TP SNORT

Sommaire

I. Présentation 2

1. Introduction : Le contexte de la sécurité réseau 2
2. Qu'est-ce que Snort ? 2
3. Les 3 modes de fonctionnement de Snort 3
4. L'anatomie d'une règle 3

II. Installation 4

1. Préparation, mise à jours et dépendances 4
2. Compilation et installation LibDAQ 5
3. Installation Snort3 7
4. Vérification de l'installation 10

III. Configuration 11

1. Configuration par défaut 11
2. Vérification de la configuration 12

IV. Règles et Snort Live 13

1. Mise en place des Snort Community Rules 13
2. Création de règles local (test ICMP) 15
3. Exécuter Snort Live 16
4. Création d'une règle SSH 20
5. Création d'une règle HTTP 21
6. Blocage d'une requête après une tentative de spam ICMP 22

I. Présentation

1. Introduction : Le contexte de la sécurité réseau

Avant de parler de Snort, il faut comprendre le problème qu'il résout. Aujourd'hui, avoir un simple pare-feu (Firewall) ne suffit plus. Un pare-feu agit comme un vigile à l'entrée d'un bâtiment : il vérifie les badges (ports, adresses IP). Mais que se passe-t-il si une personne avec un badge valide décide de faire quelque chose de malveillant une fois à l'intérieur ?

C'est là qu'interviennent les **IDS** et les **IPS**.

- **IDS (Intrusion Detection System)** : C'est le système de vidéosurveillance. Il observe le trafic réseau de l'intérieur, détecte les comportements suspects ou malveillants, et lève une alerte (mais il ne bloque rien par lui-même).
- **IPS (Intrusion Prevention System)** : C'est un IDS avec le pouvoir d'agir. S'il détecte une menace, il peut bloquer la connexion et rejeter les paquets.

2. Qu'est-ce que Snort ?

Snort est un système de détection d'intrusion réseau (NIDS) open-source. Créé en 1998 par Martin Roesch, il est aujourd'hui maintenu par Cisco. C'est tout simplement le standard de l'industrie. Même si vous utilisez d'autres solutions de sécurité plus tard, il y a de fortes chances qu'elles utilisent le moteur ou la syntaxe des règles de Snort sous le capot.

Ses principales qualités :

- Open-source et gratuit : Idéal pour l'apprentissage et très utilisé en entreprise.
- Léger et rapide : Il peut analyser des volumes de trafic importants en temps réel.
- Basé sur des règles : Il compare le trafic réseau avec une base de données de "signatures" (des empreintes de cyberattaques connues). Ou des règles définies en local.

3. Les 3 modes de fonctionnement de Snort

Snort est un outil polyvalent qui peut être utilisé de trois manières différentes, de la plus simple à la plus complexe :

- Mode "Sniffer" (Renifleur) : Snort se contente de lire les paquets qui circulent sur le réseau et de les afficher à l'écran en temps réel (un peu comme Wireshark ou tcpdump).
- Mode "Packet Logger" (Enregistreur de paquets) : Au lieu d'afficher les paquets à l'écran, Snort les enregistre dans des fichiers journaux (logs) sur le disque dur pour une analyse ultérieure.
- Mode NIDS (Network Intrusion Detection System) : C'est le mode qui nous intéresse pour ce TP. Snort analyse le trafic réseau, le compare à un ensemble de règles définies par l'administrateur, et déclenche des alertes s'il trouve une correspondance.

4. L'anatomie d'une règle

La véritable puissance de Snort réside dans ses règles. Elles sont divisées en deux parties : l'en-tête (Header) et les options.

Voici un exemple classique d'une règle Snort simple :



Décortiquons-l'en-tête (Header) :

`alert` : L'action à effectuer (ici, générer une alerte).

`tcp` : Le protocole surveillé (TCP, UDP, ICMP, IP...).

`any any` : L'adresse IP source et le port source (ici, n'importe quelle IP, n'importe quel port).

`->` : La direction du trafic.

`192.168.1.0/24 80` : L'adresse IP de destination (notre réseau local) et le port de destination (80, trafic HTTP).

Décortiquons les options (entre parenthèses) :

`msg`:"..." : Le message qui apparaîtra dans les logs de l'alerte.

