## **Overview**

This guide assumes you’ll be installing Zeek on [CentOS 8](http://www.centos.org/), given how popular CentOS tends to be in the enterprise.  However, the guide should work for any RHEL-based flavors of Linux.  For Debian-based systems, there will be some modifications required, including using apt-get vs yum for installing Linux packages.  Nothing that a search couldn’t help you figure out. 

Kicking things off, we’ll optimize CentOS to efficiently capture packets and then compile Zeek from source to start monitoring network traffic.

To do this, we’ll walkthrough these steps:

1. Enable the “network” service to apply network sniffing optimizations, including disabling NIC offloading functions to ensure Zeek sees full packet data and minimizes packet loss.
2. Setting interfaces to promiscuous mode to ensure all packets are captured and analyzed.
3. Install libmaxminddb to enable IP geolocation capability.
4. Build Zeek from source with optimizations.
5. Create a non-root Zeek user to minimize impact in the event that Zeek is compromised.
6. Deploy and run Zeek to start analyzing traffic.
7. Create a cron job to perform Zeek maintenance tasks.

## **Enable “network” service and disable NIC offloading functions**

1. Install the network-scripts package.

* sudo yum install network-scripts
* Use ethtool to determine the maximum ring parameters for your sniffing interfaces.  The example below assumes an interface named enp2s0.
* sudo ethtool -g enp2s0

Ring parameters for enp2s0:

Pre-set maximums:

RX: 4096

RX Mini: 0

RX Jumbo: 0

TX: 4096

Current hardware settings:

RX: 256

RX Mini: 0

RX Jumbo: 0

TX: 256

* As root/sudo, edit the /etc/sysconfig/network-scripts/ifcfg-<sniffinginterface> file for each sniffing network interface and change or add the following lines. Respectively, each line will disable control from NetworkManager, disable DHCP, and add appropriate ethtool options. Note that after “rx” you want to enter the maximum ring parameter as determined in the step above.

NM\_CONTROLLED=no

BOOTPROTO=none

ONBOOT=yes

IPV6INIT=no

ETHTOOL\_OPTS="-G ${DEVICE} rx <max ring parameter determined from step 1 above>; -K ${DEVICE} rx off; -K ${DEVICE} tx off; -K ${DEVICE} sg off; -K ${DEVICE} tso off; -K ${DEVICE} ufo off; -K ${DEVICE} gso off; -K ${DEVICE} gro off; -K ${DEVICE} lro off"

* Your file should now look something like this.

TYPE=Ethernet

BOOTPROTO=none

DEFROUTE=yes

IPV4\_FAILURE\_FATAL=no

IPV6INIT=no

IPV6\_AUTOCONF=yes

IPV6\_DEFROUTE=yes

IPV6\_PEERDNS=yes

IPV6\_PEERROUTES=yes

IPV6\_FAILURE\_FATAL=no

NAME=enp2s0

UUID=b22f5d92-3f1e-430b-b660-cb9376d8c0c0

DEVICE=enp2s0

ONBOOT=yes

PEERDNS=yes

PEERROUTES=yes

USERS=root

NM\_CONTROLLED=no

ETHTOOL\_OPTS="-G ${DEVICE} rx 4096; -K ${DEVICE} rx off; -K ${DEVICE} tx off; -K ${DEVICE} sg off; -K ${DEVICE} tso off; -K ${DEVICE} ufo off; -K ${DEVICE} gso off; -K ${DEVICE} gro off; -K ${DEVICE} lro off"

* Still as root/sudo, enable the “network” service.
* sudo systemctl enable network
* Finally, restart the “network” service.

1. sudo systemctl restart network

## **Set sniffing network interfaces to promiscuous mode**

1. As root/sudo, create /etc/systemd/system/promisc.service in your favorite text editor.
2. Add the following lines, assuming enp2s0 is your sniffing interface.

[Unit]

Description=Makes an interface run in promiscuous mode at boot

After=network.target

[Service]

Type=oneshot

ExecStart=/usr/sbin/ip link set dev enp2s0 promisc on

TimeoutStartSec=0

RemainAfterExit=yes

[Install]

WantedBy=default.target

1. Save the file and run the following commands to make the changes permanent and start on boot.

* sudo chmod u+x /etc/systemd/system/promisc.service

sudo systemctl start promisc.service

sudo systemctl enable promisc.service

Created symlink from /etc/systemd/system/default.target.wants/promisc.service to /etc/systemd/system/promisc.service.

* Reboot your system and verify all the changes made thus far have persisted. Verify that PROMISC is listed in the network interface status.

1. ip a show enp2s0 | grep -i promisc

3: enp2s0: < BROADCAST,MULTICAST,PROMISC,UP,LOWER\_UP > mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

## **Install Zeek Dependencies**

1. As root/sudo, edit /etc/yum.repos.d/CentOS-PowerTools.repo and set the “enabled” field to 1, to add the PowerTools repository. Your file should look something like this.

# CentOS-PowerTools.repo

#

# The mirror system uses the connecting IP address of the client and the

# update status of each mirror to pick mirrors that are updated to and

# geographically close to the client. You should use this for CentOS updates

# unless you are manually picking other mirrors.

#

# If the mirrorlist= does not work for you, as a fall back you can try the

# remarked out baseurl= line instead.

#

#

[PowerTools]

name=CentOS-$releasever - PowerTools

mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=PowerTools&infra=$infra

#baseurl=http://mirror.centos.org/$contentdir/$releasever/PowerTools/$basearch/os/

gpgcheck=1

enabled=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial

1. Add the EPEL repo.

* sudo yum --enablerepo=extras install epel-release
* Run the following yum command to download the required dependencies.
* sudo yum install cmake make gcc gcc-c++ flex bison jemalloc-devel libpcap-devel openssl-devel python3 python3-devel swig zlib-devel
* Ensure all your packages are up to date and reboot your system.

1. sudo yum update

sudo reboot

## **Configure GeoIP Support**

1. Install the libmaxminddb development library.

* sudo yum install libmaxminddb-devel
* Sign up for a [free Maxmind account](https://www.maxmind.com/en/geolite2/signup).  This is required [as of December 2019](https://blog.maxmind.com/2019/12/18/significant-changes-to-accessing-and-using-geolite2-databases/).
* [**Download**](https://www.maxmind.com/en/accounts/current/geoip/downloads) and untar the GeoLite2 database.
* tar xzvf GeoLite2-City.tar.gz
* Move the GeoLite2-City.mmdb file in the extracted GeoLite2-City\_YYYYMMDD directory to /usr/share/GeoIP.

1. sudo mv GeoLite2-City\_YYYYMMDD/GeoLite2-City.mmdb /usr/share/GeoIP/GeoLite2-City.mmdb

## **Create the zeek user and directory to install and run Zeek**

1. Create the zeek user and add it to the zeek group.

* sudo groupadd zeek

sudo useradd zeek -g zeek

* As root/sudo, set a password for the zeek user.
* sudo passwd zeek
* As root/sudo, create the /opt/zeek directory and set ownership to the zeek user.

1. sudo mkdir /opt/zeek

sudo chown -R zeek:zeek /opt/zeek

sudo chmod 750 /opt/zeek

## **Download, Compile, and Install Zeek**

1. Switch to the zeek user.

* su zeek
* We will download zeek to the /home/zeek directory. Then we will configure Zeek to install in the /opt/zeek directory and [**enable jemalloc**](http://jemalloc.net/) to improve memory and CPU usage.  As of this writing, the latest feature release is version 4.0.3.  If the download URL referenced in the wget command below no longer works, you can download the latest stable release directly from the [**Get Zeek download page**](https://zeek.org/get-zeek/).
* cd

wget https://download.zeek.org/zeek-4.0.3.tar.gz

tar -xzvf zeek-4.0.3.tar.gz

cd zeek-4.0.3

./configure --prefix=/opt/zeek --enable-jemalloc

make

make install

Note: This will take \*a while\* to compile.

* Switch back to your normal user by closing the zeek session.
* exit
* Since the zeek user is not root, give the Zeek binaries permissions to capture packets.

1. sudo setcap cap\_net\_raw=eip /opt/zeek/bin/zeek

sudo setcap cap\_net\_raw=eip /opt/zeek/bin/capstats

## **Add Zeek to PATH**

1. As root/sudo, create /etc/profile.d/zeek.sh and add the following.

pathmunge /opt/zeek/bin

1. Log out and log back in as the zeek user to update the PATH.

## **Configure Zeek**

1. Edit /opt/zeek/etc/node.cfg to configure the number of nodes.  It is recommended to use a maximum of one or two less workers than the total number of CPU cores available on your sensor.  In the example configuration below we are configuring a total of two workers, analyzing one sniffing interface.

# Example ZeekControl node configuration.

# Below is an example clustered configuration on a single host.

[logger]

type=logger

host=localhost

[manager]

type=manager

host=localhost

[proxy-1]

type=proxy

host=localhost

[worker-1]

type=worker

host=localhost

interface=enp2s0

[worker-2]

type=worker

host=localhost

interface=enp2s0

In the event you have two or more sniffing interfaces (e.g. enp2s0 and enp3s0), see the example configuration below which assigns each interface its own worker.

# Example ZeekControl node configuration.

# Below is an example clustered configuration on a single host.

[logger]

type=logger

host=localhost

[manager]

type=manager

host=localhost

[proxy-1]

type=proxy

host=localhost

[worker-1]

type=worker

host=localhost

interface=enp2s0

[worker-2]

type=worker

host=localhost

interface=enp3s0

1. Edit /opt/zeek/share/zeek/site/local.zeek to enable or disable scripts as needed.

## **Start Zeek**

1. As the zeek user, run zeekctl deploy to apply configurations and run Zeek.

* zeekctl deploy

checking configurations ...

installing ...

removing old policies in /opt/zeek/spool/installed-scripts-do-not-touch/site ...

removing old policies in /opt/zeek/spool/installed-scripts-do-not-touch/auto ...

creating policy directories ...

installing site policies ...

generating cluster-layout.zeek ...

generating local-networks.zeek ...

generating zeekctl-config.zeek ...

generating zeekctl-config.sh ...

stopping ...

stopping workers ...

stopping proxy ...

stopping manager ...

stopping logger ...

starting ...

starting logger ...

starting manager ...

starting proxy ...

starting workers ...

* If your output looks similar to what’s shown above, you should be good to go. To verify Zeek is running successfully, you can run zeekctl status.

zeekctl status

Name Type Host Status Pid Started

logger logger localhost running 1774 10 Oct 23:15:31

manager manager localhost running 1820 10 Oct 23:15:32

proxy-1 proxy localhost running 1865 10 Oct 23:15:33

worker-1-1 worker localhost running 1950 10 Oct 23:15:35

worker-1-2 worker localhost running 1951 10 Oct 23:15:35

worker-2-1 worker localhost running 1955 10 Oct 23:15:35

worker-2-2 worker localhost running 1954 10 Oct 23:15:35

If you see the following errors:

zeekctl deploy

Error: worker-1-1 terminated immediately after starting; check output with "diag"

Error: worker-1-2 terminated immediately after starting; check output with "diag"

Error: worker-2-1 terminated immediately after starting; check output with "diag"

Error: worker-2-2 terminated immediately after starting; check output with "diag"

Then try re-running the sudo setcap commands from earlier.

