Run Zeek on FreeBSD Guide

Lars Wittebrood

Version 1.0, 9 March 2021: First version

Table of Contents

Preface

Welcome to the *Run Zeek on FreeBSD Guide*.

The goal of this book is to help you start using Zeek on FreeBSD. This guide explains how to run Zeek on FreeBSD. It is based on posts about Zeek on my blog. But this book covers more topics and provides more in depth information!

This book covers several different subjects about running Zeek on FreeBSD. It includes the installation of Zeek on FreeBSD, using Zeek Packages, make advanced queries on Zeek log files and much more.

Audience

This book is for FreeBSD system administrators who want to deploy and use Zeek in their network. It can also be used by network administrators who are familiar with Zeek but want to use FreeBSD as a platform for running it.

Prerequisites

It is assumed that the reader has basic FreeBSD system administrator knowledge and experience. The reader is encouraged to consult the FreeBSD Handbook for any additional required reading and or knowledge.

Scope

The subjects covered are arranged in the following chapters:

- **Chapter 1 Introduction**: This first chapter provides background information on what Zeek is, its purpose and why you want to run it.
- **Chapter 2 Implement Zeek**: All the steps to install Zeek on FreeBSD with a basic configuration are provided.
- **Chapter 3 Run Zeek as user `zeek`**: We make the initial Zeek installation a little more secure by running Zeek as a normal user `zeek` instead of running it as user `root`.
- **Chapter 4 The first Zeek queries**: The Zeek log files are explained and we make some simple queries on some of them.
- **Chapter 5 Implement Zeek packages**: We add some new functionalities by implementing Zeek Packages
- **Chapter 6 Monitor multiple interfaces with Zeek**: We add a Zeek package and change the Zeek configuration such that we monitor more than 1 interface on our Zeek host.
- **Chapter 7 More advanced Zeek queries**: To show more of the capabilities of Zeek we discuss some more, advanced, queries on the Zeek log files.
- **Chapter 8 Add Threat Intelligence**: We can add a threat intelligence capability to our Zeek implementation by using the Zeek Intelligence Framework.

- **Chapter 9 Manage Zeek on FreeBSD**: We state some important items to be aware of when implementing and running Zeek.
- **Chapter 10 Some final words**: Some final words about this book.
- **Appendix A Resources**: I've read a lot of blog posts, articles, or just documentation about Zeek to implement and run Zeek on FreeBSD and write this book. The most important ones are listed in this chapter.
- **Appendix B Left over queries**: Some queries on Zeek log files which are in my notes but did not make the book, but are intersting enough to mention.

Enjoy reading this book and good luck with your Zeek implementation on FreeBSD!

1. Introduction

A lot of organizations have only protective measures in place to secure their network, infrastructure and data. Examples of protective meaures are a firewall and an Intrusion Prevention System. But in 2021 it is not enough to have only protective measures in place. Today detective measures are also needed. The reason is that you have to take into account that you will get breached some day. Your protective measures do not stop all attacks and will eventually fail. And therefore you need detective measures, to be able to detect the attacks which have passed your protective fences.

Zeek is a network security monitoring platform which generates rich network metadata. Zeek is not an active security device. It sits on a *sensor*, a hardware, software, virtual, or cloud platform that quietly and unobtrusively observes network traffic. Zeek interprets what it sees and creates compact, high-fidelity transaction logs, file content, and fully customized output - rich network metadata. This metadata is very valuable for general network troubleshooting, getting insight in what happens on your network, and even for incident response and forensics!

Zeek is often used in a process called Network Security Monitoring (NSM).

Richard Bejtlich was one of the first people using the term Netwerk Security Monitoring. His definition of NSM is *the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions*. See his book *The Practise of Network Security Monitoring* (no starch press - 2013) for more details.

NSM involves collecting and analyzing all sorts of network related data from different sources and tools. This data in turn gives network administrators, security analysts and the like the opportunity to detect and respond to intruders in their network.

And Zeek very much helps in this process.

Personally I found Zeek also very helpfull in just understanding my network and environment better. It provides so much usefull information, not only for the security related insights, but also very much for general infrastructure insight and working of your platform. Think here about (not conclusive) NTP, DNS, SSL/TLS (certificates), DHCP, etc.. And this alone already provides a very good reason why you should add Zeek to your platform and make use of the information it provides!

So let's get our hands dirty now in the the remaining chapters and get Zeek implemented on FreeBSD!

2. Implement Zeek on FreeBSD

2.1. Introduction

The steps to implement Zeek on FreeBSD are described in this first chapter. The implementation covers the installation of the required FreeBSD packages and the configuration of the Zeek daemon.

2.2. Technical prerequisites

The following technical prerequisites have to be in place to be able to implement Zeek:

- an up to date and supported FreeBSD system version **12.x**
- NTP and DNS are setup correctly and working
- the sudo package is installed and configured such that the user performing the Zeek implementation has `root` rights
- the FreeBSD system must have at least 2 Network Interface Cards (NICs)
- 1 NIC is used for managing and using the system. This NIC has an IP address configured. This is interface `em0`.
- (at least) 1 NIC is used to to capture packets from a network. This NIC is connected to a SPAN port or network TAP. This NIC does NOT have an IP address configured. This is interface `em1`.

The required FreeBSD system can be a physical server, a virtual machine or even a FreeBSD jail.

2.3. Requirements

No software can be implemented in the right way without defining the requirements for its intended use. So I've also defined requirements for the Zeek implementation discussed here:

- implement Zeek on 1 host (you can implement Zeek in a distributed fashion on more hosts!)
- use JSON based logging [1]
- netmap is not used (not that much of network traffic is processed in my environment to be needing this!)
- the host needs access to the internet (DNS, HTTP and HTTPS) to be able to fetch and install software

2.4. The configuration of the interfaces

The FreeBSD host which runs Zeek has 2 NICs per the requirements above. Assuming that these 2 NICs are called `em0` and `em1`, these can be setup like this: [2]

```
$ sudo sysrc ifconfig_em0="inet 10.1.2.99 netmask 255.255.255.0"
$ sudo sysrc ifconfig_em1="monitor up"
```

Find below an example diagram of how a Zeek host can be connected to the network based on the configuration items used above.

