Snort 3 on CentOS 8

Generated: 2020-03-14

Table of Contents

1. Introduction

This guide walks through installing, configuring and testing Snort 3 on CentOS version 8.1. Some of the configurations may not be applicable to production sensors. The steps in this guide should be tested first.

CentOS 8.1 Image

```
Base Image    : CentOS-8.1.1911-x86_64-dvd1.iso
Release       : CentOS Linux release 8.1.1911 (Core)
Kernel        : 4.18.0-147.5.1.el8_1.x86_64
Software      : Minimal Install
```

Snort version and build

```
Build         : Snort 3 build 270
Source        : git
```

LibDAQ version

```
Build         : 3.0.0
Source        : git
```

Paths used for installing and configuring Snort

```
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
```

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-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	Source Code
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)

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** (`pkgconfig`), **zlib** (`zlib-devel`), **LuaJIT** (`luajit-devel`), and **libmnl** (`libmnl-devel`).

```
# dnf install libpcap-devel pcre-devel libdnet-devel hwloc-devel openssl-devel zlib-devel
luajit-devel pkgconfig libmnl-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, which should result in a similar output (omitted) as demonstrated below.

```
# ./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. Prior to installing `hyperscan`, the following dependencies should be installed: **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
```

> **Info:** Installing `Ragel` version >= 7.x requires installing `colm` first. `Ragel` version 6.1.0 does not require installing `colm`.

Proceed with installing `colm` (0.13.0.7) and `ragel` (7.0.0.12).

```
# curl -LO http://www.colm.net/files/colm/colm-0.13.0.7.tar.gz
# tar xf colm-0.13.0.7.tar.gz && cd colm-0.13.0.7
# ./configure
# make -j$(nproc)
# make -j$(nproc) install
# ldconfig
# cd ../
```

```
# curl -LO http://www.colm.net/files/ragel/ragel-7.0.0.12.tar.gz
# tar xf ragel-7.0.0.12.tar.gz && cd ragel-7.0.0.12
# ./configure
# make -j$(nproc)
# make -j$(nproc) install
# ldconfig
# cd ../
```

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

```
# curl -LO https://dl.bintray.com/boostorg/release/1.72.0/source/boost_1_72_0.tar.gz
# tar xf boost_1_72_0.tar.gz
```

Download and install `hyperscan` (5.2.1).

```
# curl -Lo hyperscan-5.2.1.tar.gz https://github.com/intel/hyperscan/archive/v5.2.1.tar.gz
# tar xf hyperscan-5.2.1.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_72_0 /boost ~/sources/hyperscan-5.2.1/include/boost
# cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local ../hyperscan-5.2.1
```

Method 2 – `BOOST_ROOT`:

```
# cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local -
DBOOST_ROOT=../boost_1_72_0 ../hyperscan-5.2.1
```

Proceed with installing Hyperscan.

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

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.zip
# 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/local/lib64/pkgconfig/libsafec.pc
```

If installing Safec from source is preferred, follow the steps below.

```
# curl -LO https://github.com/rurban/safeclib/releases/download/v08112019/libsafec-
08112019.0-gad76c7.tar.gz
# tar xf libsafec-08112019.0-gad76c7.tar.gz && cd libsafec-08112019.0-gad76c7
# ./configure
# make
# make install
# cd ../
```

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.

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

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

```
--------------------------------------------------------
snort version 3.0.0

Install options:
    prefix:     /usr/local/snort
    includes:   /usr/local/snort/include/snort
    plugins:    /usr/local/snort/lib64/snort
...
Feature options:
    DAQ Modules:    Static (afpacket;bpf;dump;fst;nfq;pcap;trace)
    Flatbuffers:    ON
    Hyperscan:      ON
    ICONV:          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 3.0.0 (Build 270)
    ''''     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 1.1.1c FIPS  28 May 2019
             Using libpcap version 1.9.0-PRE-GIT (with TPACKET_V3)
             Using PCRE version 8.42 2018-03-20
             Using ZLIB version 1.2.11
             Using FlatBuffers 1.11.0
             Using Hyperscan version 5.2.1 2020-03-05
             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-12159.tar.gz https://snort.org/downloads/openappid/12159
# tar xf snort-openappid-12159.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 https://www.talosintelligence.com/documents/ip-blacklist
# mv ip-blacklist /usr/local/snort/intel/
# touch /usr/local/snort/intel/ip-whitelist
```

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
WHITE_LIST_PATH = '../../intel'
BLACK_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'

    -- 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-sql.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'

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

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 = BLACK_LIST_PATH .. '/ip-blacklist',
    whitelist = WHITE_LIST_PATH .. '/ip-whitelist'
}
```

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.

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,
    log_stats = true
}
```

6.6   Configuring file and file_log 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 } }
    }
}
```

The final step is to enable event logging for the inspector. This is accomplished with the `file_log` inspector. This inspector has two Boolean options that allow logging of packet and system time of 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' } } }
>     }
> ```

6.7   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
}
```

6.8   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. Running and Testing Snort 3

Running Snort requires setting two environment variables, `LUA_PATH` and `SNORT_LUA_PATH`. These variables point to the lua and configuration directories within the Snort installation prefix.

```
# export LUA_PATH=/usr/local/snort/include/snort/lua/\?.lua\;\;
# export SNORT_LUA_PATH=/usr/local/snort/etc/snort
```

7.1   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
```

7.2   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
> ```

7.3   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
```

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

8.1 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 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 and 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, the `ETHTOOL_CMD` parameter combined with dispatcher script is used. This will ensure that the interface ring buffers are 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
```

```bash
#!/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. Once the service is 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 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,MULTICAST,NOARP,PROMISC,UP,LOWER_UP>
```

```
# ethtool -g ifname

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

8.2   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
Environment="LUA_PATH=/usr/local/snort/include/snort/lua/?.lua;;"
Environment="SNORT_LUA_PATH=/usr/local/snort/etc/snort"
ExecStart=/usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort.lua -i ifname -l
/var/log/snort -D -u snort -g snort -k none
ExecReload=/bin/kill -SIGHUP $MAINPID
User=snort
Group=snort
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 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
```