* sudo setcap cap\_net\_raw=eip /opt/zeek/bin/zeek

sudo setcap cap\_net\_raw=eip /opt/zeek/bin/capstats

* You should now see logs being generated in /opt/zeek/logs/current.
* ls -l

total 2276

-rw-rw-r--. 1 zeek zeek 1573 Oct 10 23:15 broker.log

-rw-rw-r--. 1 zeek zeek 593 Oct 10 23:45 capture\_loss.log

-rw-rw-r--. 1 zeek zeek 1970 Oct 10 23:15 cluster.log

-rw-rw-r--. 1 zeek zeek 673435 Oct 10 23:52 conn.log

-rw-rw-r--. 1 zeek zeek 580865 Oct 10 23:52 dns.log

-rw-rw-r--. 1 zeek zeek 3830 Oct 10 23:49 dpd.log

-rw-rw-r--. 1 zeek zeek 1406 Oct 10 23:47 files.log

-rw-rw-r--. 1 zeek zeek 26108 Oct 10 23:48 http.log

-rw-rw-r--. 1 zeek zeek 24646 Oct 10 23:15 loaded\_scripts.log

-rw-rw-r--. 1 zeek zeek 753 Oct 10 23:18 notice.log

-rw-rw-r--. 1 zeek zeek 187 Oct 10 23:15 packet\_filter.log

-rw-rw-r--. 1 zeek zeek 743 Oct 10 23:46 software.log

-rw-rw-r--. 1 zeek zeek 86512 Oct 10 23:51 ssl.log

-rw-rw-r--. 1 zeek zeek 5446 Oct 10 23:50 stats.log

-rw-rw-r--. 1 zeek zeek 0 Oct 10 23:15 stderr.log

-rw-rw-r--. 1 zeek zeek 188 Oct 10 23:15 stdout.log

-rw-rw-r--. 1 zeek zeek 240866 Oct 10 23:51 weird.log

* If you’re running into issues, zeekctl diag can provide more detailed output for troubleshooting purposes.

1. zeekctl diag

## **ZeekControl Cron**

ZeekControl features a cron command to check for and restart crashed nodes and to perform other maintenance tasks.  To take advantage of this, let’s set up a cron job.

1. Edit the crontab of the non-root zeek user.

* crontab -e
* Add the following to set up a cron job that runs every five minutes.  You can adjust the frequency to your liking.

\*/5 \* \* \* \* /opt/zeek/bin/zeekctl cron

## **Set up Zeek Package Manager**

1. As the zeek user, make sure you’re in its respective home directory.

* cd
* As the zeek user, install zkg’s dependencies.

pip3 install GitPython semantic-version --user

This will install two external Python modules that zkg requires to ~/.local/lib/python3.6/site-packages. If you try to run zkg prior to installing these modules OR if you try to run zkg from a directory in which the zeek user does not have write permissions, you will receive the following error message:

* zkg

error: zkg failed to import one or more dependencies:

\* GitPython: https://pypi.org/project/GitPython

\* semantic-version: https://pypi.org/project/semantic-version

If you use 'pip', they can be installed like:

pip3 install GitPython semantic-version

* As the zeek user, configure Zeek Package Manager (zkg).

1. zkg autoconfig

This will create a configuration file in /opt/zeek/etc/zkg/config. Upon completion it should look something like the following.

zeek = https://github.com/zeek/packages

[paths]

state\_dir = /opt/zeek/var/lib/zkg

script\_dir = /opt/zeek/share/zeek/site

plugin\_dir = /opt/zeek/lib64/zeek/plugins

zeek\_dist = /home/zeek/zeek-4.0.3

## **Install the AF\_PACKET package**

1. As root/sudo, run the following yum command to install kernel development files.

* sudo yum install kernel-devel
* Switch back to the zeek user and stop Zeek if it is currently running.
* zeekctl stop
* Use zkg to install the AF\_PACKET package.

1. zkg install zeek/j-gras/zeek-af\_packet-plugin

The following packages will be INSTALLED:

zeek/j-gras/zeek-af\_packet-plugin (2.1.2)

Proceed? [Y/n] y

Running unit tests for "zeek/j-gras/zeek-af\_packet-plugin"

Installing "zeek/j-gras/zeek-af\_packet-plugin".......

Installed "zeek/j-gras/bro-af\_packet-plugin" (2.1.2)

## **Configure Zeek to use AF\_PACKET**

1. Edit /opt/zeek/etc/node.cfg to configure Zeek to use AF\_PACKET.  In the example configuration below we are configuring one worker, load balanced across two cores, analyzing one sniffing interface.

# Example ZeekControl node configuration.

# Below is an example clustered configuration on a single host.

[logger]

type=logger

host=localhost

[manager]

type=manager

host=localhost

[proxy-1]

type=proxy

host=localhost

[worker-1]

type=worker

host=localhost

interface=af\_packet::enp2s0

lb\_method=custom

lb\_procs=2

pin\_cpus=0,1

In the event you have two or more sniffing interfaces (e.g. enp2s0 and enp3s0), see the example configuration below which assigns each interface its own worker, load balanced across two cores, again using AF\_PACKET. Note the addition of unique af\_packet\_fanout\_id values for each of the sniffing interfaces.

# Example ZeekControl node configuration.

# Below is an example clustered configuration on a single host.

[logger]

type=logger

host=localhost

[manager]

type=manager

host=localhost

[proxy-1]

type=proxy

host=localhost

[worker-1]

type=worker

host=localhost

interface=af\_packet::enp2s0

lb\_method=custom

lb\_procs=2

pin\_cpus=0,1

af\_packet\_fanout\_id=2

[worker-2]

type=worker

host=localhost

interface=af\_packet::enp3s0

lb\_method=custom

lb\_procs=2

pin\_cpus=2,3

af\_packet\_fanout\_id=3

1. As root/sudo, give the Zeek binaries permissions to capture packets. This was previously done in [Part I](https://www.ericooi.com/zeekurity-zen-part-i-how-to-install-zeek-bro-on-centos-8/), however, installing AF\_PACKET requires doing this again.

* sudo setcap cap\_net\_raw=eip /opt/zeek/bin/zeek

sudo setcap cap\_net\_raw=eip /opt/zeek/bin/capstats

* As the zeek user, run zeekctl deploy to apply configurations and run Zeek.

zeekctl deploy

If you see the following errors, try re-running the sudo setcap commands from the previous step.

1. zeekctl deploy

Error: worker-1-1 terminated immediately after starting; check output with "diag"

Error: worker-1-2 terminated immediately after starting; check output with "diag"

Error: worker-2-1 terminated immediately after starting; check output with "diag"

Error: worker-2-2 terminated immediately after starting; check output with "diag"

## **[Optional] Install Additional Useful Packages (e.g. add-interfaces, ja3, and HASSH)**

We’ll install additional Zeek packages: add-interfaces, ja3, and HASSH. The install process outlined below should work for installing other packages you may be interested in.

1. As the zeek user, stop Zeek if it is currently running.

* zeekctl stop
* Use zkg to install the add-interfaces package. In situations where you are monitoring multiple network interfaces on one sensor, this adds an “\_interface” field to every log file which [labels the particular network interface that traffic is coming from](https://github.com/J-Gras/add-interfaces/).
* zkg install zeek/j-gras/add-interfaces

The following packages will be INSTALLED:

zeek/j-gras/add-interfaces (master)

Proceed? [Y/n] y

Installed "zeek/j-gras/add-interfaces" (master)

Loaded "zeek/j-gras/add-interfaces"

* Edit /opt/zeek/share/zeek/site/add-interfaces/add-interfaces.bro and modify the const enable\_all\_logs and const include\_logs: set[Log::ID] fields as shown below. Save the file when you’re finished.

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;

}

* Use zkg to install the ja3 package. This is used for [profiling SSL/TLS clients.](https://github.com/salesforce/ja3)
* zkg install zeek/salesforce/ja3

The following packages will be INSTALLED:

zeek/salesforce/ja3 (master)

Proceed? [Y/n] y

Installing "zeek/salesforce/ja3"

Installed "zeek/salesforce/ja3" (master)

Loaded "zeek/salesforce/ja3"

* Use zkg to install the HASSH package. This is used for [profiling SSH clients and servers](https://github.com/salesforce/hassh).
* zkg install zeek/salesforce/hassh

The following packages will be INSTALLED:

zeek/salesforce/hassh (master)

Proceed? [Y/n] y

Installing "zeek/salesforce/hassh"

Installed "zeek/salesforce/hassh" (master)

Loaded "zeek/salesforce/hassh"

* Edit /opt/zeek/share/zeek/site/local.zeek and add the following lines to the bottom. This will load all packages you’ve installed.

# Load Zeek Packages

@load packages

* As the zeek user, run zeekctl deploy to apply configurations and run Zeek.

1. zeekctl deploy

## **Update Installed Zeek Packages**

1. As the zeek user, stop Zeek if it is currently running.

* zeekctl stop
* Use zkg to check for updated packages.
* zkg refresh

Refresh package source: zeek

No changes

Refresh installed packages

New outdated packages:

zeek/salesforce/hassh (master)

This indicates that zeek/salesforce/hassh needs to be updated.

* Use zkg to check for updated packages.
* zkg upgrade

The following packages will be UPGRADED:

zeek/salesforce/hassh (master)

Proceed? [Y/n] y

Upgraded "zeek/salesforce/hassh" (master)

* As the zeek user, run zeekctl deploy to apply configurations and run Zeek.

zeekctl deploy

## **Output Zeek logs to JSON**

1. Stop Zeek if it is currently running.

* zeekctl stop
* Edit /opt/zeek/share/zeek/site/local.zeek and add the following.

# Output to JSON

@load policy/tuning/json-logs.zeek

* Restart Zeek and view the logs in /opt/zeek/logs/current to confirm they are now in JSON format.

