ENPM685 – Intrusion Prevention/Detection Exercises

Version 3.2 – April 1st 2022

OSSEC Exercises (30 minutes)

OSSEC Exercise - Install

Perform this exercise on your Ubuntu VM.

1. Install the required dependencies. OSSEC needs gcc (a compiler), libc (C standard library), Apache and PHP (those are already installed on the VM), and a few other things.
    a. Run: **`sudo apt-get install build-essential gcc make unzip inotify-tools mailutils sendmail libz-dev`**
    b. If this returns some errors (such as "**E: Failed to fetch […] Not found [IP: 91.189.91.26 80]**" do the following
        i. Edit /etc/apt/sources.list and comment out the "**deb cdrom**" line at the top of the file
        ii. Run "**sudo apt-get update**"
        iii. Try running the sudo-apt-get install line above again.
2. Download OSSEC - **`wget https://github.com/ossec/ossec-hids/archive/3.2.0.tar.gz`**
3. Extract the file - **`tar -xvzf 3.2.0.tar.gz`**
4. Enter the newly created directory - **`cd ossec-hids-3.2.0/`**
5. Run the installer - **`sudo sh install.sh`**
6. Select your language (I'll be using English, the default)
7. Press **Enter** to continue
8. For the kind of installation select "**local**"
9. Accept the default install location**`, /var/ossec`**
10. Select **Yes** for email notification
11. For the email address enter "**root@localhost**"
12. For the mail server enter **127.0.0.1**
13. For the "run the integrity check daemon" select **Yes**
14. For the rootkit detection engine select **Yes**
15. For active response select **Yes**
16. For the firewall-drop response select **Yes**
17. Select **No** for add more IPs to the white list
18. You'll see a summary of files that will be analyzed by OSSEC, press **Enter**

```
 3.6- Setting the configuration to analyze the following logs:
   -- /var/log/auth.log
   -- /var/log/syslog
   -- /var/log/vsftpd.log
   -- /var/log/dpkg.log
   -- /var/log/apache2/error.log (apache log)
   -- /var/log/apache2/access.log (apache log)

- If you want to monitor any other file, just change
  the ossec.conf and add a new localfile entry.
  Any questions about the configuration can be answered
  by visiting us online at http://www.ossec.net .


  --- Press ENTER to continue ---
```

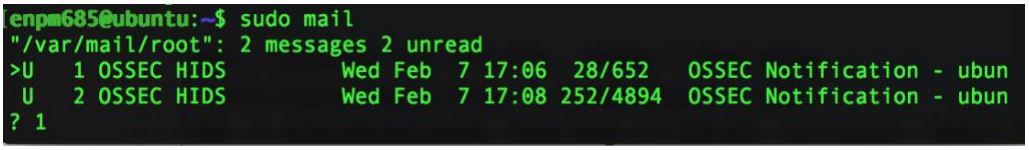
19. OSSEC will kick off the build script to compile itself. This make take a few minutes. Take a break and grab a beverage to enjoy, you deserve it.
20. When finished you'll see this:

```
- System is Debian (Ubuntu or derivative).
- Init script modified to start OSSEC HIDS during boot.

- Configuration finished properly.

- To start OSSEC HIDS:
     /var/ossec/bin/ossec-control start

- To stop OSSEC HIDS:
     /var/ossec/bin/ossec-control stop

- The configuration can be viewed or modified at /var/ossec/etc/ossec.conf


   Thanks for using the OSSEC HIDS.
   If you have any question, suggestion or if you find any bug,
   contact us at contact@ossec.net or using our public maillist at
   ossec-list@ossec.net
   ( http://www.ossec.net/main/support/ ).

   More information can be found at http://www.ossec.net

   ---  Press ENTER to finish (maybe more information below). ---
```

