How to Install Snort 3 for Most Major Debian-Based Distros

By: Tony Robinson

How to Install Snort 3 for Most Major Debian-Based Distros

Greetings! The goal of this guide will be to teach users how to compile Snort 3 for most major Debian-based Distros. Officially, this guided has been tested against:

- Ubuntu (20.04, 22.04, and 24.04)
- Debian 12
- Kali Linux 2024.1
- Remnux (Based on Ubuntu 20.04)
- Sift (Based on Ubuntu 22.04)

The version of Snort 3 I will be using for this demonstration will be 3.1.84.0 – but barring no major changes in prerequisites or functionality, these instructions should directly apply to future releases with little to no change necessary.

Contents

Overview

The major steps we'll need to complete to accomplish this task are as follows:

- Acquire the prerequisite packages necessary to compile Snort3 and its supporting libraries
- Acquire and compile the supporting libraries
- Acquire and compile Snort 3 and its additional features (Snort 3 Extras)
- Acquire Snort 3 rules via pulledpork3, and the talos_lightSPD ruleset
- Set up a user account and group to run Snort 3
- Establish a systemd service to handle start/stopping Snort 3 as well
- Extra Credit (Snort2lua)

I'll be demonstrating these steps primarily on the latest Ubuntu Server LTS release which, as of mine writing this, is 24.04. However, this documentation will be tested on Debian 12, Kali Linux 2024.1, The current 2024 releases of both Remnux (based on Ubuntu 20.04), and SIFT (based on Ubuntu 22.04), Ubuntu 22.04, and 20.04. If there are any major differences on how certain commands are run, peculiarities for a particular Linux distro, or a difference in the name of software packages, those will be noted on a per-distribution basis.

Other Thoughts (before we get started)

Here are a couple of things to think about before we proceed with installing Snort 3

Hardware Requirements

As with most software, the more system resources you can give it, the better it will perform – Better disk I/O for writing logs and alerts, more disk space to allow for more log retention time, Better and more RAM to run efficiently and support more rules and complex configurations, more CPU cores to benefit from multi-threading.

An IDS sensor of any sort should have at least two network interfaces, one interface for sniffing traffic in IDS mode from a span or network tap, and the other network interface for device management, monitoring, and/or shipping logs off the system to a central SIEM of some sort. If you're interested in IPS mode operation, then increase that number to at least three network interfaces – two to carry traffic in inline mode, and a third for device management, monitoring, and log delivery.

I've seen Snort 3 compile and run with as little as 1GB of RAM, 1 CPU core, 2 network interfaces, and 40GB of platter disk in a RAID10 array – granted it was a virtual machine for a homelab, servicing very little traffic.

For most home lab environments, with light to moderate network traffic, aim for 2GB of RAM, 2 CPU cores, 2 network interfaces, and 120GB of disk space – on an SSD if you can.

For an enterprise environment, well. All I can say is that "it varies". Depending on the size of your network, composition of your network traffic, sensor placement in your network fabric, where and how your Snort logs are processed, what kind of data Snort 3 is configured to log, and a whole host of other factors. No two networks, even with identical hardware, are exactly the same and therefore no two instances of Snort will run exactly the same. Trial and error will be the best method to determine the hardware requirements to best suit your needs.

System monitoring software such as Nagios, Cacti, or other solutions will help you determine whether or not your sensor is meeting your needs. I would recommend monitoring for the following:

- Dropped packets on network interfaces for sniffing traffic
- Consistently high system CPU usage
- Consistently high system Memory usage
- Poor disk I/O
- Free disk space available
- Poor network connectivity
- Ability to keep time consistently either via NTPd or Chrony
- Ensure services responsible for shipping logs to your SIEM of choice are performing correctly (e.g. Splunk, ELK stack, etc.)

If you notice problems with any of these, most of the time its either a lack of hardware resources, or in some cases, poor network connectivity. Use this information to better work towards tuning the hardware specs of your IDS/IPS sensors.

`root/sudo` Access

Practically all of the steps of the installation guide will require `root` access at some point. Readers will either need access to the `root` user account's credentials or `sudo` access. Be aware that by default, Ubuntu does not allow logging in as the `root` user, but instead recommends using the `sudo` command to run commands as `root`, whereas Debian does allow logging in as the `root` user, but does not come with `sudo` by default – but it is available to download via `apt-get` (`apt-get install sudo`).

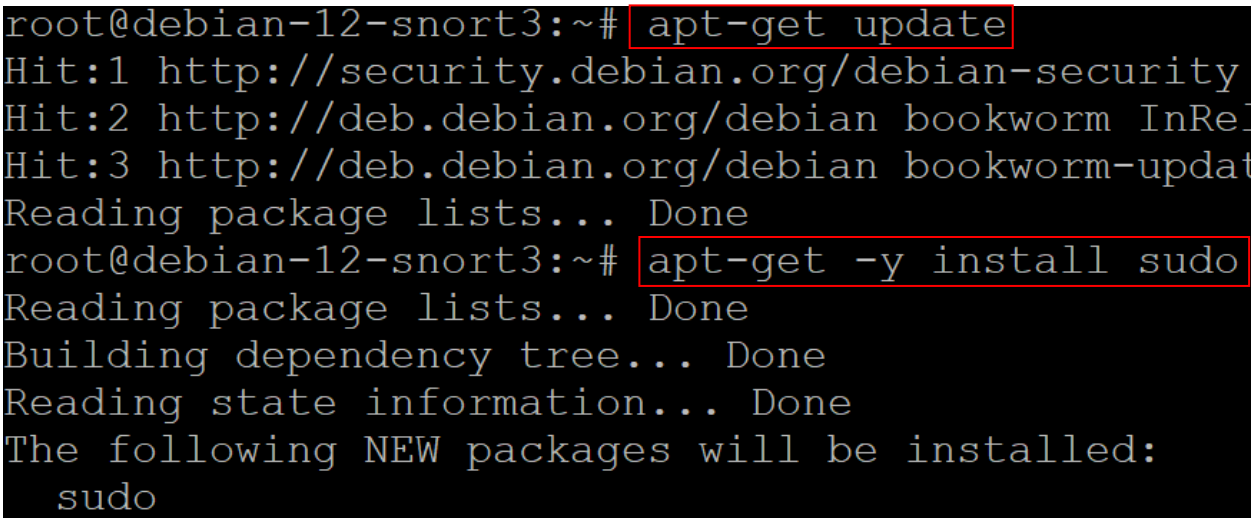
How to install/configure sudo access on Debian 12

To enable `sudo` on Debian 12, log in as the `root` user and run the following commands

```
apt-get update
apt-get install sudo
```

Note: the `apt-get` command will respond with a prompt to confirm whether or not the user wants to install the package specified and/or any of its required packages. This prompt can be skipped by adding the `-y` flag to the `apt-get` command:

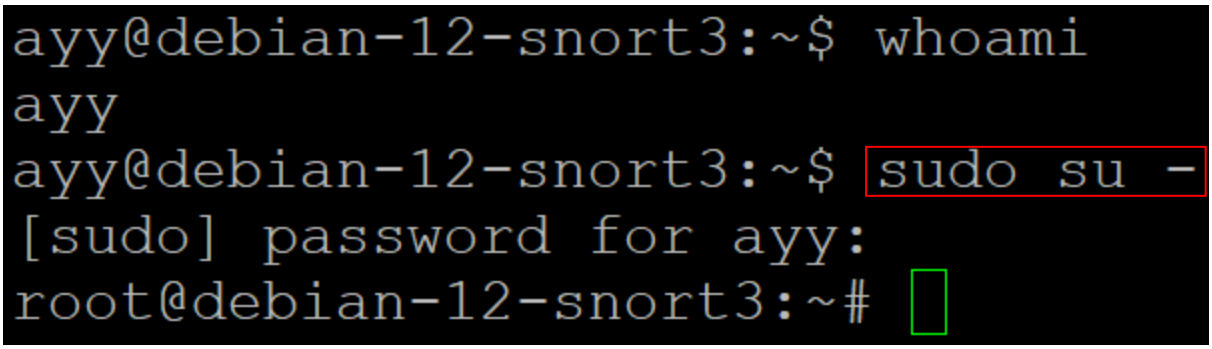
```
apt-get -y install sudo
```



1-1: To install `sudo` on Debian 12, log in as the root user, run `apt-get update`, followed by `apt-get install sudo`. If users utilize the `-y` switch, (e.g. `apt-get -y installed sudo`) the system will not supply a confirmation prompt to confirm installation of `sudo` and/or any supporting packages/libraries necessary to use the `sudo` package.

Readers will need to confirm that their desired user has `sudo` access. This is the easiest way to do so:

1. Log in to your Debian 12 system with your standard user account
2. Run the command sudo su –
3. Readers will be prompted to enter the password of their user account. After doing so, the shell prompt should change to signify that they are currently logged in to an interactive shell as the `root` user. Type `exit` to log out of the `root` shell and return to the previously logged in user account.



1-2: Testing `sudo` is pretty easy. Log in to the system with a standard user account, and run the command `sudo su -`. After entering the user's password, readers should be greeted with an interactive shell as the `root` user, provided the user is in the sudoers file, and has the correct access.

If readers receive the message "`This user is not in the sudoers file. This incident will be reported.`" Or something to that effect, first, try logging out any users that are actively logged in, then try steps 1 and 2 above once more. Sometimes, systems require that users log out before new libraries and tools work effectively.

However, if problems still persist after logging out, readers may need to modify the `/etc/sudoers` file using the `visudo` command or add the user to the sudo group manually:

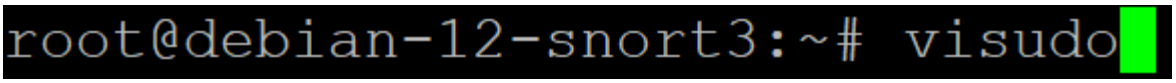
1. Log in as the `root` user, and run the command visudo
2. Debian 12 defaults to the nano command line text editor. Use the arrow keys to scroll down. There should be lines similar to:

   ```
   # Allow members of group sudo to execute any command
   %sudo   ALL=(ALL:ALL) ALL
   ```

3. If these lines are in the file, hit CTRL+X to quit editing the file. If asked to save your changes, select no. If these lines do NOT exist in the sudoers file, add them, quit the sudoers file, and save your changes.
4. Next, login as the `root` user, and run the commands:

   ```
   adduser [username] sudo
   groups [username]
   ```

Replace [username] with the name of the user to grant sudo group access to. For example: `adduser user1 sudo` will add the `user1` user to the `sudo` group. If the output from the `groups` command for the user's username includes sudo, this means that the `adduser` command was successful.




```
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

```
root@debian-12-snort3:~# adduser ayy sudo
Adding user `ayy' to group `sudo' ...
Done.
root@debian-12-snort3:~# groups ayy
ayy : ayy cdrom floppy sudo audio dip video
```

1-3: Did `sudo su -` not work? Use the `visudo` command, and check the `/etc/sudoers` config file for the lines in the middle image and confirm that members of the `sudo` group are allowed to use the `sudo` command. Finally, add the desired user to the `sudo` group via the `adduser` command, then confirm the user has `sudo` group access via the `groups` command. Afterwards, log in as the user, and try using `sudo su -` once more to confirm the user can run commands with `root` privileges.

Running commands with root privileges

Assume that most commands in this installation guide will require root user access to run, especially if they involve installing packages or change configuration aspects of the system. There are numerous ways to run commands as the root user, but these are some of the more common methods:

1. Prepend `sudo` to a single command, to run just that command as the `root` user

For example, let's say users wanted to install the `apache2` web server on their Debian-based distro. That typically involves using the `apt-get` package manager. `Apt-get` requires `root` access to install packages. If the currently logged in user has `sudo` access, then the command `sudo apt-get install apache2` will install the `apache2` web server, after entering the user's password.

2. As a non-root user with `sudo` access, run `sudo su –` to gain an interactive root shell, and run as many commands as required with `root` privileges.

There are some security minded individuals that say that doing this is dangerous, because having an interactive shell as the `root` user means it is very easy to make mistakes that can impact system stability. On the other hand, it is also very convenient in instances where users will need to run a series of commands requiring `root` privileges. For instance, there is a very high chance that after installing `apache2`, users will need to access and modify a host of configuration settings for the web server. In most cases, these files are in the directory `/etc/apache2` – including the `apache2.conf` file, which is only writable as the `root` user. To keep with this example, let's say the user is done writing changes to the `apache2.conf` file and/or other configuration files. Next, the service needs to be restarted. Typically, this is done with the `systemctl restart [command name].service` command (e.g. `systemctl restart apache2.service`) to reload the service, and re-read the configuration file. `Systemctl` requires `root` privileges to run.

