Snort 3 on CentOS 8 Stream

Generated: 2021-01-20
Author: Yaser Mansour

Table of Contents

1. Introduction

This guide walks through installing, configuring and testing Snort 3 and PulledPork on CentOS Stream. Some of the configurations may not be applicable to production sensors. The steps in this guide should be tested first.

CentOS Stream Image

Base Image	:	`CentOS-Stream-8-x86_64-20201211-dvd1.iso`
Release	:	`CentOS Stream release 8`
Kernel	:	`4.18.0-259.el8.x86_64`
Software	:	`Minimal Install`

Snort version and build

Build	:	`Snort 3.1.0.0 GA`
Source	:	`git`

LibDAQ version

Build	:	`3.0.0 GA`
Source	:	`git`

Paths used for installing and configuring Snort and PulledPork

Snort install prefix	`/usr/local/snort`
Rules directory	`/usr/local/snort/rules`
AppID directory	`/usr/local/snort/appid`
IP Reputation lists directory	`/usr/local/snort/intel`
Logging directory	`/var/log/snort`
Snort Extra Plugins directory	`/usr/local/snort/extra`
PulledPork install prefix	`/usr/local/pulledpork`

Conventions used in this guide

Info: Good to know information or suggestion.

Note: Information that requires attention.

```
Command line input
```

```
Command line output
```

```
Configuration changes
```

2. Preparation

Starting from CentOS 8, several development libraries required for successfully building LibDAQ and Snort are not in the default repositories – `AppStream`, `Base`, or `Extras`. Instead, these libraries exist in the `PowerTools` repository, which is disabled by default. Hence, the `PowerTools` repository is enabled first.

```
# dnf config-manager --add-repo /etc/yum.repos.d/ CentOS-Stream-PowerTools.repo
# dnf config-manager --set-enabled powertools
```

Additional development libraries exist in the EPEL repository. Enabling the EPEL repository reduces build time and streamlines the installation and updates of these libraries. Otherwise, packages from the EPEL repository can be built from their source code.

```
# dnf install epel-release
```

Now that all of the repositories enabled, it is time to ensure that the operating system and existing packages are up to date. This may require a reboot, especially if the updates included kernel upgrades.

```
# dnf upgrade
# reboot now
```

Since some of the packages maybe built from source, a directory is created to house the source codes.

```
# mkdir sources && cd sources
```

Next, some helper packages are installed, which are not required by Snort and can be removed later.

```
# dnf install vim git
```

Red Hat based operating systems do not include the `/usr/local/lib` and `/usr/local/lib64` in the linker caching paths, resulting in build errors since the referenced libraries cannot be found. This is corrected by creating a configuration file under `/etc/ld.so.conf.d` containing the required paths and updating the cache.

```
# vi /etc/ld.so.conf.d/local.conf
```

Add the below two lines to the newly created configuration file.

```
/usr/local/lib
/usr/local/lib64
```

After saving the configuration file, run `ldconfig`.

```
# ldconfig
```

Info: The error message typically generated by the missing linker caching paths is presented as:

```
cannot open shared object file: no such file or directory
```

The final step in the preparation is to install the build tools from the repository. These include: **flex** (`flex`), **bison** (`bison`), **gcc** (`gcc`), **c++** (`gcc-c++`), **make** (`make`), and **cmake** (`cmake`). Additionally, **autoconf** (`autoconf`), **automake** (`automake`) and **libtool** (`libtool`) packages are installed to build LibDAQ.

```
# dnf install flex bison gcc gcc-c++ make cmake automake autoconf libtool
```

3. Installing Snort 3 Dependencies

The following table summarizes the required and optional packages for building Snort and LibDAQ.

Dependency	Status	Source	Dependency	Status	Source
`dnet`	Required	Repository (PowerTools)	`LibDAQ`	Required	Source Code
`pcap`	Required	Repository (PowerTools)	`lzma`	Optional	Repository (BaseOS)
`pcre`	Required	Repository (BaseOS)	`hyperscan`	Optional	Repository (EPEL)
`OpenSSL`	Required	Repository (BaseOS)	`flatbuffers`	Optional	Source Code
`zlib`	Required	Repository (BaseOS)	`safec`	Optional	Repository (EPEL)
`pkgconfig`	Required	Repository (BaseOS)	`uuid`	Optional	Repository (BaseOS)
`LuaJIT`	Required	Repository (EPEL)	`tcmalloc`	Optional	Repository (EPEL)
`hwloc`	Required	Repository (PowerTools)	`libmnl`	Required	Repository (BaseOS)
`unwind`	Optional	Repository (EPEL)			

3.1 Required Dependencies

The following packages are installed from CentOS repositories: **pcap** (`libpcap-devel`), **pcre** (`pcre-devel`), **dnet** (`libdnet-devel`), **hwloc** (`hwloc-devel`), **OpenSSL** (`openssl-devel`), **pkgconfig** (`pkgconf`), **zlib** (`zlib-devel`), **LuaJIT** (`luajit-devel`), **libmnl** (`libmnl-devel`), and **libunwind** (`libunwind-devel`).

```
# dnf install libpcap-devel pcre-devel libdnet-devel hwloc-devel openssl-devel zlib-devel
luajit-devel pkgconf libmnl-devel libunwind-devel
```

Building LibDAQ with NFQ support requires additional packages to be installed before configuration: **libnfnetlink** (`libnfnetlink-devel`), **libnetfilter_queue** (`libnetfilter_queue-devel`).

```
# dnf install libnfnetlink-devel libnetfilter_queue-devel
```

LibDAQ

Snort 3 requires LibDAQ (>=3.0.0). Clone it and generate the configuration script.

```
# git clone https://github.com/snort3/libdaq.git
# cd libdaq/
# ./bootstrap
```

Info: Review LibDAQ configuration options to disable modules via `--disable-<name>-module` option

```
Example: ./configure --disable-netmap-module --disable-divert-module
```

Proceed with configuring LibDAQ, resulting in a similar output (omitted) as demonstrated below. The warning "`No libcmocka-1.0.0 or newer library found, cmocka tests will not be built`" can be ignored as we are not building the cmocka tests. Otherwise, install the `libcmocka-devel` package.

```
# ./configure
```

```
...
Build AFPacket DAQ module.. : yes
Build BPF DAQ module....... : yes
Build Divert DAQ module.... : no
Build Dump DAQ module...... : yes
Build FST DAQ module....... : yes
Build NFQ DAQ module....... : yes
Build PCAP DAQ module...... : yes
Build netmap DAQ module.... : no
Build Trace DAQ module..... : yes
```

Proceed to installing LibDAQ.

```
# make
# make install
# ldconfig
# cd ../
```

3.2 Optional Dependencies

LZMA and UUID

`lzma` is used for decompression of SWF and PDF files, while `uuid` is a library for generating/parsing Universally Unique IDs for tagging/identifying objects across a network.

```
# dnf install xz-devel libuuid-devel
```

Hyperscan

While `hyperscan` is an optional requirement, it is highly recommended to install it. The `hyperscan` packages are available via the EPEL repository. See the Appendix for installing hyperscan from sources.

```
# dnf install hyperscan hyperscan-devel
```

Flatbuffers

`Flatbuffers` is a cross-platform serialization library for memory-constrained apps. It allows direct access of serialized data without unpacking/parsing it first.

```
# curl -Lo flatbuffers-1.12.tar.gz https://github.com/google/flatbuffers/archive/v1.12.0.tar.gz
# tar xf flatbuffers-1.12.tar.gz
# mkdir fb-build && cd fb-build
# cmake ../flatbuffers-1.12.0
# make -j$(nproc)
# make -j$(nproc) install
# ldconfig
# cd ../
```

Safec

`Safec` is used for runtime bounds checks on certain legacy C-library calls. `Safec` package is available in the EPEL repository.