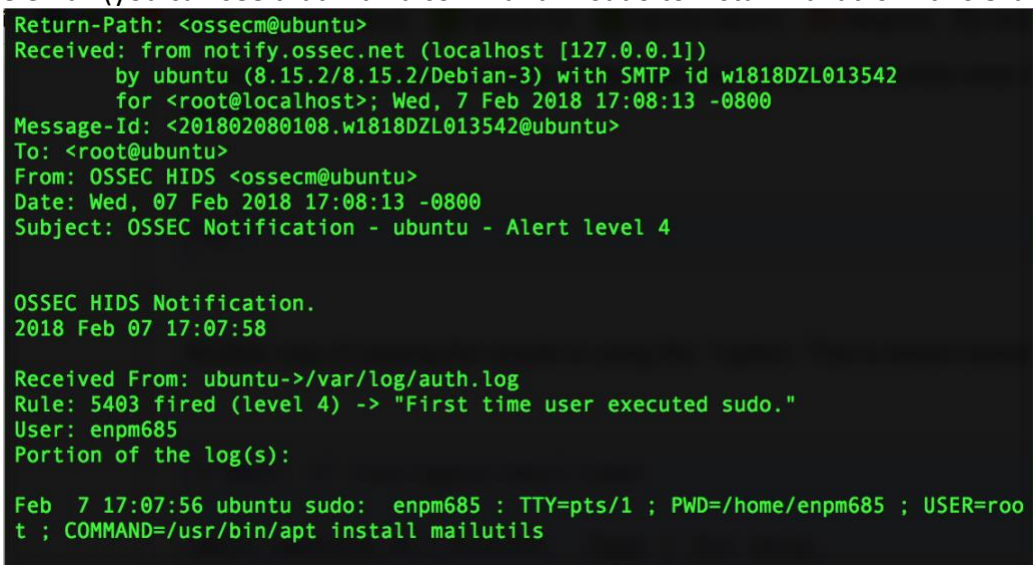
21. Press **Enter** to finish the install Script
22. Start OSSEC with **`sudo /var/ossec/bin/ossec-control start`** You'll see it the OSSEC start up output on your screen.

```
[enpm685@ubuntu:~$ sudo /var/ossec/bin/ossec-control start
[[sudo] password for enpm685:
Starting OSSEC HIDS v2.9.3 (by Trend Micro Inc.)...
Started ossec-maild...
Started ossec-execd...
Started ossec-analysisd...
Started ossec-logcollector...
Started ossec-syscheckd...
Started ossec-monitord...
Completed.
```

23. If you want to see messages that OSSEC mails out (in this case to root@localhost which we set in Step 11) type "**sudo mail**" (You may need to wait a few minutes for OSSEC mail to process, if you get a message that there is "No mail for root" continue on to the install web UI and come back later to the mail.)
24. Type the number of the email you want to read and press enter to read the email. (ex "**1**")



Example email (you can see that I ran a command w sudo to install mailutils in this example)



25. Type "**q**" to quit the mail client

Congratulations you have successfully installed OSSEC. Now let's install a web UI for it for easy management and then we'll configure and test it some.