```
ayy@debian-12-snort3:~$ sudo apt-get install apache2
[sudo] password for ayy:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libcurl4 liblua5.3-0 ssl-cert
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom www-browser
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap libcurl4 liblua5.3-0 ssl-cert
```

1-4: One of the two primary ways to run commands as the `root` user without actually logging into the `root` user account is to use the `sudo` command to get root privileges to run just a single command. It is a risk reduction technique for ensuring that access to the `root` account is limited. However, it's very irritating to have to prepend `sudo` to every single command that requires `root` privileges, and enter credentials so frequently. Especially with long, complex passwords…

```
ayy@debian-12-snort3:~$ sudo su -
[sudo] password for ayy:
root@debian-12-snort3:~# apt-get install apache2
```

1-5: …Or, the alternative is to use `sudo` to your advantage. Run `sudo su -` to run the `su` command via `sudo`, and gain an interactive `root` shell. From this `root` shell, readers can run as many commands as the `root` user as necessary. Critics of this technique say it's dangerous in a 'handguns and tequila' in the same room kind of way. As in, it's very easy to make mistakes, but it's also very convenient to run a variety of commands requiring `root` access in quick succession without the annoyance of prepending `sudo` to every single command that requires `root` access and/or entering login credentials repeatedly.

For the purposes of this Snort3 installation guide, I will be using the "`sudo su -` method" to gain a `root` shell in order to run as many commands with `root` privileges as necessary. For those wanting to play it safe and use the "`sudo` for every command" method, you'll need to modify the commands as necessary.

Install your SIEM stack on a separate system/VM

If you plan on using a SIEM such as an ELK stack, or Splunk, I highly advise placing your logging solution/stack on a separate system, and forwarding the logs from the IDS to that system. Using Splunk as an example, the Snort 3 IDS sensor should have a Splunk Universal Forwarder installed and forwarding logs to a separate Splunk Enterprise server that both ingests logs and provides search head functionality.

Acquiring Necessary Packages via `apt-get`

Our first order of business will be using `apt-get` to download the necessary packages to compile the various components of Snort3. Run the following commands:

```
apt-get update
apt-get -y dist-upgrade
apt-get -y install autoconf autotools-dev bison build-essential cmake cpputest curl ethtool
flex git jq libboost-all-dev libcmocka-dev libdumbnet-dev libhwloc-dev libluajit-5.1-dev
liblzma-dev libmnl-dev libnetfilter-queue-dev libpcap-dev libpcre3 libpcre3-dev libpcre3-dbg
libsqlite3-dev libssl-dev libtool libunwind-dev openssl pkg-config ragel uuid-dev unzip wget
xz-utils zlib1g-dev
```

The purpose of these commands are to:

1. Update the software repository listing
2. Check for any necessary system updates for packages already installed on the system and install them
3. Install the necessary packages for use to compile Snort3, its required libraries, supporting tools, as well as downloading useful utilities for the system.

I won't go over the purpose of every single package we'll be installing via `apt-get` here, but know that they are either absolutely necessary for Snort3 and/or the extras, for supporting tools to help administer the sensor, or convenient administrative tools. Note that depending on the distribution, some of these packages may already be installed.

```
root@ubuntu-2404-snort3:/usr/src# apt-get update
Hit:1 http://us.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
root@ubuntu-2404-snort3:/usr/src# apt-get -y dist-upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ubuntu-2404-snort3:/usr/src# apt-get -y install autoconf autotools-dev bison build-esse
ntial cmake cpputest curl ethtool flex git jq libboost-all-dev libcmocka-dev libdumbnet-dev
libhwloc-dev libluajit-5.1-dev liblzma-dev libmnl-dev libnetfilter-queue-dev libpcap-dev lib
pcre3 libpcre3-dev libpcre3-dbg libsqlite3-dev libssl-dev libtool libunwind-dev openssl pkg-
config ragel uuid-dev unzip wget xz-utils zlib1g-dev
```

2-1: `apt-get update` to update the software repository list, `apt-get -y dist-upgrade` to ensure the system is fully up-to-date, then run:

```
apt-get -y install autoconf autotools-dev bison build-essential cmake cpputest curl ethtool
flex git jq libboost-all-dev libcmocka-dev libdumbnet-dev libhwloc-dev libluajit-5.1-dev
liblzma-dev libmnl-dev libnetfilter-queue-dev libpcap-dev libpcre3 libpcre3-dev libpcre3-dbg
libsqlite3-dev libssl-dev libtool libunwind-dev openssl pkg-config ragel uuid-dev unzip wget
xz-utils zlib1g-dev
```

To install the grocery list of required packages for us to get started.

Acquiring Necessary Source Files via `curl`, `wget`, and `git`

Next up, we have a whole host of software to acquire, almost all of which will need to be compiled from source:

- Safeclib (https://github.com/rurban/safeclib)
- Gperftools (https://github.com/gperftools/gperftools)
- Snort3 Libdaq (https://github.com/snort3/libdaq)
- Snort3 (https://github.com/snort3/snort3)
- Snort3 extras (https://github.com/snort3/snort3_extra)
- PulledPork3 (https://github.com/shirkdog/pulledpork3)
- Vectorscan (https://github.com/VectorCamp/vectorscan)

I recommend putting the source files for all of these projects into the `/usr/src` directory on your Linux installation, so `cd` into `/usr/src` and run the following commands:

```
wget `curl --silent "https://api.github.com/repos/rurban/safeclib/releases/latest" | jq -r
'.assets[].browser_download_url' | egrep ".tar.bz2$"`
wget `curl --silent "https://api.github.com/repos/gperftools/gperftools/releases/latest" | jq
-r '.assets[0].browser_download_url'`
snort3_version=`curl -fsSL https://api.github.com/repos/snort3/snort3/releases/latest | jq -r
.tarball_url|cut -d"/" -f8`
wget `curl -fsSL https://api.github.com/repos/snort3/snort3/releases/latest | jq -r
.tarball_url` -O snort-$snort3_version.tar.gz
libdaq_version=`curl -fsSL https://api.github.com/repos/snort3/libdaq/releases/latest | jq -r
.tarball_url | cut -d"/" -f8`
wget `curl -fsSL https://api.github.com/repos/snort3/libdaq/releases/latest | jq -r
.tarball_url` -O libdaq-$libdaq_version.tar.gz
wget `curl -fsSL https://api.github.com/repos/snort3/snort3_extra/tags | jq -r --arg
snort3_version "$snort3_version" '.[] | select(.name==$snort3_version) | .tarball_url'` -O
snort3_extras-$snort3_version.tar.gz
wget https://snort.org/downloads/openappid/snort-openappid.tar.gz
git clone https://github.com/shirkdog/pulledpork3
git clone https://github.com/VectorCamp/vectorscan
```

The purpose of these commands are to just download the source packages necessary for snort3. These commands generally use `curl` to download a json file to find out where the latest release tarball (`.tar.gz` or `.tar.bz2` usually) file can be downloaded. The `jq` command is used to parse the json output returned from `curl`, extract the download url, then that output is passed to `wget` to download the source code. A couple of repositories aren't very nice about returning a nicely formatted file name (e.g. `software-version.tar.gz`), so a handful of these commands are used to set shell variables that store the current version number of the software, then we use our combination `wget/curl/jq` command with wget's `-O` to set the name of the tarball when its downloaded so that the name of the file is sane and makes sense. We also download snort's openappID, and finally, there is the `git` command that just clones a copy of the github repo for pulledpork3.

When finished, run `ls -al /usr/src | egrep -v "linux-headers|python"`. The output should look something like this:

```
root@ubuntu-2404-snort3:/usr/src# ls -al | egrep -v "linux-header|python"
total 6256
drwxr-xr-x  7 root root    4096 May  7 18:25 .
drwxr-xr-x 12 root root    4096 Apr 23 09:37 ..
-rw-r--r--  1 root root 1511014 Jan  5 17:31 gperftools-2.15.tar.gz
-rw-r--r--  1 root root  167931 Apr 27 22:39 libdaq-v3.0.14.tar.gz
drwxr-xr-x  5 root root    4096 May  7 18:25 pulledpork3
-rw-r--r--  1 root root  732903 Mar  8 10:17 safeclib-3.8.1.tar.bz2
-rw-r--r--  1 root root 3299676 Apr 27 22:31 snort-3.1.84.0.tar.gz
-rw-r--r--  1 root root   58766 Apr 27 23:29 snort3_extras-3.1.84.0.tar.gz
-rw-r--r--  1 root root  599862 Jun 16  2023 snort-openappid.tar.gz
drwxr-xr-x 15 root root    4096 May  5 19:21 vectorscan
```

3-1: If the commands above worked correctly, readers should have four `.tar.gz` files (`snort3_extras`, `snort3`, `libdaq`, and `gperftools`), one `.tar.bz2` file (`safeclib`), and two directories – `pulledpork3` and `vectorscan`.

Compiling Snort3, Snort3_extras, and all supporting libraries

Now, readers should have everything they need to get started. We'll begin by compiling the supporting libraries (gperftools, safeclib, libdaq), then snort3 itself, and finally, snort3_extras.

Gperftools

In the `/usr/src` directory, run the following command:

```
tar -xzvf gperftoools-[xxxx].tar.gz
```

Replace `[xxxx]` with the version of gperftools. In my case:

```
tar -xzvf gperftools-2.15.tar.gz
```

Note: Most modern Debian-based distros feature a shell with tab completion. So, users can start typing `tar -xzvf gperf`, hit the tab key, and the shell will complete the filename on the user's behalf.

Next, use the cd command to change directories into the gperftools-[xxxx] directory:

```
cd gperftools-2.15/
```

Once in the directory, run:

```
./configure
```

This does a bunch of checks prior to use compiling the software for use. When it finishes, run:

```
make -j$(nproc)
```

The make command compiles the software. The argument `-j` allows us to allocate multiple CPU cores to make the software compile faster. The value `$(nproc)` returns the full number of CPUs installed on the system. So, for a system with two CPU cores, `$(nproc)` will return 2, and make the command `make -j2`.

Depending on the system's performance, I/O memory, etc. This command may take a little bit of time to complete. When it's finished, run:

```
make install
```

This is the final step, and is meant to do any post-compilation cleanup, and move compiled software to default/configured directories, etc. Afterwards, use the command `cd ..` to return to `/usr/src` and prepare to compiling the next piece of software.

