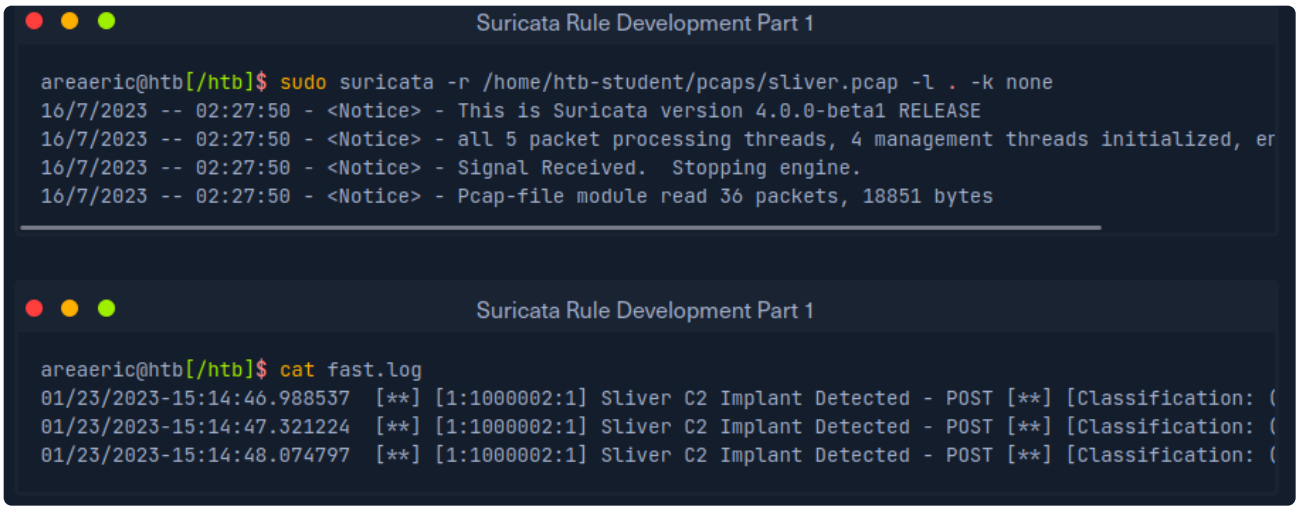
Suricata Rule Development Example 4: Detecting Sliver

```
alert tcp any any -> any any (msg:"Sliver C2 Implant Detected";
content:"POST";
pcre:"/\/(php|api|upload|actions|rest|v1|oauth2callback|authenticate|oau
th2|oauth|auth|database|db|namespaces)(.*?)
((login|signin|api|samples|rpc|index|admin|register|sign-up)\.php)\?[a-
z_]{1,2}=[a-z0-9]{1,10}/i"; sid:1000007; rev:1;)
```

Rule source: https://www.bilibili.com/read/cv19510951/

- The Suricata rule above is designed to detect certain variations of Sliver, yet another common post-exploitation framework used by attackers.
  - Let's break down the important parts of this rule to understand its workings.
- `content:"POST";`: This option instructs Suricata to look for TCP traffic containing the string "POST".
- `pcre:"/\/(php|api|upload|actions|rest|v1|oauth2callback|authenticate|oauth2|oauth|auth|database|db|namespaces)(.*?)((login|signin|api|samples|rpc|index|admin|register|sign-up)\.php)\?[a-z_]{1,2}=[a-z0-9]{1,10}/i";`: This regular expression is utilized to identify specific URI patterns in the traffic. It will match URIs that contain particular directory names followed by file names ending with a PHP extension.
  - `Sliver` is an open-source Command and Control (C2) framework. Its underpinnings can be explored via the following repository.https://github.com/BishopFox/sliver/blob/master/server/configs/http-c2.go#L294
  - Examine the `sliver.pcap` file which is located in the `/home/htb-student/pcaps` directory of this section's target using Wireshark to pinpoint the related requests.

```
areaeric@htb[/htb]$ sudo suricata -r /home/htb-student/pcaps/sliver.pcap -l . -k none
```



- The `local.rules` file contains another rule for detecting `Sliver`, located directly below the rule we just examined.

```
alert tcp any any -> any any (msg:"Sliver C2 Implant Detected - Cookie";
content:"Set-Cookie"; pcre:"/(PHPSESSID|SID|SSID|APISID|csrf-
state|AWSALBCORS)\=[a-z0-9]{32}\;/"; sid:1000003; rev:1;)
```

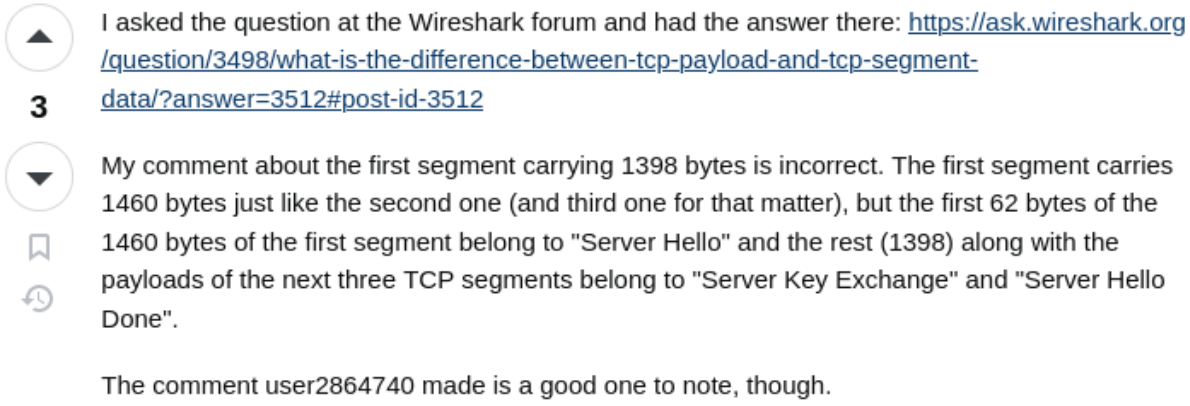
- Let's break down the important parts of this rule to understand its workings.
  - `content:"Set-Cookie";`: This option instructs Suricata to look for TCP traffic containing the string `Set-Cookie`.
  - `pcre:"/(PHPSESSID|SID|SSID|APISID|csrf-state|AWSALBCORS)\=[a-z0-9]{32}\;/";`: This is a regular expression used to identify specific cookie-setting patterns in the traffic.
    - It matches the `Set-Cookie` header when it's setting specific cookie names (PHPSESSID, SID, SSID, APISID, csrf-state, AWSALBCORS) with a value that's a 32-character alphanumeric string.

One sentence summary

- Suricata allows you to develop rules for network traffic to perform action (alert, log, pass, drop or reject) on identified traffic that could be malicious or interesting.

Useful references

- TCP payload vs TCP segment data
  https://stackoverflow.com/questions/50808140/wireshark-what-is-the-difference-between-tcp-payload-and-tcp-segment-data

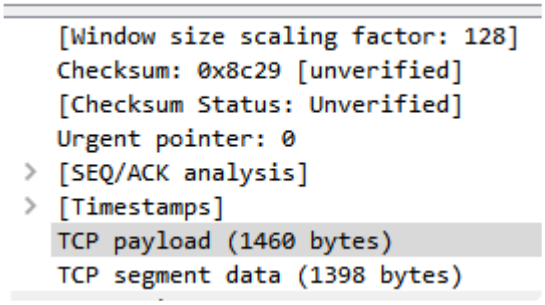


3

I asked the question at the Wireshark forum and had the answer there: https://ask.wireshark.org/question/3498/what-is-the-difference-between-tcp-payload-and-tcp-segment-data/?answer=3512#post-id-3512

My comment about the first segment carrying 1398 bytes is incorrect. The first segment carries 1460 bytes just like the second one (and third one for that matter), but the first 62 bytes of the 1460 bytes of the first segment belong to "Server Hello" and the rest (1398) along with the payloads of the next three TCP segments belong to "Server Key Exchange" and "Server Hello Done".

The comment user2864740 made is a good one to note, though.



-> TCP payload: Everything, the bytes being transmitted.

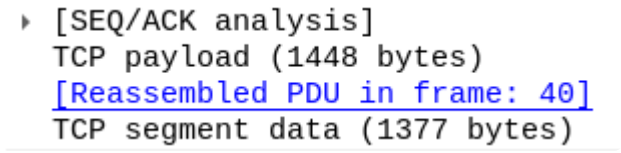
-> TCP segment data: Remaining data of the to be accepted/used by higher-level protocol that is to be used/reassembled in a further packet.

-> E.g. with TLS:

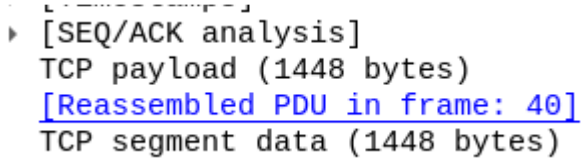
Server sends hello during an TLS, but the certificate is usually very large, so it requires multiple TCP segment/packet for the data to be send through and resassembled.

e.g.

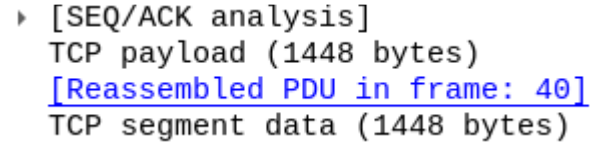
Packet 37



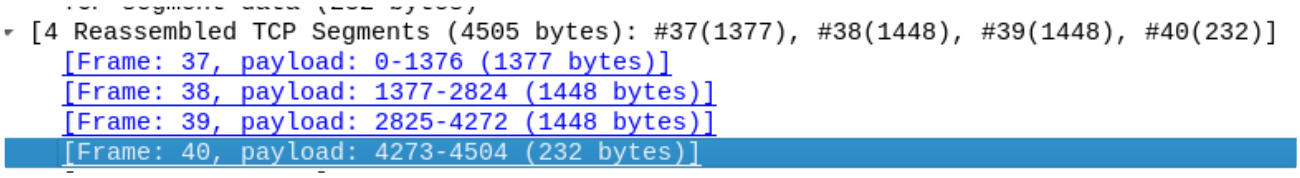
Packet 38



Packet 39



Packet 40



-> And it gets resembled at packet # 40.

Suricata Rule Development Part 2 (Encrypted Traffic)

Summary

- In the ever-evolving landscape of network security, we're often faced with a significant challenge: encrypted traffic.
  - Encrypted traffic can pose significant obstacles when it comes to effectively analyzing traffic and developing reliable Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) rules.
- There are still several aspects we can leverage to detect potential security threats.
  - Specifically, we can turn our attention to the elements within SSL/TLS certificates and the JA3 fingerprint.
- SSL/TLS certificates, exchanged during the initial handshake of an SSL/TLS connection, contain a plethora of details that remain unencrypted.
  - These details can include the issuer, the issue date, the expiry date, and the subject (containing information about who the certificate is for and the domain name).
  - Suspicious or malicious domains might utilize SSL/TLS certificates with anomalous or unique characteristics.
  - Recognizing these anomalies in SSL/TLS certificates can be a stepping stone to crafting effective Suricata rules.
- Further, we can also utilize the JA3 hash — a fingerprinting method that provides a unique representation for each SSL/TLS client.
  - The JA3 hash combines details from the client hello packet during the SSL/TLS handshake, creating a digest that could be unique for specific malware families or suspicious software.
  - Again, these hashes can be a powerful tool in formulating detection rules for encrypted traffic.

Suricata Rule Development Example 5: Detecting Dridex (TLS Encrypted)

```
alert tls $EXTERNAL_NET any -> $HOME_NET any (msg:"ET MALWARE ABUSE.CH
SSL Blacklist Malicious SSL certificate detected (Dridex)";
flow:established,from_server; content:"|16|"; content:"|0b|"; within:8;
byte_test:3,<,1200,0,relative; content:"|03 02 01 02 02 09 00|";
fast_pattern; content:"|30 09 06 03 55 04 06 13 02|"; distance:0;
pcre:"/^[A-Z]{2}/R"; content:"|55 04 07|"; distance:0; content:"|55 04
0a|"; distance:0; pcre:"/^.{2}[A-Z][a-z]{3,}\s(?:[A-Z][a-z]{3,}\s)?(?:
[A-Z](?:[A-Za-z]{0,4}?[A-Z]|(?:\.[A-Za-z]){1,3})|[A-Z]?[a-z]+|[a-z](?:\.
[A-Za-z]){1,3})\.?[01]/Rs"; content:"|55 04 03|"; distance:0;
byte_test:1,>,13,1,relative; content:!"www."; distance:2; within:4;
pcre:"/^.{2}(?P<CN>(?:(?:\d?[A-Z]?|[A-Z]?\d?)(?:[a-z]{3,20}|[a-z]{3,6}
[0-9_][a-z]{3,6})\.){0,2}?(?:\d?[A-Z]?|[A-Z]?\d?)[a-z]{3,}(?:[0-9_-][a-
z]{3,})?\.(?!com|org|net|tv)[a-z]{2,9})[01].*?(?P=CN)[01]/Rs";
```

```
content:!"|2a 86 48 86 f7 0d 01 09 01|"; content:!"GoDaddy";
sid:2023476; rev:5;)
```

- The rule above triggers an alert upon detecting a TLS session from the external network to the home network, where the payload of the session contains specific byte patterns and meets several conditions.
  - These patterns and conditions correspond to SSL certificates that have been linked to certain variations of the Dridex trojan, as referenced by the SSL blacklist on `abuse.ch`.
- No need to understand the rule in its entirety, but let's break down the important parts of it.
  - `content:"|16|"; content:"|0b|"; within:8;`: The rule looks for the hex values 16 and 0b within the first 8 bytes of the payload.
    - These represent the handshake message (0x16) and the certificate type (0x0b) in the TLS record.
  - `content:"|03 02 01 02 02 09 00|"; fast_pattern;`: The rule looks for this specific pattern of bytes in the packet, which may be characteristic of the certificates used by Dridex.
  - `content:"|30 09 06 03 55 04 06 13 02|"; distance:0; pcre:"/^[A-Z]{2}/R"`;: This checks for the 'countryName' field in the certificate's subject.
    - The content match here corresponds to an ASN.1 sequence specifying an attribute type and value for 'countryName' (OID 2.5.4.6).
    - The following PCRE checks that the value for 'countryName' begins with two uppercase letters, which is a standard format for country codes.
  - `content:"|55 04 07|"; distance:0;`: This checks for the 'localityName' field in the certificate's subject (OID 2.5.4.7).
  - `content:"|55 04 0a|"; distance:0;`: This checks for the `organizationName` field in the certificate's subject (OID 2.5.4.10).
  - `content:"|55 04 03|"; distance:0; byte_test:1,>,13,1,relative;`: This checks for the `commonName` field in the certificate's subject (OID 2.5.4.3).
    - The following byte_test checks that the length of the `commonName` field is more than 13.
  - Also give this very interesting resource on Dridex SSL certificates a look.
- The mentioned OIDs (Object Identifiers) are part of the X.509 standard for PKI and are used to uniquely identify the types of fields contained within certificates.
- The above rule is already incorporated in the `local.rules` file found in the `/home/htb-student` directory of this section's target.