OSSEC Exercise – Install web UI

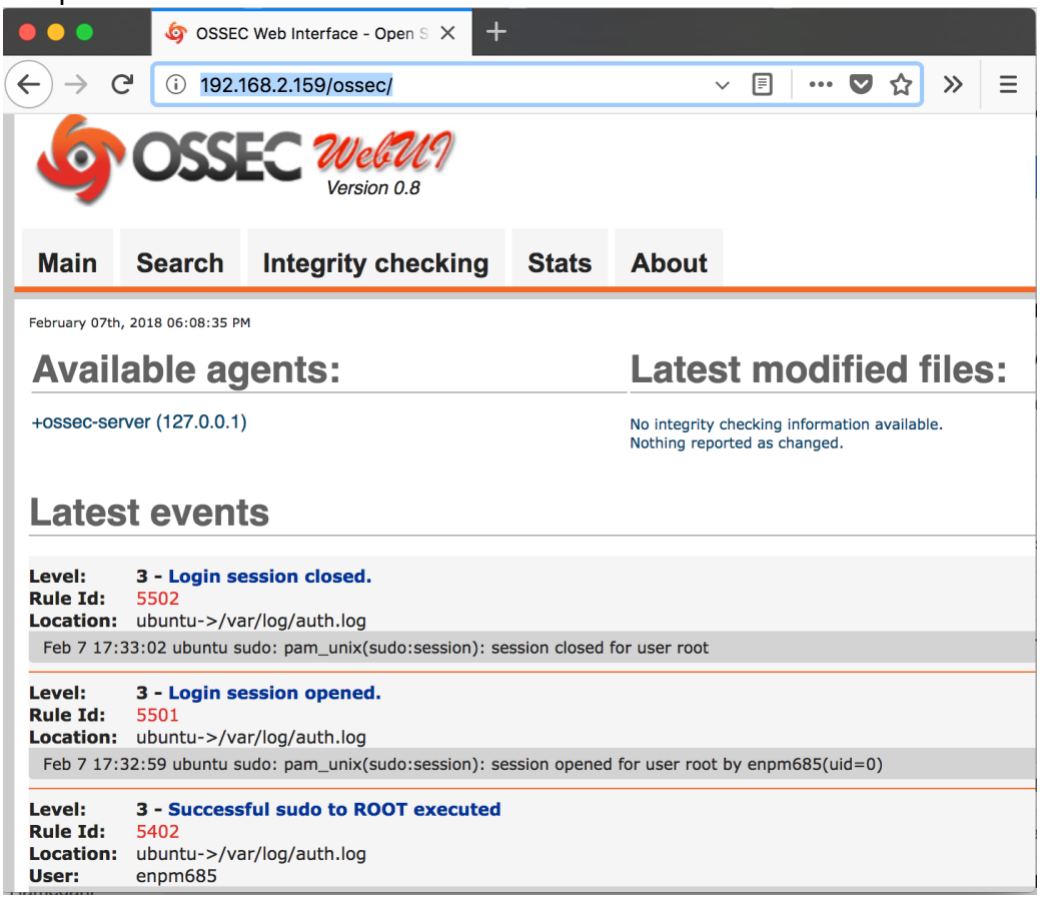
1. Download the web UI - **wget https://github.com/ossec/ossec-wui/archive/master.zip**
2. Unzip the file – **unzip master.zip**
3. Move the files we unzipped to their new home – **sudo mv ossec-wui-master /var/www/html/ossec**
4. Change to the web UI directory - **cd /var/www/html/ossec**
5. Run the setup script – **sudo ./setup.sh**
6. Enter the configuration information:

a. User: Your choice. I went with "**admin**"
b. Password: your choice. I went with "**badpassword**"
c. Web server user name: "**www-data**" (This is the user that Apache runs as on Ubuntu systems)

```
[enpm685@ubuntu:/var/www/html/ossec$ sudo ./setup.sh
 trap: SIGHUP: bad trap
 Setting up ossec ui...

[Username: admin
[New password:
[Re-type new password:
 Adding password for user admin
 Enter your web server user name (e.g. apache, www, nobody, www-data, ...)
[www-data
 You must restart your web server after this setup is done.

 Setup completed successfully.
```

7. Restart Apache with **`sudo systemctl restart apache2`**
8. Open a web browser and go to http://192.168.2.159/ossec/ (replace the IP address with the IP address of your Ubuntu VM)

Sample web UI screen shot:



9. Look around the web UI some, what do you see?

You may notice that the OSSEC Web UI is no longer maintained. Why? Most users are sending logs into a log management tool like Splunk for centralized log management and correlation with their other logs. (We'll add OSSEC logs into our Splunk install in a few weeks…)

OSSEC Exercise – Configuration and Testing

OSSEC configuration file - **/var/ossec/etc/ossec.conf**

Since we turned on all of the OSSEC monitoring if we modify a file it won't report for 22 hours (79200 seconds) by default. We can change that to real time by modifying 2 lines.