```
root@ubuntu-2404-snort3:/usr/src# tar -xzvf gperftools-2.15.tar.gz
gperftools-2.15/
root@ubuntu-2404-snort3:/usr/src# cd gperftools-2.15/
root@ubuntu-2404-snort3:/usr/src/gperftools-2.15# ./configure
root@ubuntu-2404-snort3:/usr/src/gperftools-2.15# make -j$(nproc)
root@ubuntu-2404-snort3:/usr/src/gperftools-2.15# make install
```

4-1: Compiling gperftools in four steps:

1. Use `tar -xzvf gperftools-[xxxx].tar.gz` to decompress the files
2. Use `cd gperftools-[xxxx]/` to change directories into the newly created directory
3. Run `make -j$(nproc)` to compile the software
4. Run `make install` to install libraries and documentation into the configured/default directories. Use `cd ..` to return to `/usr/src`.

Safeclib

The instructions to compile safeclib are practically identical to gperftools, but with some slight twists:

1. Run `tar -xjvf safeclib-[xxxx].tar.bz2`
2. This creates a directory `safeclib-[xxxx]-[yyyy]`. Use the `cd` command to change directories into it.
3. Run `./configure`
4. Run `make -j$(nproc)`
5. Run `make install` then `cd ..` to go back to `/usr/src`

```
----------------------------------------------------------------------
Libraries have been installed in:
   /usr/local/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
     during execution
   - add LIBDIR to the 'LD_RUN_PATH' environment variable
     during linking
   - use the '-Wl,-rpath -Wl,LIBDIR' linker flag
   - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
```

4-2: The steps for decompressing, compiling and installing safeclib are more or less the same as gperftools, the only difference to be aware of is the name of the directory, and that the `tar` command is `tar -xjvf safeclib-[xxxx].tar.bz2`. The `-j` option is specific to bzip2 compressed files. Another good sign to confirm that all of the installation steps completed successfully is to look for output from the `make install` command that looks similar to this. This is `make install` confirming that it successfully copied the libraries to the `/usr/local/lib` directory.

Libdaq

Libdaq is going to throw another slight curveball at us. Begin by running the following commands:

```
tar -xzvf libdaq-v[xxxx].tar.gz
cd snort3-libdaq-[xxxx]/
```

In the decompressed libdaq directory, users will need to run the bootstrap script with the command:

```
./bootstrap
```

Once the command finishes, continue as normal, with the following commands, to finish compiling the snort3 data acquisition libraries.

```
./configure
make -j$(nproc)
make install
cd ..
```

This time around, when running `make install`, readers should look for the text:

```
Libraries have been installed in:
    /usr/local/lib/daq
```

Vectorscan

Snort 3 can be compiled with support for hyperscan, a regular expression performance library developed by Intel. I discovered a bug with Snort3 involving `libhyperscan-dev` and the `hyperscan_literals = true` configuration setting. I've submitted bugs to multiple locations:

Snort 3 Github: https://github.com/snort3/snort3/issues/366
Vectorscan Github: https://github.com/VectorCamp/vectorscan
Ubuntu/Launchpad: https://bugs.launchpad.net/ubuntu/+source/vectorscan/+bug/2064289

To summarize things, there is some sort of a compatibility problem with Snort3 and the version of the hyperscan library available in some software repositories (version 5.4.2). This problem was not present in the older versions of libhyperscan-dev. This problem is compounded by the fact that hyperscan used to be provided with an open-source license, but has since changed, meaning fixes to hyperscan are… complicated now.

Some software developers decided to fork hyperscan. To fix problems with it, and to make it available on other CPU architectures. Thus, we have vectorscan. There are software packages available for vectorscan, but it's a work in progress currently, and those packages are not available for x86_64 systems. Maybe this will change in the future but for now, readers will have to compile vectorscan from source.

Note for Ubuntu 20.04 and Remnux (based on Ubuntu 20.04) users: The version of `cmake` available in the default package repositories is too old to compile vectorscan. Fortunately, the organization kitware maintains their own apt repos with a more update to date version of `cmake` that we'll need to compile vectorscan. Run the following commands:

```
apt-get -y remove cmake
apt-get -y autoremove
apt-get -y install gpg software-properties-common
wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc 2>/dev/null | gpg --dearmor
- | sudo tee /etc/apt/trusted.gpg.d/kitware.gpg >/dev/null
apt-add-repository 'deb https://apt.kitware.com/ubuntu/ focal main'
apt-get update
apt-get -y install cmake
cmake –version
```

These commands will add kitware's official apt repo to the apt software source repository list. If the commands above were successful, `cmake --version` should return a version number higher than 3.18.4 (minimum version required to build vectorscan). Here is the output I received:

```
cmake version 3.29.3
```

After following these instructions, readers should be able to proceed through this section as normal.

To begin, run the following commands:

```
cd /usr/src/vectorscan
mkdir build
cd build
cmake -DUSE_CPU_NATIVE=on -DFAT_RUNTIME=off -DBUILD_AVX2=ON ../
```

```
root@ubuntu-2404-snort3:/usr/src# cd /usr/src/vectorscan/
root@ubuntu-2404-snort3:/usr/src/vectorscan# mkdir build
root@ubuntu-2404-snort3:/usr/src/vectorscan# cd build
root@ubuntu-2404-snort3:/usr/src/vectorscan/build# cmake -DUSE_CPU_NATIVE=on -DFAT_RUNTIME=o
ff -DBUILD_AVX2=ON ../
```

4-5: Move into the `/usr/src/vectorscan` library. Users will need to make the `build` directory then change directories into it. Finally, we run `cmake -DUSE_CPU_NATIVE=on -DFAT_RUNTIME=off -DBUILD_AVX2=ON ../`

The options we're passing to cmake tell vectorscan to compile a version of the library that is tuned to our CPU architecture, and helps to reduce the size and how long it takes to compile significantly.

Note: There are quite a few different cmake options that can be provided to vectorscan, some are dependent on what CPU architecture readers will be building vectorscan on. For example, the option `-DBUILD_AVX2=ON` implies that the user building on a platform that the AVX2 CPU feature available on their CPU. Run the command `cat /proc/cpuinfo | grep -o avx2` and look for the red text "avx2" to appear in the list of flags to confirm the CPU supports AVX2. If readers get no output from the command, that means the CPU does not support that feature, and to remove the option -DBUILD_AVX2=ON from the cmake command.

For those interested in a complete list of cmake options available for vectorscan, check out the github project page: https://github.com/VectorCamp/vectorscan

```
root@ubuntu-2404-snort3:/usr/src# cat /proc/cpuinfo | grep -o avx2
avx2
```

4-6: This is the output that should appear from the command cat /proc/cpuinfo | grep -o avx2 if the CPU supports building vectorscan with AVX2 support. This feature is usually specific to x86_64 CPUs, and more specifically mostly recent Intel (and some AMD) CPUs. Check out the vectorscan github project page for more cmake options.

The final steps are for readers to compile and install the vectorscan libraries. Run the commands:

```
make -j$(nproc)
make install
cd /usr/src
```

To finish compiling and installing vectorscan. As mentioned earlier, even with options to trim away some of the code, compiling vectorscan is likely to take a moment or two. When finished, change directories back to the /usr/src directory.

```
root@ubuntu-2404-snort3:/usr/src/vectorscan/build# make -j$(nproc)
root@ubuntu-2404-snort3:/usr/src/vectorscan/build# make install
root@ubuntu-2404-snort3:/usr/src/vectorscan/build# cd /usr/src
```

4-7: From here, readers will run `make -j$(nproc)`, `make install`, then `cd /usr/src` to finish compiling and installing vectorscan, then change directories back to `/usr/src` for the next task.

Snort3

Compiling Snort 3 is going to be a little bit different than the other software we've compiled so far but it's still really straightforward. As usual, begin by decompressing the snort3 tarball and changing directories into the newly created directory:

```
tar -xzvf snort-[xxxx].tar.gz
cd snort3-snort3-[yyyy]/
```

Once in the newly decompressed directory, run the following command:

```
./configure_cmake.sh --prefix=/usr/local --enable-tcmalloc
```

This command tells cmake what extra things beyond the default items we want to compile support for in snort3. In this instance, we want to have support for tcmalloc (provided by gperftools) and that we want Snort3 installed under `/usr/local`. Once this script finishes running, run the following commands:

```
cd build
make -j$(nproc)
make install
cd ../..
```

Be aware that snort3 is a lot of code, and the `make -j` command is going to take a little while to finish running. With the minimum specs recommended, readers may be looking at half an hour or more compile time. The easiest way to test and see if Snort3 was installed correctly would be to run the following commands:

```
ldconfig
snort -V
```

`ldconfig` is responsible for scanning various directories where shared libraries are found on the system and making the system and applications aware of new libraries available for use. I've noticed several times in the past on a fresh compile of snort that if `ldconfig` isn't ran prior to trying to run snort that sometimes it can result in unexpected failure:

```
snort: error while loading shared libraries: libdaq.so.3: cannot open shared object file: No such file or directory
```

However, after running `ldconfig`, `snort -V` should return the following output:

```
root@ubuntu-2404-snort3:/usr/src/snort3-snort3-caf79b9/build# snort -V

   ,,_     -*> Snort++ <*-
  o"  )~   Version 3.1.84.0
   ''''    By Martin Roesch & The Snort Team
           http://snort.org/contact#team
           Copyright (C) 2014-2024 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using DAQ version 3.0.14
           Using LuaJIT version 2.1.1703358377
           Using OpenSSL 3.0.13 30 Jan 2024
           Using libpcap version 1.10.4 (with TPACKET_V3)
           Using PCRE version 8.39 2016-06-14
           Using ZLIB version 1.3
           Using Hyperscan version 5.4.2 2024-04-19
           Using LZMA version 5.4.5
```

4-8: These are the results that should return when running `snort -V`. If readers get an error about being unable to open shared object files, trying running `ldconfig`, then re-running `snort-V`. This output confirms that snort was able to find the core libraries necessary to run – DAQ, LuaJIT, OpenSSL, libpcap, PCRE, ZLIB, Hyperscan, and LZMA.

Snort3_Extras and OpenappID/ODP

Let's begin by decompressing `snort-openappid.tar.gz`:

```
tar -xzvf snort-openappid.tar.gz
```

then, we use the cp command to copy the `odp` directory, and all of its contents, to /usr/local/lib:

```
cp -R odp/ /usr/local/lib
```

We can confirm that the decompressed directory has been copied to `/usr/local/lib` by running:

```
ls -al /usr/local/lib/odp
```

```
root@ubuntu-2404-snort3:/usr/src# ls -al /usr/local/lib/odp/
total 212
drwxr-xr-x 4 root root   4096 Apr 28 16:44 .
drwxr-xr-x 7 root root   4096 Apr 28 16:44 ..
-rw-r--r-- 1 root root   1090 Apr 28 16:44 appid.conf
-rw-r--r-- 1 root root 127192 Apr 28 16:44 appMapping.data
-rw-r--r-- 1 root root    482 Apr 28 16:44 AUTHORS
drwxr-xr-x 2 root root   4096 Apr 28 16:44 libs
-rw-r--r-- 1 root root  21017 Apr 28 16:44 LICENSE
drwxr-xr-x 2 root root  32768 Apr 28 16:44 lua
-rw-r--r-- 1 root root   1269 Apr 28 16:44 README
-rw-r--r-- 1 root root     12 Apr 28 16:44 version.conf
```

4-9: These are the results that readers should expect to see in `/usr/local/lib/odp` if they followed the guidance above.

Next up, let's decompress the snort3_extras tarball, then cd into it:

```
tar -xzvf snort3_extras-[xxxx].tar.gz
cd snort3-snort3_extra-[yyyy]
```

Compiling the snort3_extras software is sort of similar to how we compiled snort3, but before we can run the `configure_cmake.sh` script, we have to use the export command to set a variable called `PKG_CONFIG_PATH`:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

From here, the experience is pretty similar to compiling snort3. Run the following commands:

```
./configure_cmake.sh --prefix=/usr/local
cd build
make -j$(nproc)
make install
cd ../..
```

The output from `make install` should have a large number of lines that all start with:

```
-- Installing: /usr/local/lib/snort/[…]
```

When `make install` finishes running, we're all done compiling and installing. Snort3, its libraries, and extras are all installed. The hard part is done, but now we have to download rules, and configure all of the different moving parts.