> **Note:** An additional step is required when installing the package version of `Safec` because the Safec EPEL package deploys a `pkg-config` file named `safec-version.pc` while Snort expects the `pkg-config` file to be named `libsafec.pc`. This additional step is not required if Safec is built from source.

```
# dnf install libsafec libsafec-devel
# ln -s /usr/lib64/pkgconfig/safec-3.3.pc /usr/lib64/pkgconfig/libsafec.pc
```

Tcmalloc

`tcmalloc` is a library created by Google (PerfTools) for improving memory handling in threaded programs. The use of the library may lead to performance improvements and memory usage reduction**.** The **gperftools** (`gperftools-devel`) package version 2.7 is available from the EPEL repository.

```
# dnf install gperftools-devel
```

4. Installing Snort 3

Now that all of the dependencies are installed, clone Snort 3 repository from GitHub.

```
# git clone https://github.com/snort3/snort3.git
# cd snort3
```

Before configuring Snort, export the `PKG_CONFIG_PATH` to include the `LibDAQ` pkgconfig path, as well as other packages' pkgconfig paths, otherwise, the build process may fail.

```
# export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH
# export PKG_CONFIG_PATH=/usr/local/lib64/pkgconfig:$PKG_CONFIG_PATH
```

Note: If `LibDAQ` or other packages were installed to a custom, non-system path, then that path should be exported to `PKG_CONFIG_PATH`, for example:

```
# export PKG_CONFIG_PATH=/opt/libdaq/lib/pkgconfig:$PKG_CONFIG_PATH
```

Proceed to building Snort 3 while enabling `tcmalloc` support. The compiler flags exported prior to building Snort are used to help improve compilation time, performance of the generated code and final Snort's binary image size. These are discussed further in section 10.5 Optimizing Performance.

```
# export CFLAGS="-O3"
# export CXXFLAGS="-O3 -fno-rtti"
# ./configure_cmake.sh --prefix=/usr/local/snort --enable-tcmalloc
```

The above command should result in an output (omitted) similar to below.

```
-------------------------------------------------------
snort version 3.1.0.0
...
Feature options:
    DAQ Modules:    Static (afpacket;bpf;dump;fst;nfq;pcap;trace)
    Flatbuffers:    ON
    Hyperscan:      ON
    ICONV:          ON
    Libunwind:      ON
    LZMA:           ON
    RPC DB:         Built-in
    SafeC:          ON
    TCMalloc:       ON
    UUID:           ON
```

Proceed to installing Snort 3.

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

Once the installation is complete, verify that Snort 3 reports the expected version and library names

```
# /usr/local/snort/bin/snort -V
```

```
   ,,_     -*> Snort++ <*-
  o"  )~   Version 3.1.0.0
   ''''    By Martin Roesch & The Snort Team
           http://snort.org/contact#team
           Copyright (C) 2014-2020 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using DAQ version 3.0.0
           Using LuaJIT version 2.1.0-beta3
           Using OpenSSL 1.1.1g FIPS  21 Apr 2020
           Using libpcap version 1.9.1 (with TPACKET_V3)
           Using PCRE version 8.42 2018-03-20
           Using ZLIB version 1.2.11
           Using FlatBuffers 1.12.0
           Using Hyperscan version 5.3.0 2020-08-10
           Using LZMA version 5.2.4
```

5. Installing Snort 3 Extra for Additional Capabilities

Snort 3 Extra is a set of C++ or Lua plugins to extend the functionality of Snort 3 in terms network traffic decoding, inspection, actions, and logging. One particular plugin is emphasized and configured in this guide is the `data_log` inspector plugin. The emphasis of this inspector is detailed in a later section.

To install Snort extras, clone its repository from GitHub.

```
# git clone https://github.com/snort3/snort3_extra.git
```

Before building the extra plugins, the environment variable `PKG_CONFIG_PATH` must be set. The path can be verified by listing Snort installation directory.

```
# cd snort3_extra
# export PKG_CONFIG_PATH=/usr/local/snort/lib64/pkgconfig:$PKG_CONFIG_PATH
# ./configure_cmake.sh --prefix=/usr/local/snort/extra
# cd build/
# make -j$(nproc)
# make -j$(nproc) install
# cd ../../
```

6. Configuring Snort 3

Snort 3 includes two main configuration files, `snort_defaults.lua` and `snort.lua`. The file `snort_defaults.lua` contains default values for rules paths, networks, ports, wizards, and inspectors, etc.

> **Info:** The `snort.lua` file contains Snort's main configuration, allowing the implementation and configuration of Snort inspectors (preprocessors), rules files inclusion, event filters, output, etc.

> **Info:** The `snort_defaults.lua` file contains default values such as paths to rules, AppID, intelligence lists, and network variables.

> **Info:** An additional file `file_magic.lua` exists in the etc/snort/ directory. This file contains pre-defined file identities based on the hexadecimal representation of the files magic headers. These help Snort identify the file types traversing the network when applicable. This file is also used by Snort main configuration file snort.lua and does not require any modifications.

The configuration changes and the respective Snort 3 Lua files are as follows.

Task	Snort Configuration File
Configure rules, reputation, and AppID paths	`snort_defaults.lua`
Configure HOME_NET and EXTERNAL_NET	`snort.lua`
Configure ips module	`snort.lua`
Enable and configure reputation inspector	`snort.lua`
Configure file_id and file_log inspectors	`snort.lua`
Configure data_log inspector	`snort.lua`
Configure logging	`snort.lua`

6.1 Global Paths for Rules, AppID, and IP Reputation

Snort rules, appid, and reputation lists will be stored in their respective directory. The `rules/` directory will contain Snort rules files, the `appid/` directory will contain the AppID detectors, and the `intel/` directory will contain IP blacklists and whitelists.

```
# mkdir -p /usr/local/snort/{builtin_rules,rules,appid,intel}
```

Snort Rules

Snort rules consist of text-based rules, and Shared Object (SO) rules and their associated text-based stubs. At the time of writing this guide, the Shared Object rules are not available yet. The rules tarball also contains Snort configuration files. The configuration files from the rules tarball will be copied to the `etc/snort/` directory, and will be used in favor of the configuration files in from Snort 3 source tarball.

Proceed by creating a directory to contain the files extracted from the rules tarball downloaded from Snort.org. Replacing the oinkcode placeholder in the below command with the official and dedicated oinkcode.

```
# mkdir rules && cd rules
# curl -Lo snortrules-snapshot-3000.tar.gz https://www.snort.org/rules/snortrules-snapshot-
3000.tar.gz?oinkcode=<YOUR OINKCODE HERE>
```

Extract the rules tarball and copy the rules to the `rules/` directory created earlier.

```
# tar xf snortrules-snapshot-3000.tar.gz
```

Extracting the rules will result in three different directories

```
├── builtins
├── etc
└── rules
```

Copy the files to their respective directories of the Snort installation paths.

```
# cp rules/*.rules /usr/local/snort/rules/
# cp builtins/builtins.rules /usr/local/snort/builtin_rules/
# cp etc/snort_defaults.lua etc/snort.lua /usr/local/snort/etc/snort/
# cd ../
```

OpenAppID (Optional)

Download and extract the OpenAppID package, and move the extracted `odp/` directory to the `appid/` directory.

```
# curl -Lo snort-openappid-15607.tar.gz https://snort.org/downloads/openappid/15607
# tar xf snort-openappid-15607.tar.gz
# mv odp/ /usr/local/snort/appid/
```

IP Reputation (Optional)

Download the IP Blacklist generated by Talos and move it to the `intel/` directory created earlier. An empty file for the IP address whitelist is also created to be configured along with the IP address blacklist.

```
# curl -Lo ip-blocklist https://www.talosintelligence.com/documents/ip-blacklist
# mv ip-blocklist /usr/local/snort/intel/
# touch /usr/local/snort/intel/ip-allowlist
```

Snort configuration file `snort_defaults.lua` needs to be modified to point to the correction locations of rules, AppID and reputation blacklists. The paths shown below follow the conventions from the beginning of this guide.