1. Edit the OSSEC configuration file (**`sudo vi /var/ossec/etc/ossec.conf`**)
2. Look for the **<syscheck>** section
3. Edit the following lines:

```
<directories check_all="yes">/etc,/usr/bin,/usr/sbin</directories>
<directories check_all="yes">/bin,/sbin,/boot</directories>
```

And add **`realtime="yes"`** in to them so they look like:

```
<directories realtime="yes"
check_all="yes">/etc,/usr/bin,/usr/sbin</directories>
<directories realtime="yes"
check_all="yes">/bin,/sbin,/boot</directories>
```

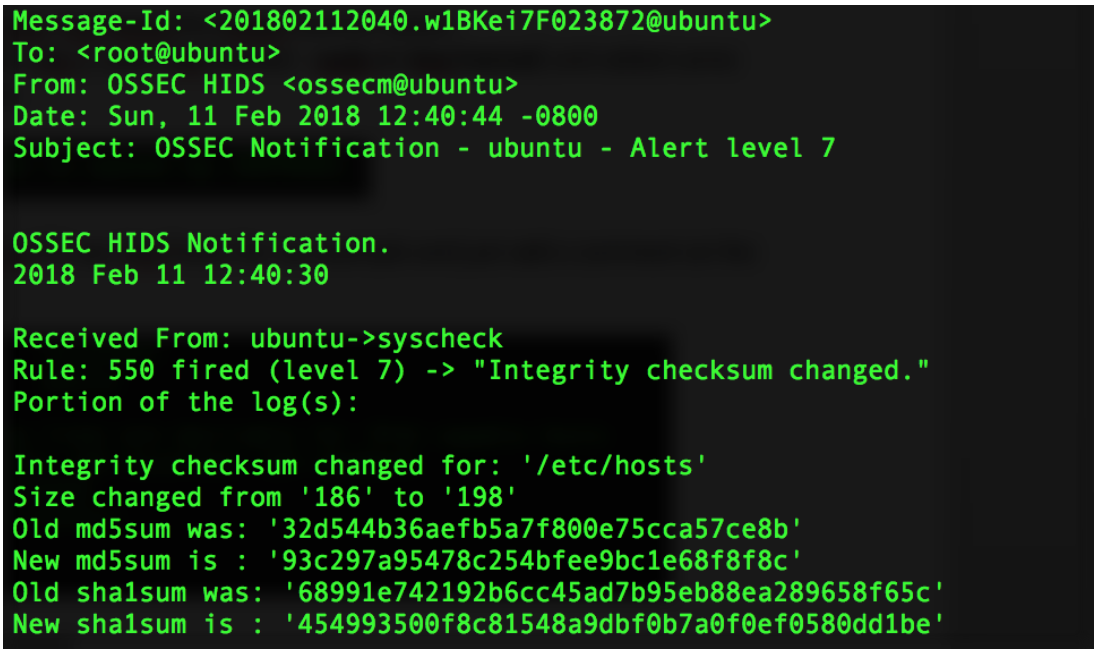
4. Save the file
5. Modify a file in /etc (I'll use /etc/hosts for my example and just add a comment at the end of the file. Ex: "**# ENPM685!**"

```
127.0.0.1       localhost
127.0.1.1       ubuntu

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# ENPM685!
```

6. Save the file you edited
7. Restart OSSEC - **`sudo /var/ossec/bin/ossec-control restart`**
8. Review the OSSEC Web UI for the new alert about /etc/hosts being modified. You can also view the mail for root as well (**sudo mail**) (This may take a few minutes to show up.)

Web UI Example:

Mail example:



Snort Exercises (30 minutes)

Snort Exercise – Configuration and Testing

NOTE: If you see an error message along the lines of "**ERROR: `/etc/snort/rules/local.rules(15) Content data needs to be enclosed in quotation marks (")!`**" review your local.rules file to see how the double quote (") is stored. Some operating systems can get squirrely with these and cause issues if you are copying and pasting.