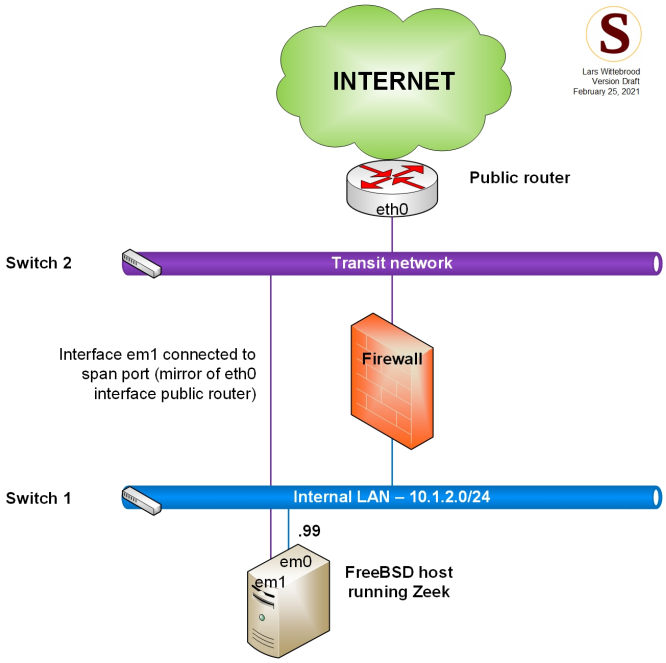

Figure 1. Example of a FreeBSD Zeek host connected to the network

2.5. Install the required packages

Before Zeek is installed and configured, some other software and packages are needed for this implementation:

- jq
- datamash

We need jq because we log in JSON format. And we want to be able to query these logs! And the best way to query JSON is using `jq`!

With datamash we can do calculations and stuff on the output of the queries we perform.

The commands to install these packages are:

```
$ sudo pkg install -y jq
$ sudo pkg install -y datamash
```

2.6. Install and configure Zeek

2.6.1. The base

Now it is time to install the Zeek software. We do this using the following command:

```
$ sudo pkg install -y zeek
```

The default Zeek package is good for most implementations. Please look at the `Makefile` of the Zeek port for configurable options.

We need to edit 3 files to configure Zeek. These files are:

- `/usr/local/etc/zeekctl.cfg`
- `/usr/local/etc/node.cfg`
- `/usr/local/etc/networks.cfg`

The `networks.cfg` file defines the local private networks of the environment `zeek` is installed in. The format of this file is like:

```
# Address range    Description
10.1.2.0/24        LAN
172.16.99.0/24     DMZ
192.168.1.0/24     WiFi
```

The `node.cfg` file defines the nodes of our zeek implementation. As per the requirement defined above we run `zeek` on 1 node and we have 1 interface which captures the network traffic (change the interface for your own configuration):

```
[zeek]
type=standalone
host=localhost
interface=em1
```

It is possible to monitor more than 1 interface with Zeek running on a single host. This is discussed later in this series.

The last configuration file is called `zeekctl.cfg` and configures the `zeekctl` utility. The following contents have been chosen for our implementation:

```
# Mail options
MailTo = admin@domain.tld
MailConnectionSummary = 1
MinDiskSpace = 5
MailHostUpDown = 1

# Logging options
LogRotationInterval = 86400
LogExpireInterval = 35day
StatsLogEnable = 1
StatsLogExpireInterval = 35

# Other options
StatusCmdShowAll = 0
CrashExpireInterval = 0
SitePolicyScripts = local.zeek
LogDir = /var/zeek/logs
SpoolDir = /var/zeek/spool
CfgDir = /usr/local/etc
```

The mail options speak for themselves. If you have a MTA running on your host (like e.g. the DMA), you can get a summary send to the address specified in the MailTo item when you have set the MailConnectionSummary setting to 1. With these logging options, the logs are roteted each day (86400 seconds) and the logs are kept for 35 days. The site policy script (a file called `local.zeek`) is in the `/usr/local/share/zeek/site` folder. The `SpoolDir` entry defines the directory of the current log files. The archived log files are in the directory `/var/zeek/logs` (`LogDir` configuration entry).

2.6.2. Enable JSON logging

So we have done the base setup of Zeek. Now we can add the logging in JSON format as we want. We do this by using the following command:

```
$ sudo tee -a /usr/local/share/zeek/site/local.zeek > /dev/null <<EOT
? # Enable JSON logging
? @load policy/tuning/json-logs
? EOT
```

2.6.3. Add a cron entry

To enable Zeek maintenance we need to add an entry to the `cron` daemon:

```
$ sudo echo "*/5 * * * * root /usr/local/bin/zeekctl cron" >> /etc/crontab
```

2.6.4. Start Zeek

To finish the base setup of Zeek we have to do one more step. And that is enable Zeek in the startup script `rc.conf`. We can do this using:

```
$ sudo sysrc zeek_enable="YES"
```

The base setup is ready now. So we can start Zeek and capture some network traffic with it. To start `zeek` we issue the command:

```
$ sudo service zeek deploy
```

We need this command as we have a new configuration. Every time the Zeek configuration changes you have to use the above command `sudo service zeek deploy`! In between you can start en stop `zeek` by using the `sudo service zeek start` and `sudo service zeek stop` commands.