Change from:

```
-- Path to your rules files (this can be a relative path)
RULE_PATH = '../rules'
BUILTIN_RULE_PATH = '../builtin_rules'
PLUGIN_RULE_PATH = '../so_rules'

-- If you are using reputation preprocessor set these
WHITE_LIST_PATH = '../lists'
BLACK_LIST_PATH = '../lists'
```

Change to:

```
-- Path to your rules files (this can be a relative path)
RULE_PATH = '../../rules'
BUILTIN_RULE_PATH = '../../builtin_rules'
PLUGIN_RULE_PATH = '../so_rules'

-- If you are using reputation preprocessor set these
ALLOW_LIST_PATH = '../../intel'
BLOCK_LIST_PATH = '../../intel'

-- Path to AppID ODP - Optional
APPID_PATH = '/usr/local/snort/appid'
```

6.2 Configuring HOME_NET and EXTERNAL_NET

The concept of home and external networks in Snort 3 is the same as in Snort 2.X. The changes made below are just an example to demonstrate the syntax.

Change from:

```
-- setup the network addresses you are protecting
HOME_NET = 'any'
```

Change to:

```
-- setup the network addresses you are protecting
HOME_NET = [[ 10.0.0.0/8 192.168.0.0/16 172.16.0.0/12 ]]
```

6.3 Configuring ips Module

The inclusion of Snort rules files (.rules) occurs within the ips module. Using the `snort.lua` copied from the Snort rules tarball, the inclusion of the rules is already configured. As a result, the changes to the ips module are minimal and involves enabling decoder and inspector alerts with the option `--enable_built_rules`, and explicitly defining the ips policy to tap mode. The ips policy governs Snort's operational mode (tap, inline, and inline-test).

Change from:

```
ips =
{
    -- use this to enable decoder and inspector alerts
    --enable_builtin_rules = true,

    -- use include for rules files; be sure to set your path
    -- note that rules files can include other rules files
    --include = 'snort3-community.rules',

    variables = default_variables,

    -- The following include syntax is only valid for BUILD_243 (13-FEB-2018) and later
    -- RULE_PATH is typically set in snort_defaults.lua
    rules = [[
        include $RULE_PATH/snort3-app-detect.rules
        include $RULE_PATH/snort3-browser-chrome.rules

        .....

        include $RULE_PATH/snort3-x11.rules
    ]]
}
```

Change to:

```
ips =
{
    mode = tap,

    -- use this to enable decoder and inspector alerts
    --enable_builtin_rules = true,

    -- use include for rules files; be sure to set your path
    -- note that rules files can include other rules files
    --include = 'snort3-community.rules',

    variables = default_variables,

    rules = [[
        include $RULE_PATH/snort.rules
    ]]
}
```

The above configuration includes one rules file `snort.rules`, which is generated later using PulledPork in Section 8 Managing Snort 3 Rules with PulledPork. All enabled rules are included in one file for simpler configuration and management.

6.4 Configuring reputation Inspector (Optional)

The reputation inspector is disabled (commented) by default. Uncomment its section and change the values of the `--blacklist` and `--whitelist` variables to point to the paths IP address lists.

Change from:

```
--[[
reputation =
{
    -- configure one or both of these, then uncomment reputation
    --blacklist = 'blacklist file name with ip lists'
    --whitelist = 'whitelist file name with ip lists'
}
--]]
```

Change to:

```
reputation =
{
    -- configure one or both of these, then uncomment reputation
    blacklist = BLOCK_LIST_PATH .. '/ip-blocklist',
    whitelist = ALLOW_LIST_PATH .. '/ip-allowlist'
}
```

Info: Enabling the Reputation inspector while in IDS mode will generate blacklist hit alert when a match occurs, and traffic may not be inspected further.

6.5 Configuring appid Inspector (Optional)

The `appid` inspector is enabled by default, however, it requires additional configuration to be fully effective; the path to the AppID package and detector are commented. Uncomment the `app_detector_dir` and change its value the global AppID path defined in the earlier in the `snort_default.lua` file. AppID logging is configured in Section 7 Configuring Snort Logging to take advantage of Snort's new AppID logging capabilities.

Change from:

```
appid =
{
    -- appid requires this to use appids in rules
    --app_detector_dir = 'directory to load appid detectors from'
}
```

Change to:

```
appid =
{
    -- appid requires this to use appids in rules
    app_detector_dir = APPID_PATH,
}
```

6.6 Configuring file Inspectors (Optional)

The `file_id` inspector (file_inspect in Snort 2.x) allows Snort to identify files and file types traversing a network stream via the file magic headers. It supports HTTP, SMTP, IMAP, POP3, FTP, and SMB protocols.

Note: Taking advantage of the `file_id` inspector involves:

- Including the file magic rules. This step is completed in the default form of the inspector.
- Configuring the inspector and defining the file policy.
- Enabling the inspector logging to generate file events.

The file inspector is configured to enable file type identification (`enable_type = true`) and file magic signature calculation (`enable_signature = true`). Finally, a file policy is configured to log all file types identified in the network traffic regardless of their type.

Change from:

```
file_id = { file_rules = file_magic }
```

Change to:

```
file_id =
{
    enable_type = true,
    enable_signature = true,
    file_rules = file_magic,
    file_policy =
    {
        { use = { verdict = 'log', enable_file_type = true, enable_file_signature = true } }
    }
}
```

7. Configuring Snort 3 Logging

Snort 3 provides several logging mechanisms natively or via Snort Extra. This section walks through configuring some of Snort 3 logging modules.

Note: Only enable the logger modules that are required for your use-case. For example, if your use-case does not require alerting to Syslog, then do not configure the `alert_syslog` logger.

7.1 Configuring Logger Module (Optional)

Snort 3 supports various logger modules natively or via the extra plugins. For this guide, the `alert_fast` logger is enabled by uncommenting its section and configuring it to log to a file. By default Snort uses `/var/log/snort` for saving log files, which also can be specified in the command line via the `-l` option.

Change from:

```
--alert_fast = { }
```

Change to:

```
alert_fast =
{
    file = true
}
```

7.2 Configuring file_log Inspector (Optional)

The `file_log` inspector accompanies the `file` inspector, i.e.: if the `file` inspector is configured, the `file_log` inspector must be configured to generate the associated logs. This inspector has two Boolean options that allow logging of packet and system time of logged file events.

```
file_log =
{
    log_pkt_time = true,
    log_sys_time = false
}
```

Info: The file policy can include multiple configurations. The below example file policy will log file identification only when a file of type PDF id = 22 or when a file with the specified SHA256 hash is observed traversing the network or capture.

```
file_policy =
    {
      { when = { file_type_id = 22 }, use = { verdict = 'log', enable_file_signature = true } },
      { when = { sha256 = "E65ECCC.....DDF3233355007" }, use = { verdict = 'log' } }
    }
```

7.3 Configuring data_log Inspector (Optional)

The `data_log` plugin is available via the extra plugins. The `data_log` is a passive inspector that does not alter data, instead, it allows for logging additional network data. The inspector can be used to log HTTP request or response headers. In Snort 2, this was possible using the `log_uri` and `log_hostname` options of the `http_inspect` preprocessor. The captured data is stored into the `data.log` within Snort's configured logging directory.