```
Install the project...
-- Install configuration: "RelWithDebInfo"
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_eapol.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_linux_sll.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_null.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_pflog.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_pbb.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_ppp.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_slip.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_token_ring.so
-- Installing: /usr/local/lib/snort/plugins/extra/codecs/cd_wlan.so
-- Installing: /usr/local/lib/snort/daq/extra/daq_socket.so
-- Installing: /usr/local/lib/snort/plugins/extra/inspectors/appid_listener.so
```

4-10: Output from make install for snort3_extras. If readers followed the instructions above, this the last step for the last piece of software we need to compile.

PulledPork3 and New Rules

Now that we have all the software in place, our next goal is to get everything we need to run snort3. Part of that involves acquiring IDS rules. We'll be using pulledpork 3 for this. Before we proceed with this. Please make sure to sign up on snort.org in order for us to download the Registered user rules at a minimum. Why?

Registered users and/or Subscribers to the paid Snort ruleset gain access to the Talos_LightSPD rules tarball. This is important because the Talos_LightSPD ruleset contain shared object rules for practically every supported release of snort 3, alongside the standard rules. Pulledpork3 is capable of finding the right shared object rules for the installed snort release and configuring them for use with our installation automatically. Be sure to log in, and grab your oink code, you'll need it to download rules.

With all of that aside, let's `mv` the `pulledpork3` directory from `/usr/src/` to `/usr/local/etc/snort`, then cd to `/usr/local/etc/snort/pulledpork3/etc/`:

```
mv /usr/src/pulledpork /usr/local/etc/snort/pulledpork/
cd /usr/local/etc/snort/pulledpork/etc/
```

Once in this directory, we're going to take the `pulledpork.conf` file and `mv` it to `pulledpork.conf.orig`, because we're going to create a new, stripped-down `pulledpork.conf`, but want to keep the original, just in case readers need to change their pulledpork configuration file later, and want a point of reference to begin:

```
mv pulledpork.conf pulledpork.conf.orig
```

```
root@ubuntu-2404-snort3:/usr/src# mv /usr/src/pulledpork3/ /usr/local/etc/snort/pulledpork3
root@ubuntu-2404-snort3:/usr/src# cd /usr/local/etc/snort/pulledpork3/etc/
root@ubuntu-2404-snort3:/usr/local/etc/snort/pulledpork3/etc# mv pulledpork.conf pulledpork.
conf.orig
root@ubuntu-2404-snort3:/usr/local/etc/snort/pulledpork3/etc# ls -al
total 16
drwxr-xr-x 2 root root 4096 Apr 28 19:17 .
drwxr-xr-x 5 root root 4096 Apr 27 23:46 ..
-rw-r--r-- 1 root root 5324 Apr 27 23:46 pulledpork.conf.orig
```

5-1: It makes more sense to place Snort-related utilities with snorts other configuration files, so we move pulledpork3 from `/usr/src/` to `/usr/local/etc/snort/`, with snort's other configuration files. Then, in `/usr/local/etc/snort/pulledpork/etc`, we take `pulledpork.conf`, and rename it to `pulledpork.conf.orig`, in case users want to use it to create a new or different configuration file for pulledpork3 later on.

Now, we need to create a new `pulledpork.conf` file. I prefer using the `vi` text editor for creating configuration files, but readers may feel free to use whatever text editor they're most comfortable with. In the `/usr/local/etc/snort/pulledpork/etc` directory, create a file named `pulledpork.conf` with the following contents:

```
LightSPD_ruleset = true
oinkcode = [your oinkcode here]
snort_blocklist = true
et_blocklist = true
blocklist_path = /usr/local/etc/lists/default.blocklist
pid_path = /var/log/snort/snort.pid
ips_policy = security
rule_mode = simple
rule_path = /usr/local/etc/rules/snort.rules
local_rules = /usr/local/etc/rules/local.rules
ignored_files = includes.rules, snort3-deleted.rules
include_disabled_rules = true
sorule_path = /usr/local/etc/so_rules/
distro = ubuntu-x64
CONFIGURATION_NUMBER = 3.0.0.3
```

Be sure to replace the text `[your oinkcode here]` with your actual oinkcode. Additionally, pay attention to the `pulledpork.conf.orig` file, and modify the `CONFIGURATION_NUMBER` as necessary to match what is in the original configuration file.

Note: the `distro` field of the pulledpork.conf file supports the following values:

```
centos-x64
debian-x64
fc-x64
opensuse-x64
ubuntu-x64
```

I highly recommend that Debian users change the distro line to `distro = debian-x64`. As for Kali Linux users, the distro appears to be able to use both `debian-x64` and `ubuntu-x64` pre-compiled shared-object rules. If users continue to experience issues related to shared object rules, consider commenting out or deleting the `distro` and `sorule_path` lines in the pulledpork.conf file.

```
root@ubuntu-2404-snort3:/usr/local/etc/snort/pulledpork3/etc# cat pulledpork.conf
LightSPD_ruleset = true
oinkcode = 35[redacted]a0
snort_blocklist = true
et_blocklist = true
blocklist_path = /usr/local/etc/lists/default.blocklist
pid_path = /var/log/snort/snort.pid
ips_policy = security
rule_mode = simple
rule_path = /usr/local/etc/rules/snort.rules
local_rules = /usr/local/etc/rules/local.rules
ignored_files = includes.rules, snort3-deleted.rules
include_disabled_rules = true
sorule_path = /usr/local/etc/so_rules/
distro = ubuntu-x64
CONFIGURATION_NUMBER = 3.0.0.3
```

5-2: The contents of your `pulledpork.conf` should look something like this. Readers are more than welcome to review the `pulledpork.conf.orig` to discover the meaning behind these configuration options, and investigate alternative configuration settings.

Because sometimes PDFs can be a bit finicky about copy/paste and formatting, I wanted to make sure that I provided an alternate source for the above configuration that readers can download directly if necessary. Please visit:
https://gist.github.com/da667/b4eb41f244e5d53519309dcba2300565

for a copy of the configuration file. Readers can download a copy of the file using wget:

https://gist.githubusercontent.com/da667/b4eb41f244e5d53519309dcba2300565/raw/5648bdb1f011a90f39a13126e84a1fb30e7bed24/pulledpork.conf

and just copy the downloaded `pulledpork.conf` directly to `/usr/local/etc/snort/pulledpork/etc/`

with the configuration file in place, there are a couple of commands we need to run, before we can download rules. Run all of the following commands:

```
mkdir /usr/local/etc/lists
mkdir /usr/local/etc/rules
mkdir /usr/local/etc/so_rules
mkdir /var/log/snort
touch /usr/local/etc/rules/snort.rules
touch /usr/local/etc/rules/local.rules
touch /usr/local/etc/lists/default.blocklist
```

These are various files and directories that need to be in place for snort and pulledpork3 to run correctly. Our next step will be to run pulledpork, and download rules for our snort3 installation. Run the following command:

```
python3 /usr/local/etc/snort/pulledpork/pulledpork.py -c
/usr/local/etc/snort/pulledpork/etc/pulledpork.conf -i -vv
```

The `-c` options allows users to specify the location of the configuration file, while `-i` allows us to ignore warnings. Pulledpork3 assumes that snort3 is running and needs to be restarted and will issue a warning if it fails to find snort3's pid file. For some reasoning, warnings are treated as failure conditions, so if this fails, the script exits immediately. Finally, `-vv` allows us to see exactly what the script is doing. The easiest way to determine if pulledpork3 completed successfully (aside from a successful exit condition) would be to check the size of the `snort.rules` file, `default.blocklist` file, and the `so_rules` directory:

```
du -h /usr/local/etc/so_rules
wc -l /usr/local/etc/rules/snort.rules
wc -l /usr/local/etc/lists/default.blocklist
```

Remember that each of these files and directories were empty just a moment ago, so if they aren't empty anymore, that's likely a good thing.

```
root@ubuntu-2404-snort3:~# du -h /usr/local/etc/so_rules/
4.2M    /usr/local/etc/so_rules/
root@ubuntu-2404-snort3:~# wc -l /usr/local/etc/rules/snort.rules
50122 /usr/local/etc/rules/snort.rules
root@ubuntu-2404-snort3:~# wc -l /usr/local/etc/lists/default.blocklist
2998 /usr/local/etc/lists/default.blocklist
```

5-3: If readers get output to the commands above that are zero bytes and/or zero lines, that's a pretty good indicator that pulledpork3 did its job correctly.

Creating and including custom.lua into snort.lua

In our installation, snort3's core configuration files are located in `/usr/local/etc/snort`. Of the files included here, the files that we will be using are the `snort.lua` and `snort_defaults.lua`. In addition to these two lua files, we'll be creating another file, `custom.lua` to define all of the configuration settings we're interested in changing for our snort3 sensor. Change directories to `/usr/local/etc/snort`, and using your preferred text editor, create the file `custom.lua` with the following contents:

```
--These configuration lines will perform the following tasks:
--enables the built-in preproc rules, and snort.rules file
--enables hyperscan as the search engine for pattern matching
--enables the IP reputation blocklist
--enables JSON alerting for snort alerts
--enables appid, the appid listener, and logging appid events.
ips =
{
    enable_builtin_rules = true,

    include = "/usr/local/etc/rules/snort.rules",

    variables = default_variables
}

search_engine = { search_method = "hyperscan" }

detection =
{

    hyperscan_literals = true,

    pcre_to_regex = true
}

reputation =
{

    blocklist = '/usr/local/etc/lists/default.blocklist',
}

alert_json =
{

    file = true,

    limit = 1000,

    fields = 'seconds action class b64_data dir dst_addr dst_ap dst_port eth_dst eth_len \
    eth_src eth_type gid icmp_code icmp_id icmp_seq icmp_type iface ip_id ip_len msg mpls \
    pkt_gen pkt_len pkt_num priority proto rev rule service sid src_addr src_ap src_port \
    target tcp_ack tcp_flags tcp_len tcp_seq tcp_win tos ttl udp_len vlan timestamp',
}

appid =
{

    app_detector_dir = '/usr/local/lib',
}

appid_listener =
{

    json_logging = true,

    file = "/var/log/snort/appid-output.log",
}
```

Just like with the `pulledpork.conf` file, I created a github gist containing the contents of the custom.lua file. Readers should cd into `/usr/local/etc/snort`, and run:

```
wget
https://gist.githubusercontent.com/da667/69e4c0bd8e8ab99d1ef851494567ac6c/raw/b39327a6e758bbe469ef198f068ff9bd8559853b/custom.lua
```

With the `custom.lua` file created, we need to include its contents into the `snort.lua` file. The two easiest ways to do this are to either:

1. User your preferred text editor, and write the line `include 'custom.lua'` at the very end of the `snort.lua` file
2. Run the command: `echo "include 'custom.lua'" >> /usr/local/etc/snort/snort.lua`

```
appid =
{
    app_detector_dir = '/usr/local/lib',
}
appid_listener =
{
    json_logging = true,
    file = "/var/log/snort/appid-output.log",
}

root@ubuntu-2404-snort3:/usr/local/etc/snort# tail -10 snort.lua

--------------------------------------------------------------------------------
-- 8. configure tweaks
--------------------------------------------------------------------------------

if ( tweaks ~= nil ) then
    include(tweaks .. '.lua')
end

include 'custom.lua'
```

6-1: Pictured above are the last few lines of the command `cat /usr/local/etc/snort/custom.lua`, and the output of `tail -10 snort.lua`. These commands confirm that the `custom.lua` file exists and was written correctly, and that the `snort.lua` file is including the `custom.lua` file as a part of its configuration.

Before we move on, I highly recommend that readers do a test run of snort, just to confirm that all of our rules and configurations options pass a "sanity" test, before moving on. Run the command:

```
snort --plugin-path=/usr/local/lib/snort --plugin-path /usr/local/etc/so_rules/ -c
/usr/local/etc/snort/snort.lua -T -v
```

This command is telling the snort to run in test (`-T`) mode, with verbose (`-v`) output. Load the OpenAppID and shared object rules from their respective directories (`--plugin-path`), and here is the configuration file (`-c`) that I want you to validate. If everything goes well, this command should return the output:

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

This means that snort didn't find any major issues with the configuration, and was able to load everything successfully. Congratulations! The hard part is done.

```
root@ubuntu-2404-snort3:/usr/src/vectorscan/build# snort --plugin-path=/usr/local/lib/snort
--plugin-path /usr/local/etc/so_rules/ -c /usr/local/etc/snort/snort.lua -T -v
```

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

6-2: Run the command: `snort --plugin-path=/usr/local/lib/snort --plugin-path /usr/local/etc/so_rules/ -c /usr/local/etc/snort/snort.lua -T -v`. This command tests to see that all of the components and rules that we have installed and configured up to this point are working correctly. We're looking for output that confirms that Snort successfully validated the configuration with zero warnings (or zero errors).

Note: If readers get output from the `snort -T` command above that looks something like:

```
ERROR: /usr/local/etc/rules/snort.rules:[xxxxx] can't compile content
…
```

Ending with messages stating there were fatal errors, it likely means that snort is running into the `libhyperscan-dev` issues I mentioned back in section 4. Here is a really quick way to see whether or not readers are running into the problem I described. Using your preferred text editor, open up file `/usr/local/etc/snort/custom.lua` and change line number 18 from:

```
    Hyperscan_literals = true,
```

To:

```
--    hyperscan_literals = true,
```

And save the file, then re-run the `snort -T` command above. If the command completes with zero errors, bad news, the version of libhyperscan available in the software repos is not working properly, and users will need to uninstall `libhyperscan-dev`, and install vectorscan, as described in section 4.

```
18     hyperscan_literals = true,
```

```
18 -    hyperscan_literals = true,
```

6-3: Run into problems with the `snort -T` command above? Did it out put a bunch of lines that contain the string "`can't compile content`"? comment out line 18 in the `custom.lua` file, then re-run the `snort -T` command listed above. If the command runs with zero warnings or errors, readers will need to follow the instructions in the section *Ubuntu 24.04 and non-x86_64 Users: Download and Compile Vectorscan*.

Cleanup and Automation

At this point, we know that Snort 3 is installed correctly, but there are some cleanup tasks that need to be done before its ready for prime time. Snort doesn't need to run as the root user, so we should create a user and group to drop privileges to once all of the tasks that require root privileges are done. That user is going to require owner ship of the /var/log/snort directory to drop log files there, so that will need to be corrected. We need to create a service file to control snort3, and have it automatically start for us on reboot. Finally, we should probably have some automation in place to automatically update the system once weekly, so let's get that covered as well.

Creating the `snort` user and group, and granting necessary file/folder permissions

Run the follow commands with root privileges:

```
groupadd snort
useradd -r -s /sbin/nologin -g snort snort
chmod 5775 /var/log/snort
chown -R snort:snort /var/log/snort
```

These commands will create the `snort` user and group, change the permissions for the `/var/log/snort` directory to only allow `root` and the `snort` user/group to modify the logs, but allow anyone else the ability to read them, then finally change the ownership of the `/var/log/snort` directory to the snort user and group. If your environment requires more restrictive or loose permissions, modify the `chmod` and `chown` commands to better suit your needs.

Creating The `snort3.service` Systemd Service

Next up, we need to define a systemd service file for Snort 3. Fortunately, I have just the thing on-hand. Take the script below and using whatever text editor the reader is most comfortable with, copy the following into a file named `snort3.service`:

```
[Unit]
Description=Snort Daemon
After=syslog.target network.target

[Service]
Type=simple
ProtectHome=true
ProtectKernelTunables=true
ProtectKernelModules=true
ProtectControlGroups=true
ExecStartPre=/usr/sbin/ip link set up promisc on arp off multicast off dev snort_iface1
ExecStartPre=/usr/sbin/ethtool -K snort_iface1 rx off tx off gro off lro off
ExecStart=/usr/local/bin/snort --plugin-path=/usr/local/lib/snort --plugin-path
/usr/local/etc/so_rules/ -c /usr/local/etc/snort/snort.lua -D -u snort -g snort -l
/var/log/snort -m 0x1b --create-pidfile -s 65535 -k none -i snort_iface1
ExecStopPost=/usr/bin/rm -f /var/log/snort/snort.pid
Restart=on-failure
RestartSec=120s

[Install]
WantedBy=multi-user.target
```

As before, to accommodate the most users, I have placed the contents of this file into a github gist. To download this file, run the following commands:

```
cd /usr/src
wget
https://gist.githubusercontent.com/da667/28ed48c59f163aad31623f319851c07c/raw/96431875ea862ff5
d1e4a2f15c6045086e38c2a4/snort3.service
```

Readers may have noticed the line **snort_iface1** in multiple places throughout the systemd service file above. Those lines need to be replaced with the name of the interface that snort will be sniffing traffic from, for users planning to use their snort 3 instance in IDS mode. (or IPS mode, for those who configure that themselves).

Recall that this guide is designed to help provide a Snort 3 sensor operating in IDS mode. Also recall that the minimum recommendation for a Snort 3 sensor is two network interfaces. There is no way I can know the name of the network interface for readers' Snort 3 sensor, so this is what you need to do:

Run the following command

```
ip --brief a | egrep -v "lo"
```

This is short hand for the Linux command `ip address`, with brief output. We only need to know the names of the interfaces and their IP addresses. We then pass that through `egrep` with the `-v` option to exclude lines of output that have the content we specify. In this case, we're not interested in seeing the `lo` (loopback) interface.

My output for this command looks like this:

```
root@ubuntu-2404-snort3:/usr/src# ip --brief a | egrep -v "lo"
ens160           UP             10.0.0.168/24 metric 100 2601:408:500:b100::f60f/128 2601:40
8:500:b100:20c:29ff:fe56:5459/64 fe80::20c:29ff:fe56:5459/64
ens192           DOWN
```

7-1: The command `ip --brief a` shows condensed output that displays a list of interface names, whether or not they are `UP` or `DOWN` (network cable plugged into to a live network port) as well as any IPv4 or IPv6 addresses assigned to that interface. If readers use remote connectivity to connect to their sensor, which IP address is used to connect to the sensor? That interface is the network management interface, while other network interfaces (if there are more than two) can be assigned to the span/tap port. To figure out which interface should be the one to replace the string `snort_iface1` in the `snort3.service` file, plug the active span/tap port (or a live network connection to your local network) into the physical interface that will be used for sniffing traffic, then re-run `ip --brief a | egrep -v "lo"`. Note which interface changes its status from `DOWN` to `UP`. That is the interface name to input into the `snort3.service` file.

`ens192` is going to be the IDS interface for my Snort sensor to sniff traffic, where `ens160` is the interface that I use for managing the sensor remotely, connecting it to a SIEM, etc. If readers are in a position where both of their interfaces have an IP address assigned, which IP address is used for remote access? `Snort_iface1` is the interface that is not being used currently. For readers who have systems with more than two network interfaces, and have no idea which interface carries network traffic, and which will be used to sniff network traffic, `ip --brief a` will display IP addresses, the interface name with the IP address used to connect to the sensor is the network management interface. Now for determining which physical interface maps to the tap/span port, the quickest way to determine that would be to plug in the tap or span port (or any network

cable with a live connection to the local network), re-run `ip --brief a`, and see which interface changes its status from `DOWN` to `UP`. Replace the string `snort_iface1` with the name of that network interface.

There are a few easy ways to do this. Of course, readers are welcome to use their text editor and manually edit lines 11,12, and 13 replacing `snort_iface1` with the name of their network interface they will use to sniff traffic in IDS mode, or you can do something like:

```
sed -i.bak 's/snort_iface1/ens192/g' snort3.service
```

The `sed` command allows one to quickly and effectively find and replace all instances of `snort_iface1` with the name of the network interface that will be used for IDS mode. This command uses the `-i` option to edit the `snort3.service` file in place. We use the argument `-i.bak` to modify the existing file, but before we do so, create a backup of the file named `snort3.service.bak` (just in case a backup copy is needed later). The expression `'s/snort_iface1/ens192/g' snort3.service` tells `sed` to replace all instances `snort_iface1` with `ens192`. (be sure to replace `ens192` with the name of the desired network interface). The `g` option ensures that this change is made throughout the entire `snort3.service` file.