Now we check if traffic is actually logged by Zeek by looking at the contents of the connections log file `conn.log`:

```
$ cd /var/zeek/logs/current
$ head -1 conn.log | jq -c
```

With the last command you should see the contents of the first logged connection in JSON format. It should look like the below:

```
{"ts": 1611708585.385558,"uid": "CWttSl429dBU80Tip6","id.orig_h":
"10.1.2.103","id.orig_p": 45341,"id.resp_h": "192.168.1.51","id.resp_p":
53,"proto": "udp","service": "dns","duration":
0.000509977340698242 2,"orig_bytes": 86,"resp_bytes": 163,"conn_state":
"SF","local_orig": true,"local_resp": true,"missed_bytes": 0,"history":
"Dd","orig_pkts": 2,"orig_ip_bytes": 142,"resp_pkts": 2, "resp_ip_bytes": 219}
```

2.6.5. The timestamp in the Zeek log files

The `ts` field in the Zeek logfiles is shown in Unix epoch, as shown in the `conn.log` example above. The Zeek configuration can be changed to show the time in the `ts` field in ISO8601 format. To do this please add the configuration lines shown below to the `/usr/local/share/zeek/site/local.zeek` configuration file:

```
redef LogAscii::json_timestamps = JSON::TS_ISO8601;
redef LogAscii::use_json = T;
```

The Zeek (JSON) logs log the UTC time in the `ts` field!

[1] the main reason why we want to log in JSON format is that in the future we want to ship the Zeek logs to a log management system.

[2] the assumption here is that the host running Zeek is connected to the 10.1.2.0/24 subnet. The em1 interface is connected to a SPAN port or network TAP.

3. Run Zeek as user zeek

3.1. Introduction

Zeek runs as user `root` by default when it is implemented on FreeBSD using the package system. But we want a more secure setup than this and run Zeek as a nomral user. Fortunatly this possible on our favorite operating system!

3.2. Configure Zeek to run as user zeek

Only a couple of commands are needed to configure Zeek to run as normal user `zeek` instead of `root` [3].

First we stop the `zeek` processes:

```
$ sudo service zeek stop
```

Then we add some lines to the `/etc/devfs.conf` file:

```
$ sudo tee -a /etc/devfs.conf > /dev/null <<EOT
? own     bpf     root:bpf
? perm    bpf     0660
? EOT
```

Now we create a new group called `bpf` and add the `zeek` user to it:

```
$ sudo pw groupadd -n bpf -g 81
$ sudo pw group mod bpf -m zeek
```

and then we restart the `devfs` service:

```
$ sudo service devfs restart
```

Next we configure to use the user `zeek` in `/etc/rc.conf`:

```
$ sudo sysrc zeek_user="zeek"
```

and check it:

```
$ cat /etc/rc.conf | grep zeek
zeek_enable="YES"
zeek_user="zeek"
```

Now we change the Zeek maintenance `cron` entry from user `root` to user `zeek`. First we remove the existing entry:

```
$ sudo sed '/zeekctl/d' /etc/crontab
```

And then we create a new one for the user `zeek`:

```
$ echo "zeek" | sudo tee -a /var/cron/allow > /dev/null
$ sudo echo "*/5 * * * * /usr/local/bin/zeekctl cron" > /var/cron/tabs/zeek
```

And we restart the `cron` daemon:

```
$ sudo service cron restart
```

We are almost ready! Last item to do is change the owner of the Zeek log directories:

```
$ sudo chown -R zeek:zeek /var/zeek/
```

And we can start the `zeek` processes:

```
$ sudo service zeek deploy
```

Important: Although you can start and stop `zeek` through the `zeekctl` command, my experience is that with Zeek running as the user `zeek` it is best to start, stop and deploy Zeek through the `service` command, as this will use the `sysrc` variables set in the system `rc.conf` configuration file!

[3] the user `zeek` is created when the `zeek` package is installed

4. First Zeek queries on FreeBSD

4.1. The Zeek log files

Zeek produces log files. It produces a seperate log file for each protocol, like e.g. HTTP and DNS. Besides the protocol logs, Zeek also produces a connection log `conn.log`, a known hosts log `known_hosts.log`, a notice log `notice.log` and many other non protocol logs. The Zeek log file overview provides a list of all the possible Zeek log files.

In our case the Zeek logs reside in the `/var/zeek/logs` and `/var/zeek/spool` directories. This is configured in the `zeekctl.cfg` configuration file (see chapter 2.6 Install and configure Zeek). And this is the standard FreeBSD configuration which comes with the Zeek package. The `/var/zeek/spool` directory holds the current log files and the `/var/zeek/logs` the archived log files. If you list the files in the `/var/zeek/spool/zeek` directory you will see all the current Zeek log files of your Zeek implementation.

The directory `/var/zeek/logs/current` is a link to the directory `/var/zeek/spool/zeek`

We will look at 3 log files here:

- `conn.log`: the log file with all the connections
- `dns.log`: the log file with all the DNS activity
- `ssl.log`: the log file with a record of all the SSL sessions, including certificates being used

Some example queries are discussed in the following chapters for each of these log files. And we start with the connection log file `conn.log`.

4.2. Query the connection log

The Zeek connection log `conn.log` provides a lot of information about all your TCP/UDP/ICMP connections. Our first query on `conn.log` shows the top 10 source IP addresses that generated the most network traffic:

First we go the the directory where the logs are:

```
$ cd /var/zeek/logs/current
```

Then we issue the query on the `conn.log` log file:

```
$ cat conn.log | jq -j '.["id.orig_h"], " ", "\n"' | \
  sort | uniq -c | sort -rn | head -n 10
```

The above command is explained in more detail below:

- `cat conn.log | jq -j '.["id.orig_h"],` selects the `id.orig_h` (is the source IP address of the connection) column from the `conn.log` log file
- `" ",` set a space between output columns
- `"\n"'` puts a newline character at the end of the line
- `| sort` uses the `sort` command to organize the rows in alphabetical order
- `| uniq -c` uses the `uniq` command with the `-c` option to remove duplicates while returning unique instances and their counts
- `| sort -rn` uses the `sort` command with the `-rn` option to organize the rows in reverse numerical order
- `| head -n 10` uses the `head` command with the `-n` option to display the 10 top most values.