In order to enable the `data_log` inspector, it must be defined in `snort.lua`. The below example will log HTTP request headers into the `data_log` file and limit the size of the log file to 100MB before a new log file is generated.

```
data_log =
{
    key = 'http_request_header_event',
    limit = 100
}
```

7.4 Configuring alert_syslog Logger (Optional)

Snort 3 can log generated alerts directly to Syslog. The below configuration example can be used to log alerts so Syslog. Edit the `snort.lua` file to add the configuration. This guide does not extend on how to use or configure the Syslog program. Note that rsyslog is not installed by default on a minimal CentOS 8 installation.

```
alert_syslog =
{
    facility = local7,
    level = alert,
    options = pid
}
```

7.5 Configuring alert_json Logger (Optional)

Logging in JSON format is favorable in several log aggregation solutions like Elastiscsearch. Snort 3 can output generated alerts into JSON format natively. The below configuration example can be used to log alerts so Syslog. Edit the `snort.lua` file to add the configuration.

> **Note:** Snort 3 allows logging various fields into JSON format. Review all of the fields and only use those that are most useful for your logging requirements. Also, note the order of fields you select as the generated log will follow the order of the fields specified in the configuration. The selected fields in the below configuration are only an example.

```
alert_json =
{
    file = true,
    limit = 100,
    fields = 'timestamp iface src_addr src_port dst_addr dst_port proto action msg priority class sid'
}
```

7.6 Configuring appid_listener Logger (Optional)

The new `appid_listener` allows generating flow logs of detected applications into a JSON format log file. The logger is part of Snort Extra plugins suite installed in Section 5 Installing Snort 3 Extra for Additional Capabilities.

The logger is configured by enabling JSON logging and specifying the log file name and path as depicted below.

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

8. Managing Snort 3 Rules with PulledPork

PulledPork allows updating and managing Snort rules and Talos open-source IP address block list in a consistent and regular manner. In order to run PulledPork, the following dependencies are installed first.

```
# dnf install perl-LWP-UserAgent-Determined perl-Net-SSLeay perl-LWP-Protocol-https perl-Sys-
Syslog perl-Archive-Tar
```

The directory structure hosting PulledPork and associated configuration files are created as specified in the Introduction section. Afterwards, PulledPork is cloned from GitHub and all of the needed files are moved to directory structure just created.

```
# mkdir -p /usr/local/pulledpork/etc
# git clone https://github.com/shirkdog/pulledpork.git
# cp pulledpork/pulledpork.pl /usr/local/pulledpork/
# cp pulledpork/etc/* /usr/local/pulledpork/etc/
```

The first item to configure is setting up the `oinkcode` acquired after registering to Snort.org in `pulledpork.conf`. Replace the marker `<oinkcode>` with the `oinkcode` tied to your Snort.org account.

```
# vi /usr/local/pulledpork/etc/pulledpork.conf
```

Change from:

```
rule_url=https://www.snort.org/reg-rules/|snortrules-snapshot.tar.gz|<oinkcode>
```

Change to:

```
rule_url=https://www.snort.org/reg-rules/|snortrules-snapshot.tar.gz|123456789
```

Note: If you are a registered or subscribed to Snort rules, then comment out the community rules URL since community rules are included in the registered ruleset, by adding the pound sign # at the beginning of the line such as below.

```
#rule_url=https://snort.org/downloads/community/|community-rules.tar.gz|Community
```

Since Snort and PulledPork are installed in custom directory layouts, paths configuration within `pulledpork.conf` must be updated to reflect the custom directory. Discussing the role of each path is out of the scope of this guide; however, the `pulledpork.conf` file is commented to explain each path. The below changes are made to have PulledPork seamlessly work with the custom directory layout of Snort installation.

Change from:

```
ignore=deleted.rules,experimental.rules,local.rules
```

Change to:

```
ignore= snort3-deleted.rules,snort3-experimental.rules
```

Change from:

```
rule_path=/usr/local/etc/snort/rules/snort.rules
```

Change to:

```
rule_path=/usr/local/snort/rules/snort.rules
```

Change from:

```
local_rules=/usr/local/etc/snort/rules/local.rules
```

Change to:

```
local_rules=/usr/local/snort/rules/local.rules
```

Change from:
```
sid_msg=/usr/local/etc/snort/sid-msg.map
```
Change to:
```
sid_msg=/usr/local/snort/etc/snort/sid-msg.map
```

Change from:
```
snort_path=/usr/local/bin/snort
```
Change to:
```
snort_path=/usr/local/snort/bin/snort
```

Change from:
```
config_path=/usr/local/etc/snort/snort.conf
```
Change to:
```
config_path=/usr/local/snort/etc/snort/snort.lua
```

Change from:
```
distro=FreeBSD-12
```
Change to:
```
distro=Centos-8
```

Change from:
```
block_list=/usr/local/etc/snort/rules/iplists/default.blocklist
IPRVersion=/usr/local/etc/snort/rules/iplists
```
Change to:
```
block_list=/usr/local/snort/intel/ip-blocklist
IPRVersion=/usr/local/snort/intel/
```

Change from:
```
# snort_version=2.9.0.0
```
Change to:
```
snort_version=3.0.0.0
```

Change from:
```
# pid_path=/var/run/snort_eth0.pid
```
Change to:
```
pid_path=/var/log/snort/snort.pid
```

The configuration files below (`enablesid.conf`, `dropsid.conf`, `disablesid.conf` and `modifysid.conf`) govern how PulledPork will process the rules. For example, `enablesid.conf` can be used to enable all rules; `disablesid.conf` can be used to disable certain rules, etc. Each file includes documentation on how to add rules SID to process.

Change from:
```
# enablesid=/usr/local/etc/snort/enablesid.conf
# dropsid=/usr/local/etc/snort/dropsid.conf
# disablesid=/usr/local/etc/snort/disablesid.conf
# modifysid=/usr/local/etc/snort/modifysid.conf
```
Change to:
```
enablesid=/usr/local/pulledpork/etc/enablesid.conf
dropsid=/usr/local/pulledpork/etc/dropsid.conf
disablesid=/usr/local/pulledpork/etc/disablesid.conf
modifysid=/usr/local/pulledpork/etc/modifysid.conf
```

After initial configurations are completed, PulledPork is invoked using the below command in order to update pull and update Snort rules.

```
# perl /usr/local/pulledpork/pulledpork.pl -c /usr/local/pulledpork/etc/pulledpork.conf -PE -
v -I security -T -H SIGHUP
```

The above command invokes PulledPork while pointing to its configuration file (using the –c option). While the other options achieve the following:

```
-P     : Process rules even if no new rules were downloaded. (Useful when updating local.rules)
-E     : Write ONLY the enabled rules to the output files. (snort.rules)
-v     : Run in verbose mode.
-H     : Reload Snort after PulledPork update rules. (SIGHUP or SIGUSR2)
-T     : Process text based rules files only. (Snort 3 does not support SO_RULES yet)
-I     : Specify a base ruleset based on rule’s policy. Policies include:
         security
         balanced
         connectivity
         max_detect
```

Depending on PulledPork configuration and the selected rules policy, PulledPork generates output similar to the blow lines.

```
...
Rule Stats...
        New:-------52696
        Deleted:---0
        Enabled Rules:----52696
        Dropped Rules:----0
        Disabled Rules:---0
        Total Rules:------52696
IP Blocklist Stats...
        Total IPs:-----885

Done
```

Note that since PulledPork option `-k` was not used, all of the enabled rules are written to a single file (`snort.rules`) as opposed to keeping the rules in separate files using same file names as found when processing them. In this case, Snort configuration must be updated to now include the `snort.rules` file and not the individual rules files as demonstrated below.

```
ips =
{
    mode = tap,

    rules = [[
        include $RULE_PATH/snort.rules
    ]]
}
```

Now we configure PulledPork to pull Snort rules and update them periodically. To achieve this, we will use `systemd timers` as opposed to `cronjobs`. First, we create a PulledPork `systemd` service, which will not be enabled (it is automatically invoked by a systemd timer as we will observe later).

```
# vi /etc/systemd/system/pulledpork.service
```

```
[Unit]
Description=PulledPork service for updating Snort 3 rules
Wants=pulledpork.timer

[Service]
Type=oneshot
ExecStart=perl /usr/local/pulledpork/pulledpork.pl -c
/usr/local/pulledpork/etc/pulledpork.conf -PE -v -I security -T -H SIGHUP

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

Afterwards, we create the systemd timer service and enable it.