- To test it, first, you need to uncomment the rule. Then, execute Suricata on the `dridex.pcap` file, which is located in the `/home/htb-student/pcaps` directory.

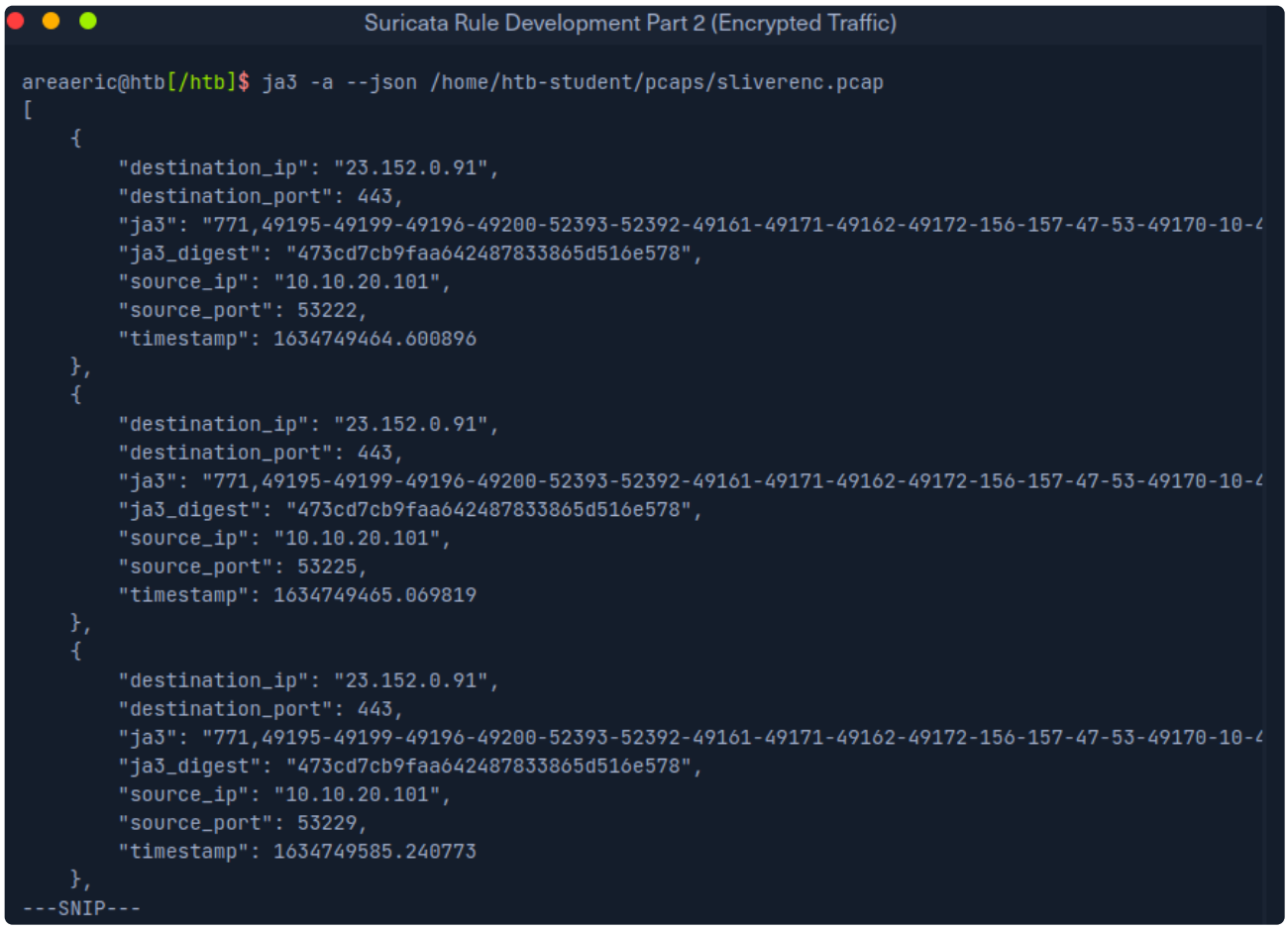
```
areaeric@htb[/htb]$ sudo suricata -r /home/htb-student/pcaps/dridex.pcap
-l . -k none
```

Suricata Rule Development Example 6: Detecting Sliver (TLS Encrypted)

```
alert tls any any -> any any (msg:"Sliver C2 SSL"; ja3.hash;
content:"473cd7cb9faa642487833865d516e578"; sid:1002; rev:1;)
```

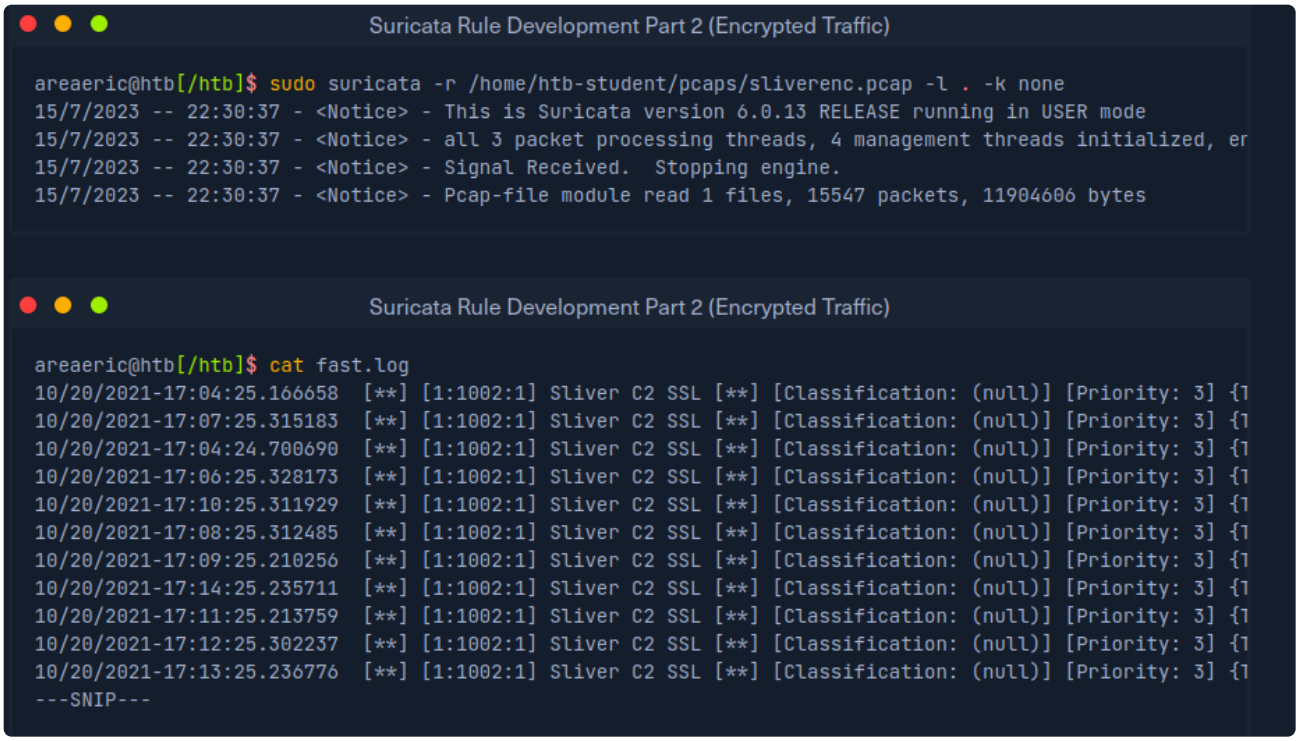
- The Suricata rule above is designed to detect certain variations of Sliver whenever it identifies a TLS connection with a specific JA3 hash.
- A PCAP file named `sliverenc.pcap` containing encrypted Sliver traffic is located in the `/home/htb-student/pcaps` directory of this section's target.
- The JA3 hash can be calculated as follows.

```
areaeric@htb[/htb]$ ja3 -a --json /home/htb-student/pcaps/sliverenc.pcap
```



- The above rule is already incorporated in the `local.rules` file found in the `/home/htb-student` directory of this section's target.
  - To test it, first, you need to uncomment the rule. Then, execute Suricata on the `sliverenc.pcap` file, which is located in the `/home/htb-student/pcaps` directory.

```
areaeric@htb[/htb]$ sudo suricata -r /home/htb-
student/pcaps/sliverenc.pcap -l . -k none
```



One sentence summary

- The beginning of TLS messages (e.g TLS handshake) can allow us to identify potential malware behaviour (weird content in certificate, JA3 hash) as these messages are not encrypted so we could analyse it.

Snort

Snort Fundamentals

Research on unfamiliar terminology

DAQ: Known as Data Acquisition Library. Abstraction that allows you to communicate with data sources passively or actively using "modules" (function). E.g. PCAP is an DAQ module.
-> When enabled allows you to use that module and interact with that data source (active or passive depending on its functionality)

Packet logger: Software that make copies of network packet.

lua: A general-purpose programming language

syslog / unified2: File format that snort can use for log files

Shared Object rules: Rules written in Shared Object language like C, allowing for more features offered than just the regular Snort rule language.

Plugins: In snort, thehre are two types of plugins, detection plugins and preprocessors. They are abstractions that provide extra functionality during the processes.

Modules: At a high level, controls how Snort processes and handles network traffic.

Unresolved Questions?

- peg? Seached online and official documentation but still quite unsure. I think it means counts of observed/performed action but I'm not too sure.

Overview

- Snort is an open-source tool, which serves as both an Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), but can also function as a packet logger or sniffer, akin to Suricata.
  - By thoroughly inspecting all network traffic, Snort has the capability to identify and log all activity within that traffic, providing a comprehensive view of the situation and detailed logs of all application layer transactions.
  - We require specific rule sets to instruct Snort on how to perform its inspection and what exactly it needs to identify.
  - Snort was created to operate efficiently on both general-purpose and custom hardware.

Snort Operation Modes

- Snort typically operates in the following modes:
  - Inline IDS/IPS
  - Passive IDS
  - Network-based IDS
  - Host-based IDS (however, Snort is not ideally a host-based IDS. We would recommend opting for more specialized tools for this.)

According to Snort's documentation:

- "With certain DAQ modules, Snort is able to utilize two different modes of operation: `passive` and `inline`.
  - `Passive mode` gives Snort the ability to observe and detect traffic on a network interface, but it prevents outright blocking of traffic.
  - `Inline mode` on the other hand, does give Snort the ability to block traffic if a particular packet warrants such an event.
- Snort will infer the particular mode of operation based on the options used at the command line.
  - For example, reading from a pcap file with the `-r` option or listening on an interface with `-i` will cause Snort to run in passive mode by default.
  - If the DAQ supports inline, however, then users can specify the `-Q` flag to run Snort inline.
- One DAQ module that supports inline mode is `afpacket`, which is a module that gives Snort access to packets received on Linux network devices."

Snort Architecture

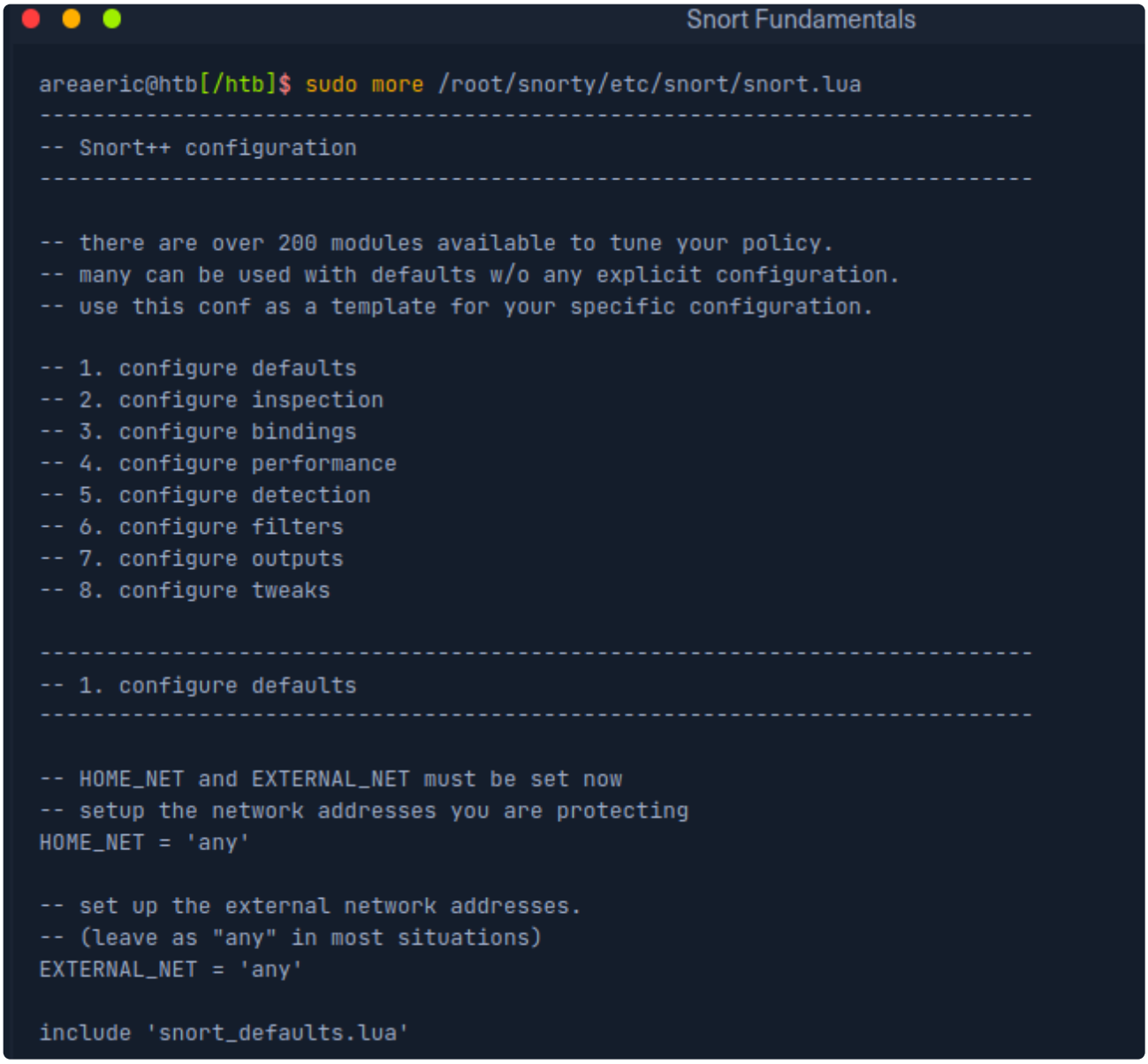
- In order for Snort to transition from a simple packet sniffer to a robust IDS, several key components were added: `Preprocessor`, `Detection Engine`, `Logging and Alerting System`, and various `Output modules`.
  - The packet sniffer (which includes the Packet Decoder) extracts network traffic, recognizing the structure of each packet.
    - The raw packets that are collected are subsequently forwarded to the `Preprocessors`.
  - `Preprocessors` within Snort identify the type or behaviour of the forwarded packets.
    - Snort has an array of `Preprocessor` plugins, like the HTTP plugin that distinguishes HTTP-related packets or the `port_scan Preprocessor` which identifies potential port scanning attempts based on predefined protocols, types of scans, and thresholds.
    - After the `Preprocessors` have completed their task, information is passed to the `Detection Engine`.
    - The configuration of these `Preprocessors` can be found within the Snort configuration file, `snort.lua`.
  - The `Detection Engine` compares each packet with a predefined set of Snort rules.
    - If a match is found, information is forwarded to the `Logging and Alerting System`.