We can also do this for destination UDP/TCP ports (column `id.resp_p` from the log file):

```
$ cat conn.log | jq -j '.["id.resp_p"], " ", "\n"' | \
  sort | uniq -c | sort -rn | head -n 10
```

Each log entry in the `conn.log` log file has also a `service` attribute, which we can query as well:

```
$ cat conn.log | jq -j '.["service"], " ", "\n"' | sort | uniq -c | sort -rn
```

These are just a couple of examples which information you can get out of the connection log file `conn.log`. Pretty amazing, right? And much more is possible! Here the current log file is used, but you can of course also query the archived connection log files in the `/var/zeek/logs/<YYYY-MM-DD>` directories.

4.3. Query the DNS log

Another log file is the `dns.log`. Zeek logs all DNS activity it sees in this log file. And we can query this log file! As first example we can make an overview of which domains have been queried with DNS (we are still in the `/var/zeek/logs/current` directory):

```
$ cat dns.log | jq -j '.["query"], " ", "\n"' | \
  sort | uniq | rev | cut -d '.' -f 1-2 | rev | \
  sort | uniq -c | sort -nr | head
```

Here is a breakdown of the above command:

- `cat dns.log | jq -j '.["query"],` : selects the query field, which tells us what domain was requested

- `" "`, set a space between output columns
- `"\n"` puts a newline character at the end of the line
- `| uniq` : remove all duplicate queries
- `| rev` : takes each query and reverses the string, so that `www.google.com` becomes `moc.elgoog.www`. The reason we do this is to strip the query down to the top level domain (TLD), like `.com` or `.net`, and the next portion of the domain.
- `| cut -d '.' -f 1-2` : Split the full query on every period and keep the first and second elements (e.g `moc.elgoog.www` => `moc.elgoog`).
- `| rev` : reverse the string again to bring it back to normal
- `| sort | uniq -c` : remove and count duplicates
- `| sort -nr | head` : output the entries with the most duplicates

Another query we can do on the `dns.log` log file is the number of unique queries for each DNS query type (i.e. A, NS, MX, TXT, etc.) using the `qtype_name` field in the `dns.log` log file:

```
$ cat dns.log | jq -j '.["qtype_name"], " ", "\n"' | \
  sort | \
  uniq -c | \
  sort -rn
```

The breakdown of this command is left to reader and can be done using the above breakdowns.

4.4. Query the SSL log

The `ssl.log` log file provides SSL/TLS handshake information. Each record has some valueble fields which we can query. The first field which we query is the `server_name` which is the URL connected to and we list the 10 most connected URLs:

```
$ cd /var/zeek/logs/current
$ cat ssl.log | jq -j '.["server_name"], " ", "\n"' | \
  sort | uniq -c | sort -rn | head -n 10
```

Another interesting field is the SSL/TLS version used. To list them with the amounts used we can issue the command:

```
$ cat ssl.log | jq -j '.version, " ", "\n"' | \
  sort | uniq -c | sort -rn | head -n 10
```

The breakdown of these commands is left to reader and can be done using the above breakdowns.

5. Implement Zeek packages on FreeBSD

5.1. Introduction

A Zeek package can contain Zeek scripts, Zeek plugins, or `ZeekControl` plugins. Any number or combination of those components can be included within a single package. You can write your own package(s). But the Zeek project also keeps a repository of already existing packages: Zeek Packages on Github. The Zeek Package Manager makes it easy for Zeek user to implement and manage Zeek Packages.

5.2. Install the Zeek Package Manager

The Zeek Package Manager is **not** available as a FreeBSD package. So we must implement it in another way. Luckely this is not difficult and is perfectly doable on FreeBSD. The Zeek Package Manager must be installed through the Python Package Installer so we have to install Pyhon first:

```
$ sudo pkg install python37
```

This does not install a `python` binary, so we make a `link`:

```
$ sudo ln -s /usr/local/bin/python3.7 /usr/local/bin/python
```

Now we can install the Python Package Installer and create a `link`:

```
$ sudo pkg install py37-pip
$ sudo ln -s /usr/local/bin/pip-3.7 /usr/local/bin/pip
```

We need git to be able to install Zeek Packages so we install that as well:

```
$ sudo pkg install git
```

Now we have everything in place to install the Zeek Package Manager itself and we install this using the `root` user (!):

```
$ sudo -i
# cd /root
# pip install zkg --user
# .local/bin/zkg autoconfig
# ln -s /root/.local/bin/zkg /usr/local/bin/zkg
```

That is it! We have installed the Zeek Package Manager. Now we can progress with actually installing the needed (or wanted) Zeek Packages!

5.3. Install Zeek packages

The Zeek Packages available can be found on the Zeek Packages Browser web site. Depending on your requirements and environment you can install any package listed here you want.

Here we install two Zeek Packages:

- ja3
- hassh

Both packages create fingerprints of encrypted traffic which I find very usefull. More details on both can be found in the links provided above.

The following commands are used to install these packages:

```
$ sudo -i
# zkg install zeek/salesforce/ja3
# zkg install zeek/salesforce/hassh
```

5.4. Enable the packages in Zeek

So we've installed the Zeek Packages we want. Now we have to enable them. We do this by adding one line of text to `/usr/local/share/zeek/site/local.zeek` configuration file:

```
$ sudo tee -a /usr/local/share/zeek/site/local.zeek > /dev/null <<EOT
? @load packages
? EOT
```

This one line enables all the packages we install, now or later. So we have to do this only once. Because we have changed the Zeek configuration we have to deploy it:

```
$ sudo service zeek stop
$ sudo service zeek deploy
```

5.5. Check if all ok

The ja3 package add 2 fields to the `ssl.log` log file called `ja3` and `ja3s`. And the hassh adds the following fields to the `ssh.log` log file: `hasshVersion`, `hassh`, `hasshServer` and `hasshAlgorithms`. If you see these fields in both log files the installation of both Zeek Packages succeeded!