```
# vi /etc/systemd/system/pulledpork.timer
```

```
[Unit]
Description=PulledPork service timer for updating Snort 3 rules
Requires=pulledpork.service

[Timer]
Unit=pulledpork.service
OnCalendar=*-*-* 00:10:00
AccuracySec=1us

[Install]
WantedBy=timers.target
```

The above systemd timer will invoke the PulledPork service everyday of every month of every year 10 minutes after midnight, with a time span accuracy of one microsecond from time the timer is configure to run.

```
# systemctl daemon-reload
# systemctl enable pulledpork.timer
```

Note: The above timer schedule is just an example. Use your own schedule to help distribute Snort rules update requests across varying time spans and not have all requests hit Snort rules servers at once.

Expanding on systemd timers is beyond the scope of this guide.

9. Running and Testing Snort 3

9.1 Running against PCAP Files

Snort can process a single packet capture PCAP file via the `-r` option, while specifying the configuration file via the `-c` option, the log directory via the `-l` option, and the extra plugins directory (for the `data_log` inspector) via `--plugin-path` option.

```
# /usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort.lua -r test.pcap -l
/var/log/snort --plugin-path /usr/local/snort/extra -k none
```

Snort can also process multiple PCAP files stored in a specific directory in bulk. This involves specifying the directory containing the PCAP files via the `--pcap-dir` option and filtering only the PCAP files in that directory via the `--pcap-filter` option.

```
# /usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort.lua --pcap-dir pcaps/ --
pcap-filter '*.pcap' -l /var/log/snort --plugin-path /usr/local/snort/extra -k none
```

9.2 Running against an Interface

Snort can be run against a listening interface via the `-i` option while specifying the capture network interface.

```
# /usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort.lua -i eth0 -l
/var/log/snort --plugin-path /usr/local/snort/extra -k none
```

Info: Snort can run and process network from more than one network interface via the `-i` option, while taking advantage of Snort's multiple packets processing threads via `--max-packet-threads` or `-z` options:

Multiple Interfaces:

```
snort -c snort.lua -i eth0 eth1 -z 2
```

Inline Pairs:

```
snort -c snort.lua -i eth0:eth1 -z 2
```

9.3 Running Snort 3 Demo

Snort 3 demo contains usage examples and tests against Snort 3 in an automated fashion using bats – Bash Automated Testing System. Bats can be installed using the below steps.

```
# git clone https://github.com/sstephenson/bats.git
# cd bats/
# ./install.sh /usr/local
```

Now, clone Snort 3 demo project and run the tests.

```
# git clone https://github.com/snort3/snort3_demo.git
# cd snort3_demo/
# ./run_test.sh /usr/local/snort
```

10. Configuring Snort Network Interfaces, User, Service and Logging

10.1 Configuring Network Capturing Interfaces

The network capture interface that Snort will utilize to inspect traffic is setup with minimal configurations as shown below. Replace the `ifname` with the actual interface name

```
TYPE=Ethernet
BOOTPROTO=none
IPV4_FAILURE_FATAL=no
IPV6INIT=no
IPV6_FAILURE_FATAL=no
NAME=ifname
DEVICE=ifname
ONBOOT=yes
```

If an existing interface is modified, ensure that `NetworkManager` can read the changes and have them applied.

```
# nmcli con load /etc/sysconfig/network-scripts/ifcfg-ifname
# nmcli con up ifname
```

Network Capturing Interface and NIC Offloading

NIC offloads are options that allow the stack to transmit packets that are larger than the normal MTU for resources optimization. In doing so, network traffic is potentially altered – (re)segmentation, IP fragmentation, reassembly, etc. – by the receiving host's network interface instead of the CPU. This could lead to packet errors potentially allowing IDS evasion scenarios. In order to avoid these issues and allow Snort to monitor the same packets destined to the receiving host, it is recommended to disable NIC offloading options.

> **Info:** Network scripts are deprecated in CentOS 8 and are replaced with `NetworkManager` through the `nmcli` tool. The deprecated network scripts will not be installed/used in this guide.

In CentOS 8 with `NetworkManager` present, this can be achieved with the following command, replacing the `ifname` with the capturing interface name.

```
# nmcli con mod ifname ethtool.feature-lro off ethtool.feature-gro off ethtool.feature-tso off
ethtool.feature-gso off ethtool.feature-sg off ethtool.feature-rx off ethtool.feature-tx off
ethtool.feature-rxvlan off ethtool.feature-txvlan off
```

This permanently modifies the interface's configuration file `ifcfg-ifname` with the `ETHTOOL_OPTS` parameter.

```
ETHTOOL_OPTS="-K ifname gro off gso off lro off rx off rxvlan off sg off tso off tx off txvlan off"
```

Depending on the hardware, interface type and driver, it is possible to increase the size of the receive ring buffer, `rx`, to the maximum value the interface is capable of, increasing the number of stored incoming packets, thus, potentially improving capture performance. Determining the ring buffer size can be done using `ethtool` with the `-g` option as shown in the below example, replacing the `ifname` with the capturing interface name.

```
# ethtool -g ifname
```

```
Ring parameters for ifname:
Pre-set maximums:
RX:            4096
RX Mini:       2048
RX Jumbo:      4096
TX:            4096
Current hardware settings:
RX:            1024
RX Mini:       128
RX Jumbo:      256
TX:            512
```

From the output, the interface is set to 1024 while the maximum is 4096. The `NetworkManager` does not support adapting ring buffers. Instead, using the `ETHTOOL_CMD` parameter combined with dispatcher script ensures that the interface ring buffers are adjusted permanently.

First, the interface is configured with the `ETHTOOL_CMD` parameter.

```
# vi /etc/sysconfig/network-scripts/ifcfg-ifname
```

```
ETHTOOL_OPTS="-K ifname gro off gso off lro off rx off rxvlan off sg off tso off tx off txvlan off"
ETHTOOL_CMD="-G ifname rx 4096"
```

Second, an executable network dispatcher script is created, which will pass the configured `ETHTOOL_CMD` string from the interface's configuration file to the `ethtool` program.

```
# vi /etc/NetworkManager/dispatcher.d/99-ethtool.sh
```

```
#!/bin/bash
# BEGIN 99-ethtool.sh
if [[ $2 == up ]]; then
    SCRIPT="$(basename "$0")"
    if [[ -e $CONNECTION_FILENAME ]]; then
        source $CONNECTION_FILENAME
        if [[ -n $ETHTOOL_CMD ]]; then
            ETHTOOL_CMD="/usr/sbin/ethtool $ETHTOOL_CMD"
            if $ETHTOOL_CMD; then
                logger "$SCRIPT: success: $ETHTOOL_CMD"
            else
                logger "$SCRIPT: failed: $ETHTOOL_CMD"
            fi
        else
            logger "$SCRIPT: ETHTOOL_CMD not in $CONNECTION_FILENAME, skipping"
        fi
    else
        logger "$SCRIPT: $CONNECTION_FILENAME does not exist?"
    fi
fi
```

Finally, the script must be made executable.