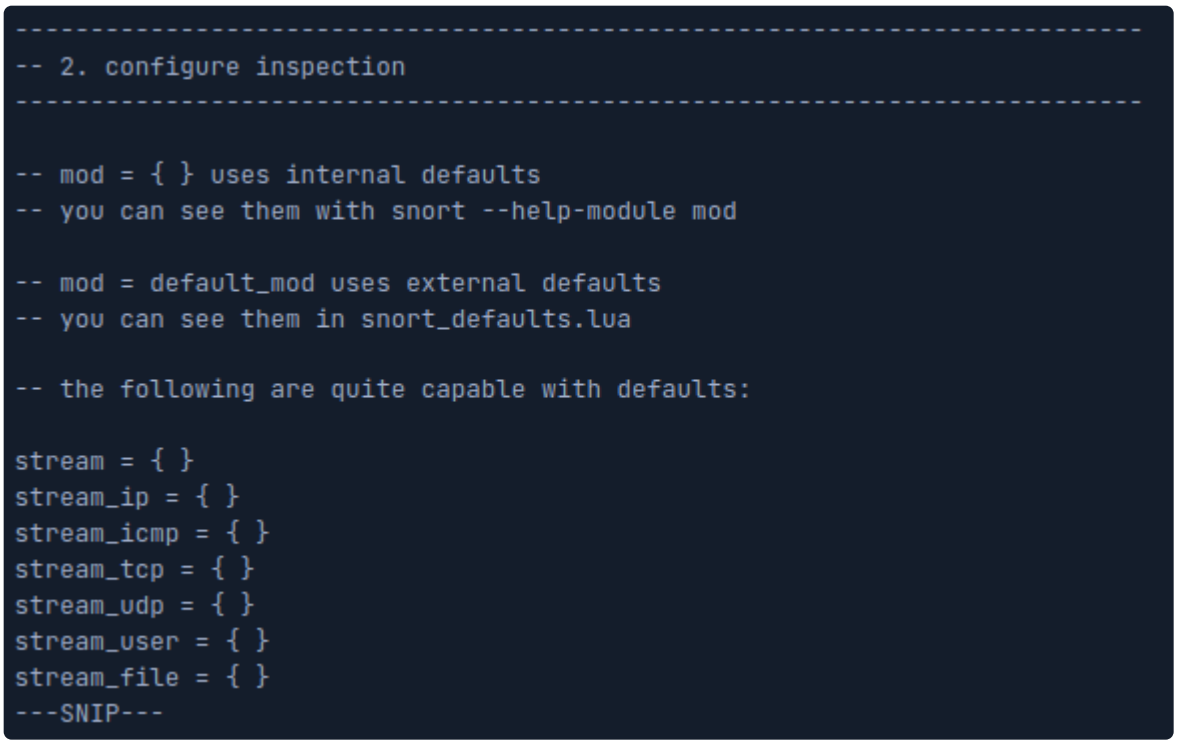
- The `Logging and Alerting System` and `Output modules` are in charge of recording or triggering alerts as determined by each rule action.
  - Logs are generally stored in `syslog` or `unified2` formats or directly in a database.
  - The `Output modules` are configured within the Snort configuration file, `snort.lua`.

Snort Configuration & Validating Snort's Configuration

- Snort offers a wide range of configuration options, and fortunately, the open-source Snort 3 provides users with pre-configured files to facilitate a quick start.
  - These default configuration files, namely `snort.lua` and `snort_defaults.lua`, serve as the foundation for setting up Snort and getting it operational in no time.
  - They provide a standard configuration framework for Snort users.
- The `snort.lua` file serves as the principal configuration file for Snort.
  - This file contains the following sections:
  - Network variables
  - Decoder configuration
  - Base detection engine configuration
  - Dynamic library configuration
  - Preprocessor configuration
  - Output plugin configuration
  - Rule set customization
  - Preprocessor and decoder rule set customization
  - Shared object rule set customization
- Let's browse the `snort.lua` file residing in this section's target as follows.

```
areaeric@htb[/htb]$ sudo more /root/snorty/etc/snort/snort.lua
```


Snort Fundamentals
areaeric@htb[/htb]$ sudo more /root/snorty/etc/snort/snort.lua
-- Snort++ configuration
-- there are over 200 modules available to tune your policy.
-- many can be used with defaults w/o any explicit configuration.
-- use this conf as a template for your specific configuration.
-- 1. configure defaults
-- 2. configure inspection
-- 3. configure bindings
-- 4. configure performance
-- 5. configure detection
-- 6. configure filters
-- 7. configure outputs
-- 8. configure tweaks
-- 1. configure defaults
-- HOME_NET and EXTERNAL_NET must be set now
-- setup the network addresses you are protecting
HOME_NET = 'any'
-- set up the external network addresses.
-- (leave as "any" in most situations)
EXTERNAL_NET = 'any'
include 'snort_defaults.lua'


-- 2. configure inspection
-- mod = { } uses internal defaults
-- you can see them with snort --help-module mod
-- mod = default_mod uses external defaults
-- you can see them in snort_defaults.lua
-- the following are quite capable with defaults:
stream = { }
stream_ip = { }
stream_icmp = { }
stream_tcp = { }
stream_udp = { }
stream_user = { }
stream_file = { }
---SNIP---

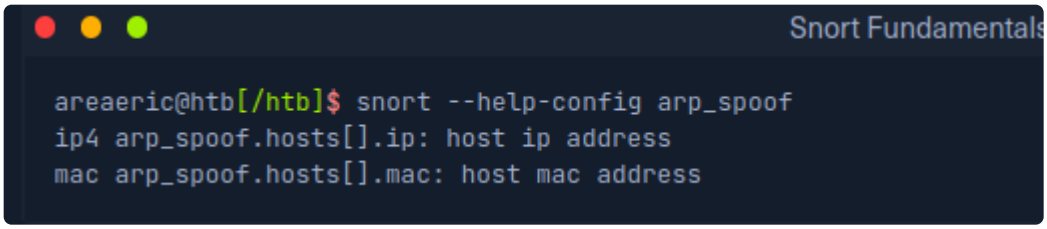
- Enabling and fine-tuning Snort `modules` is a significant aspect of the configuration process.
  - To explore the complete list and get a brief description of all Snort 3 modules, you can use the following command.

```
areaeric@htb[/htb]$ snort --help-modules
```

```
areaeric@htb[/htb]$ snort --help-modules
ack (ips_option): rule option to match on TCP ack numbers
active (basic): configure responses
address_space_selector (policy_selector): configure traffic processing based on address space
alert_csv (logger): output event in csv format
alert_fast (logger): output event with brief text format
alert_full (logger): output event with full packet dump
alert_json (logger): output event in json format
alert_syslog (logger): output event to syslog
alert_talos (logger): output event in Talos alert format
alert_unixsock (logger): output event over unix socket
alerts (basic): configure alerts
appid (inspector): application and service identification
appids (ips_option): detection option for application ids
arp (codec): support for address resolution protocol
arp_spoof (inspector): detect ARP attacks and anomalies
---SNIP---
```

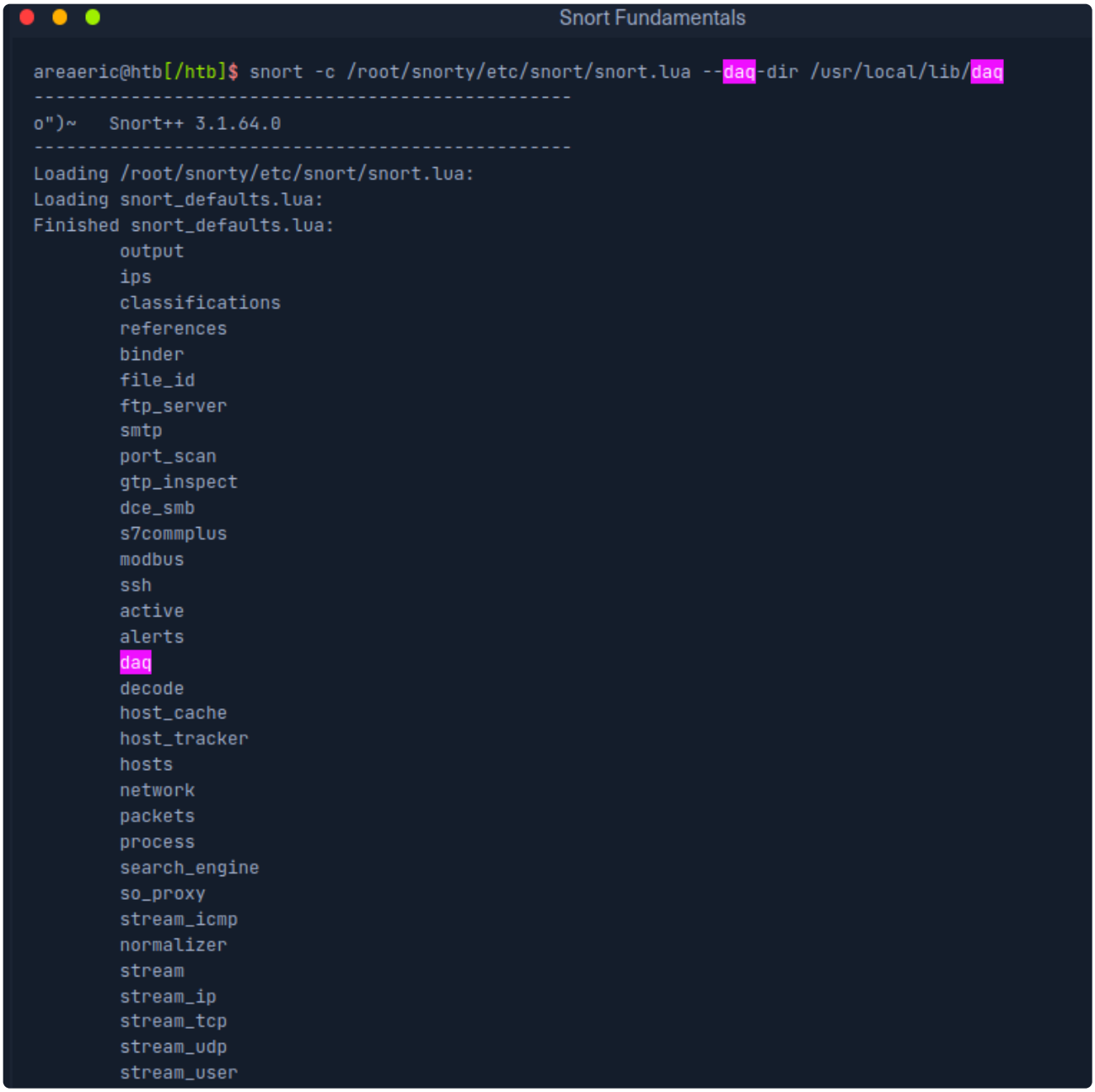
- These modules are enabled and configured within the `snort.lua` configuration file as Lua table literals.
  - If a module is initialized as an empty table, it implies that it is utilizing its predefined "default" settings.
  - To view these default settings, you can utilize the following command.

```
areaeric@htb[/htb]$ snort --help-config arp_spoof
```



- Passing (and validating) configuration files to Snort can be done as follows.

```
areaeric@htb[/htb]$ snort -c /root/snorty/etc/snort/snort.lua --daq-dir
/usr/local/lib/daq
```


Snort Fundamentals
areaeric@htb[/htb]$ snort -c /root/snorty/etc/snort/snort.lua --daq-dir /usr/local/lib/daq
o")~   Snort++ 3.1.64.0
Loading /root/snorty/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
output
ips
classifications
references
binder
file_id
ftp_server
smtp
port_scan
gtp_inspect
dce_smb
s7commplus
modbus
ssh
active
alerts
daq
decode
host_cache
host_tracker
hosts
network
packets
process
search_engine
so_proxy
stream_icmp
normalizer
stream
stream_ip
stream_tcp
stream_udp
stream_user

```
Finished /root/snorty/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
---------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     208       0     208    /root/snorty/etc/snort/snort.lua
---------------------------------------------------
rule counts
       total rules loaded: 208
              text rules: 208
            option chains: 208
            chain headers: 1
---------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      208     208
                    total:      208     208
---------------------------------------------------
fast pattern groups
                to_server: 1
                to_client: 1
---------------------------------------------------
search engine (ac_bnfa)
                instances: 2
                 patterns: 416
            pattern chars: 2508
               num states: 1778
         num match states: 370
             memory scale: KB
             total memory: 68.5879
           pattern memory: 18.6973
        match list memory: 27.3281
        transition memory: 22.3125
appid: MaxRss diff: 3084
appid: patterns loaded: 300
---------------------------------------------------
pcap DAQ configured to passive.

Snort successfully validated the configuration (with 0 warnings).
o")~   Snort exiting
```

Note: `--daq-dir /usr/local/lib/daq` is not required to pass and validate a configuration file. It is added so that we can replicate the command in this section's target.