1. zeekctl deploy

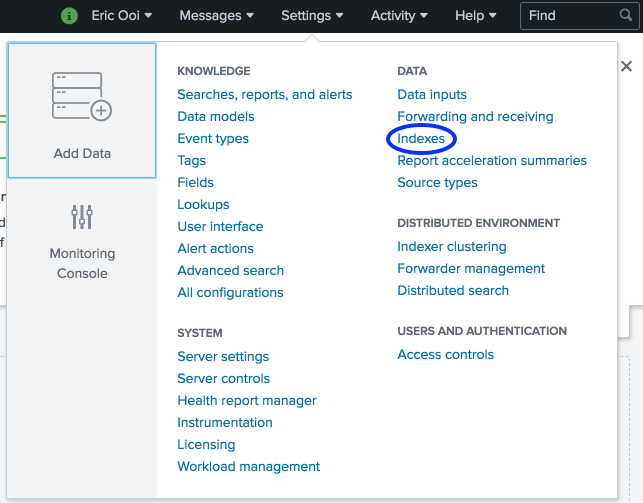
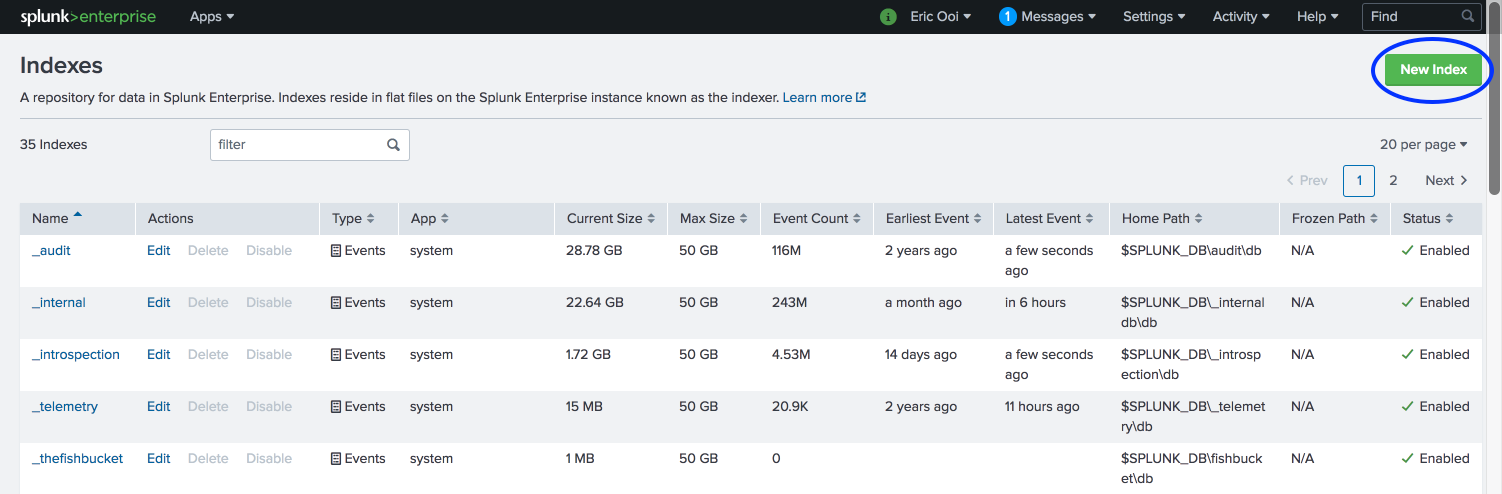
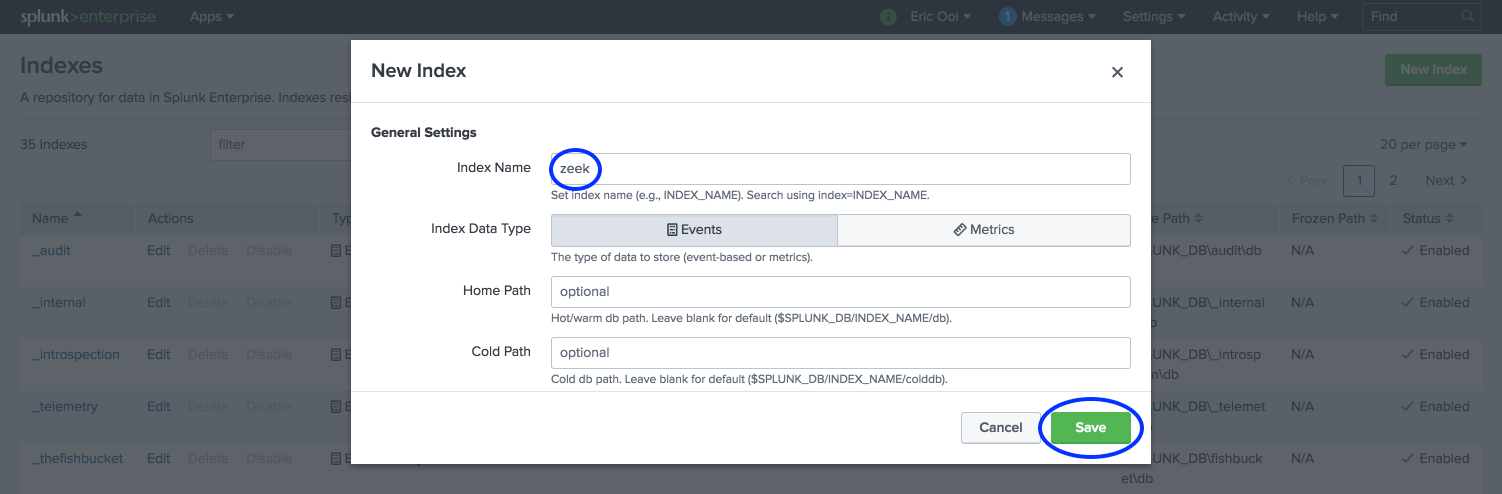
cd /opt/zeek/logs/current

less conn.log

## **Next. Log to SPLUNK**

## **Create an index in Splunk for Zeek data**

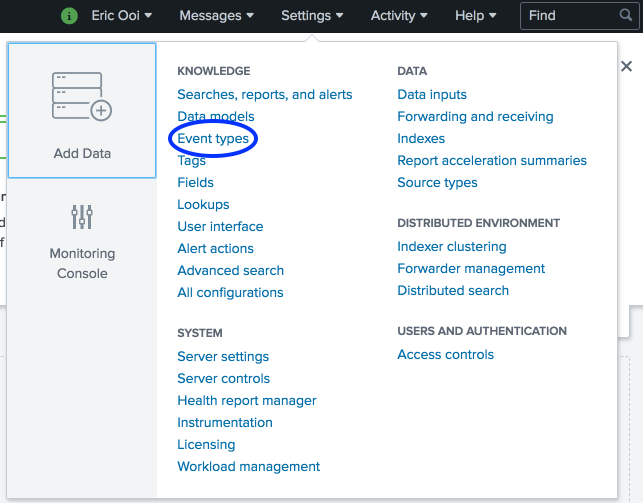
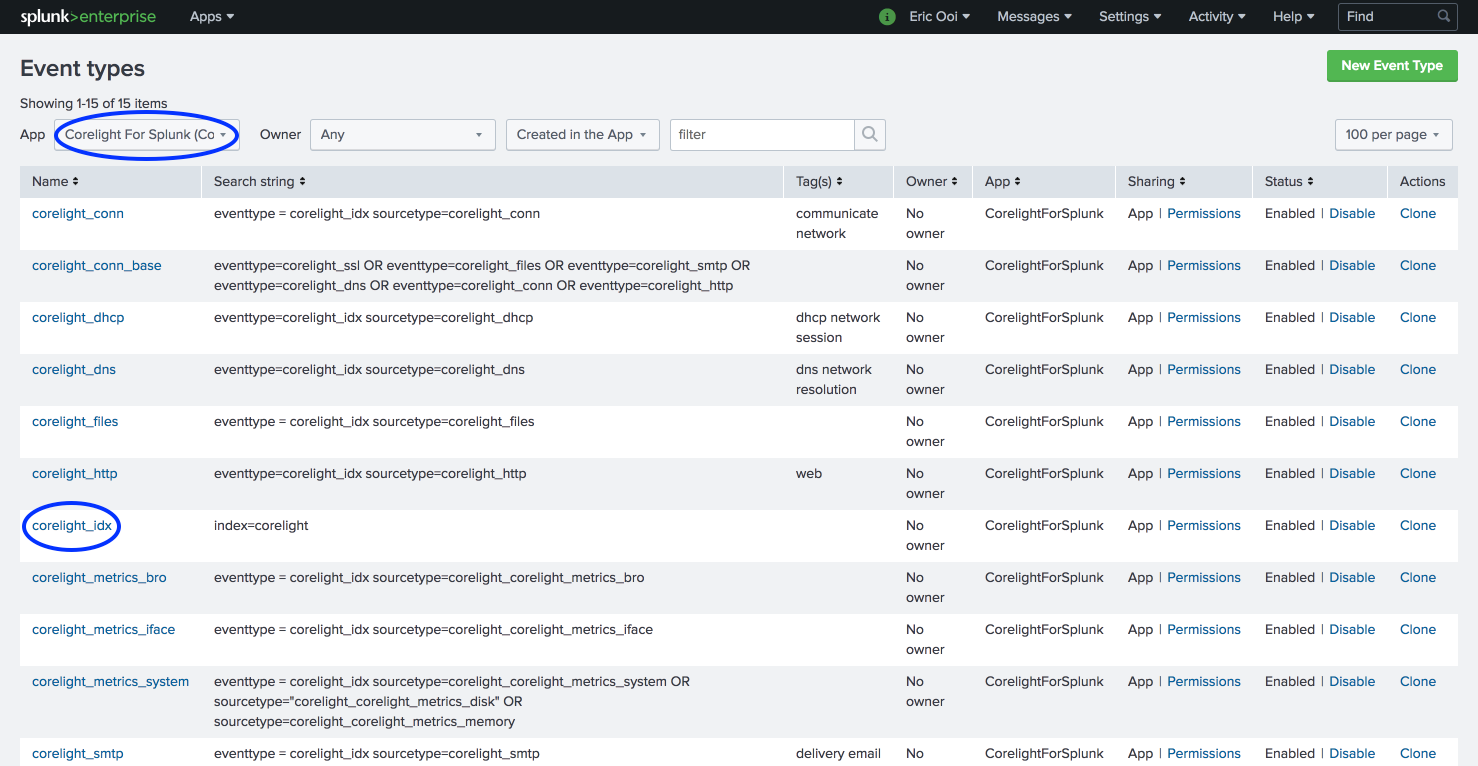
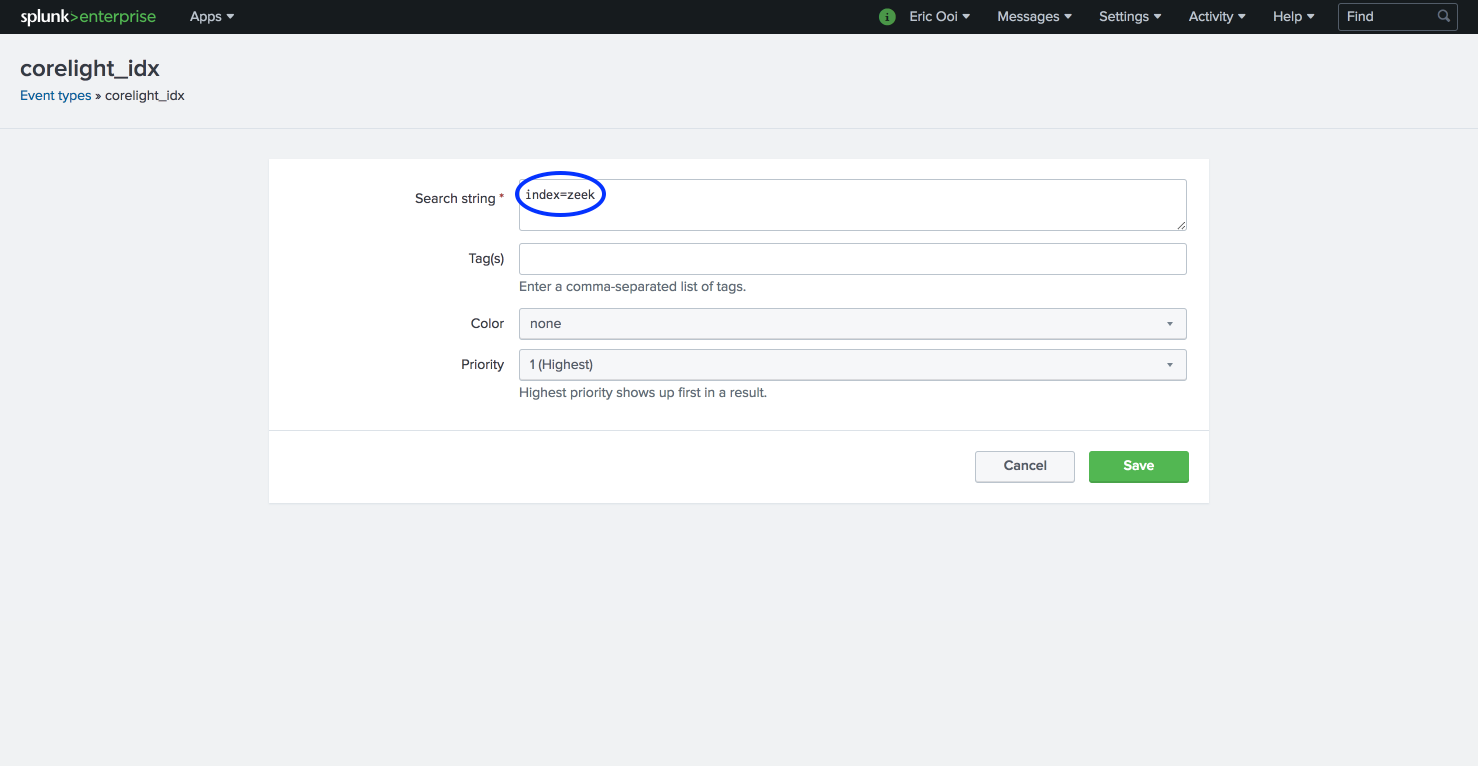
It’s best practice to create separate indexes for different types of Splunk data.