1. Let'd start by editing **/etc/snort/rules/local.rules** and add a Snort rule to monitor for TCP traffic:

   `alert tcp any any -> any any (msg:"TCP Packet"; sid:1;)`

```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures.  Put your local
# additions here.

alert tcp any any -> any any (msg:"TCP Packet sent"; sid:1;)
```

2. Run Snort: **`sudo snort -c /etc/snort/snort.conf -A fast -l /var/log/snort`**
3. On that same VM try to connect to another host host (open a web page, SSH to something, etc – for my example I ran **`wget http://www.umd.edu`**)
4. Type **Control + C** to stop snort from running. You'll see the snort statistics run by, you can review them if you wish for some interesting information. The **Action Stats** are most likely the most interesting since it will show if any network traffic was alerted on.

```
===============================================================================
Packet I/O Totals:
   Received:          274
   Analyzed:          273 ( 99.635%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            1 (  0.365%)
   Injected:            0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:          275 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:          271 ( 98.545%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:            4 (  1.455%)
        TCP:          251 ( 91.273%)
        IP6:            0 (  0.000%)
```

```
===============================================================================
Action Stats:
     Alerts:          136 ( 49.455%)
     Logged:          136 ( 49.455%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            6
Verdicts:
      Allow:          272 ( 99.270%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:            0 (  0.000%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
      Retry:            0 (  0.000%)
```

5. Review **/var/log/snort/alert** (**`less /var/log/snort/alert`**) – what do you see? An example from my logs are below:

```
02/07-18:45:35.543858  [**] [1:1:0] TCP Packet sent [**] [Priority: 0] {TCP} 192
.168.2.1:52700 -> 192.168.2.159:22
02/07-18:45:35.543863  [**] [1:1:0] TCP Packet sent [**] [Priority: 0] {TCP} 192
.168.2.1:52700 -> 192.168.2.159:22
02/07-18:45:35.544270  [**] [1:1:0] TCP Packet sent [**] [Priority: 0] {TCP} 192
.168.2.1:52700 -> 192.168.2.159:22
02/07-18:45:35.544367  [**] [1:1:0] TCP Packet sent [**] [Priority: 0] {TCP} 192
.168.2.1:52700 -> 192.168.2.159:22
02/07-18:45:35.544495  [**] [1:1:0] TCP Packet sent [**] [Priority: 0] {TCP} 192
.168.2.1:52700 -> 192.168.2.159:22
02/07-18:45:35.544641  [**] [1:1:0] TCP Packet sent [**] [Priority: 0] {TCP} 192
.168.2.1:52700 -> 192.168.2.159:22
02/07-18:45:35.544775  [**] [1:1:0] TCP Packet sent [**] [Priority: 0] {TCP} 192
.168.2.1:52700 -> 192.168.2.159:22
```

Getting More Interesting With Snort

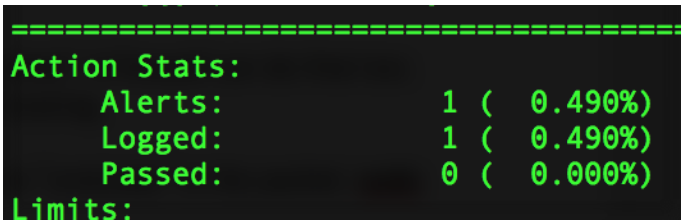
So far, we have shown we can alert/block on TCP traffic. So what? A firewall can do that too. Let's try alerting on the content of a packet which is more interesting.

1. Let's create a rule that looks for TCP traffic that contains "umd.edu" in the packet. **`sudo vi /etc/snort/rules/local.rules`**
2. The rule should look like:

```
alert tcp any any -> any any (msg:"umd.edu Content Detected";
content:"umd.edu"; sid:2;)
```

3. Run snort again - **`sudo snort -c /etc/snort/snort.conf -A fast -l /var/log/snort`**

4. Open another SSH terminal into the host you are running snort on and download the main page of www.umd.edu with one of the following:
    a. **wget http://www.umd.edu**
    b. **curl http://www.umd.edu**
5. After the wget/curl command gets the UMD home page in the snort window press Control + C to stop snort from running.
6. If you review the snort output you should see one or more alerts



7. Review /var/log/snort/alert, you should see one or more alerts for your "umd.edu" rule (**less /var/log/snort/alert** or **tail /var/log/snort/alert**)

```
02/11-19:43:56.394159  [**] [1:3:0] umd.edu Content Detected [**]
[Priority: 0] {TCP} 13.33.252.65:80 -> 192.168.2.159:58314
```

In this case the "umd.edu" is matched in the request to connect and download the umd.edu home page. This is innocent traffic but imaging if umd.edu was a C&C controller for a botnet? Or we can use that content rule option for more interesting things as we'll see in the next example.