- Since we mentioned `DAQ`, Snort 3 should know where to find the appropriate `LibDAQ`.
  - `LibDAQ` is the "Data Acquisition Library", and at a high-level, it's an abstraction layer used by `modules` to communicate with both hardware and software network data sources.
- We highly recommend taking the time to read the comments inside the `snort.lua` file as they provide valuable insights.

Snort Inputs

- To observe Snort in action, the easiest method is to execute it against a packet capture file.
  - By providing the name of the pcap file as an argument to the `-r` option in the command line, Snort will process the file accordingly.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -r /home/htb-student/pcaps/icmp.pcap
```

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --daq-dir /usr/local/lib/daq -r /home/htb-studen
[sudo] password for htb-student:
--------------------------------------------------
o")~   Snort++ 3.1.64.0
--------------------------------------------------
Loading /root/snorty/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
---SNIP---
Finished /root/snorty/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     208       0     208    /root/snorty/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 208
              text rules: 208
            option chains: 208
            chain headers: 1
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      208     208
                    total:      208     208
--------------------------------------------------
fast pattern groups
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 2
                 patterns: 416
            pattern chars: 2508
               num states: 1778
         num match states: 370
             memory scale: KB
             total memory: 68.5879
           pattern memory: 18.6973
         match list memory: 27.3281
         transition memory: 22.3125
```

```
pcap DAQ configured to read-file.
Commencing packet processing
++ [0] /home/htb-student/pcaps/icmp.pcap
-- [0] /home/htb-student/pcaps/icmp.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                   pcaps: 1
                received: 8
                analyzed: 8
                   allow: 8
                rx_bytes: 592
--------------------------------------------------
codec
                   total: 8            (100.000%)
                     eth: 8            (100.000%)
                   icmp4: 8            (100.000%)
                    ipv4: 8            (100.000%)
--------------------------------------------------
Module Statistics
--------------------------------------------------
appid
                 packets: 8
        processed_packets: 8
           total_sessions: 1
--------------------------------------------------
binder
               new_flows: 1
                inspects: 1
--------------------------------------------------
detection
                analyzed: 8
--------------------------------------------------
port_scan
                 packets: 8
                trackers: 2
--------------------------------------------------
stream
                   flows: 1
--------------------------------------------------
stream_icmp
                sessions: 1
                     max: 1
--------------------------------------------------
Summary Statistics
--------------------------------------------------
timing
                  runtime: 00:00:00
                  seconds: 0.033229
                pkts/sec: 241
o")~   Snort exiting
```

- Snort also has the capability to listen on active network interfaces.
  - To specify this behavior, you can utilize the `-i` option followed by the names of the interfaces on which Snort should run.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -i ens160
```

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --daq-dir /usr/local/lib/daq -i ens160
--------------------------------------------------
o")~   Snort++ 3.1.64.0
--------------------------------------------------
Loading /root/snorty/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
---SNIP---
Finished /root/snorty/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     208       0     208    /root/snorty/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 208
              text rules: 208
            option chains: 208
            chain headers: 1
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      208     208
                    total:      208     208
--------------------------------------------------
fast pattern groups
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 2
                 patterns: 416
            pattern chars: 2508
               num states: 1778
         num match states: 370
             memory scale: KB
             total memory: 68.5879
           pattern memory: 18.6973
        match list memory: 27.3281
        transition memory: 22.3125
appid: MaxRss diff: 2820
```

```
pcap DAQ configured to passive.
Commencing packet processing
++ [0] ens160
^C** caught int signal
== stopping
-- [0] ens160
---------------------------------------------------
Packet Statistics
---------------------------------------------------
daq
                received: 33
                analyzed: 33
                   allow: 33
                    idle: 9
                rx_bytes: 2756
---------------------------------------------------
codec
                   total: 33          (100.000%)
                discards: 2           (  6.061%)
                     arp: 13          ( 39.394%)
                     eth: 33          (100.000%)
                   icmp4: 12          ( 36.364%)
                   icmp6: 3           (  9.091%)
                    ipv4: 17          ( 51.515%)
                    ipv6: 3           (  9.091%)
                     tcp: 5           ( 15.152%)
---------------------------------------------------
Module Statistics
---------------------------------------------------
appid
                 packets: 18
        processed_packets: 16
          ignored_packets: 2
           total_sessions: 3
---------------------------------------------------
arp_spoof
                 packets: 13
---------------------------------------------------
binder
              raw_packets: 15
                new_flows: 3
                 inspects: 18
---------------------------------------------------
```

```
detection
                analyzed: 33
-------------------------------------------------
port_scan
                 packets: 20
                trackers: 8
-------------------------------------------------
stream
                   flows: 3
-------------------------------------------------
stream_icmp
                sessions: 2
                     max: 2
                 created: 2
                released: 2
-------------------------------------------------
stream_tcp
                sessions: 1
                     max: 1
                 created: 1
                released: 1
             instantiated: 1
                   setups: 1
            data_trackers: 1
              segs_queued: 1
            segs_released: 1
                max_segs: 1
               max_bytes: 64
-------------------------------------------------
tcp
        bad_tcp4_checksum: 2
-------------------------------------------------
Appid Statistics
-------------------------------------------------
detected apps and services
             Application: Services   Clients    Users      Payloads   Misc       Referred
                 unknown: 1          0          0          0          0          0
-------------------------------------------------
Summary Statistics
-------------------------------------------------
process
                 signals: 1
```

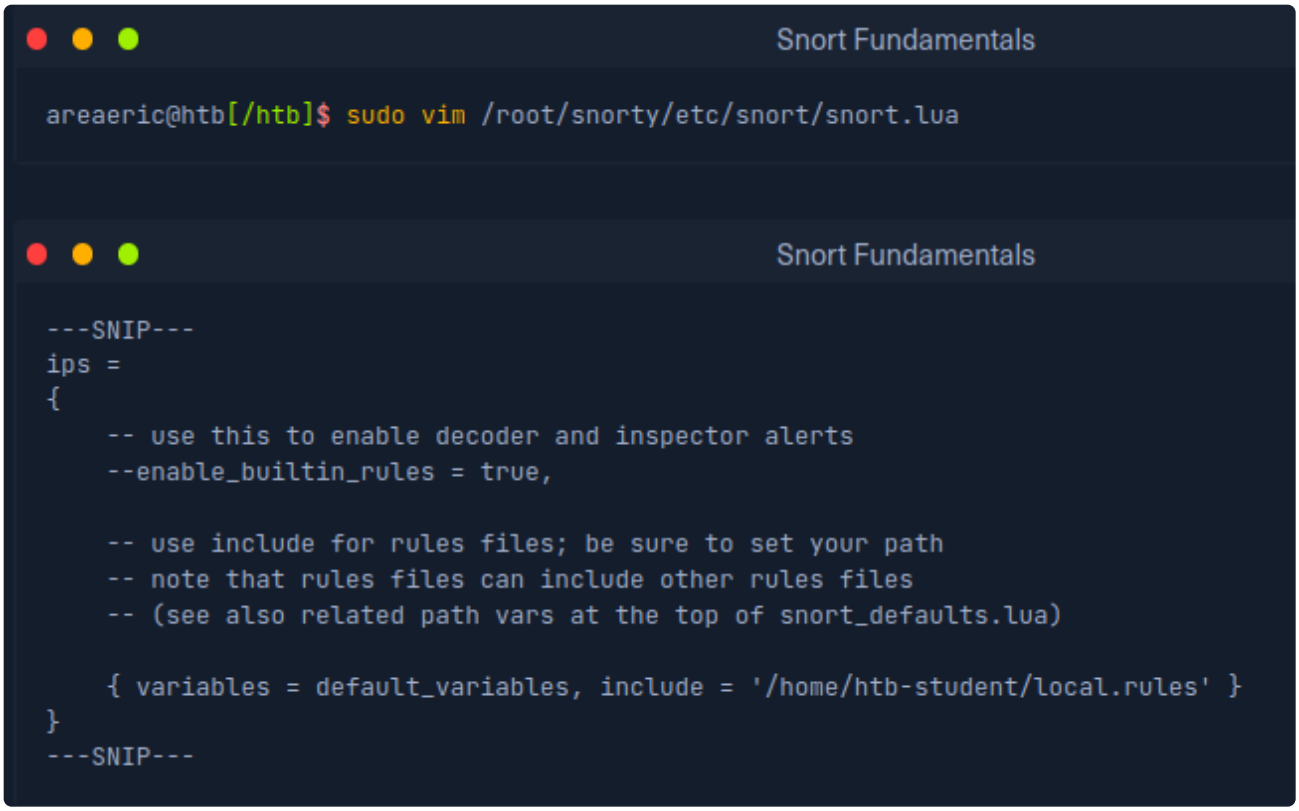
```
timing
                  runtime: 00:00:25
                  seconds: 25.182182
                 pkts/sec: 1
o")~   Snort exiting
```

Snort Rules

- Snort rules, which resemble Suricata rules, are composed of a `rule header` and `rule options`.
  - Even though Snort rules share similarities with Suricata rules, we strongly suggest studying Snort rule writing from the following resources: https://docs.snort.org/, https://docs.suricata.io/en/latest/rules/differences-from-snort.html.

    - The most recent Snort rules can be obtained from the Snort website or the Emerging Threats website.
- In Snort deployments, we have flexibility in managing rules.
    - It's possible to place rules (for example, `local.rules` residing at `/home/htb-student`) directly within the `snort.lua` configuration file using the `ips` module as follows.

```
areaeric@htb[/htb]$ sudo vim /root/snorty/etc/snort/snort.lua
```

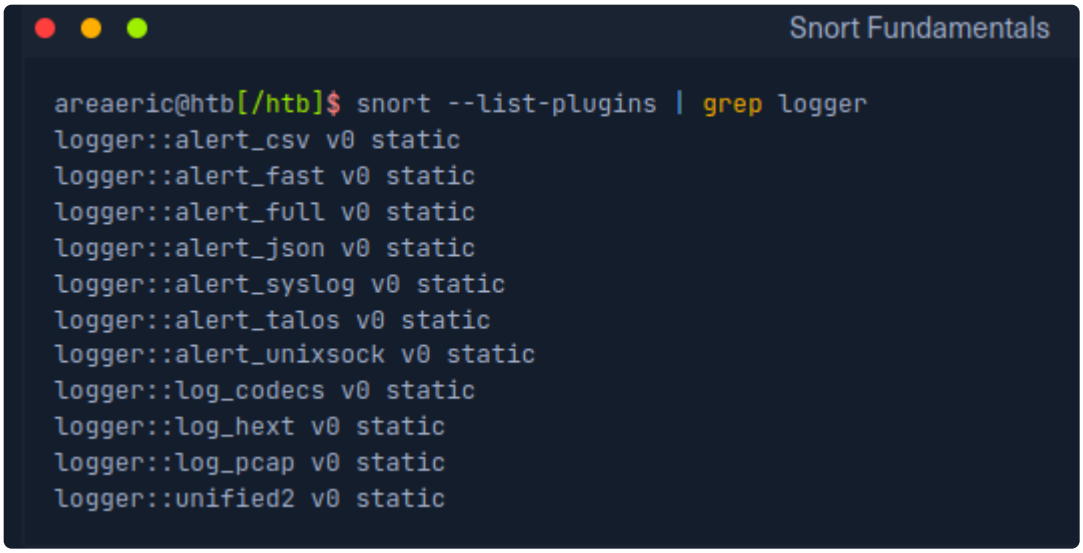


- Then, the "included" rules will be automatically loaded.
- In our Snort deployment, we have an alternative approach to incorporate rules directly from the command line.
    - We can pass either a single rules file or a path to a directory containing rules files directly to Snort.
    - This can be achieved using two options:
- For a single rules file, we can use the `-R` option followed by the path to the rules file.
    - This allows us to specify a specific rules file to be utilized by Snort.
- To include an entire directory of rules files, we can use the `--rule-path` option followed by the path to the rules directory.
    - This enables us to provide Snort with a directory containing multiple rules files.

Snort Outputs

- In our Snort deployment, we may encounter a significant amount of data.
  - To provide a summary of the core output types, let's explore the key aspects:
  - `Basic Statistics`: Upon shutdown, Snort generates various counts based on the configuration and processed traffic. This includes:
    - `Packet Statistics`: It includes information from the DAQ and decoders, such as the number of received packets and UDP packets.
    - `Module Statistics`: Each module keeps track of activity through peg counts, indicating the frequency of observed or performed actions.
      - Examples include the count of processed HTTP GET requests and trimmed TCP reset packets.
    - `File Statistics`: This section provides a breakdown of file types, bytes, and signatures.
    - `Summary Statistics`: It encompasses the total runtime for packet processing, packets per second, and, if configured, profiling data.
  - `Alerts`: When rules are configured, it is necessary to enable alerting (using the `-A` option) to view the details of detection events.
    - There are multiple types of alert outputs available, including:
      - `-A cmg`: This option combines `-A fast -d -e` and displays alert information along with packet headers and payload.
      - `-A u2`: This option is equivalent to `-A unified2` and logs events and triggering packets in a binary file, which can be used for post-processing with other tools.
      - `-A csv`: This option outputs fields in comma-separated value format, providing customization options and facilitating pcap analysis.
    - To discover the available alert types, we can execute the following command.

```
areaeric@htb[/htb]$ snort --list-plugins | grep logger
```



- `Performance Statistics`: Beyond the aforementioned outputs, additional data can be obtained.
  - By configuring the `perf_monitor` module, we can capture a customizable set of `peg` counts during runtime.
  - This data can be fed into an external program to monitor Snort's activity without interrupting its operation.
  - The `profiler` module allows tracking of time and space usage by modules and rules.
  - This information is valuable for optimizing system performance.
  - The profiler output appears under `Summary Statistics` during shutdown.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -r /home/htb-student/pcaps/icmp.pcap -A cmg
```

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --daq-dir /usr/local/lib/daq -r /home/htb-student/
--------------------------------------------------
o")~   Snort++ 3.1.64.0
--------------------------------------------------
Loading /root/snorty/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
--SNIP---
Finished /root/snorty/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading /home/htb-student/local.rules:
Finished /home/htb-student/local.rules:
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     209       0     209    /root/snorty/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 209
              text rules: 209
            option chains: 209
            chain headers: 2
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any       0       0       1       0
   total       0       0       1       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      208     208
                    total:      208     208
--------------------------------------------------
fast pattern groups
                to_server: 1
                to_client: 1
```