1. Login to your Splunk instance.
2. In the top right menu navigate to Settings -> Data -> Indexes.  
   [](https://www.ericooi.com/wp-content/uploads/2019/01/Settings_Indexes.png)
3. In the Indexes page, click on New Index.  
   [](https://www.ericooi.com/wp-content/uploads/2019/01/New_Index.png)
4. Type “zeek” for Index Name and click Save to create your new index.  
   [](https://www.ericooi.com/wp-content/uploads/2019/01/Zeek_Index.png)

## **[Optional] Install and configure the Corelight For Splunk app**

The [Corelight For Splunk](https://splunkbase.splunk.com/app/3884/) app is developed by the Corelight team for use with Corelight (enterprise Zeek) and open-source Zeek sensors. We’ll use this app to help parse, index, and visualize Zeek logs. Note that it is completely optional to use this app. You are free to skip this section entirely.

Note that Splunk has also published their own [Splunk Add-on for Zeek aka Bro](https://splunkbase.splunk.com/app/1617/) app which helps to ingest Zeek logs but does not feature any sort of dashboards or reports.

1. Download and install the [Corelight for Splunk](https://splunkbase.splunk.com/app/3884/) app onto your Splunk server.  This can either be done within the Splunk server itself or by manually downloading and installing as you would all other Splunk apps.
2. You can navigate to the app to verify that it is installed correctly.  However, since we have not yet configured our sensor to send data, the dashboards will be blank.
3. In the top right menu navigate to Settings -> Knowledge -> Event types.[](https://www.ericooi.com/wp-content/uploads/2019/01/Settings_Event_types.png)
4. In the App dropdown menu, select Corelight For Splunk and click on corelight\_idx.  
   [](https://www.ericooi.com/wp-content/uploads/2019/01/Corelight_Event_types-1.png)
5. In the Search string field type index=zeek.  This tells the Corelight for Splunk app to search for data in the “zeek” index we created earlier.  
   [](https://www.ericooi.com/wp-content/uploads/2019/01/Corelight_Index.png)

## **Create the splunk user to run the Splunk Universal Forwarder**

1. Back in the Zeek sensor, create a splunk user and add it to the splunk and zeek groups.

* sudo groupadd splunk

sudo useradd splunk -g splunk -G zeek

* As root/sudo, set a password for the splunk user.

1. sudo passwd splunk

## **Install and configure a Splunk Universal Forwarder**

1. Login to your Splunk account and download the latest [**Splunk Universal Forwarder**](https://www.splunk.com/en_us/download/universal-forwarder.html).  Once logged in, click “Download Now” for the Linux 64-bit .rpm installer.  Note that Splunk also generates a convenient wget command you can use from the sensor itself once you accept the license agreement. As of this writing, the latest release is version 8.2.0.  If the download URL referenced in the wget command below no longer works, download directly as noted above.

* wget -O splunkforwarder-8.2.0-e053ef3c985f-linux-2.6-x86\_64.rpm 'https://www.splunk.com/bin/splunk/DownloadActivityServlet?architecture=x86\_64&platform=linux&version=8.2.0&product=universalforwarder&filename=splunkforwarder-8.2.0-e053ef3c985f-linux-2.6-x86\_64.rpm&wget=true'
* Install the forwarder to /opt/splunkforwarder using the rpm command.
* sudo rpm -i splunkforwarder-<…>-linux-2.6-x86\_64.rpm
* As root/sudo set ownership of /opt/splunkforwarder to the splunk user.
* sudo chown -R splunk:splunk /opt/splunk
* Switch to the splunk user.
* su splunk
* Start the forwarder to accept the license agreement and create an administrative password.
* cd /opt/splunkforwarder/bin

./splunk start --accept-license

* Stop the forwarder.
* ./splunk stop
* Remove the default data processing limit.  Edit /opt/splunkforwarder/etc/system/local/limits.conf and add the following lines.Note that given the volume of data that Zeek generates, the forwarder may never process all log data if the default limit is not removed.

[thruput]

maxKBps = 0 # means unlimited

* Edit /opt/splunkforwarder/etc/system/local/inputs.conf to monitor desired Zeek logs.  An example inputs.conf is below but may or may not include the logs you wish to ingest. Note that the index and sourcetype fields are leveraging the “zeek” naming scheme to match the “zeek” index we created in Splunk. If you intend to use the Corelight For Splunk app, you’ll want to replace the “zeek” sourcetype prefix with “corelight” as this is what the app is expecting (e.g. replace “zeek\_conn” with “corelight\_conn”). Modify the index and sourcetype configurations to your needs.

[default]

host = sensor

[monitor:///opt/zeek/logs/current/conn.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_conn

[monitor:///opt/zeek/logs/current/dns.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_dns

[monitor:///opt/zeek/logs/current/software.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_software

[monitor:///opt/zeek/logs/current/smtp.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_smtp

[monitor:///opt/zeek/logs/current/ssl.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_ssl

[monitor:///opt/zeek/logs/current/ssh.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_ssh

[monitor:///opt/zeek/logs/current/x509.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_x509

[monitor:///opt/zeek/logs/current/ftp.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_ftp

[monitor:///opt/zeek/logs/current/http.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_http

[monitor:///opt/zeek/logs/current/rdp.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_rdp

[monitor:///opt/zeek/logs/current/smb\_files.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_smb\_files

[monitor:///opt/zeek/logs/current/smb\_mapping.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_smb\_mapping

[monitor:///opt/zeek/logs/current/snmp.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_snmp

[monitor:///opt/zeek/logs/current/sip.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_sip

[monitor:///opt/zeek/logs/current/files.log]

\_TCP\_ROUTING = \*

index = zeek

sourcetype = zeek\_files

* Edit /opt/splunkforwarder/etc/system/local/outputs.conf to send data to your Splunk server.  In the sample file below, replace each instance of splunkserver:9997 with your own server name/IP and port number.

[tcpout]

defaultGroup = default-autolb-group

[tcpout:default-autolb-group]

server = splunkserver:9997

[tcpout-server://splunkserver:9997]

* Start the forwarder as the splunk user and confirm it successfully ran.  You can check /opt/splunkforwarder/var/log/splunk/splunkd.log for any issues.

cd /opt/splunkforwarder/bin

./splunk start

## **Enable Zeek’s Intelligence Framework**

1. Edit /opt/zeek/share/zeek/site/local.zeek and add the following lines to the bottom. This enables the Intelligence Framework and tells Zeek to load intelligence from /opt/zeek/intel/zeek\_intel.txt.

# Load Zeek Intelligence Framework

@load policy/frameworks/intel/seen

redef Intel::read\_files += { "/opt/zeek/intel/zeek\_intel.txt" };

1. As the zeek user, create the /opt/zeek/intel directory.

* mkdir /opt/zeek/intel
* As the zeek user, stop zeek.
* zeekctl stop
* As the zeek user, apply the new settings and start zeek.

1. zeekctl deploy

## **Create a simple intelligence file**

1. As the zeek user, open your favorite file editor to create a text file and save it to /opt/zeek/intel/zeek\_intel.txt. This file must be tab delimited, including the #fields header row. The indicator and indicator\_type fields are required while the meta headers are optional and can include any additional information you feel would be useful to describe an indicator. For our purposes, we will populate our test file with zeek.org as the indicator and Intel::DOMAIN as the indicator\_type. Note that there are [several options for indicator\_type](https://docs.zeek.org/en/current/scripts/base/frameworks/intel/main.zeek.html" \l "type-Intel::Type), depending on the indicator you intend to use.

#fields indicator indicator\_type meta.source meta.desc

zeek.org Intel::DOMAIN zeek-intel-test Zeek Intelligence Framework Test

Note: If you copy and paste my example above, your file will only have spaces. Given this, you will need to manually ensure that all rows are tab delimited, including the #fields header row.

Note: You can continue adding new indicators to this file without restarting Zeek. A restart is required only if you want to remove an indicator.

## **Review intelligence match results**

1. With our newly created intel file, visit zeek.org to see what happens. If all went well, you should now have a newly generated intel.log file in your current log directory. Below is a sample intel.log file in JSON format showing three separate log entries.