Replaying a packet capture (.pcap) file through Snort

If you remember from the lecture slides, Snort can process packet captures (.pcaps) as well as live network traffic. You process .pcaps with the -r command line option:

sudo snort -c /etc/snort/snort.conf -A fast -l /var/log/snort -r *pcap*

Why is this useful? A few reasons:

- For developing and testing new rules
- You are able to capture network traffic somewhere but are unable to run Snort there
- In larger environments it often makes more sense to capture traffic in 5+ minute "chunks" and then process those "chunks" vs running in real time.

1. We're going to review a packet capture of the SSL Heartbleed exploit. This is preloaded on the Ubuntu VM and is located at **/var/www/html/pcaps/heartbleed.pcap** (or you can download it from the running web server under the Packet Captures section – "Heartbleed Example" example.)

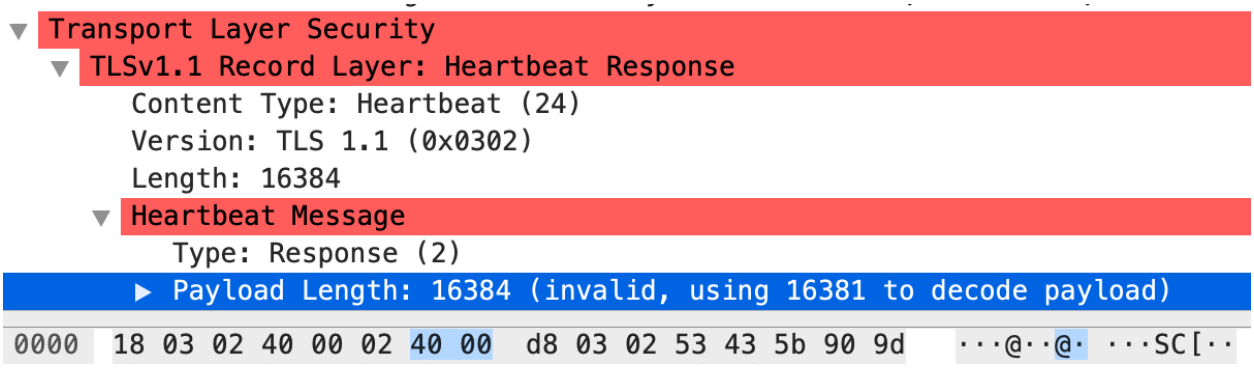
2. First comment out (with a "#" at the start of a line) any existing rules you have in **/etc/snort/rules/local.conf** (We're doing this to make reviewing the alert log easier.)
3. We're going to create a new rule to look for signs that a server is responding to an OpenSSL Heartbleed request with a large amount of data. This is a sign that the system responding is vulnerable to the Heartbleed vulnerability. The rule looks like:

```
alert tcp any 443 -> any any (msg: "OpenSSL Heartbleed response";
content:"|18 03 02|"; depth:3; dsize:>40; reference:cve,2014-0160;
sid:3;)
```

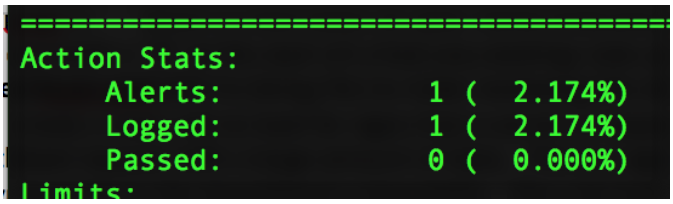
Rule breakdown:

- **`any 443 -> any any`** = Any IP address on port 443 sending traffic out
- **`msg`** = The signature name
- **`content:"|18 03 02|"`** = The packet has the hexadecimal values of "18 03 02" in the packet
- **`depth:3`** = Only look at the first 3 bytes for the content
- **`dsize:>40`** = Only look at packets that are larger than 40 bytes in size
- **`reference:cve,2014-0160`** = a CVE reference for the Heartbleed vulnerability. Useful metadata
- **`sid:3`** = The signature ID

Wireshark can be helpful to visualize the packet and how it all works. Highlighted here is the large payload size to help understand why the **`dsize:>40`** is important for this rule.