6. Monitor multiple interfaces with Zeek

6.1. Introduction

So far we have configured Zeek to monitor only one network interface. But Zeek can also monitor multiple network interfaces on a host. In this chapter we explain how to change the Zeek configuration to monitor multiple interfaces.

6.2. Add a Zeek package

Before we change the Zeek configuration we install another Zeek Package:

```
$ sudo -i
# zkg install zeek/j-gras/add-interfaces
```

This package adds a field called `_interface` to the Zeek log file `conn.log`. If required you can add this field to any (other) Zeek log file by changing the configuration of the `zeek/j-gras/add-interfaces` package. The configuration file of this package is `/usr/local/share/zeek/site/packages/add-interfaces/add-interfaces.zeek`. To enable the `_interfaces` field for all Zeek log files change the `export` part to:

```
export {
   ## Enables interfaces for all active streams
   const enable_all_logs = T &redef;
   ## Streams not to add interfaces for
   const exclude_logs: set[Log::ID] = { } &redef;
   ## Streams to add interfaces for
   const include_logs: set[Log::ID] = {  } &redef;
}
```

6.3. Change the Zeek configuration

The Zeek configuration file `node.cfg` needs to change for Zeek to monitor more than 1 network interface. The `node.cfg` we used so far looks like:

```
[zeek]
type=standalone
host=localhost
interface=em1
```

Now we still want to use 1 server running Zeek, but want to use multiple interfaces to monitor network traffic. In the example below we use 3 network interfaces (`em1`, `em2` and `em3`):

```
[logger]
type=logger
host=localhost

[manager]
type=manager
host=localhost

[proxy-1]
type=proxy
host=localhost

[worker-1]
type=worker
host=localhost
interface=em1

[worker-2]
type=worker
host=localhost
interface=em2

[worker-3]
type=worker
host=localhost
interface=em3
```

6.4. Enable the changes

To enable the above change we have to deploy this new configuration again with the following commands:

```
$ sudo service zeek stop
$ sudo service zeek deploy
```

And we are done!

The layout of the directory structure where the Zeek log files reside, the directory `/var/zeek/spool`, changes after you have configured Zeek to monitor more interfaces. Each item from your `node.cfg` configuration file, logger, manager, proxy-1, worker-1, etc., is now visible in the `/var/zeek/spool` directory with its own subdirectory!

7. More advanced Zeek queries

7.1. Introduction

We have discussed some basic queries on some of the Zeek log files in chapter 4. To show more of the capabilities of Zeek we discuss some more, advanced, queries in this chapter. Each query discussed in this chapter has value in *every* environment, as all the queries discussed are generic and can be applied in every situation!

The advanced Zeek query topics covered below are:

- More DNS log queries
- Find bad clients

7.2. More DNS log queries

The DNS is a valuable resource to verify if malicious activity is happening on your network. Although there is a push to encrypt DNS, it is still largely unencrypted on local networks. And it remains one of the most effective methods for detecting malicious activity on your network. The DNS log file of Zeek provides a lot of valuable information about a DNS query and its response.

Below we cover the following DNS topics:

- large queries
- large responses
- responses with NXDOMAIN

7.2.1. Large queries

The length of a DNS query can indicate bad behaviour by the requestor. A large DNS query may be an indicator of possible DNS tunneling or exfiltration. Therefore it is good to measure the length of the DNS queries on your network. We can do this using Zeek in its simplest form by using the following BASH script:

```
#!/usr/local/bin/bash ①
varQueries=$(cat /var/zeek/spool/logger/dns.log | jq -j '.["query"], "\n"' |
sort -u) ②
for varQ in ${varQueries} ③
do
   varQLength=`echo ${varQ} | wc -c` ④
   if [ ${varQLength} -gt 60 ] ⑤
```

```
   then
      echo "The length of the query ${varQ} is " ${varQLength} ⑥
   fi
done
```

where:

① Make this a BASH script
② Declare the variable `varQueries` using a query on the Zeek `dns.log` log file
③ For every `query` entry in the `dns.log` (is the vriable `varQ`) we do
④ We determine the length of the query `varQ` (varQLength)
⑤ We check if the length of the query is greater than 60 characters
⑥ We print the length of the query `varQ` if the length is greater than 60 characters

The threshold to be used, in this example 60, is different for every environment. So please research and play with it and use what is best for your situation and environment.

7.2.2. Large responses

Whenever you query a DNS server, you get an answer back, the response. Also large responses can be an indicator of something wrong going on. Therefore it is good to measure the length of the DNS response on your network. We can do this using Zeek in its simplest form by using the following BASH script:

```
#!/usr/local/bin/bash ①
varAnswers=$(cat /var/zeek/spool/logger/dns.log | jq -j '.["answers"], "\n"' |
sort -u) ②
for varA in ${varAnswers} ③
do
   varALength=`echo ${varA} | wc -c` ④
   if [ ${varALength} -gt 100 ] ⑤
   then
      echo "The length of the answer ${varA} is " ${varALength} ⑥
   fi
done
```

where:

① Make this a BASH script
② Declare the variable `varAnswers` using a query on the Zeek `dns.log` log file
③ For every `answer` entry in the `dns.log` (is the vriable `varA`) we do
④ We determine the length of the answer `varA` (varALength)

⑤ We check if the length of the answer is greater than 100 characters

⑥ We print the length of the answer `varA` if the length is greater than 100 characters

The threshold to be used, in this example 100, is different for every environment. So please research and play with it and use what is best for your situation and environment.