```
root@ubuntu-2404-snort3:/usr/src# grep snort_iface1 snort3.service
ExecStartPre=/usr/sbin/ip link set up promisc on arp off multicast off dev snort_iface1
ExecStartPre=/usr/sbin/ethtool -K snort_iface1 rx off tx off gro off lro off
ExecStart=/usr/local/bin/snort --plugin-path=/usr/local/lib/snort --plugin-path /usr/local/e
tc/so_rules/ -c /usr/local/etc/snort/snort.lua -D -u snort -g snort -l /var/log/snort -m 0x1
b --create-pidfile -s 65535 -k none -i snort_iface1
root@ubuntu-2404-snort3:/usr/src# sed -i.bak 's/snort_iface1/ens192/g' snort3.service
root@ubuntu-2404-snort3:/usr/src# grep ens192 snort3.service
ExecStartPre=/usr/sbin/ip link set up promisc on arp off multicast off dev ens192
ExecStartPre=/usr/sbin/ethtool -K ens192 rx off tx off gro off lro off
ExecStart=/usr/local/bin/snort --plugin-path=/usr/local/lib/snort --plugin-path /usr/local/e
tc/so_rules/ -c /usr/local/etc/snort/snort.lua -D -u snort -g snort -l /var/log/snort -m 0x1
b --create-pidfile -s 65535 -k none -i ens192
```

7-2: This screen capture demonstrates how the command `sed -i.bak 's/snort_iface1/ens192/g' snort3.service` works. This command finds all instances of snort_iface1 in the file, and replaces them with ens192. This systemd service script is now ready to be used.

Note: In section 5, I notified readers that if they are having problems with shared object rules not operating properly, that I recommend avoiding their use entirely. For users who disabled shared object rules in pulledpork.conf earlier, remove the option `--plugin-path=/usr/local/etc/so_rules/` from the `ExecStart=` line to avoid any problems.

Next up, we need to put the snort3.service file into place, and tell systemd its ready for use. Run the following commands:

```
cp /usr/src/snort3.service /etc/systemd/system/
systemctl daemon-reload
systemctl enable snort3.service
systemctl start snort3.service
systemctl status snort3.service
```

```
root@ubuntu-2404-snort3:/usr/src# cp /usr/src/snort3.service /etc/systemd/system/
root@ubuntu-2404-snort3:/usr/src# systemctl daemon-reload
root@ubuntu-2404-snort3:/usr/src# systemctl enable snort3.service
Created symlink /etc/systemd/system/multi-user.target.wants/snort3.service → /etc/systemd/sy
stem/snort3.service.
root@ubuntu-2404-snort3:/usr/src# systemctl start snort3.service
root@ubuntu-2404-snort3:/usr/src# systemctl status snort3.service
● snort3.service - Snort Daemon
     Loaded: loaded (/etc/systemd/system/snort3.service; enabled; preset: enabled)
     Active: active (running) since Mon 2024-05-06 01:59:28 UTC; 1min 5s ago
    Process: 7699 ExecStartPre=/usr/sbin/ip link set up promisc on arp off multicast off de>
    Process: 7701 ExecStartPre=/usr/sbin/ethtool -K ens192 rx off tx off gro off lro off (c>
   Main PID: 7703 (snort3)
```

7-3: Most of these commands will have no output, except for the last one. We're looking for green text indicating that the service is enabled, its active, and the main paid is still running.

Next, to be *really* sure that snort3 is running, let's run:

ps -ef | grep snort; cat /var/log/snort/snort.pid

```
root@ubuntu-2404-snort3:/usr/src# ps -ef | grep snort; cat /var/log/snort/snort.pid
snort       7703       1 19 01:59 ?        00:00:55 /usr/local/bin/snort --plugin-path=/usr/
local/lib/snort --plugin-path /usr/local/etc/so_rules/ -c /usr/local/etc/snort/snort.lua -D
-u snort -g snort -l /var/log/snort -m 0x1b --create-pidfile -s 65535 -k none -i ens192
root        7723    2287  0 02:04 pts/0    00:00:00 grep --color=auto snort
7703
```

7-4: The `ps -ef` command is used to get a complete list of running processes, we then `grep` for `snort`. Two lines should appear – one for the `snort` process itself, and a second for our `grep` command. The second command reads the contents of `/var/log/snort/snort.pid`, which is a text file that contains the process ID number for the snort process while its running. Notice that the number 7703 matches the number next to the snort command is the `ps -ef` output, and that it matches the `Main PID` number in the `systemctl status snort3.sevice` command from *figure 7-3*.

Alright, now let's be sure that there are brakes on this train. Run the command:

```
systemctl stop snort3.service
ls -al /var/log/snort/ | grep snort.pid
ps -ef | grep snort
```

The first command might take a moment or two to come back. Snort has a lot of stuff it needs to clean up before the process can die cleanly.

```
root@ubuntu-2404-snort3:/usr/src# systemctl stop snort3.service
```

```
root@ubuntu-2404-snort3:/usr/src# ls -al /var/log/snort | grep snort.pid
root@ubuntu-2404-snort3:/usr/src# ps -ef | grep snort
root        7761    2287  0 02:11 pts/0    00:00:00 grep --color=auto snort
```

7-5: The first command is systemd telling snort 3 to shut down gracefully. Sometimes, this takes a minute, as there is a lot of stuff snort 3 has to do before shutting down gracefully. The other two commands are to confirm that there isn't an active pid file for snort 3, and that the snort process isn't still in the process list. With this, we can confirm that the systemd3 service installed correctly and that snort3 is capable of being started, stopped, and restarted by systemd correctly.

With this, we've got service control for snort 3 wrapped up cleanly.

Automatic System and Rule Updates via updater and /etc/cron.weekly

Keeping IDS systems up to date is an important task. Automating this task to be done regularly is a good idea to ensure that system updates, and new IDS rules are applied on a regular basis. Most Debian-based systems have a set of directories `/etc/cron.[hourly|daily|weekly|monthly|yearly]` that can be used to execute tasks on a regular basis. We're going to take advantage of this, and place a custom script called `updater` into the `/etc/cron.weekly` directory. If readers want to see when exactly the scripts these various directories will be executed, run the command cat `/etc/crontab`. Learning about the various aspects of the crontab are outside of the scope of this document, but a good starting point is the documentation in the crontab itself, as well as the website https://crontab.guru. Additionally, if readers want system updates and/or rules to be applied more or less frequently, the updater script can be placed into the other `/etc/cron.[time period]` directories. Begin by running cd `/etc/cron.weekly/` to change directories. Next up, readers can use their favorite text editor, create a file in the `/etc/cron.weekly/` directory titled `updater`, and input the following contents:

```
#!/bin/bash
#updater.sh - Weekly update script
#checks for updates, downloads them, then reboots the system.
#place this script in /etc/cron.weekly, ensure it is owned by root (chown root:root
/etc/cron.weekly/updater)
#ensure the script has execute permissions (chmod 700 /etc/cron.weekly/updater)
#if you want updates to run once daily or monthly, you could also place this script into
cron.daily, or cron.weekly.
#alternatively, edit /etc/crontab to create a crontab entry.

export DEBIAN_FRONTEND=noninteractive
apt-get -q update
apt-get -y -q dist-upgrade
python3 /usr/local/etc/snort/pulledpork3/pulledpork.py -c
/usr/local/etc/snort/pulledpork3/etc/pulledpork.conf -i -vv
logger updater cron job ran successfully. rebooting system
init 6
exit 0
```

Save the contents of the file, then run the command:

```
chmod 700 /etc/cron.weekly/updater
```

And the script should automatically execute on the next cron.weekly interval. As always, A copy of this script is available as a github gist. Run the following commands to download it, if there are difficulties transcribing the script above:

```
cd /etc/cron.weekly/
wget
https://gist.githubusercontent.com/da667/5f03ce60c51ccbae6365159637db7ed6/raw/9f62367cb9c826c5
c237e6a66cab47543215b592/updater
chmod 700 updater
```

```
root@ubuntu-2404-snort3:~# cd /etc/cron.weekly/
root@ubuntu-2404-snort3:/etc/cron.weekly# wget https://gist.githubusercontent.com/da667/5f03
ce60c51ccbae6365159637db7ed6/raw/9f62367cb9c826c5c237e6a66cab47543215b592/updater
--2024-05-06 16:22:01--  https://gist.githubusercontent.com/da667/5f03ce60c51ccbae6365159637
db7ed6/raw/9f62367cb9c826c5c237e6a66cab47543215b592/updater
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.108.133, 185.19
9.109.133, 185.199.110.133, ...
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.108.133|:443..
. connected.
HTTP request sent, awaiting response... 200 OK
Length: 751 [text/plain]
Saving to: 'updater'

updater               100%[===========================>]     751  --.-KB/s    in 0s

2024-05-06 16:22:01 (17.3 MB/s) - 'updater' saved [751/751]

root@ubuntu-2404-snort3:/etc/cron.weekly# chmod 700 updater
```

7-6: Change directories to `/etc/cron.weekly`, then either using a text editor, to copy the `updater` script, or by using `wget` to download it, place the updater script into `/etc/cron.weekly`, then use `chmod 700 updater` to grant full permissions (including execute) to the file owner (`root`).

For users who want to confirm that the script runs correctly, inside of the `/etc/cron.weekly` directory, run the command:

```
./updater
```

This tells the current shell to execute the name of the file in the current directory. Please be aware that this ***will*** cause the system to reboot via the `init 6` command.

```
root@ubuntu-2404-snort3:/etc/cron.weekly# ./updater
```

```
Program execution complete.
root@ubuntu-2404-snort3:/etc/cron.weekly#
```

7-7: Kicking off the updater script manually is the easiest way to ensure it works as intended. In the `/etc/cron.weekly` directory run `./updater` to execute the commands in the shell script. Upon finishing, the script should give the user control of the command line session once more, before immediately rebooting the system.

On reboot, log back in, and become the `root` user, then run:

```
cat /var/log/syslog | grep updater
```

```
2024-05-06T16:36:45.931984+00:00 ubuntu-2404-snort3 root: updater cron job ran successfully.
 rebooting system
```

7-8: Surefire confirmation on Ubuntu and Ubuntu-like systems that the updater script ran successfully.

Note: On Debian 12, `/var/log/messages` and `/var/log/syslog` are not written to `/var/log` by default, use the command `journalctl -g updater` instead:

```
root@debian-12-snort3:~# journalctl -g updater
Apr 22 12:54:39 debian-12-snort3 root[1888]: updater cron job ran successfully. rebooting s>
-- Boot a2d60417d0ac4403992fa667378f2234 --
-- Boot d0468a50641d419d92eb766553195690 --
-- Boot 9064f267907a4b4b8f42223547f90a79 --
Apr 28 06:47:05 debian-12-snort3 root[7356]: updater cron job ran successfully. rebooting s>
-- Boot 71c9299a7e754ed39438afa290262e86 --
May 05 06:48:28 debian-12-snort3 root[4133]: updater cron job ran successfully. rebooting s>
-- Boot 2e6bdb1104664446a3f9c1e4add5f8e1 --
```

7-9: Debian 12 doesn't write `/var/log/syslog` by default, instead letting `journalctl` handle it. Run the command `journalctl -g updater` to find entries that contain the text "updater", instead.

Next up, we can run the same set of commands we ran previously to confirm snort3 is running:

```
ps -ef | grep snort; cat /var/log/snort/snort.pid
systemctl status snort3.service
```