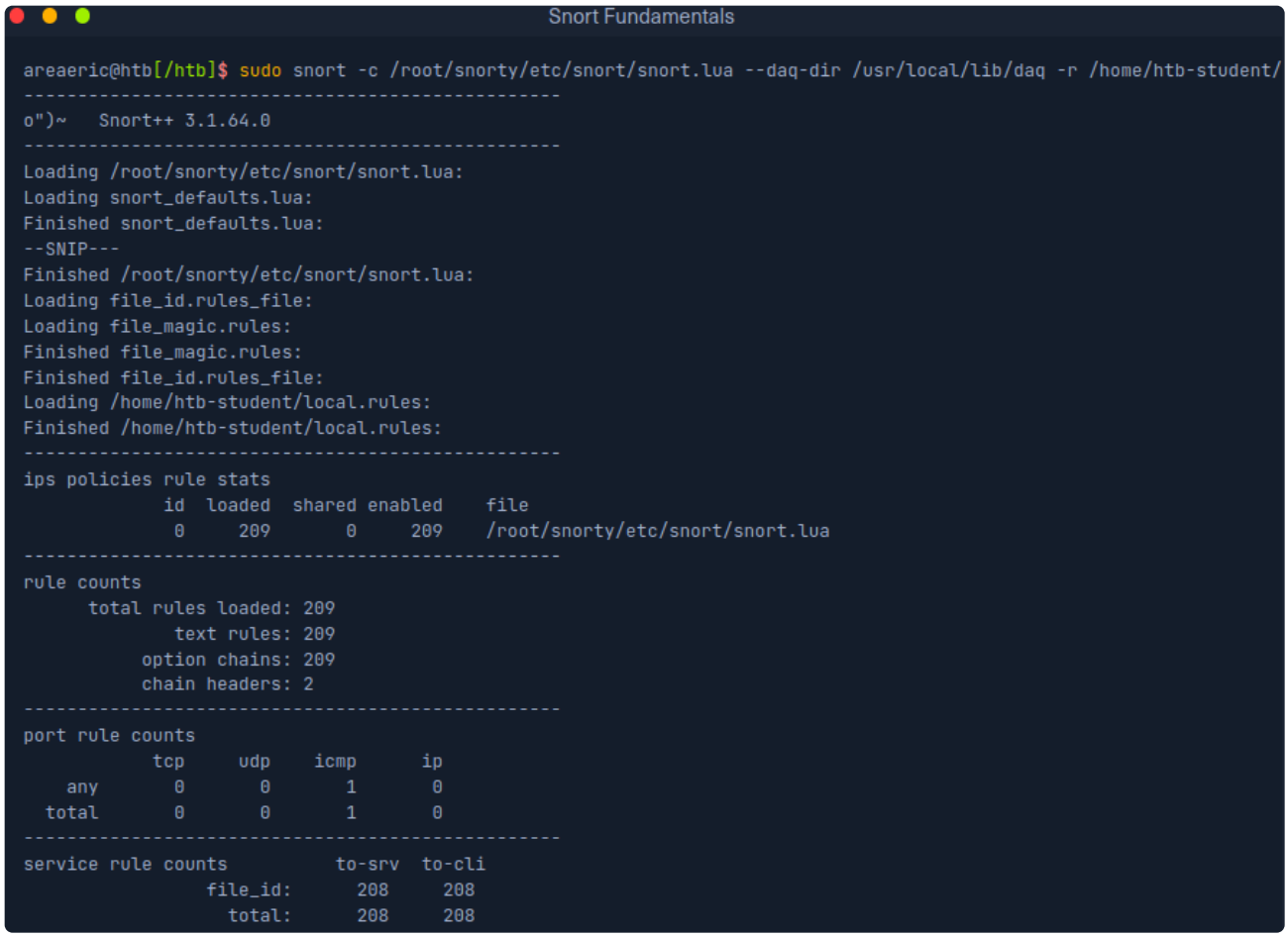
```
fast pattern groups
               to_server: 1
               to_client: 1
-------------------------------------------------
search engine (ac_bnfa)
                instances: 2
                 patterns: 416
           pattern chars: 2508
              num states: 1778
         num match states: 370
             memory scale: KB
             total memory: 68.5879
          pattern memory: 18.6973
        match list memory: 27.3281
        transition memory: 22.3125
appid: MaxRss diff: 3024
appid: patterns loaded: 300
-------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
++ [0] /home/htb-student/pcaps/icmp.pcap
06/19-08:45:56.838904 [**] [1:1000001:1] "ICMP test" [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 1
00:0C:29:34:0B:DE -> 00:50:56:E0:14:49 type:0x800 len:0x4A
192.168.158.139 -> 174.137.42.77 ICMP TTL:128 TOS:0x0 ID:55107 IpLen:20 DgmLen:60
Type:8  Code:0  ID:512   Seq:8448  ECHO

snort.raw[32]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -
61 62 63 64 65 66 67 68  69 6A 6B 6C 6D 6E 6F 70  abcdefgh ijklmnop
71 72 73 74 75 76 77 61  62 63 64 65 66 67 68 69  qrstuvwa bcdefghi
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -

06/19-08:45:57.055699 [**] [1:1000001:1] "ICMP test" [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 1
00:50:56:E0:14:49 -> 00:0C:29:34:0B:DE type:0x800 len:0x4A
174.137.42.77 -> 192.168.158.139 ICMP TTL:128 TOS:0x0 ID:30433 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:8448  ECHO REPLY
```

```
-- [0] /home/htb-student/pcaps/icmp.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                   pcaps: 1
                received: 8
                analyzed: 8
                   allow: 8
                rx_bytes: 592
--------------------------------------------------
codec
                   total: 8            (100.000%)
                     eth: 8            (100.000%)
                   icmp4: 8            (100.000%)
                    ipv4: 8            (100.000%)
--------------------------------------------------
Module Statistics
--------------------------------------------------
appid
                 packets: 8
        processed_packets: 8
           total_sessions: 1
--------------------------------------------------
binder
               new_flows: 1
                inspects: 1
--------------------------------------------------
detection
                analyzed: 8
              hard_evals: 8
                  alerts: 8
             total_alerts: 8
                  logged: 8
--------------------------------------------------
```

```
-------------------------------------------------
port_scan
                  packets: 8
                 trackers: 2
-------------------------------------------------
search_engine
         qualified_events: 8
-------------------------------------------------
stream
                    flows: 1
-------------------------------------------------
stream_icmp
                 sessions: 1
                      max: 1
                  created: 1
                 released: 1
-------------------------------------------------
Summary Statistics
-------------------------------------------------
timing
                  runtime: 00:00:00
                  seconds: 0.042473
                 pkts/sec: 188
o")~   Snort exiting
```

- The same command but using a `.rules` files that may not be "included" in `snort.lua` is the following.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -r /home/htb-student/pcaps/icmp.pcap -R
/home/htb-student/local.rules -A cmg
```



Snort Key Features

- Key features that bolster Snort's effectiveness include:
  - Deep packet inspection, packet capture, and logging
  - Intrusion detection
  - Network Security Monitoring
  - Anomaly detection
  - Support for multiple tenants
  - Both IPv6 and IPv4 are supported

One sentence summary

- Snort is like Suricata but with some differences in rule set and development, they both serve as an network monitoring tool and can serve as an IDS/IPS.

Snort Rule Development

Overview

- A Snort rule, at its core, is a powerful tool that we use to identify and flag potential malicious activity in network traffic.
- As already discussed, Snort rules resemble Suricata rules.
  - They are composed of a `rule header` and `rule options`.
  - Even though Snort rules share similarities with Suricata rules, we strongly suggest studying Snort rule writing from the following resources: https://docs.snort.org/, https://docs.suricata.io/en/latest/rules/differences-from-snort.html.
- Let's move forward and explore some examples of crafting Snort rules to counter real-world malware threats.

Snort Rule Development Example 1: Detecting Ursnif (Inefficiently)

```
alert tcp any any -> any any (msg:"Possible Ursnif C2 Activity";
flow:established,to_server; content:"/images/", depth 12; content:"_2F";
content:"_2B"; content:"User-Agent|3a 20|Mozilla/4.0 (compatible|3b|
MSIE 8.0|3b| Windows NT"; content:!"Accept"; content:!"Cookie|3a|";
content:!"Referer|3a|"; sid:1000002; rev:1;)
```

- The Snort rule above is designed to detect certain variations of Ursnif malware.
  - The rule is inefficient since it misses HTTP sticky buffers.
  - Let's break down the important parts of this rule to understand its workings.
  - `flow:established,to_server;` ensures that this rule only matches established TCP connections where data is flowing from the client to the server.
  - `content:"/images/", depth 12;` instructs Snort to look for the string `/images/` within the first `12` bytes of the packet payload.
  - `content:"_2F";` and `content:"_2B";` direct Snort to search for the strings `_2F` and `_2B` anywhere in the payload.
  - `content:"User-Agent|3a 20|Mozilla/4.0 (compatible|3b| MSIE 8.0|3b| Windows NT";` is looking for a specific `User-Agent`. The `|3a 20|` and `|3b|` in the rule are hexadecimal representations of the `:` and `;` characters respectively.
  - `content:!"Accept"; content:!"Cookie|3a|"; content:!"Referer|3a|";` look for the absence of certain standard HTTP headers, such as `Accept`, `Cookie:` and `Referer:`. The `!` indicates negation.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -R /home/htb-student/local.rules -r
/home/htb-student/pcaps/ursnif.pcap -A cmg
```

```
++ [0] /home/htb-student/pcaps/ursnif.pcap
07/21-19:27:47.161230 [**] [1:1000002:1] "Possible Ursnif C2 Activity" [**] [Priority: 0] {TCP} 10.10.16
00:1F:E2:10:8B:C9 -> 00:0C:29:C9:67:00 type:0x800 len:0x18C
10.10.10.104:49191 -> 192.42.116.41:80 TCP TTL:128 TOS:0x0 ID:20640 IpLen:20 DgmLen:382 DF
***AP*** Seq: 0x12E06BB  Ack: 0xE061E225  Win: 0x4029  TcpLen: 20

snort.raw[342]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - -
47 45 54 20 2F 69 6D 61  67 65 73 2F 70 32 52 55  GET /ima ges/p2RU
52 68 5F 32 2F 42 6B 32  76 72 31 4F 59 52 46 31  Rh_2/Bk2 vr1OYRF1
57 47 75 35 6E 67 5F 32  46 73 66 73 2F 57 4F 57  WGu5ng_2 Fsfs/WOW
72 47 54 45 54 79 4B 2F  37 4D 7A 4D 5F 32 42 6E  rGTETyK/ 7MzM_2Bn
47 5A 51 52 32 6A 73 67  50 2F 73 5F 32 42 70 53  GZQR2jsg P/s_2BpS
34 31 4B 37 41 67 2F 4F  75 4A 6A 51 66 41 61 63  41K7Ag/O uJjQfAac
32 64 2F 76 6D 46 5F 32  46 31 4B 42 50 72 4B 5F  2d/vmF_2 F1KBPrK_
32 2F 46 36 36 38 32 64  67 64 69 61 47 31 7A 75  2/F6682d gdiaG1zu
56 43 7A 37 47 68 64 2F  62 4C 37 36 57 66 35 71  VCz7Ghd/ bL76Wf5q
64 71 77 56 35 76 7A 52  2F 32 65 31 41 38 79 42  dqwV5vzR /2e1A8yB
49 64 6B 6D 49 5F 32 42  2F 6F 67 79 7A 4E 55 57  IdkmI_2B /ogyzNUW
33 47 72 2F 67 4B 42 5A  57 58 78 2E 67 69 66 20  3Gr/gKBZ WXx.gif
48 54 54 50 2F 31 2E 31  0D 0A 55 73 65 72 2D 41  HTTP/1.1 ..User-A
67 65 6E 74 3A 20 4D 6F  7A 69 6C 6C 61 2F 34 2E  gent: Mo zilla/4.
30 20 28 63 6F 6D 70 61  74 69 62 6C 65 3B 20 4D  0 (compa tible; M
53 49 45 20 38 2E 30 3B  20 57 69 6E 64 6F 77 73  SIE 8.0;  Windows
20 4E 54 20 36 2E 31 29  0D 0A 48 6F 73 74 3A 20   NT 6.1) ..Host:
62 6C 75 65 77 61 74 65  72 73 74 6F 6E 65 2E 72  bluewate rstone.r
75 0D 0A 43 6F 6E 6E 65  63 74 69 6F 6E 3A 20 4B  u..Conne ction: K
65 65 70 2D 41 6C 69 76  65 0D 0A 43 61 63 68 65  eep-Aliv e..Cache
2D 43 6F 6E 74 72 6F 6C  3A 20 6E 6F 2D 63 61 63  -Control : no-cac
68 65 0D 0A 0D 0A                                 he....
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - -
```

Snort Rule Development Example 2: Detecting Cerber

```
alert udp $HOME_NET any -> $EXTERNAL_NET any (msg:"Possible Cerber
Check-in"; dsize:9; content:"hi", depth 2, fast_pattern; pcre:"/^[af0-9]
{7}$/R"; detection_filter:track by_src, count 1, seconds 60;
sid:2816763; rev:4;)
```

- The Snort rule above is designed to detect certain variations of Cerber malware.
  - Let's break down the important parts of this rule to understand its workings.
  - `$HOME_NET any -> $EXTERNAL_NET any` signifies the rule applies to any `UDP` traffic going from any port in the home network to any port on external networks.
  - `dsize:9;`: This is a condition that restricts the rule to UDP datagrams that have a payload data size of exactly `9` bytes.
  - `content:"hi", depth 2, fast_pattern;`: This checks the payload's first `2` bytes for the string `hi`.
    - The `fast_pattern` modifier makes the pattern matcher search for this pattern before any others in the rule, optimizing the rule's performance.
  - `pcre:"/^[af0-9]{7}$/R";`: This stands for Perl Compatible Regular Expressions.

- The rule is looking for seven hexadecimal characters (from the set `a-f` and `0-9`) starting at the beginning of the payload (after the `hi`), and this should be the complete payload (signified by the `$` end anchor).