4. Run snort against the pcap file: **`sudo snort -c /etc/snort/snort.conf -A fast -l /var/log/snort -r /var/www/html/pcaps/heartbleed.pcap`**
5. If you review the output under "**Action Stats**" you should see 1 packet was alerted on and logged.



6. Review the log file – **`less /var/log/snort/alert`** or **`tail /var/log/snort/alert`** . You should see a log line like:

`06/24-06:40:46.159756 [**] [1:3:0] "OpenSSL Heartbleed response" [**] [Priority: 0] {TCP} 128.8.164.168:443 -> 192.168.103.128:50470`

If you don't see the above line in **/var/log/snort/alert** verify your `local.rules` file and that the rule has been entered correctly.

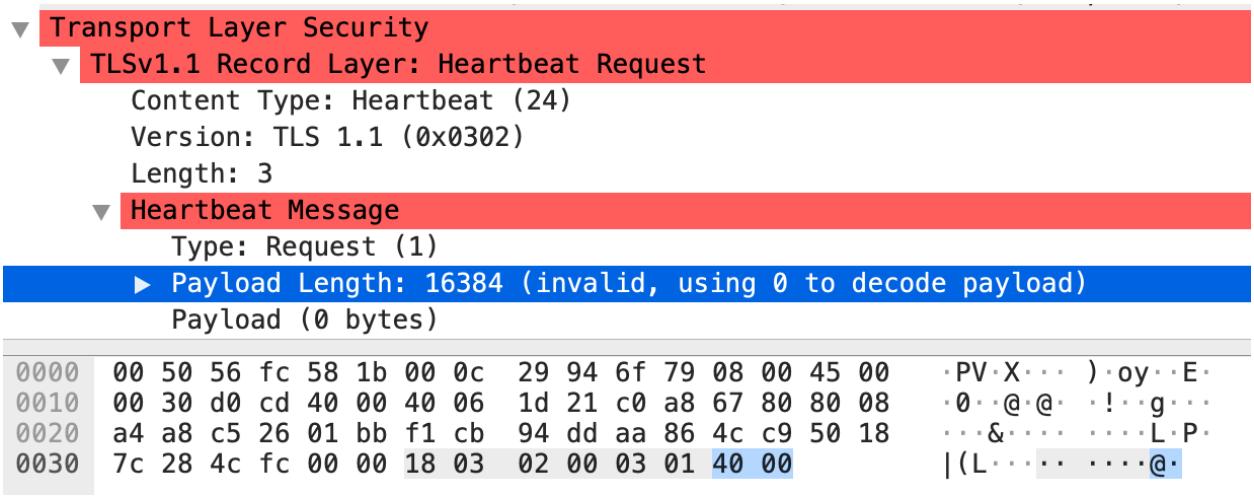
One last improvement, this only works for TLS v1.1. What can we do for other TLS versions? Two options here:

- Make a regular expression to match all of the SSL/TLS versions
- Multiple rules for each version of SSL/TLS

Which is better? In my opinion the multiple rules option is better since it is simple content matches which performance wise is many times faster than using regular expressions.

Why is this alert looking for a Heartbleed response vs a request? (Why is this looking for an exploit reply vs an exploit attempt?) Is one better than the other? Why?

How would you write a rule to detect an inbound Heartbleed exploit attempt? Packet #8 of **`heartbleed.pcap`** can be useful here to see what a Heartbleed exploit attempt looks like from a network packet perspective.



content:"|18 03 02|"; depth:3; This is from the previous rule, we want to check the first 3 bytes of the packet for a heartbeat TLS v 1.1 packet, this is marked with the hex **|18 03 02|**

content:"|01|"; distance:3; After the first match we want to jump 3 bytes ahead and if it is equal to **|01|** then we have a heartbeat request.

byte_test:2,>,50,3; The last test is looking for a large payload length. Most heartbeat messages are small so if it's larger than 50 (meaning the reply will be 50 bytes or more) this is most likely an exploit attempt.