```
root@ubuntu-2404-snort3:~# ps -ef | grep snort; cat /var/log/snort/snort.pid
snort        826       1  8 16:37 ?        00:01:10 /usr/local/bin/snort --plugin-path=/usr/
local/lib/snort --plugin-path /usr/local/etc/so_rules/ -c /usr/local/etc/snort/snort.lua -D
-u snort -g snort -l /var/log/snort -m 0x1b --create-pidfile -s 65535 -k none -i ens192
root        1050     989  0 16:50 pts/0    00:00:00 grep --color=auto snort
826
root@ubuntu-2404-snort3:~# systemctl status snort3.service
● snort3.service - Snort Daemon
     Loaded: loaded (/etc/systemd/system/snort3.service; enabled; preset: enabled)
     Active: active (running) since Mon 2024-05-06 16:37:15 UTC; 13min ago
    Process: 749 ExecStartPre=/usr/sbin/ip link set up promisc on arp off multicast off dev>
    Process: 783 ExecStartPre=/usr/sbin/ethtool -K ens192 rx off tx off gro off lro off (co>
   Main PID: 826 (snort3)
```

7-10: After rebooting, we can see that snort 3 is in the process list with pid 826. This matches the output from the `snort.pid` file. Additionally, can confirm that `snort3.service` is running normally, and that the main PID of the snort process is 826. Of course, readers' PID numbers will different, but the `ps -ef`, `snort.pid`, and `systemctl status` PID should all be identical.

At this point, readers have a fully functioning Snort3 sensor, that automatically updates itself, and updates its rules, and we're done. The snort 3 sensor may need some tuning and configuration to better fit the environment it will be analyzing traffic in, but tuning is outside of the scope of this guide. However, there are some additional things that can be configured. Check out the Extra Credit section, to see if any of the subjects covered are of interest.

Extra Credit

As the name of this section implies, Readers should be done at this point and have a functional Snort 3 sensor that they can tinker with as they desire. This section provides users with a couple of extra things they can do with their sensors. Specifically, there is a section on how to integrate the Proofpoint Emerging Threats ruleset into snort 3 via the tool, `snort2lua`. Additionally, we will talk about how to integrate the newly created Snort3 sensor into Splunk.

`snort2lua` and the Emerging Threats Snort 2.9 Ruleset

The `snort2lua` tool is available with the snort 3 source. When snort3 was compiled earlier, this should have also resulted in a copy of `snort2lua` being compiled and installed on the system. This can be confirmed with the command:

```
which snort2lua
```

```
root@ubuntu-2404-snort3:/var/log/snort# which snort2lua
/usr/local/bin/snort2lua
```

8-1: The `which` command checks all the filesystem paths defined in the `PATH` variable for the current session. We installed snort to `/usr/local/bin`, and `snort2lua` should have been installed to the same directory.

Next up, readers need a copy of the emerging threats ruleset. Let's do a little housekeeping, first. Run these commands:

```
cd /usr/src
mkdir -p et_rules/snort2lua
cd et_rules
```

We're just creating a directory to store the Emerging Threats rule tar file, then a subdirectory for readers to perform their conversion in. Next up, in the `/usr/src/et_rules` directory, readers will need to download a copy of the Emerging Threats ruleset for snort 2.9.0. If readers have an Emerging Threats oinkcode, run the following command:

```
wget https://rules.emergingthreatspro.com/[16 character oinkcode]/snort-
2.9.0/etpro.rules.tar.gz
```

Or, if readers do not have an ETPRO oinkcode, or just wish to utilize the ETOPEN ruleset:

```
wget http://rules.emergingthreats.net/open/snort-2.9.0/emerging.rules.tar.gz
```

The end result will either be `etpro.rules.tar.gz`, or `emerging.rules.tar.gz`. For this demonstration I will be assuming that users will be using the ETOPEN ruleset. The next thing we need to do is run

```
tar -xzvf [filename]
```

Of course, change `[filename]` to either `emerging.rules.tar.gz`, or `etpro.rules.tar.gz`. This will decompress the tarball, and create a `rules` directory full of `.rules` files.

```
root@ubuntu-2404-snort3:~/snort2lua/rules# cd /usr/src/
root@ubuntu-2404-snort3:/usr/src# mkdir -p et_rules/snort2lua
root@ubuntu-2404-snort3:/usr/src# cd et_rules
root@ubuntu-2404-snort3:/usr/src/et_rules# wget http://rules.emergingthrea
2.9.0/emerging.rules.tar.gz
--2024-05-06 20:52:32--  http://rules.emergingthreats.net/open/snort-2.9.0
r.gz
Resolving rules.emergingthreats.net (rules.emergingthreats.net)... 44.196.
.217, 50.19.69.237, ...
Connecting to rules.emergingthreats.net (rules.emergingthreats.net)|44.196
nnected.
HTTP request sent, awaiting response... 200 OK
Length: 4289270 (4.1M) [application/octet-stream]
Saving to: 'emerging.rules.tar.gz'

emerging.rules.tar.gz  100%[=========================>]   4.09M  3.10MB

2024-05-06 20:52:33 (3.10 MB/s) - 'emerging.rules.tar.gz' saved [4289270/4

root@ubuntu-2404-snort3:/usr/src/et_rules# tar -xzvf emerging.rules.tar.gz
```

```
root@ubuntu-2404-snort3:/usr/src/et_rules# ls
emerging.rules.tar.gz  rules  snort2lua
```

8-2: Change directories back to `/usr/src` then run `mkdir -p et_rules/snort2lua` -- this creates the `et_rules` directory, followed by a `snort2lua` subdirectory within the `et_rules` directory. Next, change directories into the newly created `/usr/src/et_rules` directory, then use the `wget` command to download either the ETPRO or ETOPEN rules tarball for snort 2.9.X. Once downloaded, use `tar -xzvf [filename]` to decompress the rule tarball, and create a new directory `/usr/src/et_rules/rules`, that will be full of `.rules` files. We'll be processing these rules in a moment.

Next, there is a particular file file in both the ETPRO and the ETOPEN rules directory that needs to be moved. For readers who downloaded the ETPRO ruleset, run the following command from the `/usr/src/et_rules` directory:

```
mv rules/deleted.rules rules/deleted.txt
```

For readers who downloaded the ETOPEN ruleset, run:

```
mv rules/emerging-deleted.rules mv rules/emerging-deleted.txt
```

Next, run the following command from the `/usr/src/et_rules` directory:

```
cat rules/*.rules >> snort2lua/et_all_290.rules
cd /usr/src/et_rules/snort2lua
```

This will concatenate (combine) all of the individual .rules files from `/usr/src/et_rules/rules` into a single file `et_all_290.rules` and place that resulting file into the `/usr/src/et_rules/snort2lua` directory. Next, we change directories into the `/usr/src/et_rules/snort2lua` directory, so we can convert the rules.

```
root@ubuntu-2404-snort3:/usr/src/et_rules# mv rules/emerging-deleted.rules rules/emerging-de
leted.txt
root@ubuntu-2404-snort3:/usr/src/et_rules# cat rules/*.rules >> snort2lua/et_all_290.rules
root@ubuntu-2404-snort3:/usr/src/et_rules# cd snort2lua/
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# ls
et_all_290.rules
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# wc -l et_all_290.rules
97265 et_all_290.rules
```

8-3: Users will need to use the `mv` command to rename `deleted.rules` or `emerging-deleted.rules` to `deleted.txt/emerging-deleted.txt` – the deleted rules category should not be used in a production environment and will definitely cause problems with `snort2lua`. Next up, readers use the cat command to combine every `.rules` file in the `/usr/src/et_rules/rules` directory into the file `et_all_290.rules` and place that into the directory `/usr/src/et_rules/snort2lua`. Finally, we change directories into the `/usr/src/et_rules/snort2lua` directory.

Next, students will need to run snort2lua, and let it do all the heavy lifting:

```
snort2lua -c et_all_290.rules -r et_snort3.rules
```

```
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# snort2lua -c et_all_290.rules -r et_sno
rt3.rules
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# ls -al
total 67432
drwxr-xr-x 2 root root     4096 May  6 21:59 .
drwxr-xr-x 4 root root     4096 May  6 21:58 ..
-rw-r--r-- 1 root root 30768319 May  6 21:59 et_all_290.rules
-rw-r--r-- 1 root root 30293483 May  6 21:59 et_snort3.rules
-rw-r--r-- 1 root root  7976288 May  6 21:59 snort.lua
```

8-4: Next, readers will run the command `snort2lua -c et_all_290.rules -r et_snort3.rules`. If everything works as expected, there will be zero output, and the prompt will just return. Users will need to run the `ls` command. There should be two new rules in the `snort2lua` directory: `et_snort3.rules`, and `snort.lua`.

Running the `ls` command should reveal two new files in the directory: `et_snort3.rules`, and `snort.lua`. The `snort.lua` file is a massive file containing a list of event_filters for every single rule in the emerging threats ruleset. Even the rules that are disabled. By default, this file is massive, but with a few command line utilities, we can trim the file down considerably.

We will also be renaming this file to `et_thresholds.lua`, moving it to `/usr/local/etc/snort/et_thresholds.lua`, and creating an include statement for it in the `custom.lua` file. Additionally, we'll be moving the `et_snort3.rules` file to `/usr/local/etc/rules`, and adding in an include statement for those rules in `custom.lua` as well. Finally, the ET ruleset makes some slight assumptions about the default HOME_NET variable that can make some rules react poorly with snort3, and its default `HOME_NET` value of `any`. We will work together on redefining the default `HOME_NET` value to prevent this error from occurring, and finally, run snort 3 in test mode (`-T`) to confirm that it is able to read all of the new Emerging Threats rules successfully, and validate the new configuration settings. It's a tall order, but I'll help guide you through it. First things first, let's fix the snort.lua file that snort2lua made, and turn it into `et_thresholds.lua`. in the `/usr/src/et_rules/snort2lua` directory, run the following commands:

```
sed '/^--\[\[/,/\]\]/d;s/--\[\[.*$//' snort.lua >> et_thresholds.tmp.lua
sed -i 1,26d et_thresholds.tmp.lua
head -n -6 et_thresholds.tmp.lua >> et_thresholds.lua
```

```
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# du -h snort.lua
7.7M    snort.lua
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# sed '/^--\[\[/,/\]\]/d;s/--\[\[.*$//' s
nort.lua >> et_thresholds.tmp.lua
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# sed -i 1,26d et_thresholds.tmp.lua
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# head -n -6 et_thresholds.tmp.lua >> et_
thresholds.lua
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# du -h et_thresholds.lua
340K    et_thresholds.lua
```

8-5: This series of moderately complex sed and head commands with output redirection take a file that is nearly 8MB of text, and trims it down to about 340KB. Which is still a lot of text, but much more manageable.

Okay, so what did those commands just do? We ran a pretty complex `sed` statement against `snort.lua`, and redirected the output to `et_thresholds.tmp.lua`. In a nutshell, there are a lot of commented out lines in this file that gets generated for rules that are commented out. We don't need these lines, and we don't want them, so we surgically remove them. Next, the header of this file is kind of pointless. All we need are the `event_filter` statements. Finally, the head command we run tells head to read all of the contents of the `et_thresholds.tmp.lua`, except for the final six lines. Just like the first 26 lines of the file, we don't about these lines and we want them trimmed from the file. We take the output from this command, and redirect it to a file named `et_thresholds.lua`, our finished product.

Before going any further, readers might be wondering: what is an `event_filter`? In layman's terms, it's a way of defining how many times a rule's criteria needs to match in network traffic before generating an alert, how many times a rule is allowed to generate alerts for a given time period, or in some cases, both. Snort 2.9 has the ability to define suppressions/thresholds in a separate file, or to do so inline, in the rule body itself via the `threshold` keyword. The `threshold` keyword was deprecated with snort 3, and replaced with another option called `detection_filter`, that is similar, but not exactly the same. The `snort2lua` tool takes `threshold` rule options, and converts then to `event_filter` arguments and stuffs them all into a lua configuration file (`snort.lua`).

Now that we have the files et_thresholds.lua, and et_snort3.rules, let's copy them to /usr/local/etc/snort, and `/usr/local/etc/rules`, respectively. In the `/usr/src/etc_rules/snort2lua` directory, run the commands:

```
cp et_thresholds.lua /usr/local/etc/snort/
cp et_snort3.rules /usr/local/etc/rules
cd /usr/local/etc/snort
```

```
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# cp et_thresholds.lua /usr/local/etc/sno
rt/
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# cp et_snort3.rules /usr/local/etc/rules
/
root@ubuntu-2404-snort3:/usr/src/et_rules/snort2lua# cd /usr/local/etc/snort/
```

8-6: use the `cp` command to copy the `et_thresholds.lua` file to `/usr/local/etc/snort`, and `et_snort3.rules` to `/usr/local/etc/rules`, then change directories to `/usr/local/etc/snort.`

Our next tasks involve editing both `snort.lua`, and `custom.lua`. Readers should use their preferred text editor and open `snort.lua`. Near the beginning of the file is a section titled:

```
1. configure defaults
```

And not far under that is a line that should read:

```
HOME_NET = 'any'
```

We need to change this line to:

HOME_NET = ’10.0.0.0/8,172.16.0.0/12,192.168.0.0/16’

Collectively, these three network ranges are known as RFC1918 addresses, or “Private” IP addresses. If readers know what their actual home network range is, they should enter that into the HOME_NET field instead. For example, my HOME_NET is 172.16.2.0/24, so instead of covering the ENTIRE RFC1918 address space, I can set HOME_NET to a single /24 network like this:

HOME_NET = ‘172.16.2.0/24’

But for those readers who have no idea what their HOME_NET value should be, use value provided above (The entire RFC1918 IPv4 address space). When finished, save changes to the snort.lua file, and exit the text editor.

```
---------------------------------------------------
-- 1. configure defaults
---------------------------------------------------

-- HOME_NET and EXTERNAL_NET must be set now
-- setup the network addresses you are protecting
HOME_NET = '10.0.0.0/8,172.16.0.0/12,192.168.0.0/16'
```

8-7: Using their preferred text editor, students should navigate to this portion of the snort.lua file in /usr/local/etc/snort, and change the HOME_NET variable from its default value of ‘any’ to either their desired HOME_NET value, or if that is not known, set it to the RFC1918 address range: ’10.0.0.0/8,172.16.0.0/12,192.168.0.0/16’. When finished, save and exit the snort.lua file.

Next up, we need to edit a couple of lines in the custom.lua file we created earlier. Again, readers should open their preferred text editor and find the following lines:

```
ips =
{
    enable_builtin_rules = true,


    include = "/usr/local/etc/rules/snort.rules",
    variables = default_variables
}
```

Add the lines below in **bold**:

```
ips =
{
    enable_builtin_rules = true,
    rules = [[
        include /usr/local/etc/rules/snort.rules
        include /usr/local/etc/rules/et_snort3.rules
        include /usr/local/etc/rules/local.rules
        ]],

    variables = default_variables
}
```

next, scroll to the very bottom of the `custom.lua` file:

```
appid_listener =
{
    json_logging = true,
    file = "/var/log/snort/appid-output.log",
}
```

and add the following line in **bold**:

```
appid_listener =
{
    json_logging = true,
    file = "/var/log/snort/appid-output.log",
}
include 'et_thresholds.lua'
```

Save changes to the file `custom.lua` file, and exit the text editor.

```
ips =
{
    enable_builtin_rules = true,

    rules = [[
        include /usr/local/etc/rules/snort.rules
        include /usr/local/etc/rules/et_snort3.rules
        include /usr/local/etc/rules/local.rules
        ]],

    variables = default_variables
}
```

```
appid_listener =
{
    json_logging = true,
    file = "/var/log/snort/appid-output.log",
}
include 'et_thresholds.lua'
```

8-8: Readers should open `custom.lua` in their preferred text editor. In the ips section, modify the portion where the snort.rules file is being included to:

```
    rules = [[
        include /usr/local/etc/rules/snort.rules
        include /usr/local/etc/rules/et_snort3.rules
        include /usr/local/etc/rules/local.rules
        ]],
```

inside of the curly braces. Next, at the very bottom of the file, add the line:

`include 'et_thresholds.lua'`

Save changes, and exit the text editor.

The include statements in the ips section tell snort to use the et_snort3.rules file. Additionally, in the event readers want to try their hand at creating their own custom rules for snort to use in the future, we create an include statement for the local.rules file as well. Its empty for now, but that shouldn't cause any problems for Snort 3 on startup. For future reference, it is highly recommended for users creating their own custom rules to use sid values between 1000000 and 1999999. For more information, check out sidallocation.org. One last task remains: validating the changes to the snort 3 configuration files. Rune the following command:

```
snort --plugin-path=/usr/local/lib/snort --plugin-path /usr/local/etc/so_rules/ -c
/usr/local/etc/snort/snort.lua -T -v
```

Once again, we're looking for the output:

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

`snort2lua`, and `snort.rej` entries

Occasionally, readers might run into issues with `snort2lua` being able to convert all of the emerging threats rules cleanly. If `snort2lua` failed to convert a rule successfully, this will be mentioned in the command output, and a `snort.rej` file will be created in the directory `snort2lua` was ran from (`/usr/src/et_rules/snort2lua`):

```
ERROR: 4 errors occurred while converting
ERROR: see snort.rej for details
```

8-9: If `snort2lua` can't make sense of a particular snort rule, its vocal about informing the user, but does not stop processing rules being converted.

The `snort.rej` file will look something like this:

```
--[[    FAILED RULES CONVERSIONS:
 These rules have invalid rule options:


    Failed to convert rule: alert tcp $HOME_NET any -> $EXTERNAL_NET
        $HTTP_PORTS (msg:"ET TROJAN BunnyLoader 3.0 Initial Checkin";
        flow:established,to_server; flowbits:set,ET.BunnyLoaderv3.Checkin;
        content:"GET"; pcre:"/^\/[A-Za-z0-9]{12,14}/U"; http_method;
        content:"/gate.php?ipaddress|3d|"; http_uri; fast_pattern;
        content:"hostname|3d|"; http_uri; content:"version|3d|"; http_uri;
        content:"system|3d|"; http_uri; content:"privileges|3d|"; http_uri;
        content:"arch|3d|"; http_uri; content:"antivirus|3d|"; http_uri;
        content:"key|3d|"; http_uri; content:"enc_key|3d|"; http_uri;
        reference:url,twitter.com/ViriBack/status/1757953408747569644;
        reference:md5,194118c43c65faad06bf5ff6cd9b52a2;
        classtype:trojan-activity; sid:2050885; rev:1;
        metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit,
        affected_product Windows_11, attack_target Client_and_Server,
        created_at 2024_02_15, deployment Perimeter, former_category MALWARE,
        malware_family BunnyLoader, performance_impact Low, confidence High,
        signature_severity Major, updated_at 2024_02_15;)
    ^^^^ unknown_option=http_method
```

8-10: Some output from a `snort.rej` file, indicating which rules failed to convert, and why.

The entirety of the rule that failed to convert will be dumped into the file, followed by lines that look like:

```
^^^^ [root cause of failure]
```

For example, in figure 8-10:

```
^^^^ unknown_option=http_method
```

In some cases, the modifier that snort2lua refers to may be used in the rule more than once, making troubleshooting a little bit hard, but in this case, `snort2lua` doesn't like that the `http_method` is after `pcre` pattern. Moving that instance of `http_method` to where its's called immediately after the GET content match would correct this problem, and is how I corrected this problem in the ET ruleset.

Unless readers are specifically seeking a challenge and would like to debug rules that `snort2lua` is failing on themselves, feel free to contact the Emerging Threats team whenever `snort2lua` generates a `snort.rej` file for Emerging Threats rules. The best way to receive support for rule transformation failures would be to either:

1) Register and post the contents of the `snort.rej` file on https://community.emergingthreats.net, specifically under the *Feeback & Support* category
2) Open a feedback ticket, with the issue type of *Other* on https://feedback.emergingthreats.net/feedback, and include the contents of the `snort.rej` file

Be aware that Snort 3, with all of our configuration settings is trying to load somewhere in the neighborhood of nearly 60,000 rules, and that on systems with limited CPU, RAM, and/or Disk I/O, that this means the time needed to start and stop snort is going to increase significantly. My recommendation is to begin tuning your ruleset.

- Disable rules that do not apply to your environment
- Set `detection_filters` or `event_limits` for noisy rules that alert excessively
- If it's decided that 60,000 rules is excessive, or that there is too much rule overlap, or perhaps one ruleset fits the needs of the environment better than the other, Try using either the Cisco Talos LightSPD ruleset, or the Emerging Sets ruleset exclusively
- Be as accurate as possible with regards to the `HOME_NET` variable.
- Pay attention to where the IDS sensor is deployed, and the goal of that sensor deployment. Tune the sensor's ruleset to achieve deployment goals

Conclusion

At this point, readers have a fully functional Snort 3 sensor complete with OpenAppID, and json output. Additionally, the Snort 3 sensor is configured to receive system and rule updates once weekly. I was originally going to include a guide for Splunk integration here, but honestly, I have no idea what Cisco is planning to do to Splunk.

Will there still be a free version?

Will developer licenses still exist?

Will Splunk.com accounts eventually be forced into becoming Cisco TAC accounts?

I have no answers to these questions, and as of mine writing this in May of 2024, this year's Splunk .conf conference where they'll probably be laying out a roadmap for this is still a month away. I don't want to commit my time creating instructions that may be outdated in less than a month. However, here are some of the general steps:

- If readers don't already have a Splunk system/search head established, Install Splunk Enterprise on a separate system from the Snort3 Sensor, and establish a listener for the Universal Forwarder that will be installed on the Snort 3 Sensor
- Install the Splunk Universal Forwarder on the Snort 3 IDS, and add the Splunk Enterprise server as a destination for forwarding logs
- Install the Snort 3 JSON Alerts app on both the Splunk Enterprise system, and the Snort 3 IDS.
    - https://splunkbase.splunk.com/app/4633

If this guidance is not enough for you, I am the author of a work called Building Virtual Machine Labs: A Hands-On Guide. Physical copies of my book can be found on Amazon, while virtual copies can be found on https://leanpub.com/avatar2. Readers can pay any price they feel comfortable with paying for the digital release (Including for free – If money is tight, or those who want to try it before they buy it).

Alternatively, Noah Dietrich provided an installation guide for snort 3.1.0.0 on Ubuntu 18.04 and 20.04 many moons ago. He describes how to set up Splunk on the Snort 3 IDS, but I really don't recommend running Splunk and Snort 3 on the same system. I leave that choice up to the reader, however. His guide can be found here (for now):

https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/012/147/original/Snort_3.1.8.0_on_Ubuntu_18_and_20.pdf

Good luck, and Happy Hunting!

Tony Robinson

Sr. Security Researcher

Proofpoint Emerging Threats