- `detection_filter:track by_src, count 1, seconds 60;`: The `detection_filter` keyword in Snort rule language is used to suppress alerts unless a certain threshold of matched events occurs within a specified time frame.
  - In this rule, the filter is set to track by source IP (`by_src`), with a count of `1` and within a time frame of `60` seconds.
  - This means that the rule will trigger an alert only if it matches more than one event (specifically, more than count events which is `1` here) from the same source IP address within `60` seconds.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -R /home/htb-student/local.rules -r
/home/htb-student/pcaps/cerber.pcap -A cmg
```

```
07/22-02:17:56.486795 [**] [1:2816763:4] "Possible Cerber Check-in" [**] [Priority: 0] {UDP} 10.0.2.15:1
08:00:27:A9:8C:97 -> 52:54:00:12:35:02 type:0x800 len:0x3C
10.0.2.15:1046 -> 31.184.234.2:6892 UDP TTL:128 TOS:0x0 ID:84 IpLen:20 DgmLen:37
Len: 9

snort.raw[9]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -
68 69 30 30 37 32 38 39  35                      hi007289 5
---SNIP---
-- [0] /home/htb-student/pcaps/cerber.pcap
-----------------------------------------------
```

Snort Rule Development Example 3: Detecting Patchwork

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"OISF TROJAN Targeted
AutoIt FileStealer/Downloader CnC Beacon"; flow:established,to_server;
http_method; content:"POST"; http_uri; content:".php?profile=";
http_client_body; content:"ddager=", depth 7; http_client_body;
content:"&r1=", distance 0; http_header; content:!"Accept"; http_header;
content:!"Referer|3a|"; sid:10000006; rev:1;)
```

- The Snort rule above is designed to detect certain variations of malware used by the Patchwork APT. Please notice the use of HTTP sticky buffers in this rule. Let's break down the important parts of this rule to understand its workings.
  - `flow:established,to_server;`: This keyword is used to specify the direction of the traffic we are interested in.

    - In this case, we're looking at established connections where the traffic is going from the client to the server.
- `http_method; content:"POST";`: We are looking for HTTP traffic where the method used is `POST`.
- `http_uri; content:".php?profile=";`: This specifies that we're looking for HTTP URIs that contain the string `.php?profile=`.
- `http_client_body; content:"ddager=", depth 7;`: We're examining the body of the HTTP request.
    - Specifically, we're looking for the string `ddager=` within the first `7` bytes of the body.
- `http_client_body; content:"&r1=", distance 0;`: We're still examining the body of the HTTP request, but now we're looking for the string `&r1=` immediately following the previous content match.
- `http_header; content:!"Accept"; http_header; content:!"Referer|3a|";`: These conditions are looking for the absence of the `Accept` and `Referer` HTTP headers.
    - The `!` before the content means `not`, so we're looking for situations where these headers are not present.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -R /home/htb-student/local.rules -r
/home/htb-student/pcaps/patchwork.pcap -A cmg
```

```
http_inspect.http_header[167]:
- - - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
43 6F 6E 6E 65 63 74 69  6F 6E 3A 20 4B 65 65 70  Connecti on: Keep
2D 41 6C 69 76 65 0D 0A  43 6F 6E 74 65 6E 74 2D  -Alive.. Content-
54 79 70 65 3A 20 61 70  70 6C 69 63 61 74 69 6F  Type: ap plicatio
6E 2F 78 2D 77 77 77 2D  66 6F 72 6D 2D 75 72 6C  n/x-www- form-url
65 6E 63 6F 64 65 64 0D  0A 55 73 65 72 2D 41 67  encoded. .User-Ag
65 6E 74 3A 20 4D 6F 7A  69 6C 6C 61 2F 35 2E 30  ent: Moz illa/5.0
20 46 69 72 65 66 6F 78  20 28 4C 69 6B 65 20 53   Firefox  (Like S
61 66 61 72 69 2F 57 65  62 6B 69 74 29 0D 0A 43  afari/We bkit)..C
6F 6E 74 65 6E 74 2D 4C  65 6E 67 74 68 3A 20 36  ontent-L ength: 6
34 0D 0A 48 6F 73 74 3A  20 32 31 32 2E 31 32 39  4..Host:  212.129
2E 31 33 2E 31 31 30                             .13.110
- - - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

http_inspect.http_client_body[64]:
- - - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
64 64 61 67 65 72 3D 30  26 72 31 3D 56 30 6C 4F  ddager=0 &r1=V0lO
58 7A 63 3D 26 72 32 3D  57 44 59 30 26 72 33 3D  Xzc=&r2= WDY0&r3=
4D 53 34 78 26 72 34 3D  4D 41 3D 3D 26 72 35 3D  MS4x&r4= MA==&r5=
49 43 41 3D 26 72 36 3D  56 48 4A 31 5A 51 3D 3D  ICA=&r6= VHJ1ZQ==
- - - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

06/01-19:25:09.059391 [**] [1:10000006:1] "OISF TROJAN Targeted AutoIt FileStealer/Downloader CnC Beacon
0
```

Snort Rule Development Example 4: Detecting Patchwork (SSL)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Patchwork SSL Cert
Detected"; flow:established,from_server; content:"|55 04 03|";
content:"|08|toigetgf", distance 1, within 9; classtype:trojan-activity;
sid:10000008; rev:1;)
```

- The Snort rule above is designed to detect certain variations of malware used by the Patchwork APT.
  - Let's break down the important parts of this rule to understand its workings.
  - `flow:established,from_server;`: This keyword pair signifies that we're interested in observing `established` flows of traffic originating from the server.
  - `content:"|55 04 03|";`: This rule is looking for the specific hex values `55 04 03` within the payload of the packet.
    - These hex values represent the ASN.1 (Abstract Syntax Notation One) tag for the "common name" field in an X.509 certificate, which is often used in SSL/TLS certificates to denote the domain name that the certificate applies to.
  - `content:"|08|toigetgf", distance 1, within 9;`: Following the common name field, this rule looks for the string `toigetgf`.
    - The distance `1` means that Snort should start looking for the string `toigetgf` `1` byte after the end of the previous content match.
    - The within `9` sets an upper limit on how far into the packet's payload Snort should search, starting from the beginning of this content field.

```
areaeric@htb[/htb]$ sudo snort -c /root/snorty/etc/snort/snort.lua --
daq-dir /usr/local/lib/daq -R /home/htb-student/local.rules -r
/home/htb-student/pcaps/patchwork.pcap -A cmg
```

```
06/01-19:25:44.259156 [**] [1:10000008:1] "Patchwork SSL Cert Detected" [**] [Classification: A Network

ssl.stream_tcp[807]:
- - - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
16 03 01 00 51 02 00 00  4D 03 01 BA 21 1C 95 C6  ....Q... M...!...
8A 83 F3 C8 31 16 EF 25  32 C4 63 7E 82 B0 7D D0  ....1..% 2.c~..}.
01 EF 6C 7B DC A3 CD 53  97 18 62 20 C6 FE 9D B0  ..l{...S ..b ....
B0 F4 9B 9A 6E 1D 6B 74  64 4D E4 CC 30 2E 05 B8  ....n.kt dM..0...
30 1A 34 D0 30 94 5B FA  AB 64 27 09 00 2F 00 00  0.4.0.[. .d'../..
05 FF 01 00 01 00 16 03  01 02 C3 0B 00 02 BF 00  ........ ........
02 BC 00 02 B9 30 82 02  B5 30 82 01 9D A0 03 02  .....0.. .0......
01 02 02 08 44 0E 87 29  65 41 6D 2A 30 0D 06 09  ....D..) eAm*0...
2A 86 48 86 F7 0D 01 01  0B 05 00 30 13 31 11 30  *.H..... ...0.1.0
0F 06 03 55 04 03 0C 08  74 6F 69 67 65 74 67 66  ...U.... toigetgf
30 1E 17 0D 31 34 31 31  31 38 30 36 30 38 33 38  0...1411 18060838
5A 17 0D 32 34 31 31 31  35 30 36 30 38 33 38 5A  Z..24111 5060838Z
30 13 31 11 30 0F 06 03  55 04 03 0C 08 74 6F 69  0.1.0... U....toi
67 65 74 67 66 30 82 01  22 30 0D 06 09 2A 86 48  getgf0.. "0...*.H
86 F7 0D 01 01 01 05 00  03 82 01 0F 00 30 82 01  ........ .....0..
0A 02 82 01 01 00 C6 B9  1B 97 5C 6E DA 23 4C 02  ........ ..\n.#L.
EC A6 A8 09 56 FA 85 5E  35 75 A8 BB 63 B5 81 30  ....V..^ 5u..c..0
90 91 45 D7 19 36 0B 20  DF 70 37 0E 91 05 FB 86  ..E..6.  .p7.....
22 E8 56 3D A5 89 BA 13  01 60 DF 43 A6 F0 05 7B  ".V=.... .`.C...{
5A 04 7F 53 14 80 C1 64  EA 9C 09 98 A2 B8 99 EA  Z..S...d ........
91 26 52 81 62 D3 BB CE  A2 4E E7 BB 97 C9 19 D2  .&R.b... .N......
EF 61 8A A5 50 9A D7 6B  9F 9D 54 7B AE E2 6F 53  .a..P..k ..T{..oS
BB 7A B4 D2 93 06 73 96  CD 04 19 55 D3 7A DA 34  .z....s. ...U.z.4
8F 05 2D 2E 98 7F 6C 9E  0B C8 41 A2 49 BA FB CC  ..-...l. ..A.I...
A4 20 BD 8A E5 18 27 88  BB 87 F9 F6 F3 56 8F 73  . ....'. .....V.s
D6 BA 92 29 F9 F0 A6 AB  F5 FD 5F E0 92 C6 96 2D  ...)....  .._....-
41 80 FA 0B 4C C9 9B AE  2D 69 F7 9D B5 4B 14 81  A...L... -i...K..
AD F8 71 6F 2B A8 59 66  6E FD B5 8B 3B 14 09 F7  ..qo+.Yf n...;...
B8 FC 20 EF 7D A0 D5 40  D6 66 BB 65 B6 FC 92 3A  .. .}..@ .f.e...:
71 F5 BA 5B F1 07 A5 FD  E3 11 F2 A9 51 6C 16 8F  q..[.... ....Ql..
```

One sentence summary

- Snort rule development is like Suricata with different syntax and looking at network packets of malware can help develop effective signatures.

Zeek

Zeek Fundamentals

Summary

- Zeek, as defined by its developers, is an open-source network traffic analyzer.
  - It is typically employed to scrutinize every bit of traffic on a network, digging deep for any signs of suspicious or malicious activity.
  - But, Zeek isn't limited to just that.
  - It can also be a handy tool for troubleshooting network issues and conducting a variety of measurements within a network.

    - Once we deploy Zeek, our defensive cybersecurity teams (or blue teams) can immediately tap into a wealth of log files, which offer an elevated view of all types of network activity.
    - Specifically, these logs contain detailed records of every connection made over the network, along with transcripts of application-layer activities such as DNS queries and responses, HTTP sessions, and more.
    - But Zeek's capabilities extend beyond mere logging. It comes bundled with a suite of functions for analyzing and detecting network activities.
- What sets Zeek apart is its highly capable scripting language, which allows users to craft Zeek scripts that are functionally similar to Suricata rules.
    - This powerful feature enables Zeek to be fully customizable and extendable, letting blue team members develop their own logic and strategies for network analysis and intrusion detection.
- Considering Zeek's functionality, combined with its ability to run on standard hardware and the powerful scripting language, it becomes clear that we're not looking at another signature-based IDS.
    - Instead, Zeek is a platform that can facilitate semantic misuse detection, anomaly detection, and behavioral analysis..

Zeek's Operation Modes

- Zeek operates in the following modes:
    - Fully passive traffic analysis
    - libpcap interface for packet capture
    - Real-time and offline (e.g., PCAP-based) analysis
    - Cluster support for large-scale deployments

Zeek's Architecture

- Zeek's architecture comprises two primary components: the `event engine` (or core) and the `script interpreter`.
- The `event engine` takes an incoming packet stream and transforms it into a series of high-level `events`.
    - In Zeek's context, these `events` describe network activity in policy-neutral terms, meaning they inform us of what's happening, but they don't offer an interpretation or evaluation of it.
    - For instance, an HTTP request will be transformed into an `http_request` event.

    - While this event provides all the details of the request, it doesn't offer any judgement or analysis, such as whether a port corresponds to a port known to be used by malware.
- Such interpretation or analysis is provided by Zeek's `script interpreter`, which executes a set of event handlers written in Zeek's scripting language (Zeek scripts).
    - These scripts express the site's security policy, defining actions to be taken upon the detection of certain events.
- Events generated by Zeek's core are queued in an orderly manner, awaiting their turn to be processed on a first-come, first-served basis.
    - Most of Zeek's events are defined in `.bif` files located in the `/scripts/base/bif/plugins/` directory. For a more comprehensive list of events, refer to the following resource: https://docs.zeek.org/en/stable/scripts/base/bif/

Zeek Logs

- As for Zeek's logging, when we use Zeek for offline analysis of a PCAP file, the logs will be stored in the current directory.
- Among the diverse array of logs Zeek produces, some familiar ones include:
    - `conn.log`: This log provides details about IP, TCP, UDP, and ICMP connections.
    - `dns.log`: Here, you'll find the details of DNS queries and responses.
    - `http.log`: This log captures the details of HTTP requests and responses.
    - `ftp.log`: Details of FTP requests and responses are logged here.
    - `smtp.log`: This log covers SMTP transactions, such as sender and recipient details.
- Let's consider the `http.log` as an example. It is chock-full of valuable data like:
    - `host`: The HTTP domain/IP.
    - `uri`: The HTTP URI.
    - `referrer`: The referrer of the HTTP request.
    - `user_agent`: The client's user agent.
    - `status_code`: The HTTP status code.
- For a more exhaustive list of common Zeek logs and their respective fields, refer to the following resource: https://docs.zeek.org/en/master/logs/index.html
- It's noteworthy to mention that Zeek, in its standard configuration, applies gzip compression to log files every hour.
    - The older logs are then transferred into a directory named in the `YYYY-MM-DD` format.

