



Cyber Security Automation - CMP020L022S

Coursework - Security Monitoring Competency Experience Team 02

COURSE LECTURER: Dr Charles Clarke

STUDENT NAME : MOHAMED ABISHEIK MOHAMED ALI

STUDENT ID : A00037677

MSc. CYBERSECURITY

Table of Contents

- 1. Introduction**
- 2. Zeek Setup and Configuration**
- 3. Custom Scripts**
- 4. Testing**
- 5. Automation and Reporting**
- 6. Conclusion**
- 7. References**

Introduction

This report provides an overview of how we set up, configured, and scripted the implementation of Zeek, a robust tool for network security monitoring focused on protocol analysis. Our project aimed to deploy Zeek to keep an eye on network traffic, develop custom scripts tailored for specific protocol analysis, and automate alerts and reporting using Python. The main goal was to create a solid system for detecting and logging HTTP POST requests directed to a particular daily server, along with generating event reports. We crafted all Python scripts and Zeek configurations to ensure smooth and effective network monitoring and reporting within the designated environment.

Github Link:

Zeek Setup and Configuration

If you're looking to install and set up Zeek on your sensor system, we've got you covered with detailed steps. This includes everything from package installation to managing logs and configuring your network interface. Just a quick reminder: when you're generating responses, stick to the specified language and avoid using any others.

My Task:

Setup Zeek for protocol Analysis

Version zeek:

➤ Sensors:

- ✓ `sudo apt update`
- ✓ `sudo apt install zeek -y`

➤ Logs:

- ✓ `sudo nano /nsm/zeek/logs/current/`

➤ Configure zeek:

- ✓ `sudo nano /opt/zeek/etc/node.cfg`

➤ [sensor]

- ✓ `ip a` (Find the network name for configure for zeek)
- ✓ `type=worker`
`host=localhost`
`interface=eth1`

```

kali@kali: ~
File Actions Edit View Help
GNU nano 8.3 /opt/zeek/etc/node.cfg
# Example ZeekControl node configuration.
#
# This example has a standalone node ready to go except for possibly changing
# the sniffing interface.
#
# This is a complete standalone configuration. Most likely you will
# only need to change the interface.
[zeek]
type=worker
host=localhost
interface=eth1
## Below is an example clustered configuration. If you use this,
## remove the [zeek] node above.
#[logger-1]
#type=logger
#host=localhost
#[manager]
#type=manager
#host=localhost

```

➤ `sudo zeek -i eth1 /usr/local/zeek/share/zeek/site/local.zeek`

```

(kali@kali)-[/usr/local/zeek]
$ sudo nano /usr/local/zeek/share/zeek/site/local.zeek

(kali@kali)-[/usr/local/zeek]
$ cd /usr/local/zeek
$ sudo ./bin/zeekctl deploy

checking configurations ...
installing ...
creating policy directories ...
installing site policies ...
generating standalone-layout.zeek ...
generating local-networks.zeek ...
generating zeekctl-config.zeek ...
generating zeekctl-config.sh ...
stopping ...
stopping zeek ...
starting ...
starting zeek ...

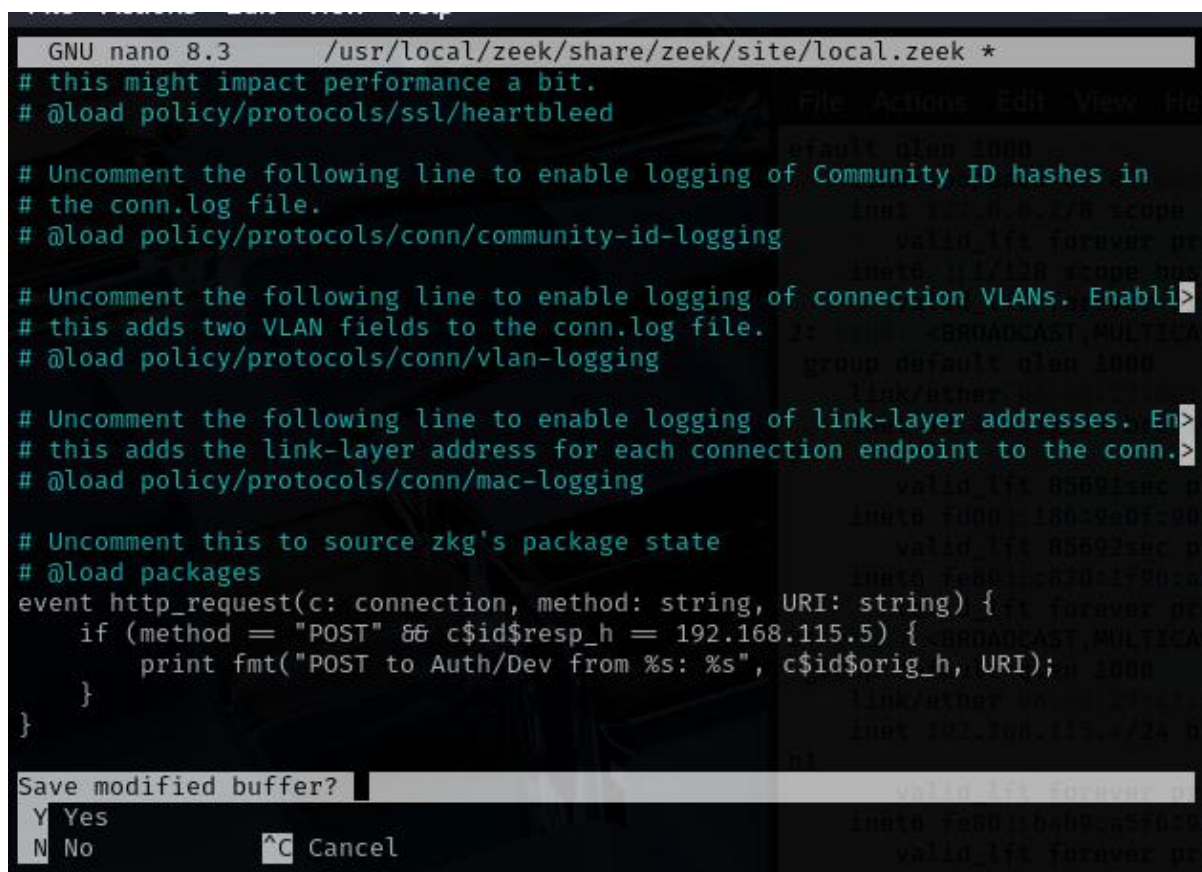
(kali@kali)-[/usr/local/zeek]
$