`sid:1000001` : L'identifiant unique de la règle (Snort ID). Les règles personnalisées doivent avoir un SID > 1 000 000.

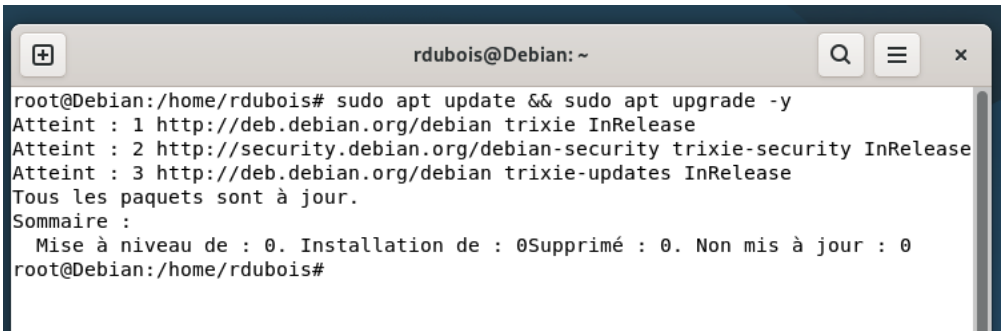
`rev:1` : Le numéro de révision de la règle (utile quand on la modifie).

II. Installation

1. Préparation, mise à jours et dépendances

Pour installer Snort sur Debian 13, nous allons devoir le télécharger et le compiler nous même avec make.

Pour commencer, nous allons vérifier les mises à jour puis nous allons télécharger les dépendances nécessaires pour Snort (et pour compiler des applications en C)

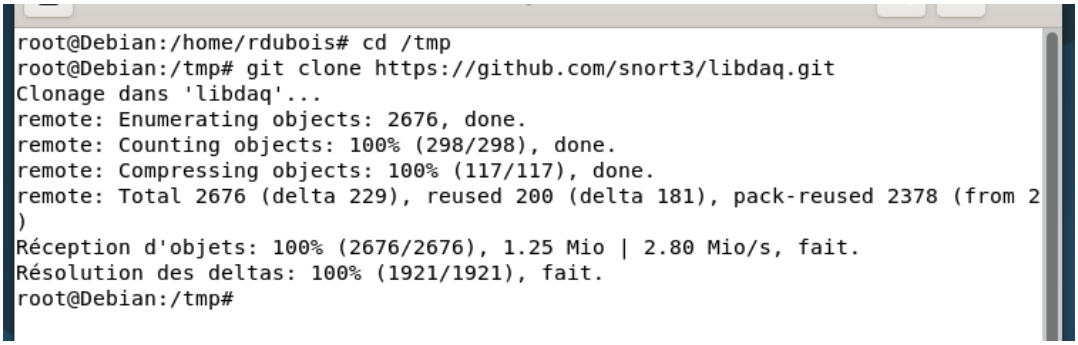


Commande utilisée pour le téléchargement des dépendances :

```
sudo apt install -y build-essential autotools-dev libdumbnet-dev \
libluajit-5.1-dev libpcap-dev libpcre2-dev zlib1g-dev pkg-config \
libhwloc-dev cmake liblzma-dev openssl libssl-dev cpputest \
libsqlite3-dev uuid-dev libcmocka-dev libnetfilter-queue-dev \
libmnl-dev autoconf libtool git flex bison
```

2. Compilation et installation LibDAQ

Une fois nos dépendances installées, nous allons devoir compiler et installer libdaq (DAQ (Data Acquisition) Library) étant nécessaire pour le bon fonctionnement de Snort3. Pour cela, nous allons le cloner avec git clone dans le dossier /tmp :