    - When dealing with these compressed logs, alternative tools like `gzcat` for printing logs or `zgrep` for searching within logs can come in handy.
- You can find examples on how to work with gzip-compressed Zeek logs at this link: https://blog.rapid7.com/2016/06/02/working-with-bro-logs-queries-by-example/
- As stated earlier, when interacting with Zeek log files, we can utilize native Unix commands such as cat or grep.
    - However, Zeek also provides a specialized tool known as `zeek-cut` for handling log files.
    - This utility accepts Zeek log files via standard input using pipelines or stream redirections and then delivers the specified columns to the standard output.
- If you're interested in exploring Zeek examples, use cases, and learning the basics of writing Zeek scripts, take a look at the following link: https://docs.zeek.org/en/stable/examples/index.html
- For a quick start guide to Zeek, refer to the following link: https://docs.zeek.org/en/stable/quickstart/index.html

Zeek Key Features

- Key features that bolster Zeek's effectiveness include:
    - Comprehensive logging of network activities
    - Analysis of application-layer protocols (irrespective of the port, covering protocols like HTTP, DNS, FTP, SMTP, SSH, SSL, etc.)
    - Ability to inspect file content exchanged over application-layer protocols
    - IPv6 support
    - Tunnel detection and analysis
    - Capability to conduct sanity checks during protocol analysis
    - IDS-like pattern matching
    - Powerful, domain-aware scripting language that allows for expressing arbitrary analysis tasks and managing network state over time
    - Interfacing that outputs to well-structured ASCII logs by default and offers alternative backends for ElasticSearch and DataSeries
    - Real-time integration of external input into analyses
    - External C library for sharing Zeek events with external programs
    - Capability to trigger arbitrary external processes from within the scripting language

One sentence summary

- Zeek is highly flexible and extensible tool for network-based intrusion detection and investigation.

Intrusion Detection With Zeek

Overview

- As already discussed, Zeek, formerly known as Bro, is a powerful network security monitoring tool that allows us to delve deep into our network traffic and extract useful insights.
- The flexibility and extensibility of Zeek make it a cornerstone of advanced network-based intrusion detection and investigation.
    - With its rich set of logs and extensive scripting capabilities, we can customize it to suit our specific detection requirements and continuously improve our security posture.

Intrusion Detection With Zeek Example 1: Detecting Beaconing Malware

- Beaconing is a process by which malware communicates with its command and control (C2) server to receive instructions or exfiltrate data.
    - It's usually characterized by a consistent or patterned interval of outbound communications.
- By analyzing connection logs (`conn.log`), we can look for patterns in outbound traffic.
    - These patterns can include repetitive connections to the same destination IP or domain, constant data size in the sent data, or the connection timing.
    - These are all indicative of potential beaconing behavior.
    - Anomalies can be further explored using Zeek scripts specifically designed to spot beaconing patterns.

```
areaeric@htb[/htb]$ /usr/local/zeek/bin/zeek -C -r /home/htb-
student/pcaps/psempire.pcap
```

```
areaeric@htb[/htb]$ cat conn.log
```

```
areaeric@htb[/htb]$ cat conn.log
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   conn
#open   2023-07-16-12-15-40
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       proto   serv
#types  time    string  addr    port    addr    port    enum    string  interval        count   coun
1511269439.125289       CuQYC98rE69BBb7jb       192.168.56.14   50436   51.15.197.127   80      tcp
1511269436.547667       CTc2Qc2kleCjVaU0V1      fe80::ec23:e8b7:91cb:974d       61431   ff02::1:3
1511269436.548234       Cd4aTbH02m0iB9XS9       192.168.56.14   64755   224.0.0.252     5355    udp
1511269445.266039       CjKEcE3tUWBHhYM93d      192.168.56.14   50437   51.15.197.127   80      tcp
1511269446.190550       CNahóo4OZz5Sr5wGq1      192.168.56.14   50438   51.15.197.127   80      tcp
1511269451.891317       CA6iaN3UgCDI0Sy9x4      192.168.56.14   50439   51.15.197.127   80      tcp
1511269457.130160       ChcRft1mTccVo2yQfa      192.168.56.14   50440   51.15.197.127   80      tcp
1511269462.359918       Clja4s40wWlk8bkAW4      192.168.56.14   50441   51.15.197.127   80      tcp
1511269467.593242       CyE3Th1j6AunL5E3Pl      192.168.56.14   50442   51.15.197.127   80      tcp
1511269472.881671       CwuY7B34I442zmY0hf      192.168.56.14   50443   51.15.197.127   80      tcp
1511269478.120597       CVPMlj3atDGkCy1xyk      192.168.56.14   50444   51.15.197.127   80      tcp
1511269483.366011       Ckn8aZn8c67nuAE19       192.168.56.14   50445   51.15.197.127   80      tcp
1511269488.593094       CfeRTH1oejGgógA5Li      192.168.56.14   50446   51.15.197.127   80      tcp
1511269493.824701       CWmiwT4QR8u71xp0h       192.168.56.14   50447   51.15.197.127   80      tcp
1511269499.116879       C113hs4IWwgOGe0hxf      192.168.56.14   50448   51.15.197.127   80      tcp
1511269504.350011       CHI1AHQLm8ybrQ5ti       192.168.56.14   50449   51.15.197.127   80      tcp
1511269509.574454       CIpoyx4Sx3XEyiKbjh      192.168.56.14   50450   51.15.197.127   80      tcp
1511269514.842106       CDLbbm2nnHbr3RO9F9      192.168.56.14   50451   51.15.197.127   80      tcp
1511269520.114079       CipCM33FvSh7hWVHnc      192.168.56.14   50452   51.15.197.127   80      tcp
1511269525.359633       CwMhAI3giczB1gTeR2      192.168.56.14   50453   51.15.197.127   80      tcp
1511269530.579134       CowTCpq1Lon2Rp5T2       192.168.56.14   50454   51.15.197.127   80      tcp
1511269535.801940       CpYwvc23LmuhPsuuMc      192.168.56.14   50455   51.15.197.127   80      tcp
1511269541.044084       Cnvsqj2QcNcWSLBqH7      192.168.56.14   50456   51.15.197.127   80      tcp
1511269546.278570       CgbzqfgMFulKOP1xe       192.168.56.14   50457   51.15.197.127   80      tcp
1511269551.506084       CUlFD949mmSfKATJpc      192.168.56.14   50458   51.15.197.127   80      tcp
1511269556.712264       Cbc1No4FSSjiRQFoNc      192.168.56.14   50459   51.15.197.127   80      tcp
1511269561.963394       CYwON91Js8ssOY4503      192.168.56.14   50460   51.15.197.127   80      tcp
1511269567.178812       Cryspb4A5KOHbIWDY       192.168.56.14   50461   51.15.197.127   80      tcp
1511269572.442802       CRpXje3yCwJlayfhnj      192.168.56.14   50462   51.15.197.127   80      tcp
1511269577.652288       CkudtR34qPv7fMdMJ6      192.168.56.14   50463   51.15.197.127   80      tcp
1511269582.860772       Cmmlvl4K523e8SjCN6      192.168.56.14   50464   51.15.197.127   80      tcp
```

- If we look carefully enough we will notice connections being made to `51.15.197.127:80` approximately every 5 seconds, which is indicative of a malware beaconing.
- The `psempire.pcap` file, which is located in the `/home/htb-student/pcaps` directory includes traffic related to PowerShell Empire.
    - PowerShell Empire indeed beacons every 5 seconds in its default configuration.

Intrusion Detection With Zeek Example 2: Detecting DNS Exfiltration

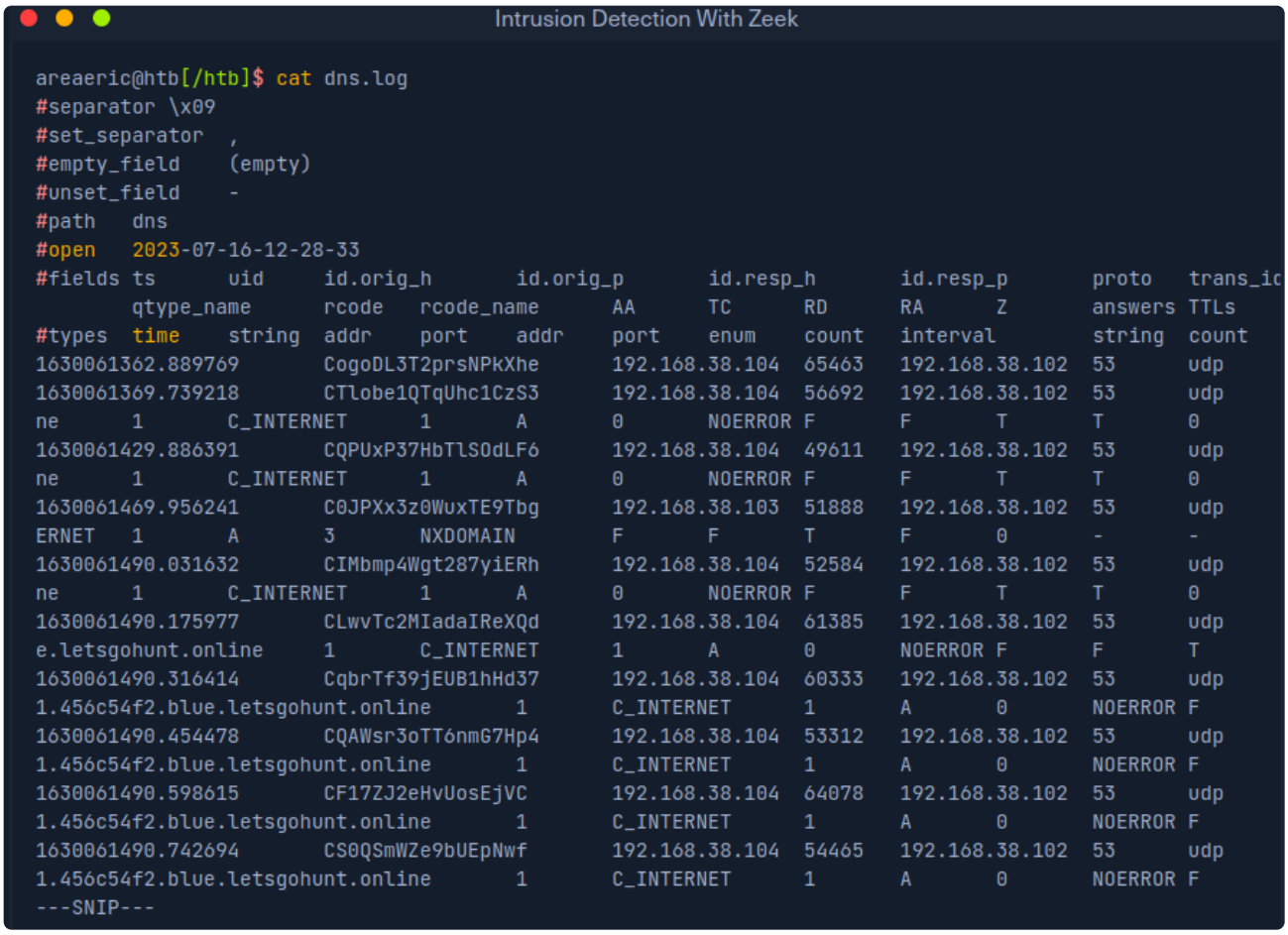
- Zeek is also useful when we suspect data exfiltration.
    - Data exfiltration can be difficult to detect as it often mimics normal network traffic. However, with Zeek, we can analyze our network traffic at a deeper level.

- Zeek's files.log can be used to identify large amounts of data being sent to an unusual external destination, or over non-standard ports, which may suggest data exfiltration.
  - The http.log and dns.log can also be utilized to identify potential covert exfiltration channels, such as DNS tunneling or HTTP POST requests to a suspicious domain.
- Furthermore, Zeek's ability to reassemble files transferred over the network (regardless of the protocol used) can assist in identifying the nature of the data being exfiltrated.

PCAP credits to: Oleh Levytskyi and Bogdan Vennyk

```
areaeric@htb[/htb]$ /usr/local/zeek/bin/zeek -C -r /home/htb-student/pcaps/dnsexfil.pcapng
```

```
areaeric@htb[/htb]$ cat dns.log
```



- Let's focus on the requested (sub)domains by leveraging `zeek-cut` as follows.

```
areaeric@htb[/htb]$ cat dns.log | /usr/local/zeek/bin/zeek-cut query |
cut -d . -f1-7
```

- Upon close inspection, it becomes evident that the domain `letsgohunt.online` possesses a significant number of subdomains, similar to cloud providers.
  - However, it's worth noting that interactions with dozens or even hundreds of subdomains are generally not considered typical behavior.
- The `dnsexfil.pcapng` file, which is located in the `/home/htb-student/pcaps` directory includes traffic related to DNS exfiltration.