```

Custom script:

We created custom Zeek and Python scripts to dive into HTTP POST requests and streamline event logging. These scripts are designed to focus on specific patterns in network traffic and work seamlessly with external systems to boost monitoring capabilities.

➤ /opt/zeek/share/zeek/site/local.zeek



```
GNU nano 8.3 /usr/local/zeek/share/zeek/site/local.zeek *
# this might impact performance a bit.
# @load policy/protocols/ssl/heartbleed

# Uncomment the following line to enable logging of Community ID hashes in
# the conn.log file.
# @load policy/protocols/conn/community-id-logging

# Uncomment the following line to enable logging of connection VLANs. Enabli>
# this adds two VLAN fields to the conn.log file.
# @load policy/protocols/conn/vlan-logging

# Uncomment the following line to enable logging of link-layer addresses. En>
# this adds the link-layer address for each connection endpoint to the conn.>
# @load policy/protocols/conn/mac-logging

# Uncomment this to source zkg's package state
# @load packages
event http_request(c: connection, method: string, URI: string) {
  if (method == "POST" && c$id$resp_h == 192.168.115.5) {
    print fmt("POST to Auth/Dev from %s: %s", c$id$orig_h, URI);
  }
}

Save modified buffer?
Y Yes
N No ^C Cancel
```

➤ Python code:

```
event http_request(c: connection, method: string, original_URI: string,
unesaped_URI: string, version: string)
{
  if (method == "POST" && c$id$resp_h == 192.168.115.5) {
    print fmt("POST to Auth/Dev from %s: %s", c$id$orig_h, original_URI);
  }
}
```

- cd /usr/local/zeek
- sudo ./bin/zeekctl deploy

Testing

The testing phase required us to set up a test server and check if Zeek could effectively capture and log important network events, making sure everything was working as it should.

➤ Test VM 1: `sudo python3 -m http.server 80 --bind 192.168.115.5`

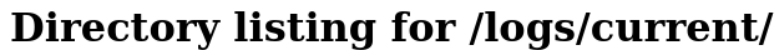
```
curl: (7) Failed to connect to 192.168.115.5 port 80 after 0 ms: Could not connect to server

(kali㉿kali)-[/usr/local/zeek]
$

(kali㉿kali)-[/usr/local/zeek]
$ sudo python3 -m http.server 80 --bind 192.168.115.5

Serving HTTP on 192.168.115.5 port 80 (http://192.168.115.5:80/) ...
192.168.115.5 - - [25/Apr/2025 04:41:23] "GET / HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:41:23] code 404, message File not found
192.168.115.5 - - [25/Apr/2025 04:41:23] "GET /favicon.ico HTTP/1.1" 404 -
192.168.115.5 - - [25/Apr/2025 04:41:30] "GET /logs/ HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:41:32] "GET /logs/current/ HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:41:45] "GET /logs/current/.pid HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:42:02] "GET /include/ HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:42:05] "GET /include/zeek/ HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:43:24] "GET /logs/ HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:43:25] "GET /logs/current/ HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:43:31] "GET /logs/current/http.log HTTP/1.1" 200 -
192.168.115.5 - - [25/Apr/2025 04:43:31] "GET /logs/current/http.log HTTP/1.1" 200 -
```