```
# chmod +x /etc/NetworkManager/dispatcher.d/99-ethtool.sh
```

Network Capturing Interface and Promiscuous Mode

Another task involves setting up the interface in promiscuous mode permanently using a custom `oneshot` `systemd` service. The service will also disable `ARP` and `multicast`. Once created, reload `systemd` and enable it.

```
# vi /etc/systemd/system/promisc.service
```

```
[Unit]
Description=Snort 3 interface promiscuous mode during boot service
After=network.target

[Service]
Type=oneshot
ExecStart=/usr/sbin/ip link set dev ifname arp off
ExecStart=/usr/sbin/ip link set dev ifname multicast off
ExecStart=/usr/sbin/ip link set dev ifname promisc on
TimeoutStartSec=0
RemainAfterExit=yes

[Install]
WantedBy=default.target
```

```
# systemctl daemon-reload
# systemctl enable promisc.service
```

Finally, reboot the host and verify that all of the changes were successfully applied. The below outputs demonstrate the expected behavior of the above tasks, replacing the `ifname` with the capturing interface name.

```
# systemctl status promisc.service
● promisc.service - Snort 3 interface promiscuous mode during boot service
   Loaded: loaded (/etc/systemd/system/promisc.service; enabled; vendor preset: disabled)
   Active: active (exited) since Wed 2020-03-04 08:29:18 UTC; 6 days ago
  Process: 1284 ExecStart=/usr/sbin/ip link set dev ifname promisc on (code=exited, status=0/SUCCESS)
  Process: 1275 ExecStart=/usr/sbin/ip link set dev ifname arp off (code=exited, status=0/SUCCESS)
```

```
# ip link show ifname
ifname: <BROADCAST,NOARP,PROMISC,UP,LOWER_UP>
```

```
# ethtool -g ifname
Ring parameters for ifname:
Pre-set maximums:
RX:             4096
...
Current hardware settings:
RX:             4096
...
```

10.2 Creating Snort User, Logging Directory and Systemd Startup Service

Preparing Snort for production also involves running Snort with a regular system user and not as root. The following steps will create a group and a user under which the Snort process will run.

```
# groupadd snort
# useradd snort -r -M -g snort -s /sbin/nologin -c SNORT_SERVICE_ACCOUNT
```

By default, Snort writes the generated logs into /var/log/snort directory. The following steps involved creating the directory and then assigning its ownership to the Snort user and group created in the previous step along with appropriate permissions.

```
# mkdir /var/log/snort
# chmod -R 5700 /var/log/snort
# chown -R snort:snort /var/log/snort
```

Note: If a custom logging directory is created outside of `/var/log`, then SELINUX may block Snort from writing logs to the custom directory. The label for the directory can be viewed using the `ls -Z` command as demonstrated below.

```
# ls -Z /var/log | grep snort
unconfined_u:object_r:var_log_t:s0 snort
```

In this case, the SELINUX label and context must be configured for the custom logging directory. The example below replicates the SELINUX label and context of the directory `/var/log` to the custom Snort logging directory without having to disable SELINUX.

```
# chcon --reference /var/log /opt/log/snort
```

In order to run Snort as a startup service, a `systemd` unit file is created. The unit file specifies the environment variables required for running Snort via the `Environment` option (one per line), the user and group that the service and ultimately Snort will be running as, and the capabilities that will be granted to the service and user.

Info: Programs running with a regular user (non-root) must have capabilities granted externally, such as granting the Snort user network-capturing capabilities. This is achieved by using the `CapabilityBoundingSet` and `AmbientCapabilities` in Snort's systemd unit file. The `AmbientCapabilities` grants the configured capabilities automatically while the `CapabilityBoundingSet` limits the capabilities to only those configured.

Create the `systemd` unit file under `/etc/systemd/system` as follows.

```
# vi /etc/systemd/system/snort.service
```

```
[Unit]
Description=Snort 3 Intrusion Detection and Prevention service
After=syslog.target network.target

[Service]
Type=simple
ExecStart=/usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort.lua --plugin-path
/usr/local/snort/extra -i ifname -l /var/log/snort -D -u snort -g snort --create-pidfile -k
none
ExecReload=/bin/kill -SIGHUP $MAINPID
User=snort
Group=snort
Restart=on-failure
RestartSec=5s
CapabilityBoundingSet=CAP_NET_ADMIN CAP_NET_RAW CAP_IPC_LOCK
AmbientCapabilities=CAP_NET_ADMIN CAP_NET_RAW CAP_IPC_LOCK

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

Reload `systemd` to pick up the new service and then enable the service.

```
# systemctl daemon-reload
# systemctl enable snort.service
```

Many of Snort configurations can be supplied either at run-time via the command line or via its configuration file. For example, in Snort's `systemd` unit file, the command line options `-D`, `-u` snort, and `-g` snort were supplied to run Snort process in daemon mode under the user and group `snort`, respectively. The same can be configured in `snort.lua` via the `process` module (optional) as the below example demonstrates.

```
process =
{
    --same as -D
    daemon = true,
    --same as -u
    set_uid = 'snort',
    --same as -g
    set_gid = 'snort',
    utc = true
}
```

The last option, `utc`, configures Snort to log timestamps in UTC instead of the host's configured time zone.

Start Snort service and verify that it is active and running.

```
# systemctl start snort.service
# systemctl status snort.service
```

```
● snort.service - Snort 3 Intrusion Detection and Prevention service
   Loaded: loaded (/etc/systemd/system/snort.service; enabled; vendor preset: disabled)
   Active: active (running) since Sat 2020-03-07 08:51:44 UTC; 9min ago
 Main PID: 2333 (snort)
    Tasks: 2 (limit: 26213)
   Memory: 257.3M
   CGroup: /system.slice/snort.service
           └─2333 /usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort.lua -i ifname -l
/var/log/snort -D -k none
```

Verify that Snort process is running as the Snort user

```
# ps aux | grep snort
```

```
snort     2333  8.3  4.1 361272 253304 ?       Ssl  11:51   0:16 /usr/local/snort/bin/snort -c
/usr/local/snort/etc/snort/snort.lua -i ens224 -l /var/log/snort -D -k none
```

11. Optimizing Performance (Optional)

Configurations in this section attempt to optimize the operating system, kernel, and network IO performance to accommodate IDS/IPS functions. Configuring the CPU governor is the safest configuration to be performed.

> **Note:** These configurations should be treated with care as they might in fact cause performance issues if not thoroughly tested and correctly implemented against the monitored environment. Also, the configurations are dependent on the underlying hardware, i.e.: Intel vs. Mellanox NICs, and whether Snort is running in IDS (passive/tap) or IPS (inline). Finally, optimizations through these configurations may increase utilization on other components, i.e.: CPU and memory, or may be negligible to the point they are not worth it.

11.1 Configuring CPU Governor

`Tuned` manages the CPU governor with a selection of performance profiles required for certain workloads. The default profile may vary depending on whether the host (sensor) is a virtual machine or bare metal. Out of the many profiles available, the `throughput-performance` and `network-throughput` are the prime candidates. In this guide, the `network-throughput` is selected since it is based on the `throughput-performance` profile, and it additionally increases kernel network buffers. The governor can be set with the below command.

```
# tuned-adm profile network-throughput
```

To verify the current active profile, use the below command.

```
# tuned-adm active
Current active profile: network-throughput
```

11.2 Kernel Networking Management

The default Linux kernel configurations may limit the total available throughput. Adapting these defaults may increase the ability of the Linux kernel to transmit (IPS) or receive data (IDS/IPS). These configurations are generally hardware dependent. Hence, they are provided as suggestions or as starting point for tuning network performance. For example, the below configurations may be suitable for **Intel 10G** `ixgbe` network cards.