7.2.3. Responses with NXDOMAIN

A DNS query doesn't always get a positive response. A response with *non existing domain* or NXDOMAIN is also possible. If you see a lot of these from a specific system it might be that the DNS settings of this system are wrong or is trying to resolve a malicious domain which does not exist any more.

We can filter the Zeek `dns.log` for these NXDOMAIN answers and make a list which clients has the most:

```
$ cd cat /var/zeek/spool/logger
$ cat dns.log | jq 'select(.["rcode_name"] == "NXDOMAIN")' | \
  jq -j '.["id.orig_h"], "\n"' |
  sort -nr | uniq -c | sort -rn | head -n 10
```

Assuming that the IP address of the client with the most NXDOMAIN answers is `10.10.10.116`, we can then look at what the DNS where:

```
$ cat dns.log | jq 'select(.["rcode_name"] == "NXDOMAIN")' | \
  jq 'select(.["id.orig_h"] == "10.10.10.116")' | \
  jq -j '.["id.orig_h"], ",", .["id.resp_h"], ",", .["proto"], ",",
.["query"], "\n"' | less
```

7.3. Find bad clients

Nowadays the majority of the network traffic on the internet, or any other network for that matter, is encrypted. This means that we cannot check the payload of the traffic for maliciousness. But there are still other ways to be able to find out if a client is infected with e.g. malware or not.

For *all* traffic coming from any client, so **including** encrypted traffic, we can measure the following network traffic characteristics:

- the length of the connections
- the number of connections
- the amount of data

If for any chosen timeframe, i.e. 5 minutes, 15 minutes or 1 hour, a client is high on all of these

3 lists, the possibility that this client is infected is quite high! Especially when one acts suspicious.

We can monitor each of these characteristics with Zeek! And the `conn.log` provides us all the needed information here!

7.3.1. Length of the connections

Having long lasting connections or TCP sessions may be an indication of malicious command and control (C2) activity. And therefore we want to know if these are happening. But out of the box Zeek is not configured correctly to be able to show long connections in the right way. This is explained in detail in an article on ActiveCountermeasures, so I will not repeat this. To overcome this we can change the Zeek configuration as explained in this article. What we need to do is add the below configuration line to the `local.zeek` configuration file in the `/usr/local/share/zeek/site` directory:

```
# Redefine activity timeout
redef tcp_inactivity_timeout = 60 min;
```

And because we have changed the Zeek configuration we have to `deploy` this:

```
$ sudo service zeek stop
$ sudo service zeek deploy
```

Now we can query the Zeek connection log file `conn.log` to get these long connections:

```
$ cd /var/zeek/logs/current ①
$ cat conn.log | \ ②
  jq 'select(.["_interface"] == "em2")' | \ ③
  jq -j '.["id.orig_h"], ",", .["id.resp_h"], ",", .["id.resp_p"], ",",
.proto, ",", .duration, "\n"' | \ ④
  grep -v "null" | grep -v ":" | tr ',' '\t' | \ ⑤
  awk 'BEGIN{ FS="\t" } ⑥
  { arr[$1 FS $2 FS $3 FS $4] += $5 } ⑦
  END{ for (key in arr) printf "%s%s%s\n", key, FS, arr[key] }' | \ ⑧
  sort -nrk 5 | head -n 10 ⑨
```

where:

① We change to the directory with the current log files

② Concatenate the `conn.log` log file

③ Select only traffic on interface `em2` from the log file

④ Query and show the `id.orig_h`, `id.resp_h`, `id.resp_p`, `.proto` and `.duration` fields and use the `,` as field seperator

⑤ Filter out `null` and `:` responses with `grep` and change the field seperator from `,` to '\t' using `tr`

⑥ Start of an `awk` script and we use the tab character as field seperator (FS). `BEGIN` means we do this only once

⑦ Here we create an array (named arr) and add up the duration (duration is our fifth field $5). Important here is that we use all first 4 fields ($1 through $4) as our array key. This means that as long as the source and destination IP addresses, destination port, and the protocol remain the same it will add the duration to the total. `awk` does this repeatedly for every line of data

⑧ Here we are looping through all the elements in the array and printing out the results. `END` signifies that awk only executes this instruction one time, after processing all the data.

⑨ Sort the numbers of the fifth field in reverse order and show the 10 highest

7.3.2. Number of connections

A lot of connections between two IP addresses in a certain time frame can slo be an indication of something wrong going on. Therefore it is good to be able to measure this:

```
$ cd /var/zeek/logs/current ①
$ cat conn.log | \ ②
  jq 'select(.["_interface"] == "em2")' | \ ③
  jq -j '.["id.orig_h"], ",", .["id.resp_h"], ",", .["id.resp_p"], ",",
.proto, "\n"' | \ ④
  grep -v "null" | grep -v ":" | tr ',' '\t' | \ ⑤
  awk 'BEGIN{ FS="\t" } ⑥
  { arr[$1 FS $2 FS $3 FS $4] += 1 } ⑦
  END{ for (key in arr) printf "%s%s%s\n", key, FS, arr[key] }' | \ ⑧
  sort -nrk 5 | head -n 10 ⑨
```

where

① We change to the directory with the current log files

② Concatenate the `conn.log` log file

③ Select only traffic on interface `em2` from the log file

④ Query and show the `id.orig_h`, `id.resp_h`, `id.resp_p` and `.proto` fields and use the `,` as field seperator

⑤ Filter out `null` and `:` responses with `grep` and change the field seperator from `,` to '\t' using `tr`

⑥ Start of an `awk` script and we use the tab character as field seperator (FS). `BEGIN` means we do this only once

⑦ Here we create an array (named arr) with the number of connections. Important here is that we use all first 4 fields ($1 through $4) as our array key and concatenate each equal occurance. This means that as long as the source and destination IP addresses, destination port, and the protocol remain the same it counts the connections under the same key. `awk` does this repeatedly for every line of data

⑧ Here we are looping through all the elements in the array and printing out the results. `END` signifies that awk only executes this instruction one time, after processing all the data.