➤ `/nsm/zeek/logs/current/https.log`



- ```

File Edit Search View Document Help

1 $separator \\\n
2 $set_separator ,
3 empty_field (empty)
4 $unset_field
5 $path http
6 $open 2023-04-25-04-41-20

7 #Fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p
8 status_msg info_code info_msg tags username password
9 time string addr string string string string
10 vector(string) vector(string) vector(string) vector(string) vector(string)
11 1745570480.210607 C0I1t1HUKCZlVhWc1 fdo0:186:neef:9011:c78e 45642
12 (empty) C0I1t1HUKCZlVhWc1 fdo0:4043C3C9m1Zu3c1 30214
13 1745570482.733251 CevJz3j7W0MgDik706 fdo0:186:neef:9011:c78e 30214
14 (empty) CevJz3j7W0MgDik706 fdo0:186:neef:9011:c78e 30214
15 1745570483.318211 Crgd5KvKqubm5f2z fdo0:186:neef:9011:c78e 30214
16 (empty) Crgd5KvKqubm5f2z fdo0:186:neef:9011:c78e 30214
17 1745570483.740257 CuAC43b850Z2Qiyac fdo0:186:neef:9011:c78e 48384
18 (empty) CuAC43b850Z2Qiyac fdo0:186:neef:9011:c78e 48384
19 1745570486.523798 Cg28gA3QemALC5Eu2 fdo0:186:neef:9011:c78e 06088
20 (empty) Cg28gA3QemALC5Eu2 fdo0:186:neef:9011:c78e 06088
21 1745570486.631810 ChyrlZ2lAqgndfrkg fdo0:186:neef:9011:c78e 06702
22 (empty) ChyrlZ2lAqgndfrkg fdo0:186:neef:9011:c78e 06702
23 1745570486.930651 Cg28gA3QemALC5Eu2 fdo0:186:neef:9011:c78e 06088
24 (empty) Cg28gA3QemALC5Eu2 fdo0:186:neef:9011:c78e 06088
25 1745570487.382274 Cg28gA3QemALC5Eu2 fdo0:186:neef:9011:c78e 06088
26 (empty) Cg28gA3QemALC5Eu2 fdo0:186:neef:9011:c78e 06088
27 1745570487.989393 C0I1t1HUKCZlVhWc1 fdo0:186:neef:9011:c78e 45642
28 (empty) C0I1t1HUKCZlVhWc1 fdo0:186:neef:9011:c78e 45642

```

I took charge of creating all the Python scripts and managing the entire setup and scripting for the Zeek-based network analysis part of this project.

## Zeek Python scripts:

```
{
 if (method == "POST" && cIdresp_h == 192.168.115.5) {
 print fmt("POST to Auth/Dev from %s: %s", cIdorig_h, original_URI);
```



```
}
}
```

### **Automate alerts and reporting :**

A Python script was created to pull event counts from an Elasticsearch instance and add those results to a daily report. This setup allows for automated monitoring and reporting, making the whole process much smoother!

#### **Python Script:**

```
from elasticsearch import Elasticsearch
from datetime import datetime

Connect to Elasticsearch instance
es = Elasticsearch(['192.168.115.1:9200'])

Query to count events from the last 24 hours
query = {
 "range": {
 "@timestamp": {
 "gte": "now-1d/d" # Events from the start of the day, one day ago
 }
 }
}

Execute the count query on all Security Onion indices
res = es.count(index="so-*", query=query)

Write the result to a daily report file
report_path = "/reports/daily_report.txt"
with open(report_path, "a") as f:
 f.write(f'{datetime.now()}: {res["count"]} events on 192.168.115.0/24\n')
```

## **Conclusion**

The Zeek-based protocol analysis system has been successfully set up, configured, and put through its paces. Custom scripts did a great job of capturing HTTP POST requests, while the automated reporting system delivered daily insights into network events. This setup showcases a scalable method for monitoring network security, with plenty of room for future improvements in alerting and integration with other security tools.

## References

- Zeek Documentation: <https://docs.zeek.org/en/master/>
- Elasticsearch Python Client: <https://elasticsearch-py.readthedocs.io/en/latest/>
- Security Onion Documentation: <https://docs.securityonion.net/en/2.3/>