The final rule to detect a TLS v1.1 based Heartbleed exploit attempt would like like:

```
alert tcp any 443 -> any any (msg: "OpenSSL Heartbleed exploit
attempt"; content:"|18 03 02|"; depth:3; content:"|01|"; distance:3;
byte_test:2,>,50,3; reference:cve,2014-0160;  sid:8;)
```

Zeek Exercises (20 minutes)

Some sample data used for this exercise is stored in the home directory of the account you used to build the VM used for class (most likely **`enpm685`**.)

NOTE: This exercise will utilize a number of built in Unix commands to process and manipulate data outputs if you are unfamiliar with them review the *Linux Primer* in ELMS.

1. Login to the Ubuntu VM with the account you used to build the VM for class

2. Change directory to zeek_exercise – **`cd zeek_exercise`**

3. Run **`ls`** to see a listing of the files, these are from after zeek processed a packet capture and processed out many of the various protocols. The original pcap is there along with a number of log files which are tab separated (similar to a CSV file) for various protocols and network connections from that packet capture.

4. The log files are plain text so you can view them with any text viewer. A "cheatsheet" for the log formats is available here: https://github.com/corelight/bro-cheatsheets/blob/master/Corelight-Bro-Cheatsheets-2.6.pdf

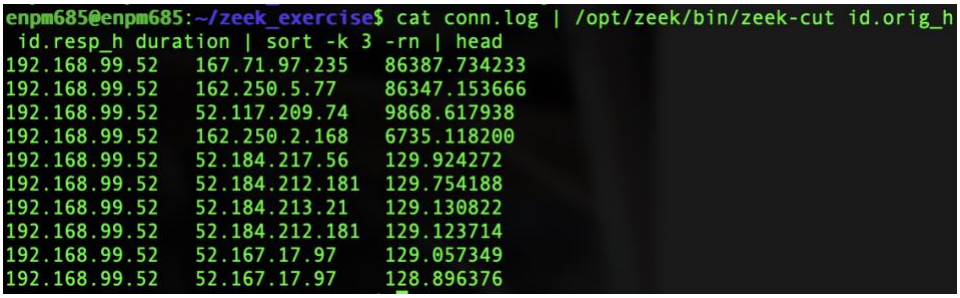
5. In the scenario for this exercise this is a packet capture of a section of a fictitious company's network and there is a suspected compromise by a sophisticated attacker. We want to process the logs looking for signs of an attacker inside the network.

6. We can review the connection log – conn.log – with **`less conn.log`**. While we may eventually find signs of an attack this way it's going to be very hard to find a needle in a haystack. (We could also analyze the ~309MB pcap but that's also trying to find a needle in a haystack.)

7. Let's look for long running connections – a typical sign of an ongoing command and control (C2) session from a compromised systems back out to an attacker's C2 infrastructure. We can do this with:

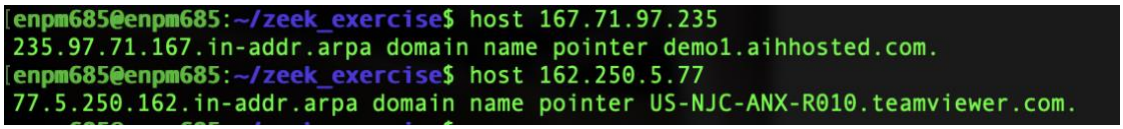
```
cat conn.log | /opt/zeek/bin/zeek-cut id.orig_h id.resp_h duration |
sort -k 3 -rn | head
```

8. Analyze the results:



We see 2 connections that have significantly long connection times. Let's find out more about each host.

9. Type **`host` *`ip.address`*** for the top 2 connections.



10. For this example, let's say that **demo1.aihosted.com** is a business partner/SaaS solution the organization uses and that is expected activity. But the other one?

11. What is TeamViewer? The organization does not use it. What is it used for? It is utilized by attackers? Do you think the organization has been compromised?