⑨ Sort the numbers of the fifth field in reverse order and show the 10 highest

7.3.3. Amount of data

We want to make a list with source and destination IP addresses with the amoubnt of data tranferred between each unique pair of these addresses. To create such a list we issue the query on the `conn.log` log file below:

```
$ cd /var/zeek/logs/current
$ cat conn.log |
  jq 'select(.["_interface"] == "em2")' | \
  jq -j '.["id.orig_h"], ",", .["id.resp_h"], ",", .["orig_bytes"], "\n"' | \
  grep -v "null" | grep -v ":" | tr ',' '\t' | \
  awk 'BEGIN{ FS="\t" }
  { arr[$1 FS $2] += $3 }
  END{ for (key in arr) printf "%s%s%s\n", key, FS, arr[key] }' | \
  sort -nrk 3 | head -n 10
```

The principle of this command is the same as the queries above, so the breakdown of this one is left to the reader.

8. Add Threat Intelligence

8.1. Introduction

So far we have learned a lot about our network using Zeek and gained new insights as well. But Zeek has even more to offer. We can add a threat intelligence capability to our Zeek implementation by using the Zeek Intelligence Framework. With the Zeek Intelligence Framework we can give Zeek intelligence data and make it available such that we can match it with our own traffic which is captured in the Zeek log files. Data in the Zeek Intelligence Framework can be e.g. an IP address, a URL, a domain name or an e-mail address. Matched data is logged in the Zeek log file `intel.log`.

8.2. Threat Intelligence resources

The internet hosts a lot of threat intelligence resources. It is important that the indicators in the intelligence resources are well maintained and not stale, e.g. an IP address that was malicious a year ago, may be recycled and is benign today. You do not want a threat intelligence resource to provide bad and unreliable information.

And where can you find these threat intelligence resources? The Awesome Threat Intelligence resource list is a good place to start. It provides a curated list of sources, formats, frameworks and more. The Zeek Intelligence Framework requires the feeds it consumes to be in a specific format. A threat intelligence resource which provides a feed in the required Zeek format is from Critical Path Security. This feed is specifically tailored for Zeek by Critical Path Security and consists of well known and reliable sources like ABUSE|ch and Alienvault. It is also well maintained at the moment (9 March 2021). Updates appear daily! And that is what you want.

I show in the next chapter how to implement this feed for our Zeek instance.

8.3. Implement a Threat Intel resource

The `git` package is needed to install or pull the feeds from Github. We already installed this in chapter 5, but if you skipped that, we can do it here now:

```
$ sudo pkg install git
```

The steps to install the Critical Path Security feed are:

```
$ sudo -i
# cd /usr/local/share/zeek/site
# mkdir Zeek-Intelligence-Feeds
# git clone https://github.com/CriticalPathSecurity/Zeek-Intelligence-
Feeds.git ./Zeek-Intelligence-Feeds/
```

The Zeek script `main.zeek` in the `Zeek-Intelligence-Feeds` folder has some directory names which need to be fixed to reference the FreeBSD folder structure. We need to change `/usr/local/zeek/share/zeek` into `/usr/local/share/zeek`:

```
# cd Zeek-Intelligence-Feeds
# sed -i .bak 's+/usr/local/zeek/share/zeek+/usr/local/share/zeek+g' main.zeek
```

We are almost there! Next is to make sure Zeek loads the installed feeds. We do this by adding a line to the `/usr/local/share/zeek/site/local.zeek` script:

```
# cd /usr/local/share/zeek/site
# echo "@load Zeek-Intelligence-Feeds" >> local.zeek
```

The configuration part is done now. We can re-deploy the Zeek configuration now:

```
$ sudo service zeek stop
$ sudo service zeek deploy
```

As a last step in our initial implementation we check if the feeds are loaded:

```
$ cd /var/zeek/logs/current
$ grep "Intelligence" loaded_scripts.log
```

The outcome should be something like the below:

```
{"name":"  /var/zeek/spool/installed-scripts-do-not-touch/site/Zeek-
Intelligence-Feeds/__load__.zeek"}
{"name":"    /var/zeek/spool/installed-scripts-do-not-touch/site/Zeek-
Intelligence-Feeds/main.zeek"}
```

We are not completely finished yet. We have nothing in place to update the feed we just installed. But we can write a small script, which we schedule with a `cron` job, to do this for us.

Our script looks like the below:

```
#!/usr/local/bin/bash
# Variables
varGit="/usr/local/bin/git"
varRm="/bin/rm"
varTouch="/usr/bin/touch"
varChmod="/bin/chmod"
varSed="/usr/bin/sed"
```

```
varService="/usr/sbin/service"
varDate="/bin/date"

varIntelFeedsDir="/usr/local/share/zeek/site/Zeek-Intelligence-Feeds"
varLog="/tmp/_zeek-intel-feed-update.log"

varStateOK=0
varStateWARNING=1
varStateCRITICAL=2
varStateUNKNOWN=3

# Check root
if [ "$(id -u)" != "0" ]
then
   echo "This script must be run by root." 2>&1
   exit ${varStateWARNING}
fi

# The script
if [ -f ${varLog} ]
   then
      ${varRm} ${varLog}
   fi
${varTouch} ${varLog}
${varChmod} 0600 ${varLog}

${varDate} > ${varLog} 2>&1
cd ${varIntelFeedsDir} >> ${varLog} 2>&1
${varGit} fetch origin master >> ${varLog} 2>&1
${varGit} reset --hard FETCH_HEAD >> ${varLog} 2>&1
${varGit} clean -df >> ${varLog} 2>&1

${varSed} -i .bak 's+/usr/local/zeek/share/zeek+/usr/local/share/zeek+g'
main.zeek >> ${varLog} 2>&1

${varService} zeek stop >> ${varLog} 2>&1
${varService} zeek deploy >> ${varLog} 2>&1

${varDate} >> ${varLog} 2>&1
```

Assuming we name our script `zeek-intel-update.sh` and put it in the directory `/usr/local/sbin`, we can schedule this with the commands:

```
$ sudo echo "2 0 * * * root /usr/local/sbin/zeek-intel-update.sh" >>
/etc/crontab
$ sudo service cron restart
```

9. Manage Zeek on FreeBSD

9.1. Introduction

As with every daemon running on a Unix or FreeBSD system also Zeek needs to be managed.