> **Note:** Store the default values of kernel entries before making modifications to revert to original defaults if necessary.

Add a new configuration under `/etc/sysctl.d/` in order to persist them.

```
# vi /etc/sysctl.d/ixgb.conf
```

```
# Number of unprocessed RX packets before kernel starts dropping them, default = 1000
net.core.netdev_max_backlog = 300000
# turn TCP timestamp support off, default 1, reduces CPU use
net.ipv4.tcp_timestamps = 0
# turn SACK support off, default on
net.ipv4.tcp_sack = 0
# Increase size of RX socket buffer, default = 212992
net.core.rmem_default = 524287
# Increase Max size of RX socket buffer, default = 212992
net.core.rmem_max = 524287
# TCP buffer space pages (not bytes), default = 67932 90576 135864
net.ipv4.tcp_mem = 1048576 4194304 16777216
# TCP read buffer in bytes kernel auto-tuning, default = 4096 87380 16777216
net.ipv4.tcp_rmem = 1048576 4194304 16777216
# Don't cache ssthresh from previous connection, turn off route metrics
net.ipv4.tcp_no_metrics_save = 1
# enable BPF JIT to speed up packet filtering
net.core.bpf_jit_enable = 1
# disable source validation
net.ipv4.conf.[iface].accept_local = 1
net.ipv4.conf.[iface].rp_filter = 0
```

Finally, apply the entries using the `sysctl` utility.

```
# vi /etc/sysctl.d/ixgb.conf
```

11.3 UDP Multi-queue Hashing Algorithms

Modern network cards provide multiple receive RX queues where each queue can be pinned to a dedicated CPU or core. This allows packets to flow through all RX queues utilizing all CPUs or cores. The distribution of packets to queues is accomplished via hashing algorithms on the packet headers such as source/destination IPv4/IPv6 addresses/ports (tuple).

Some network cards default to hashing algorithms that consider the source and destination IP addresses only. This limits the NIC's ability to take to advantage of all available RX queues, thus, not utilizing all available CPUs or cores. An example of such a case is demonstrated below.

```
# ethtool -n ifname rx-flow-hash udp4
 UDP over IPV4 flows use these fields for computing Hash flow key:
 IP SA
 IP DA
```

The hashing algorithm can be changed to include source and destination ports in addition to the source and destination IPv4/IPv6 addresses. Note that not all NICs support the `ethtool -n/-N` options.

```
# ethtool -N ifname rx-flow-hash udp4 sdfn
# ethtool -N ifname rx-flow-hash udp6 sdfn
```

11.4 Network Card PCI Bus Tuning

The configuration involves increasing the Maximum Memory Read Byte Count (MMRBC) in PCI-X configuration space to increase transmit burst lengths on the bus. Note that this is a run-time configuration only, and it also depends on the network card hardware. The process initially involves identifying the vendor and device IDs of the monitoring network interface. Once these are identified, the adapter registers - MMRBC field – is modified.

First, identify the hardware ID of the network interface using `lspci.` The below examples show the output of an Intel I350 1G igb network card, with the hardware ID highlighted in green.

```
# lspci | grep "Ethernet"
 08:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
```

Second, identify the vendor and device IDs from PCI devices. The below output highlights the hardware, vendor, and device IDs (minified).

```
# grep 0800 /proc/bus/pci/devices
 0800  80861521  30  d9b00000 0  0  d9ff0000  0  0  d9000000  100000  0  0  4000  ...  igb
```

```
Vendor ID: 8086 = Intel
Device ID: 1521 = igb, 1a48 = ixgb
```

Finally, increase MMRBC based on the network card, for example 4K for Intel 10G `ixgbe` network cards.

```
# setpci -v -d 8086:1a48 e6.b=2e
```

Note that:

```
-d     : Location of Ethernet Interface on PCI-X Bus Structure
e6.b   : Address of PCI-X Command Register
2e     : Value to be set. Possible value are:
         MM   Value (Bytes)
         22   512  ← Default
         26   1024
         2a   2048
         2e   4096 ← for Intel 10GbE.
```

Since the above configuration is applied at runtime, persisting the changes requires a startup script or service. To revert the changes, restart the host or manually set the MMRBC to its default value using the below command.

```
# setpci -v -d 8086:1a48 e6.b=22
```

11.5 Optimizing Snort 3 at Build Time

During building Snort in section 4 Installing Snort 3 a number of compiler flags were exported prior to building the source.

```
# export CFLAGS="-O3"
# export CXXFLAGS="-O3 -fno-rtti"
```

The `-O3` flag includes all of the optimizations in `-O1` and `-O2`. It informs the compiler to reduce code size and execution time and increase the performance of the generated code. Discussing individual optimization flags is outside the scope of this guide.

The `-fno-rtti` flag disables the generation of Run Time Type Information (RTTI) identification features in C++. This allows for reduced binary image size.

Additionally, `tcmalloc` was enabled while building Snort 3. As discussed in previous sections, `tcmalloc` enables performance improvements and memory usage reduction. Thus, allowing reduced processing time.

```
# ./configure_cmake.sh --enable-tcmalloc
```

11.6 Optimizing Snort 3 at Run Time

Improve Snort 3 Performance with Hyperscan

Hyperscan can improve Snort 3 performance as follows. Note that enabling hyperscan will result in longer times for Snort's startup process loading the rules.

1. Boost Snort 3's IPS fast pattern matching.
2. Faster search engine than existing search engines.
3. Assist in application identification and HTTP inspection.
4. Faster literal content searches and pcre matches during signature evaluation.

The configuration changes to Snort 3 in order to enable `hyperscan` are implemented via tweaks. Snort 3 tweaks are tunable configurations that allows configuring Snort relative to the default configurations. Tweaks are expressed in Lua files and use Snort configuration syntax. Some of the built-in tweaks include `talos.lua`, `security.lua`, and `max_detect.lua`. For this guide, the configurations demonstrated below are added to a custom tweaks Lua file within Snort's configurations directory.

```
# touch /usr/local/snort/etc/snort/custom_tweaks.lua
```

Tweaks are then incorporated into Snort with the command line option `--tweaks tweak_name`. Using the above custom tweaks file, the custom tweaks are incorporated using `--tweaks custom_tweaks`.

To enable hyperscan, edit the `custom_tweaks.lua` file to include the below lines

```
-- Enable hyperscan for IPS, AppID and HTTP inspection
-- Enable hyperscan for pcre/regex matches
search_engine = { search_method = "hyperscan" }
detection = { hyperscan_literals = true, pcre_to_regex = true }
```

12. Snort 3 Use-case Configurations and Tweaks

The following tweaks explore added-value configurations to Snort 3. They are scenario based, and as such may not be applicable or maybe optional within a given environment.

These tweaks are incorporated by editing the existing `custom_tweaks.lua` file created earlier.

```
# vi /usr/local/snort/etc/snort/custom_tweaks.lua
```

12.1 File Inspection over SMB

File inspection over SMB sessions is disabled by default. To enable it, we configure `binder` is configured to bind observed TCP traffic over port 445 to be inspected by the `dce_smb` inspector by adding the below line to the custom tweaks file `custom_tweaks.lua`. Enabling and controlling file inspection is achieved using the

```
-- Add SMB port binding to dce_smb inspector
table.insert(
    binder, 2,
    { when = { proto = 'tcp', ports = '445', role='any' }, use = { type = 'dce_smb' } })

--  Enable SMB file inspection (unlimited file size inspection example)
dce_smb.smb_file_depth = 0
dce_smb.policy = 'Win7'
```

Finally, ensure that `file` inspector is configured as discussed in section 6.6 Configuring file Inspectors.