{

"ts": 1593537714.074326,

"uid": "CNQFv73BiKAd14YY11",

"id.orig\_h": "10.2.2.23",

"id.orig\_p": 44425,

"id.resp\_h": "10.2.2.1",

"id.resp\_p": 53,

"seen.indicator": "zeek.org",

"seen.indicator\_type": "Intel::DOMAIN",

"seen.where": "DNS::IN\_REQUEST",

"seen.node": "worker-1-2",

"matched": [

"Intel::DOMAIN"

],

"sources": [

"zeek-intel-test"

]

}

{

"ts": 1593537714.182831,

"uid": "CvtoU61JPTygBQUb3c",

"id.orig\_h": "10.2.2.23",

"id.orig\_p": 51786,

"id.resp\_h": "192.0.78.212",

"id.resp\_p": 443,

"seen.indicator": "zeek.org",

"seen.indicator\_type": "Intel::DOMAIN",

"seen.where": "SSL::IN\_SERVER\_NAME",

"seen.node": "worker-1-2",

"matched": [

"Intel::DOMAIN"

],

"sources": [

"zeek-intel-test"

]

}

{

"ts": 1593537714.182831,

"uid": "CvtoU61JPTygBQUb3c",

"id.orig\_h": "10.2.2.23",

"id.orig\_p": 51786,

"id.resp\_h": "192.0.78.212",

"id.resp\_p": 443,

"seen.indicator": "zeek.org",

"seen.indicator\_type": "Intel::DOMAIN",

"seen.where": "X509::IN\_CERT",

"seen.node": "worker-1-2",

"matched": [

"Intel::DOMAIN"

],

"sources": [

"zeek-intel-test"

],

"fuid": "FWJEqd4JisAzgNZD42",

"file\_mime\_type": "application/x-x509-user-cert",

"file\_desc": "192.0.78.212:443/tcp"

}

1. Most of the fields should look familiar, so we’ll focus on the new fields.
   * seen.indicator (e.g., zeek.org): The indicator that we told Zeek to look for.
   * seen.indicator\_type (e.g., Intel::DOMAIN): The indicator’s type.
   * seen.where (e.g., DNS::IN\_REQUEST, SSL::IN\_SERVER\_NAME, X509::IN\_CERT): Where Zeek detected the indicator.  We see that there are three distinct log entries showing that Zeek found “zeek.org” in a DNS request, in a TLS server name, and in a X.509 certificate.
   * seen.node (e.g., worker-1-2): The Zeek node that detected the indicator.
   * matched (e.g., Intel::DOMAIN): The indicator type that was matched.
   * sources (e.g., zeek-intel-test): What we populated for the meta.source field in our zeek\_intel.txt file.

## **Use actionable intelligence sources with automation**

We used a simple example above to demonstrate how to use Zeek’s Intelligence Framework to generate a log of intelligence matches.  But how do we take this to the next level and instead use large actionable intelligence data sets?  How do we also scale our process to automatically consume and generate intelligence files that Zeek can ingest?

To answer these questions, we can leverage the [awesome-threat-intelligence](https://github.com/hslatman/awesome-threat-intelligence) list which includes intelligence sources, formats, frameworks and platforms, and tools.  It’s a mix of open source and commercial intelligence resources. If you’re brand new to threat intelligence, the list can be overwhelming. If you’re absolutely unsure where to start, I suggest checking out [MISP](https://www.misp-project.org/) — an open source threat intelligence platform.  It supports a wide variety of sources and formats, and can output intelligence to Zeek and other security platforms.  Otherwise, read through the descriptions and note any sources or platforms that interest you.  When it comes to intelligence sources, you want indicators that are well maintained and not stale (e.g., an IP address that was malicious a year ago, may be recycled and benign today).  In terms of frameworks, you’ll want to choose one that can ingest a variety of intelligence sources/formats and can output this data into Zeek-formatted intelligence.

## **Enable file hashing and Team Cymru’s Malware Hash Registry lookups**

1. By default, automatic file hashing and Team Cymru’s Malware Hash Registry lookups are enabled.  To confirm this, open /opt/zeek/share/zeek/site/local.zeek and look for the following lines. Ensure they appear as below and the @load lines are not commented out (e.g., do not have a # symbol in front). Update the file if needed.

# Enable MD5 and SHA1 hashing for all files.

@load frameworks/files/hash-all-files

# Detect SHA1 sums in Team Cymru's Malware Hash Registry.

@load frameworks/files/detect-MHR

## **Enable SHA256 hashing for all files**

1. SHA256 hashing is not enabled by default.  We will enable this by creating a simple Zeek script.  As the zeek user, create a new file /opt/zeek/share/zeek/site/hash\_sha256.zeek, add the following lines, and then save the file.

##! Perform SHA256 hashing on all files.

@load base/files/hash

event file\_new(f: fa\_file)

{

Files::add\_analyzer(f, Files::ANALYZER\_SHA256);

}

1. As the zeek user, edit /opt/zeek/share/zeek/site/local.zeek, add the following lines, and then save the file.

# Add SHA256 hash for files

@load hash\_sha256

1. As the zeek user, stop zeek.

* zeekctl stop
* As the zeek user, apply the new settings and start zeek.

1. zeekctl deploy

## **Understand files.log**

1. Take a look at your own files.log and note the types of files that are hashed.  Below is a sample files.log file in JSON format.

{

"ts": 1597593633.224633,

"fuid": "FB4Sx62yaleypxnhIb",

"tx\_hosts": [

"23.246.2.148"

],

"rx\_hosts": [

"10.2.2.23"

],

"conn\_uids": [

"CUgYfkjoZLP4BR8Ol"

],

"source": "HTTP",

"depth": 0,

"analyzers": [

"JPEG",

"SHA1",

"MD5",

"SHA256"

],

"mime\_type": "image/jpeg",

"duration": 0.01756000518798828,

"local\_orig": false,

"is\_orig": false,

"seen\_bytes": 58175,

"total\_bytes": 58175,

"missing\_bytes": 0,

"overflow\_bytes": 0,

"timedout": false,

"md5": "0671e92b0fb8ffe5724579c229a43689",

"sha1": "e855561e88f0bc57733eafa05a9d7681d276e55a",

"sha256": "fc58cf109988af3b3dbc499001ff300584eff638cb120405558d3df69c22fdf4"

}

1. Let’s examine some of the key fields to better understand how we can use them to analyze files on our own network.  For a full listing, check out the [official Zeek documentation](https://docs.zeek.org/en/current/scripts/base/frameworks/files/main.zeek.html" \l "type-Files::Info).
   * fuid (e.g., FB4Sx62yaleypxnhIb): The file’s unique ID.  Note that this is not the same as the uid commonly found in other Zeek logs.
   * tx\_hosts (e.g., 23.246.2.148): The host that transferred the file.
   * rx\_hosts (e.g., 10.2.2.23): The host that received the file.
   * conn\_uids (e.g., CUgYfkjoZLP4BR8Ol): This is equivalent to the uid or unique ID that’s used to correlate activity across conn.log and other Zeek logs.
   * source (e.g., HTTP): This indicates which protocol the file was transferred over.
   * analyzers (e.g., JPEG, SHA1, MD5, SHA256): The file analyzers used to analyze this file.
   * mime\_type (e.g., image/jpeg): What Zeek believes the MIME type of the file is.
   * seen\_bytes (e.g., 58175): The number of bytes that Zeek observed.
   * total\_bytes (e.g., 58175): The total number of bytes that the file should be.
   * missing\_bytes (e.g., 0): The number of bytes that were missing in the analysis, likely due to dropped packets.
   * overflow\_bytes (e.g., 0): The number of bytes that were not analyzed either due to overlapping bytes or reassembly errors.
   * md5 (e.g., 0671e92b0fb8ffe5724579c229a43689): The MD5 hash of the file.
   * sha1 (e.g., e855561e88f0bc57733eafa05a9d7681d276e55a): The SHA1 hash of the file.
   * sha256 (e.g., fc58cf109988af3b3dbc499001ff300584eff638cb120405558d3df69c22fdf4): The SHA256 hash of the file.

## **Enable automatic file extraction**

1. As the zeek user, stop Zeek if it is currently running.

* zeekctl stop
* Use zkg to install the [**file extraction package**](https://github.com/hosom/file-extraction).
* zkg install zeek/hosom/file-extraction

The following packages will be INSTALLED:

zeek/hosom/file-extraction (2.0.3)

Proceed? [Y/n] y

Installing "zeek/hosom/file-extraction".

Installed "zeek/hosom/file-extraction" (2.0.3)

Loaded "zeek/hosom/file-extraction"

* Configure file extraction options by editing /opt/zeek/share/zeek/site/file-extraction/config.zeek. Below is a sample config.zeek that will set the directory to store extracted files to /opt/zeek/extracted/ and set the files we want to automatically extract to commonly exploited file types (e.g., Java, PE, Microsoft Office, and PDF).

# All configuration must occur within this file.

# All other files may be overwritten during upgrade

module FileExtraction;

# Configure where extracted files will be stored

redef path = "/opt/zeek/extracted/";

# Configure 'plugins' that can be loaded

# these are shortcut modules to specify common

# file extraction policies. Example:

# @load ./plugins/extract-pe.bro

@load ./plugins/extract-common-exploit-types

* Create the directory to save all extracted files. It must match what we set in config.zeek.
* mkdir /opt/zeek/extracted
* If this is your first time installing a Zeek package, edit /opt/zeek/share/zeek/site/local.zeek and add the following lines to the bottom. This will load all packages you’ve installed. You will only need to do this once.

# Load Zeek Packages

@load packages

* As the zeek user, apply the new settings and start zeek.

1. zeekctl deploy

## **Real World Example**

So how could we use this in the real world? Imagine a user was sent a malicious link via their email that claimed to be this quarter’s employee bonus payouts.  The user proceeds to click on this link and immediately downloads a file.  We want to know whether the file was malicious and if so, determine what actions we can take to prevent other systems from downloading the same file.  Since we’ve got our Zeek instance automatically configured to hash all files, extract Windows PE files, and perform Team Cymru Malware Hash Registry lookups, we’re confident that we can perform a thorough analysis of the event.

1. We’re first alerted to suspicious activity through an alert raised in notice.log. The log entry below tells us the file’s MIME type is “application/x-dosexec”, that the notice is in regards to a “TeamCymruMalwareHashRegistry::Match”, and that there’s a Team Cymru detection rate of 38%. Additionally, the notice provides a direct [**VirusTotal link to the suspicious file**](https://www.virustotal.com/en/search/?query=5ff465afaabcbf0150d1a3ab2c2e74f3a4426467) that shows virtually every scanner detecting this file as malicious.  From the detection names, we see that this is related to the WannaCry ransomware. The notice also conveniently tells us where the file originated from (149.202.220.122) and which host downloaded the file (10.2.2.23).

{

"ts": 1597850503.829048,

"uid": "CO3tTx2lknzNvQe7P3",

"id.orig\_h": "10.2.2.23",

"id.orig\_p": 56197,

"id.resp\_h": "149.202.220.122",

"id.resp\_p": 80,

"fuid": "F1sCdV2rXJ9afKdlP2",

"file\_mime\_type": "application/x-dosexec",

"file\_desc": "http://s000.tinyupload.com/download.php?file\_id=91645583928538055155&t=9164558392853805515507216",

"proto": "tcp",

"note": "TeamCymruMalwareHashRegistry::Match",

"msg": "Malware Hash Registry Detection rate: 38% Last seen: 2020-06-05 08:29:39",

"sub": "https://www.virustotal.com/en/search/?query=5ff465afaabcbf0150d1a3ab2c2e74f3a4426467",

"src": "10.2.2.23",

"dst": "149.202.220.122",

"p": 80,

"peer\_descr": "worker-1-2",

"actions": [

"Notice::ACTION\_LOG"

],

"suppress\_for": 3600

}

1. Using the uid (CO3tTx2lknzNvQe7P3) from the notice, let’s search our logs for related activity and see what comes up.  You could search for this in Splunk or use grep to search through your raw logs.  Assuming we use grep, we find related activity in conn.log, http.log, and files.log as shown below.
   * conn.log  
     First, we confirm the connection metadata detailed in notice.log and observe that the file was transferred via HTTP.