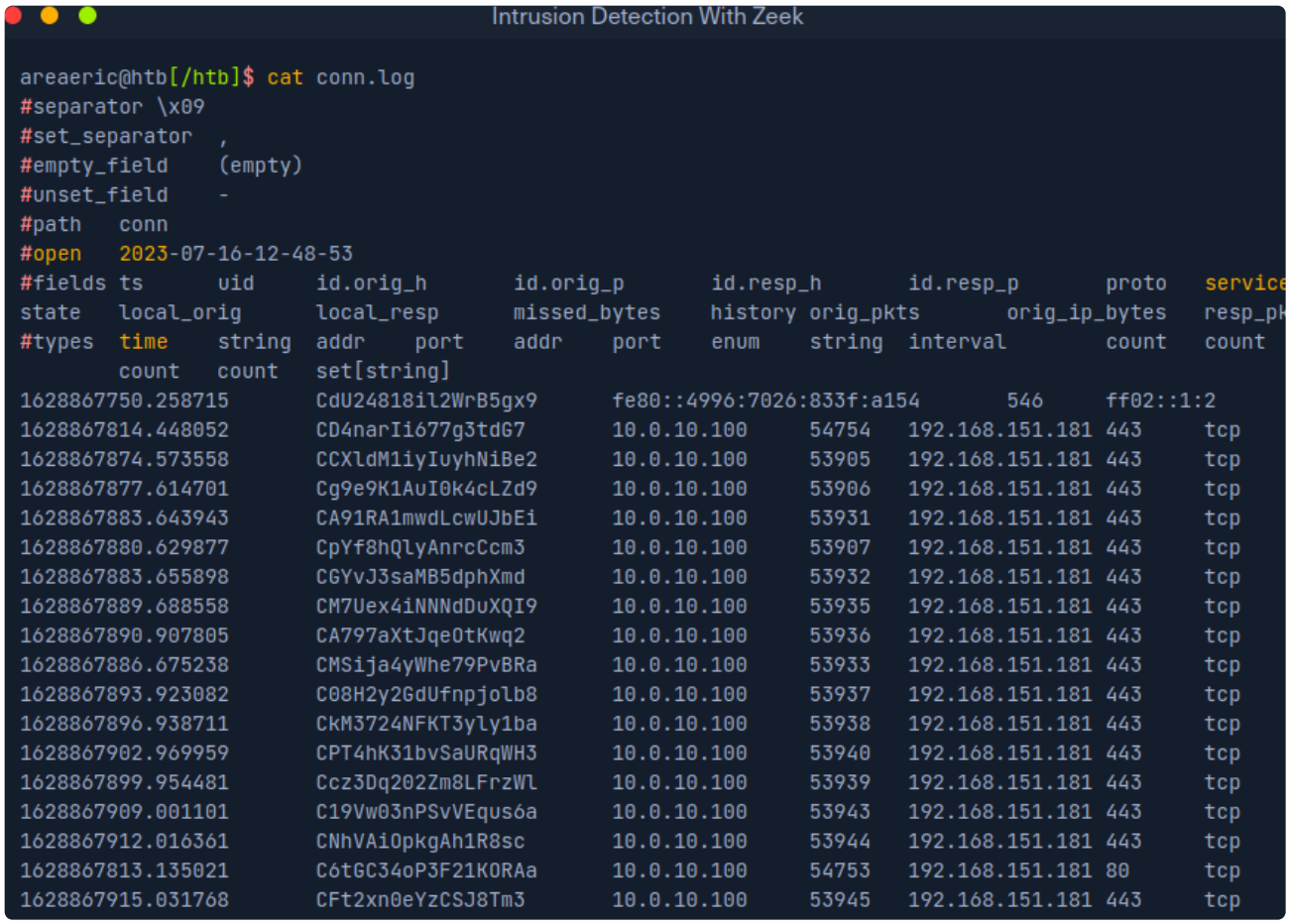
Intrusion Detection With Zeek Example 3: Detecting TLS Exfiltration

PCAP credits to: Oleh Levytskyi and Bogdan Vennyk

- Let's now go over an example of detecting data exfiltration over TLS.

```
areaeric@htb[/htb]$ /usr/local/zeek/bin/zeek -C -r /home/htb-
student/pcaps/tlsexfil.pcap
```

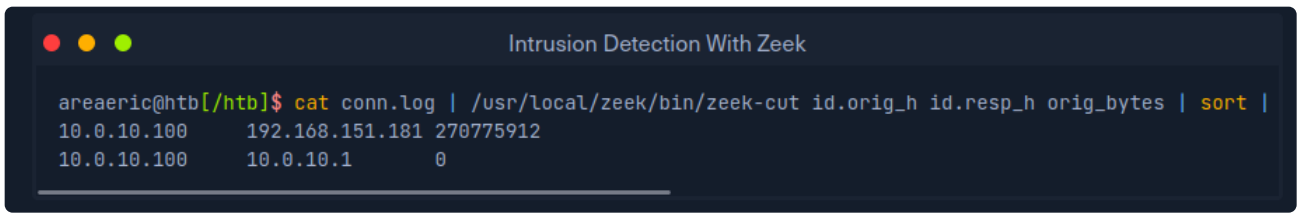
```
areaeric@htb[/htb]$ cat conn.log
```



- The output is a bit tricky to analyze. Let's narrow things down by using `zeek-cut` one more time.

One-liner source: activecountermeasures

```
areaeric@htb[/htb]$ cat conn.log | /usr/local/zeek/bin/zeek-cut
id.orig_h id.resp_h orig_bytes | sort | grep -v -e '^$' | grep -v '-' |
datamash -g 1,2 sum 3 | sort -k 3 -rn | head -10
```



Let's analyze the command above.

- `cat conn.log`: This command is used to read the content of the conn.log file.

    - The `conn.log` file, generated by Zeek, provides a record of all connections that have taken place in our network.
- `/usr/local/zeek/bin/zeek-cut id.orig_h id.resp_h orig_bytes`: In this case, we're extracting the `id.orig_h` (originating host), `id.resp_h` (responding host), and `orig_bytes` (number of bytes sent by the originating host) fields.
- `sort`: This command is used to sort the output from the previous command.
    - By default, sort will arrange the lines in ascending order based on the contents of the first field (in this case `id.orig_h`).
- `grep -v -e '^$'`: This command filters out any empty lines.
    - The `-v` option inverts the selection, the `-e option allows for a regular expression, and` '^$'` matches empty lines.
- `grep -v '-'`: This command filters out lines containing a dash `-`.
    - In the context of Zeek logs, a dash often represents a missing value or an undefined field.
- `datamash -g 1,2 sum 3`: `datamash` is a command-line tool that performs basic numeric, textual, and statistical operations.
    - The `-g 1,2` option groups the output by the first two fields (the IP addresses of the originating and responding hosts), and `sum 3` computes the sum of the third field (the number of bytes sent) for each group.
- `sort -k 3 -rn`: This command sorts the output of the previous command in descending order (`-r`) based on the numerical value (`-n`) of the third field (`-k 3`), which is the sum of `orig_bytes` for each pair of IP addresses.
- `head -10`: This command is used to limit the output to the top 10 lines, thus showing the top 10 pairs of IP addresses by total bytes sent from the originating host to the responding host.
- We notice that ~270 MB (actually a bit less) of data have been sent to `192.168.151.181`.

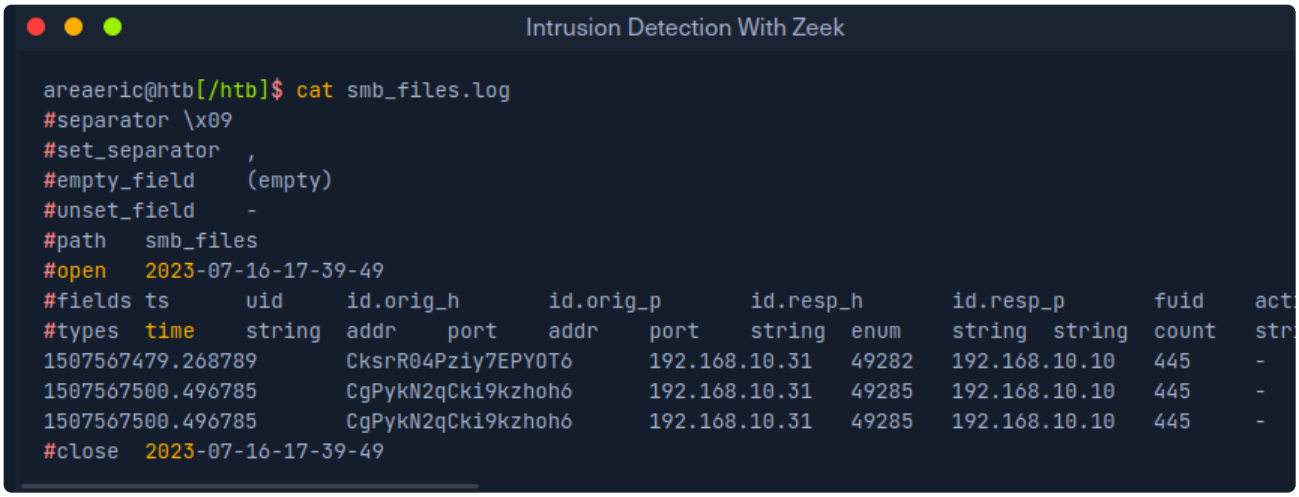
Intrusion Detection With Zeek Example 4: Detecting PsExec

- `PsExec`, a part of the Sysinternals Suite, is frequently used for remote administration within Active Directory environments.
    - Given its powerful capabilities, it's no surprise that adversaries often prefer `PsExec` when they carry out remote code execution attacks.
- To illustrate a typical attack sequence, let's consider this: an attacker transfers the binary file `PSEXESVC.exe` to a target machine using the `ADMIN$` share, a special shared folder used in Windows networks, via the SMB (Server Message Block) protocol.

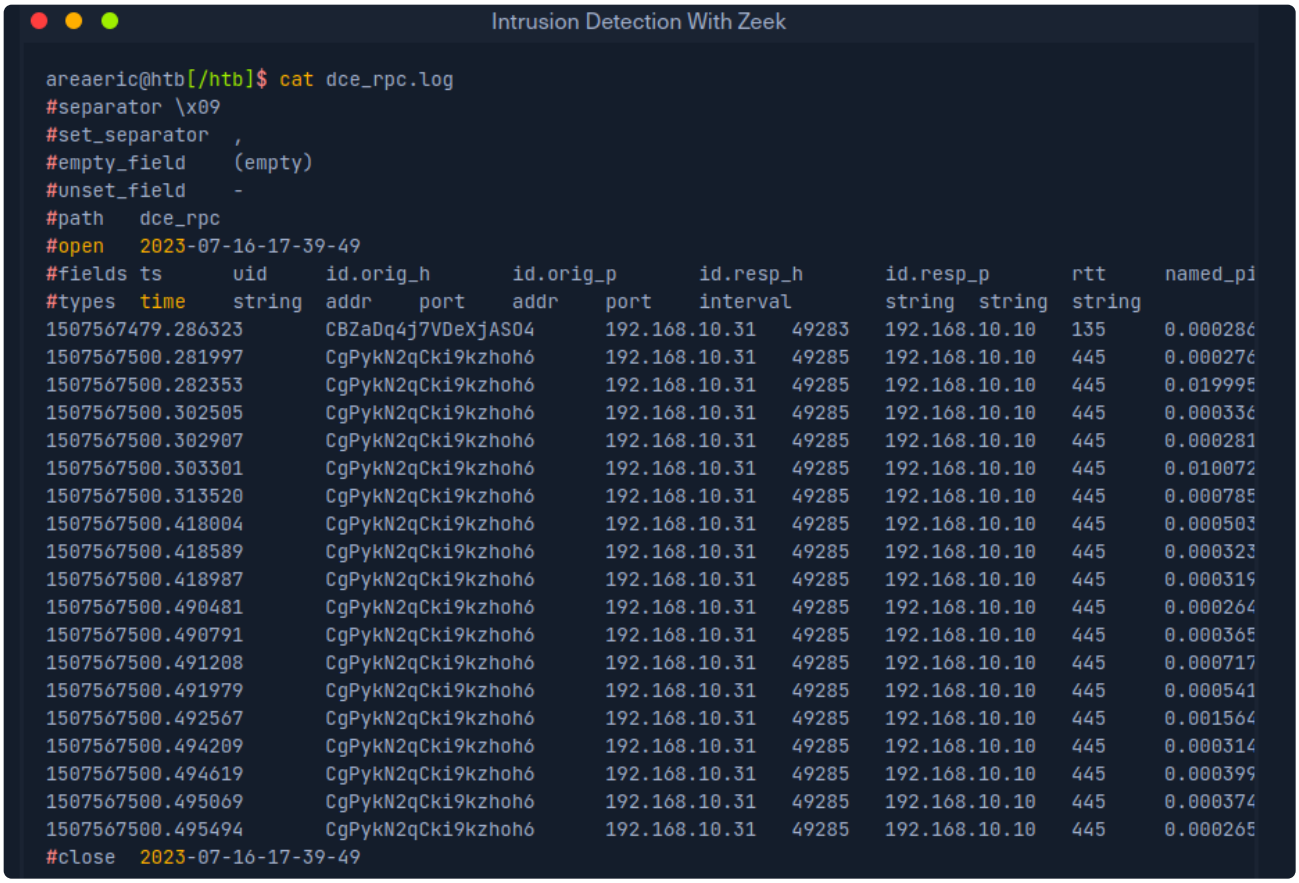
    - Following this, the attacker remotely launches this file as a temporary service by utilizing the `IPC$` share, another special shared resource that enables Inter-Process Communication.
- We can identify SMB transfers and the typical use of `PsExec` using Zeek's `smb_files.log`, `dce_rpc.log`, and `smb_mapping.log` as follows.

```
areaeric@htb[/htb]$ /usr/local/zeek/bin/zeek -C -r /home/htb-student/pcaps/psexec_add_user.pca
```

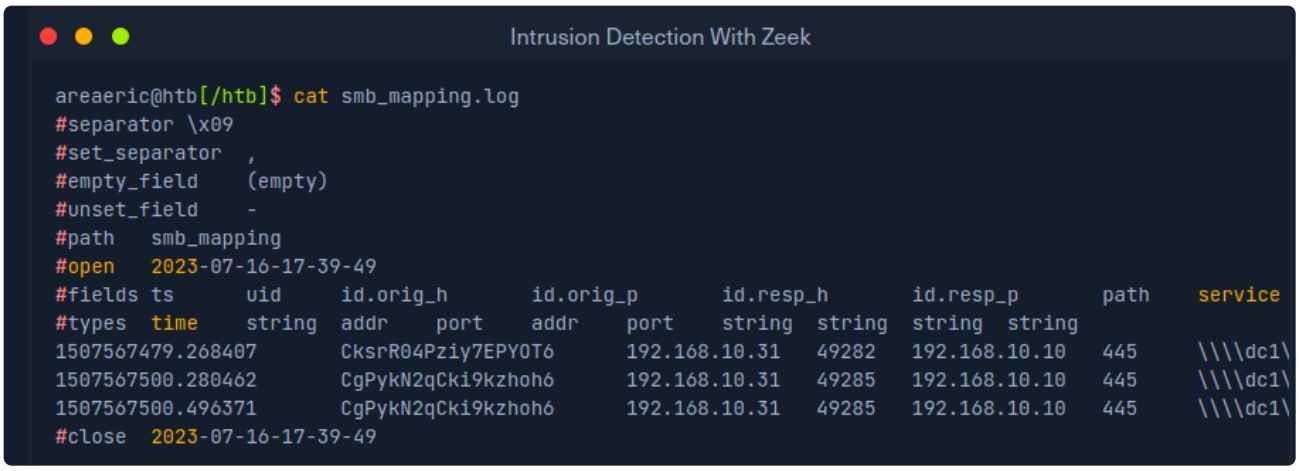
```
areaeric@htb[/htb]$ cat smb_files.log
```




```
areaeric@htb[/htb]$ cat dce_rpc.log
```




```
areaeric@htb[/htb]$ cat smb_mapping.log
```



- The temporary service creation is apparent in the last two logs above.

One sentence Summary

- Zeek is not just IDS/IPS, it can do much more and generate lots of different useful logs for various purpose (e.g. dns exfiltration look at dns.log with zeek-cut, TLS exfiltration

look at conn.log with zeek-cut, malware beaconing look at conn.log with time, psexec activity with smb logs) with useful built-in functionality (zeek-cut) that snort and suricata might not be able to do or do so efficiently.