In the below chapters we state some important items to be aware of when implementing and running Zeek.

9.2. Initial setup

The manual steps to implement Zeek on FreeBSD are described in chapter 2 of this book. For a single host deployment of Zeek this is probably just fine, but when you need to deploy multiple Zeek systems or need to implement a distributed Zeek deployment you are better of using an automation and configuration management tool like Ansible, Salt or Puppet. All 3 are in the FreeBSD Ports, such that you can run them.

We currently use Salt here at *SoCruel.NU*

The advantages of using an automation and configuration management tool are:

- your configurations are stored centrally and better documented
- you can deploy a new system faster and more precise then doing it manually from written down procedures
- you have the tools in place to be up and running more quickly when disaster strikes
- it can help to make day to day management tasks easier

The most important items to be aware of when deploying Zeek on FreeBSD are:

- make sure you design and have enough disk space available for the Zeek log files, best is to use seperate disks for the log files
- computing hardware is very powerfull today, but make sure you design and implement enough CPU and memory for your Zeek use case

9.3. Monitoring

When you have Zeek up and running you want to monitor a couple of items. This makes you aware that it runs smoothly and keeps running that way.

The most important items to monitor Zeek are:

- monitor the Zeek processes
- monitor the disk space and or disk usage

Both items can be done with monitoring tools available in the FreeBSD Ports. Examples are

Nagios, Incinga, Zabbix or Sensu.

We currently use Nagios here at *SoCruel.NU*

10. Some final words

This book covers only the tip of the iceberg looking at what is possible with Zeek. The goal of this book is by no means to cover *everything* there is to tell about running Zeek on FreeBSD. It is just to get the reader going with an initial implementation of Zeek on FreeBSD and covers basic functionality of Zeek out of the box. Some examples of topics which are not covered in this book, but are for sure very interesting, are:

- the Zeek File Analysis Framework
- forward Zeek log files to a central log solution like i.e. the ELK stack and do analysis there
- writing our own Zeek scripts
- design and implement a distributed Zeek platform
- show examples of real threat hunting examples with Zeek
- cover more Zeek log files

I consider this book a live document, so maybe not yet finished. So, who knows, one of these topics gets added to this book one day.

Nevertheless, I hope that you have enjoyed reading this book in it's current status and got out of it what you were after!

This book is created with Asciidoctor on FreeBSD using the Open Sans, Noto Serif and Courier Prime fonts using the *book* document type.

With kind regards,
Lars Wittebrood

Appendix A: Resources

I've read a lot of blog posts, articles, or just documentation about Zeek and also listened to some webinars to implement and run Zeek on FreeBSD and write this book. The most important ones of these are listed below:

- Zeek Documentation
- FreeBSD ifconfig monitor option
- Zeek Network Security Monitor Tutorial
- The jq command-line JSON processor
- Parsing Zeek JSON logs with jq
- Zeek log file overview
- Bro/Zeek log file cheat sheets
- Zeek Package Browser
- Zeek Package Manager
- JA3 - A method for profiling SSL/TLS Clients
- ActiveCountermeasures Threat Hunting Labs
- Zeek Lab Series
- Critical Path Security Zeek Intelligence Feed
- The Power of Open Source Zeek
- Examining aspects of encrypted traffic through Zeek logs

Appendix B: Some left over queries

Please find below some, left over, queries you can use on the Zeek log files, which have not been covered in the book. These are queries which are in my notes and did not make the book, but are interesting enough to be mentioned in this appendix!

What are the issuers of the ceritifcates used on my network?

```
$ cat ssl.log | jq -j '.["issuer"], "\n"' | grep -v "null" | sort | uniq -c |
sort -rn
```

The *total* number of HTTPS (TCP port 443) connections to a web server with IP address `10.12.14.80`:

```
$ cat conn.log | jq 'select(.["id.resp_h"] == "10.12.14.80")' | \
  jq 'select(.["id.resp_p"] == 443)' | \
  jq -j '.["id.orig_h"], ",", .["id.resp_h"], ",", .["id.resp_p"], ",",
.proto, "\n"' | \
  grep -v "null" | grep -v ":" | \
  tr ',' '\t' | \
  awk 'BEGIN{ FS="\t" } { arr[$1 FS $2 FS $3 FS $4] += 1 } END{ for (key in
arr) printf "%s%s%s\n", key, FS, arr[key] }' | \
  sort -nrk 5 | \
  datamash -W sum 5
```

Show the algorithms used to sign the certificates used in our network:

```
$ cat x509.log | jq -j '.["certificate.sig_alg"], " ", "\n"' | sort | uniq -c
| sort -nr
```

Get the HTTP user agents used in our network:

```
$ cat http.log | jq -j '.["user_agent"], " ", "\n"' | sort -u
```

Get all the TCP port 443 connections from a specific IP address (in this case `10.20.30.102`):

```
$ cat conn.log | jq 'select(.id.orig_h = "10.11.12.102" and .id.resp_p ==
"443" and .proto == "tcp")'
```