{

"ts": 1597850493.368458,

"uid": "CO3tTx2lknzNvQe7P3",

"id.orig\_h": "10.2.2.23",

"id.orig\_p": 56197,

"id.resp\_h": "149.202.220.122",

"id.resp\_p": 80,

"proto": "tcp",

"service": "http",

"duration": 113.54712104797363,

"orig\_bytes": 624,

"resp\_bytes": 3514699,

"conn\_state": "RSTR",

"local\_orig": true,

"local\_resp": false,

"missed\_bytes": 0,

"history": "ShADadfr",

"orig\_pkts": 1398,

"orig\_ip\_bytes": 73512,

"resp\_pkts": 2433,

"resp\_ip\_bytes": 3641211

}

* + http.log  
    Next, we see that the user (10.2.2.23) made a GET request to s000.tinyupload.com to download a file.  Note the file information that Zeek includes in this log, the file’s unique ID (F1sCdV2rXJ9afKdlP2), the file’s name (bonus.exe), and the file’s MIME type (application/x-dosexec).

{

"ts": 1597850493.556732,

"uid": "CO3tTx2lknzNvQe7P3",

"id.orig\_h": "10.2.2.23",

"id.orig\_p": 56197,

"id.resp\_h": "149.202.220.122",

"id.resp\_p": 80,

"trans\_depth": 1,

"method": "GET",

"host": "s000.tinyupload.com",

"uri": "/download.php?file\_id=91645583928538055155&t=9164558392853805515507216",

"referrer": "http://s000.tinyupload.com/index.php?file\_id=91645583928538055155",

"version": "1.1",

"user\_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Safari/537.36",

"request\_body\_len": 0,

"response\_body\_len": 3514368,

"status\_code": 200,

"status\_msg": "OK",

"tags": [],

"resp\_fuids": [

"F1sCdV2rXJ9afKdlP2"

],

"resp\_filenames": [

"bonus.exe"

],

"resp\_mime\_types": [

"application/x-dosexec"

]

}

* + files.log  
    Finally, we again see the same file information that the http.log provided — unique ID, name, and MIME type.  But now we also see the MD5, SHA1, and SHA256 hashes of the file.  Since we’ve also enabled automatic file extraction for commonly exploited file types, we see a new field named “extracted” that tells us where Zeek extracted a copy of the file to (/opt/zeek/extracted/HTTP-F1sCdV2rXJ9afKdlP2.exe).  Note that the filename is formatted SOURCE-fuid.  We confirm that “seen\_bytes” matches “total\_bytes” and that there are zero “missing\_bytes”, ultimately telling us that Zeek was able to successfully analyze and fully extract the file in its entirety.

{

"ts": 1597850493.672357,

"fuid": "F1sCdV2rXJ9afKdlP2",

"tx\_hosts": [

"149.202.220.122"

],

"rx\_hosts": [

"10.2.2.23"

],

"conn\_uids": [

"CO3tTx2lknzNvQe7P3"

],

"source": "HTTP",

"depth": 0,

"analyzers": [

"SHA1",

"EXTRACT",

"PE",

"MD5",

"SHA256"

],

"mime\_type": "application/x-dosexec",

"filename": "bonus.exe",

"duration": 10.055749893188477,

"local\_orig": false,

"is\_orig": false,

"seen\_bytes": 3514368,

"total\_bytes": 3514368,

"missing\_bytes": 0,

"overflow\_bytes": 0,

"timedout": false,

"md5": "84c82835a5d21bbcf75a61706d8ab549",

"sha1": "5ff465afaabcbf0150d1a3ab2c2e74f3a4426467",

"sha256": "ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa",

"extracted": "/opt/zeek/extracted/HTTP-F1sCdV2rXJ9afKdlP2.exe",

"extracted\_cutoff": false

}

1. From here, we can use our endpoint security systems to determine if the user executed the file or examine additional Zeek logs to identify subsequent suspicious behavior.  To prevent other systems from downloading this file, we can block the identified file hashes or IP/URL in our network and endpoint security platforms.  Additionally, since we have a copy of the raw file we can perform deeper analysis and generate additional IOCs and threat intelligence, further strengthening our defenses.  Pretty cool, huh?

## **Troubleshooting**

If you find that files aren’t properly captured in files.log or automatically extracted, there are two likely causes:

1. You’re not actually performing full packet capture. In [Part I of this series](https://www.ericooi.com/zeekurity-zen-part-i-how-to-install-zeek-on-centos-8/), we enabled network optimizations to ensure your sensor is performing full packet capture and not utilizing any “NIC offloading functions.”  Refer to the steps in the section titled “Enable network service and disable NIC offloading functions” and confirm they’re applied properly on your system.  Zeek will typically warn you in reporter.log if it believes that NIC offloading functions have not been disabled.
2. You’re dropping packets. This could be due to an underpowered Zeek sensor or an overwhelmed network mirror/tap.  Make sure your Zeek sensor uses appropriately sized hardware for the traffic it’s monitoring and that your network mirror/TAP is capable of handling your network’s traffic volume.

## **Enable X.509 Logging of The Full Certificate Chain**

1. By default, Zeek only logs the X.509 certificate of the actual host and will exclude any intermediary certificates.  To enable logging of the full certificate chain, we need to edit /opt/zeek/share/zeek/site/local.zeek and comment out the following lines by adding a # symbol in front.  Then save the file.

# This script prevents the logging of SSL CA certificates in x509.log

# @load protocols/ssl/log-hostcerts-only

1. As the zeek user, stop zeek.

* zeekctl stop
* As the zeek user, apply the new setting and start zeek.

1. zeekctl deploy

## **Understand ssl.log and x509.log**

As we noted earlier, Zeek cannot directly decrypt encrypted SSL/TLS traffic, but that doesn’t prevent it from observing the SSL/TLS handshakes or capturing certificate details in cleartext.

1. Take a look at your own ssl.log and note the encrypted SSL/TLS connections and details that are captured.  Below is a sample ssl.log file in JSON format.

{

"ts": 1600552002.022206,

"uid": "CY188jv7sKlzAMNNc",

"id.orig\_h": "10.0.1.8",

"id.orig\_p": 54458,

"id.resp\_h": "104.244.42.1",

"id.resp\_p": 443,

"version": "TLSv12",

"cipher": "TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256",

"curve": "secp256r1",

"server\_name": "twitter.com",

"resumed": false,

"next\_protocol": "h2",

"established": true,

"cert\_chain\_fuids": [

"FzHo3q14WMvHTjCc79",

"F5woTi2O9Ea05rRxhj"

],

"client\_cert\_chain\_fuids": [],

"subject": "CN=twitter.com,OU=atla,O=Twitter\\, Inc.,L=San Francisco,ST=California,C=US",

"issuer": "CN=DigiCert SHA2 High Assurance Server CA,OU=www.digicert.com,O=DigiCert Inc,C=US",

"validation\_status": "ok",

"ja3": "b32309a26951912be7dba376398abc3b",

"ja3s": "8d2a028aa94425f76ced7826b1f39039"

}

1. Let’s examine some of the key fields to better understand how we can use them to analyze SSL/TLS connections on our own network.  For a full listing, check out the [official Zeek documentation](https://docs.zeek.org/en/current/scripts/base/protocols/ssl/main.zeek.html" \l "type-SSL::Info).
   * uid (e.g., CY188jv7sKlzAMNNc): This is equivalent to the uid or unique ID that’s used to correlate activity across conn.log and other Zeek logs.
   * version (e.g., TLSv12): The SSL/TLS version the server chose for the connection.
   * cipher (e.g., TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256): The SSL/TLS cipher suite the server chose for the connection.
   * curve (e.g., secp256r1): The elliptic curve the server chose when using ECDH/ECDHE.
   * server\_name (e.g., twitter.com): The hostname listed in the Server Name Indication (SNI) extension.
   * resumed (e.g., false): Indicates if the connection was resumed or not.
   * next\_protocol (e.g., h2): The protocol the server chose to perform over the secure connection via application-layer protocol negotiation (ALPN).  In this example, “h2” refers to “HTTP/2.”
   * established (e.g., true): Indicates whether the encrypted session was successfully established.
   * cert\_chain\_fuids (e.g., FzHo3q14WMvHTjCc79, F5woTi2O9Ea05rRxhj): The unique file IDs (fuids) of all certificates offered by the server.
   * subject (e.g., CN=twitter.com,OU=atla,O=Twitter\\, Inc.,L=San Francisco,ST=California,C=US): The subject of the certificate.
   * issuer (e.g., CN=DigiCert SHA2 High Assurance Server CA,OU=www.digicert.com,O=DigiCert Inc,C=US): The issuer/signer of the certificate.
   * validation\_status (e.g., ok): Indicates the status of a certificate (e.g., ok, expired, self-signed, etc.).
2. We can find the associated X.509 certificates by searching on the cert\_chain\_fuids. Below is a sample x509.log file in JSON format for each of the two cert\_chain\_fuids from our ssl.log above.  
   cert\_chain\_fuid: FzHo3q14WMvHTjCc79

{

"ts": 1600552002.062889,

"id": "FzHo3q14WMvHTjCc79",

"certificate.version": 3,

"certificate.serial": "0FBABC6DFD509287B4B260CE67C6F292",

"certificate.subject": "CN=twitter.com,OU=atla,O=Twitter\\, Inc.,L=San Francisco,ST=California,C=US",

"certificate.issuer": "CN=DigiCert SHA2 High Assurance Server CA,OU=www.digicert.com,O=DigiCert Inc,C=US",

"certificate.not\_valid\_before": 1580968800,

"certificate.not\_valid\_after": 1612548000,

"certificate.key\_alg": "rsaEncryption",

"certificate.sig\_alg": "sha256WithRSAEncryption",

"certificate.key\_type": "rsa",

"certificate.key\_length": 2048,

"certificate.exponent": "65537",

"san.dns": [

"twitter.com",

"www.twitter.com"

],

"basic\_constraints.ca": false

}

cert\_chain\_fuid: F5woTi2O9Ea05rRxhj

{

"ts": 1600552002.062889,

"id": "F5woTi2O9Ea05rRxhj",

"certificate.version": 3,

"certificate.serial": "04E1E7A4DC5CF2F36DC02B42B85D159F",

"certificate.subject": "CN=DigiCert SHA2 High Assurance Server CA,OU=www.digicert.com,O=DigiCert Inc,C=US",

"certificate.issuer": "CN=DigiCert High Assurance EV Root CA,OU=www.digicert.com,O=DigiCert Inc,C=US",

"certificate.not\_valid\_before": 1382461200,

"certificate.not\_valid\_after": 1855846800,

"certificate.key\_alg": "rsaEncryption",

"certificate.sig\_alg": "sha256WithRSAEncryption",

"certificate.key\_type": "rsa",

"certificate.key\_length": 2048,

"certificate.exponent": "65537",

"basic\_constraints.ca": true,

"basic\_constraints.path\_len": 0

}

Note: The first X.509 is the host server’s certificate and the second X.509 is the intermediate certificate that signed and issued the first. If we had skipped the steps in “Enable X.509 Logging of The Full Certificate Chain,” we would only have the X.509 certificate for the host server and not the intermediate certificate that issued it.