Note: Using unlimited file size or depth inspection may introduce throughput and performance penalties. Use reasonable file depth values to achieve desired functionality while maintaining optimized performance.

12.2 ZIP, SWF and PDF Decompression

Snort supported in-place decompression of SWF (Adobe Flash content) and PDF files streams since Snort 2. Snort 3 adds decompression of ZIP files, across multiple inspectors such as `http_inspect`, `smtp`, `imap` and `pop`. This allows Snort to inspect decompressed content against file and IPS.

The below configuration demonstrates enabling decompression for the `http_inspect` and `smtp` inspectors. Other inspectors use the same configuration keywords and patterns, and can be configured in a similar manner.

```
-- Enable ZIP, PDF and SWF decompression in http_inspect
http_inspect.decompress_pdf = true
http_inspect.decompress_swf = true
http_inspect.decompress_zip = true

-- Enable ZIP, PDF and SWF decompression in smtp
smtp.decompress_pdf = true
smtp.decompress_swf = true
smtp.decompress_zip = true
```

12.3 Logging of Email Headers and Attachment Names

Snort 3 can log forensic and investigative details about email connections in `smtp`, `imap` and `pop` protocols other than source and destination IP addresses and ports. These include email headers, attachment names, mail form, and recipient to. Having these logs ready can accelerate detection and incident response.

By default, logging is disabled across the `smtp`, `imap`, and `pop` inspectors. Edit the custom tweaks file `custom_tweaks.lua` and add the below line to enable logging in `smtp` inspector.

```
-- Enable logging of email headers and attachments in smtp
smtp.log_email_hdrs = true
smtp.log_filename = true
smtp.log_mailfrom = true
smtp.log_rcptto = true
```

The generated logs are only applicable in the `unified2` logging format. Sample output is demonstrated below.

```
# /usr/local/snort/bin/u2spewfoo unified2.log
```

```
(ExtraData)
        sensor id: 0    event id: 28    event second: 1223906143
        type: 5 datatype: 1     bloblength: 43  SMTP Attachment Filename: file.pdf
(ExtraDataHdr)
        event type: 4   event length: 58
(ExtraData)
        sensor id: 0    event id: 28    event second: 1223906143
        type: 6 datatype: 1     bloblength: 34  SMTP MAIL FROM Addresses: <abc@addr.test>
(ExtraDataHdr)
        event type: 4   event length: 47
(ExtraData)
        sensor id: 0    event id: 28    event second: 1223906143
        type: 7 datatype: 1     bloblength: 23  SMTP RCPT TO Addresses: <def@addr.test>
```

12.4 Multi-thread Packet Processing

Snort 3 can run multiple packet processing threads on PCAP files or interfaces. Using the new option `--max-packet-threads` or `-z` Snort will start N packet processing threads, where N is the number of threads specified with a maximum of eight threads. The example below runs four threads against a directory (`--pcap-dir`) containing PCAP files while filtering only for PCAP files (`--pcap-filter '*.pcap'`).

```
# snort -c snort.lua --pcap-dir ./pcaps --pcap-filter '*.pcap' --plugin-path /extra -k none -z 4
```

Reviewing Snort threads with the `top` program displays the four threads specified in the example above, plus an additional thread for logging as a result of using the `-l` option.

```
PID   USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
17079 root      20   0 1297372   1.0g   8560 R 98.0 18.0   0:04.43 snort
17094 root      20   0 1297372   1.0g   8560 R 35.3 18.0   0:01.06 snort
17095 root      20   0 1297372   1.0g   8560 R 34.0 18.0   0:01.02 snort
17097 root      20   0 1297372   1.0g   8560 R  8.0 18.0   0:00.24 snort
17028 root      20   0 1297372   1.0g   8560 S  1.7 18.0   0:15.40 snort
```

Note that when using multiple threads while logging to files, each thread will generate its own set of log files,

```
-rw-------. 1 root root  1107 Aug 11 19:41 0_alert_json.txt
-rw-------. 1 root root    72 Aug 11 19:41 0_appid_stats.log
-rw-------. 1 root root     0 Aug 11 19:41 0_data_log
-rw-------. 1 root root     0 Aug 11 19:41 0_file.log
-rw-------. 1 root root  3131 Aug 11 19:41 1_alert_json.txt
-rw-------. 1 root root   328 Aug 11 19:41 1_appid_stats.log
-rw-------. 1 root root     0 Aug 11 19:41 1_data_log
-rw-------. 1 root root     0 Aug 11 19:41 1_file.log
...
```

If the `--id-subdir` option is used, then each thread will create a directory named after the thread's ID under the specified log directory.

```
├── 0
│   ├── alert_json.txt
│   ├── appid_stats.log
│   ├── data_log
│   ├── file.log
├── 1
│   ├── alert_json.txt
│   ├── appid_stats.log
│   ├── data_log
│   ├── file.log
...
```

12.5 Snort 3 Inline (IPS) with DAQ afpacket

The setup and configuration for running Snort 3 inline does not differ from Snort 2. The below configuration example sets the IPS policy mode to inline and configures DAQ to run in inline mode with an Inline Pair as the inputs interface.

```
ips =
{
    mode = inline,
    ...
}
```

```
daq =
{
    module_dirs =
    {
        '/usr/local/lib/daq',
    },
    modules =
    {
        {
            name = 'afpacket',
            mode = 'inline',
            variables =
            {
                'fanout_type=hash'
            }
        }
    },
    inputs =
    {
        'ens192:ens224',
    }
}
```

The above configurations can be executed from the command line directly as show below.

```
# snort -c snort.lua --daq-dir /usr/local/lib/daq --daq afpacket --daq-var fanout_type=hash -i
ens192:ens224 -Q
```

Update the `normalizer` inspector module to reflect the following configuration.

```
normalizer =
{
    tcp =
    {
        ips = true,
    }
}
```

Appendix

Installing hyperscan from Sources

Prior to installing `hyperscan`, the following dependencies should be installed: **Colm** (`colm`), **Ragel** (`ragel`), **Boost** and **sqlite3** (`sqlite-devel`). CentOS 8 does not come with Python preinstalled. Building `hyperscan` requires a python interpreter, **python3** (`python3`) installed.

```
# dnf install python3 sqlite-devel colm ragel
```

The remaining dependency is `boost`, which is downloaded and decompressed without building it.

```
# curl -LO https://dl.bintray.com/boostorg/release/1.73.0/source/boost_1_73_0.tar.gz
# tar xf boost_1_73_0.tar.gz
```

Download and install `hyperscan` (5.3.0).

```
# curl -Lo hyperscan-5.3.0.tar.gz https://github.com/intel/hyperscan/archive/v5.3.0.tar.gz
# tar xf hyperscan-5.3.0.tar.gz
# mkdir hs-build && cd hs-build
```

There are two methods to make hyperscan aware of the Boost headers: 1) `Symlink`, **or** 2) Passing `BOOST_ROOT` pointing to the root directory of the `boost` headers to `cmake`. Both methods are shown below.

Method 1 – `Symlink`:

```
# ln -s ~/sources/boost_1_73_0 /boost ~/sources/hyperscan-5.3.0/include/boost
# cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local ../hyperscan-5.3.0
```

Method 2 – `BOOST_ROOT`:

```
# cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local -
DBOOST_ROOT=../boost_1_73_0 ../hyperscan-5.3.0
```

Proceed with installing Hyperscan.

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