1. Let’s examine some of the key fields to better understand how we can use them to analyze X.509 certificates on our own network.  For a full listing, check out the [official Zeek documentation](https://docs.zeek.org/en/current/script-reference/file-analyzers.html?" \l "type-X509::Certificate).
   * id (e.g., FzHo3q14WMvHTjCc79 / F5woTi2O9Ea05rRxhj): This is the cert\_chain\_fuid or unique file ID that’s used to correlate files across Zeek logs.
   * certificate.version (e.g., 3): The certificate’s version number.
   * certificate.serial (e.g., 0FBABC6DFD509287B4B260CE67C6F292 / 04E1E7A4DC5CF2F36DC02B42B85D159F): The certificate’s serial number.
   * certificate.subject (e.g., CN=twitter.com,OU=atla,O=Twitter\\, Inc.,L=San Francisco,ST=California,C=US / CN=DigiCert SHA2 High Assurance Server CA,OU=www.digicert.com,O=DigiCert Inc,C=US): The certificate’s subject.
   * certificate.issuer (e.g., CN=DigiCert SHA2 High Assurance Server CA,OU=www.digicert.com,O=DigiCert Inc,C=US / CN=DigiCert High Assurance EV Root CA,OU=www.digicert.com,O=DigiCert Inc,C=US): The issuer of the certificate.  Note again that the first certificate was issued by the second certificate which itself was issued by a root certificate that is presumably trusted by the client.
   * certificate.not\_valid\_before (e.g., 1580968800 / 1382461200): The timestamp in Unix epoch time for when the certificate is first valid.
   * certificate.not\_valid\_after (e.g., 1612548000 / 1855846800): The timestamp in Unix epoch time for when the certificate is no longer valid.
   * certificate.key\_alg (e.g., rsaEncryption): The certificate’s encryption key algorithm.
   * certificate.sig\_alg (e.g., sha256WithRSAEncryption): The certificate’s signature algorithm.
   * certificate.key\_type (e.g., rsa): The certificate’s encryption key type.
   * certificate.key\_length (e.g., 2048): The certificate’s encryption key length.
   * certificate.exponent (e.g., 65537): The certificate’s exponent, if it is an RSA-certificate.
   * san.dns (e.g., twitter.com, www.twitter.com): The certificate’s list of Subject Alternative Name (SAN) DNS entries.
   * basic\_constraints.ca (e.g., false / true): This identifies whether or not the certificate is a certificate authority.  Note again, that the first certificate is not while the second is.
   * basic\_constraints.path\_len (e.g., 0): The certificate’s maximum path length.

We’ll discuss the remaining fields in the next section.

## **Fingerprint SSL/TLS with JA3**

In [Part VI](https://www.ericooi.com/zeekurity-zen-part-vi-zeek-file-analysis-framework/), we learned how to leverage Zeek’s incredible File Analysis Framework to automatically hash and uniquely fingerprint all files on the network.  This enabled us to easily identify known malicious files and quickly analyze unknown files by their hash.  Since the goal of encrypted traffic is confidentiality, wouldn’t it be great to have a way to fingerprint encrypted SSL/TLS traffic to identify and investigate for malicious activity?  Fortunately, the team at Salesforce took this idea and [developed a fingerprinting method to uniquely identify encrypted TLS traffic](https://github.com/salesforce/ja3) using the information Zeek already captures in ssl.log.  At a high level, [by combining cleartext elements of the SSL/TLS negotiation (e.g. version, cipher, elliptic curve, etc.) and MD5 hashing them](https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967), they’ve created the JA3 fingerprint for the client side and the JA3S fingerprint for the server’s response.

Our sample ssl.log from above includes these ja3/ja3s fields.

* ja3 (b32309a26951912be7dba376398abc3b): Client-side JA3 fingerprint.
* ja3s (e.g., 8d2a028aa94425f76ced7826b1f39039): Server-side JA3S fingerprint.

So how do we integrate JA3 into Zeek?  If you followed the “Install Additional Useful Packages” section in [Part II: Zeek Package Manager](https://www.ericooi.com/zeekurity-zen-part-ii-zeek-package-manager/) then you’re all set!  Your ssl.log will already have the JA3/JA3S fingerprints included.  From there, you can use the JA3 fingerprints just as you would with the file hashes — [**searching your logs for known malicious JA3 fingerprints**](https://sslbl.abuse.ch/ja3-fingerprints/) or [**searching on identified JA3 fingerprints**](https://ja3er.com/) to investigate notable activity.  While there are [JA3 evasion techniques](https://www.bc-security.org/post/ja3-s-signatures-and-how-to-avoid-them/), it remains a great method for analyzing encrypted SSL/TLS traffic and is increasingly supported by a number of security platforms.

## **Analyze Decrypted HTTP/2 traffic**

What if you’re one of the few organizations that actively decrypts web traffic through a network appliance such as a firewall?  If you’re already doing this, you can also send a copy of that decrypted traffic to Zeek to have it inspected as usual.  You’ll notice additional logging in http.log, but you may find that you’re not seeing everything you expected — especially from the internet’s major destinations.  It turns out that is due to most major web properties using HTTP/2 instead of the traditional HTTP/1.1.  Zeek cannot natively parse HTTP/2, but fortunately for us, the team at MITRE has developed an [excellent Zeek package for analyzing HTTP/2 traffic](https://github.com/MITRECND/bro-http2).

The following steps detail how to install the HTTP/2 analyzer package. As a reminder, for this to be useful, you will need to already be decrypting HTTPS traffic outside of Zeek and then sending a copy of this decrypted traffic to Zeek.  Zeek cannot decrypt traffic on its own.

1. As root/sudo, install the required dependencies.

* sudo yum install brotli libnghttp2-devel
* As the zeek user, stop zeek.
* zeekctl stop
* As the zeek user, use zkg to install the HTTP/2 analyzer package.
* zkg install zeek/mitrecnd/bro-http2

The following packages will be INSTALLED:

zeek/mitrecnd/bro-http2 (0.5.1)

Verify the following REQUIRED external dependencies:

(Ensure their installation on all relevant systems before proceeding):

from zeek/mitrecnd/bro-http2 (0.5.1):

libnghttp2>=1.11.0 libbrotlidec>=1.0.0

Proceed? [Y/n] y

Running unit tests for "zeek/mitrecnd/bro-http2"

Installing "zeek/mitrecnd/bro-http2".........................

Installed "zeek/mitrecnd/bro-http2" (0.5.1)

Loaded "zeek/mitrecnd/bro-http2"

* Edit/opt/zeek/share/zeek/site/local.zeek and add the following lines.  Note that you must specifically add these lines to local.zeek as the package is not enabled by simply using @load packages (which typically enables all Zeek packages):

@load http2

@load http2/intel

* As the zeek user, run zeekctl deploy to apply configurations and run Zeek.

1. zeekctl deploy
2. Assuming you’re sending decrypted HTTP/2 traffic to Zeek, you should now see a new http2.log in /opt/zeek/logs/current.  Below is a sample http2.log file in JSON format.  Note the version field’s value of “2.0” confirming that this is in fact HTTP/2 traffic.

{

"ts": 1600987697.997372,

"uid": "C6IFL331Q4qxw5sacf",

"id.orig\_h": "10.0.1.9",

"id.orig\_p": 55837,

"id.resp\_h": "172.217.9.164",

"id.resp\_p": 443,

"stream\_id": 1,

"method": "GET",

"host": "www.google.com",

"uri": "/",

"version": "2.0",

"user\_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36",

"request\_body\_len": 0,

"response\_body\_len": 220105,

"status\_code": 200,

"status\_msg": "",

"encoding": "br",

"push": false

}

Note: Even after following the steps above, you may find that you don’t see much Google traffic.  This is because Google tends to use the QUIC protocol and while there is a [Zeek package for analyzing QUIC traffic](https://github.com/salesforce/GQUIC_Protocol_Analyzer), it only logs metadata about the activity as it cannot perform decryption directly.  Unless you have a way to directly decrypt QUIC, a workaround is to block QUIC with a firewall and force Google to use TLS and HTTP/2 instead.  From there, you can decrypt web traffic as normal and send this decrypted traffic directly to Zeek for analysis.

## **Understand ssh.log**

Just as with SSL/TLS, while Zeek cannot directly decrypt SSH traffic, it can still observe and capture details from the SSH handshake — even going as far as heuristically determining whether an authentication attempt was successful.

1. Take a look at your own ssh.log and note the encrypted SSH connections and details that are captured.  Below is a sample ssh.log file in JSON format.

{

"ts": 1600955934.041752,

"uid": "C6JIXO2J1rHy9J0jAd",

"id.orig\_h": "10.0.1.99",

"id.orig\_p": 60774,

"id.resp\_h": "10.0.1.102",

"id.resp\_p": 22,

"version": 2,

"auth\_success": true,

"auth\_attempts": 2,

"client": "SSH-2.0-OpenSSH\_8.1",

"server": "SSH-2.0-OpenSSH\_7.4",

"cipher\_alg": "chacha20-poly1305@openssh.com",

"mac\_alg": "umac-64-etm@openssh.com",

"compression\_alg": "none",

"kex\_alg": "curve25519-sha256",

"host\_key\_alg": "ecdsa-sha2-nistp256",

"host\_key": "15:44:af:61:9f:fd:d6:51:f4:e8:35:3e:90:e7:9d:d5",

"hasshVersion": "1.1",

"hassh": "ec7378c1a92f5a8dde7e8b7a1ddf33d1",

"hasshServer": "6832f1ce43d4397c2c0a3e2f8c94334e",

"cshka": "ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-ed25519-cert-v01@openssh.com,rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-rsa-cert-v01@openssh.com,ssh-ed25519,rsa-sha2-512,rsa-sha2-256,ssh-rsa",

"hasshAlgorithms": "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,ext-info-c;chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com;umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1;none,zlib@openssh.com,zlib",

"sshka": "ssh-rsa,rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,ssh-ed25519",

"hasshServerAlgorithms": "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1;chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,blowfish-cbc,cast128-cbc,3des-cbc;umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1;none,zlib@openssh.com"

}

1. Let’s examine some of the key fields to better understand how we can use them to analyze SSH connections on our own network.  For a full listing, check out the [official Zeek documentation](https://docs.zeek.org/en/current/scripts/base/protocols/ssh/main.zeek.html" \l "type-SSH::Info).
   * uid (e.g., CY188jv7sKlzAMNNc): This is equivalent to the uid or unique ID that’s used to correlate activity across conn.log and other Zeek logs.
   * version (e.g., 2): The SSH version in use.
   * auth\_success (e.g., true): Using heuristics, Zeek determines whether the SSH session authenticated successfully.
   * auth\_attempts (e.g., 2): Using heuristics, Zeek determines the number of authentication attempts made.
   * client (e.g., SSH-2.0-OpenSSH\_8.1): The SSH client in use.
   * server (e.g., SSH-2.0-OpenSSH\_7.4): The SSH server in use.
   * cipher\_alg (e.g., chacha20-poly1305@openssh.com): The encryption algorithm used.
   * mac\_alg (e.g., umac-64-etm@openssh.com): The signing algorithm (MAC) used.
   * compression\_alg (e.g., none): The compression algorithm used.
   * kex\_alg (e.g., curve25519-sha256): The key exchange algorithm used.
   * host\_key\_alg (e.g., ecdsa-sha2-nistp256): The server’s key algorithm.
   * host\_key (e.g., 15:44:af:61:9f:fd:d6:51:f4:e8:35:3e:90:e7:9d:d5): The server’s key fingerprint.

We’ll discuss the remaining fields in the section below.

1. Zeek also includes built-in scripts that take advantage of its SSH capabilities — detecting SSH to/from specific countries and [detecting brute force attacks](https://corelight.blog/2019/05/07/how-zeek-can-provide-insights-despite-encrypted-communications/).  You can confirm these are enabled by viewing /opt/zeek/share/zeek/site/local.zeek and verifying the @load statements are uncommented as they are below.

# If you have GeoIP support built in, do some geographic detections and

# logging for SSH traffic.

@load protocols/ssh/geo-data

# Detect hosts doing SSH bruteforce attacks.

@load protocols/ssh/detect-bruteforcing

## **Fingerprint SSH with HASSH**

Similar to how JA3 can be used to fingerprint SSL/TLS connections, [HASSH](https://github.com/salesforce/hassh) can be used to [fingerprint SSH connections](https://engineering.salesforce.com/open-sourcing-hassh-abed3ae5044c).  Just as with JA3, you can use these fingerprints to profile and identify suspicious SSH activity.  Our sample ssh.log from above includes HASSH fields that we’ll review below.

* hasshVersion (e.g., 1.1): The HASSH version in use.
* hassh (e.g., ec7378c1a92f5a8dde7e8b7a1ddf33d1): The client-side HASSH fingerprint.
* hasshServer (e.g., 6832f1ce43d4397c2c0a3e2f8c94334e): The server-side HASSH fingerprint.
* cshka (e.g., ecdsa-sha2-nistp256-cert-v01@openssh.com): The client host key algorithms.
* hasshAlgorithms (e.g., curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,ext-info-c;chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com;umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1;none,zlib@openssh.com,zlib): The concatenated list of client-side algorithms including key exchange methods, encryption, message auth, and compression.  The client-side HASSH fingerprint is a result of taking the MD5 hash of this list.
* sshka (e.g., ssh-rsa, rsa-sha2-512): The server host key algorithms.
* hasshServerAlgorithms (e.g., curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1;chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,blowfish-cbc,cast128-cbc,3des-cbc;umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1;none,zlib@openssh.com): The concatenated list of server-side algorithms including key exchange methods, encryption, message auth, and compression.  The server-side HASSH fingerprint is a result of taking the MD5 hash of this list.

## **Next. Log to ELK**

## **Output Zeek logs to JSON**

1. Stop Zeek if it is currently running.

* zeekctl stop
* Edit /opt/zeek/share/zeek/site/local.zeek and add the following.

# Output to JSON

@load policy/tuning/json-logs.zeek

* Restart Zeek and view the logs in /opt/zeek/logs/current to confirm they are now in JSON format.

1. zeekctl deploy

cd /opt/zeek/logs/current

less conn.log

## **Configure Filebeat**

This guide assumes you have already installed Filebeat. If not, refer to [Elastic’s documentation](https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-installation-configuration.html) and then come back here when you’re done.

1. Switch back to your normal user.

* su eric
* Stop Filebeat if it is currently running.
* sudo systemctl stop filebeat
* Enable Filebeat’s Zeek module.
* sudo filebeat modules enable zeek
* Add Zeek’s log paths (e.g., /opt/zeek/logs/current/http.log) to the Zeek Filebeat configuration file. Assuming you installed Filebeat from the standard Elastic repositories, the configuration file will be in /etc/filebeat/modules.d/zeek.yml. For each log type you want to capture:
* Set the enabled field to true.
* Set the var.paths field to the log’s specific path.

The sample configuration file below enables all log types currently available in the Filebeat Zeek module. Note that while the “signature” log type is listed, it is currently not available and will result in an error if enabled. This [Github issue](https://github.com/elastic/beats/issues/18868) discusses this in further detail.

# Module: zeek

# Docs: https://www.elastic.co/guide/en/beats/filebeat/7.x/filebeat-module-zeek.html

- module: zeek

capture\_loss:

enabled: true

var.paths: ["/opt/zeek/logs/current/capture\_loss.log"]

connection:

enabled: true

var.paths: ["/opt/zeek/logs/current/conn.log"]

dce\_rpc:

enabled: true

var.paths: ["/opt/zeek/logs/current/dce\_rpc.log"]

dhcp:

enabled: true

var.paths: ["/opt/zeek/logs/current/dhcp.log"]

dnp3:

enabled: true

var.paths: ["/opt/zeek/logs/current/dnp3.log"]

dns:

enabled: true

var.paths: ["/opt/zeek/logs/current/dns.log"]

dpd:

enabled: true

var.paths: ["/opt/zeek/logs/current/dpd.log"]

files:

enabled: true

var.paths: ["/opt/zeek/logs/current/files.log"]

ftp:

enabled: true

var.paths: ["/opt/zeek/logs/current/ftp.log"]

http:

enabled: true

var.paths: ["/opt/zeek/logs/current/http.log"]

intel:

enabled: true

var.paths: ["/opt/zeek/logs/current/intel.log"]

irc:

enabled: true

var.paths: ["/opt/zeek/logs/current/irc.log"]

kerberos:

enabled: true

var.paths: ["/opt/zeek/logs/current/kerberos.log"]

modbus:

enabled: true

var.paths: ["/opt/zeek/logs/current/modbus.log"]

mysql:

enabled: true

var.paths: ["/opt/zeek/logs/current/mysql.log"]

notice:

enabled: true

var.paths: ["/opt/zeek/logs/current/notice.log"]

ntlm:

enabled: true

var.paths: ["/opt/zeek/logs/current/ntlm.log"]

ntp:

enabled: true

var.paths: ["/opt/zeek/logs/current/ntp.log"]

ocsp:

enabled: true

var.paths: ["/opt/zeek/logs/current/oscp.log"]

pe:

enabled: true

var.paths: ["/opt/zeek/logs/current/pe.log"]

radius:

enabled: true

var.paths: ["/opt/zeek/logs/current/radius.log"]

rdp:

enabled: true

var.paths: ["/opt/zeek/logs/current/rdp.log"]

rfb:

enabled: true

var.paths: ["/opt/zeek/logs/current/rfb.log"]

signature:

enabled: false

var.paths: ["/opt/zeek/logs/current/signature.log"]

sip:

enabled: true

var.paths: ["/opt/zeek/logs/current/sip.log"]

smb\_cmd:

enabled: true

var.paths: ["/opt/zeek/logs/current/smb\_cmd.log"]

smb\_files:

enabled: true

var.paths: ["/opt/zeek/logs/current/smb\_files.log"]

smb\_mapping:

enabled: true

var.paths: ["/opt/zeek/logs/current/smb\_mapping.log"]

smtp:

enabled: true

var.paths: ["/opt/zeek/logs/current/smtp.log"]

snmp:

enabled: true

var.paths: ["/opt/zeek/logs/current/snmp.log"]

socks:

enabled: true

var.paths: ["/opt/zeek/logs/current/socks.log"]

ssh:

enabled: true

var.paths: ["/opt/zeek/logs/current/ssh.log"]

ssl:

enabled: true

var.paths: ["/opt/zeek/logs/current/ssl.log"]

stats:

enabled: true

var.paths: ["/opt/zeek/logs/current/stats.log"]

syslog:

enabled: true

var.paths: ["/opt/zeek/logs/current/syslog.log"]

traceroute:

enabled: true

var.paths: ["/opt/zeek/logs/current/traceroute.log"]

tunnel:

enabled: true

var.paths: ["/opt/zeek/logs/current/tunnel.log"]

weird:

enabled: true

var.paths: ["/opt/zeek/logs/current/weird.log"]

x509:

enabled: true

var.paths: ["/opt/zeek/logs/current/x509.log"]

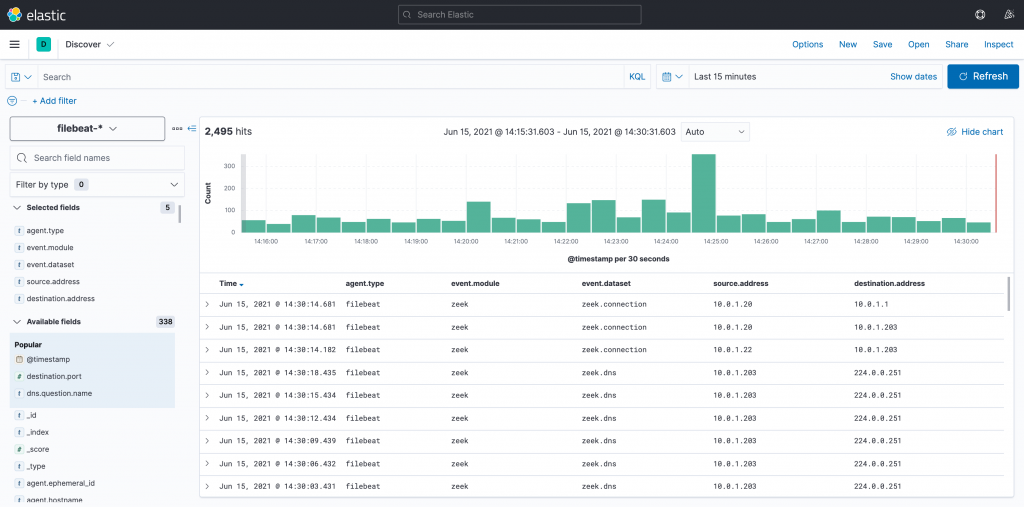
# Set custom paths for the log files. If left empty,

# Filebeat will choose the paths depending on your OS.

#var.paths:

* Restart Filebeat to apply the configuration and confirm your Zeek logs are now properly ingested into Elasticsearch and available for analysis in Kibana.

1. sudo systemctl restart filebeat

[](https://www.ericooi.com/wp-content/uploads/2021/06/zzz_partviii_kibana_discover.png)

## **Simple Kibana Queries**

Once Zeek logs are flowing into Elasticsearch, we can write some simple Kibana queries to analyze our data. Let’s convert some of our previous [sample threat hunting queries](https://www.ericooi.com/zeekurity-zen-part-iv-threat-hunting-with-zeek/) from Splunk SPL into Elastic KQL. Try taking each of these queries further by creating relevant visualizations using [Kibana Lens](https://www.elastic.co/guide/en/kibana/current/lens.html).

### Connections To Destination Ports Above 1024

event.module:zeek AND event.dataset:zeek.connection AND destination.port>1024

### Query Responses With NXDOMAIN

event.module:zeek AND event.dataset:zeek.dns AND dns.response\_code:NXDOMAIN

### Expired Certificates

event.module:zeek AND event.dataset:zeek.x509 AND file.x509.not\_after < now