Maintenant, nous allons nous rendre dans le répertoire libdaq qui a été créer, et puis nous allons lancer les scripts **bootstrap** et **configure** pour tout préparer avant la compilation :

```
root@Debian:/tmp# cd libdaq
root@Debian:/tmp/libdaq# ./bootstrap
+ autoreconf -ivf --warnings=all
autoreconf: export WARNINGS=all
autoreconf: Entering directory '.'
autoreconf: configure.ac: not using Gettext
autoreconf: running: aclocal --force -I m4
autoreconf: configure.ac: tracing
autoreconf: running: libtoolize --copy --force
libtoolize: putting auxiliary files in '.'.
libtoolize: copying file './ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4'.
libtoolize: copying file 'm4/libtool.m4'
libtoolize: copying file 'm4/ltoptions.m4'
libtoolize: copying file 'm4/ltsugar.m4'
libtoolize: copying file 'm4/ltversion.m4'
libtoolize: copying file 'm4/lt~obsolete.m4'
autoreconf: configure.ac: not using Intltool
autoreconf: configure.ac: not using Gtkdoc
autoreconf: running: aclocal --force -I m4
autoreconf: running: /usr/bin/autoconf --force
configure.ac:27: warning: The macro 'AC_PROG_CC_C99' is obsolete.
configure.ac:27: You should run autoupdate.
./lib/autoconf/c.m4:1662: AC_PROG_CC_C99 is expanded from...
configure.ac:27: the top level
autoreconf: running: /usr/bin/autoheader --force
autoreconf: running: automake --add-missing --copy --force-missing
configure.ac:29: installing './ar-lib'
configure.ac:26: installing './compile'
configure.ac:34: installing './config.guess'
configure.ac:34: installing './config.sub'
configure.ac:19: installing './install-sh'
configure.ac:19: installing './missing'
api/Makefile.am: installing './depcomp'
parallel-tests: installing './test-driver'
autoreconf: Leaving directory '.'
root@Debian:/tmp/libdaq#
```

```
config.status: creating modules/afpacket/libdaq_static_afpacket.pc
config.status: creating modules/bpf/libdaq_static_bpf.pc
config.status: creating modules/dump/libdaq_static_dump.pc
config.status: creating modules/fst/libdaq_static_fst.pc
config.status: creating modules/nfq/libdaq_static_nfq.pc
config.status: creating modules/pcap/libdaq_static_pcap.pc
config.status: creating modules/savefile/libdaq_static_savefile.pc
config.status: creating modules/trace/libdaq_static_trace.pc
config.status: creating modules/gwlb/libdaq_static_gwlb.pc
config.status: creating Makefile
config.status: creating api/daq_version.h
config.status: creating api/Makefile
config.status: creating example/Makefile
config.status: creating modules/Makefile
config.status: creating test/Makefile
config.status: creating libdaq.pc
config.status: creating config.h
config.status: executing depfiles commands
config.status: executing libtool commands

    libdaq 3.0.25

    prefix:         /usr/local
    sysconfdir:     ${prefix}/etc
    libdir:         ${exec_prefix}/lib
    includedir:     ${prefix}/include

    cc:             gcc
    cppflags:
    am_cppflags:     -fvisibility=hidden -Wall -Wmissing-declarations -Wpointer-arith -Wcast-align -Wcast-qual -Wformat -Wformat-nonliteral -Wformat-security -Wundef -Wwrite-strings -Wextra -Wsign-compare -Wno-u
nused-parameter -fno-strict-aliasing -fdiagnostics-show-option
    cflags:         -g -O2
    am_cflags:       -Wstrict-prototypes -Wmissing-prototypes -Wold-style-definition -Wnested-externs
    ldflags:
    am_ldflags:
    libs:

    code_coverage_enabled:  no
    code_coverage_cppflags:
    code_coverage_cflags:
    code_coverage_ldflags:

    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 netmap DAQ module.... : no
    Build NFQ DAQ module....... : yes
    Build PCAP DAQ module...... : yes
    Build Savefile DAQ module.. : yes
    Build Trace DAQ module..... : yes
    Build GWLB DAQ module...... : yes

root@Debian:/tmp/libdaq#
```

Maintenant que nos 2 deux scripts ont été exécuté, nous pouvons lancer la compilation avec make. Commande : **make -j$(nproc)**. L'argument -j$(nproc) permet d'accélérer la compilation en utilisant tous les cœurs disponibles sur notre machine :

```
make[2] : on quitte le répertoire « /tmp/libdaq/example »
Making all in test
make[2] : on entre dans le répertoire « /tmp/libdaq/test »
make[2]: Rien à faire pour « all ».
make[2] : on quitte le répertoire « /tmp/libdaq/test »
make[2] : on entre dans le répertoire « /tmp/libdaq »
make[2]: Rien à faire pour « all-am ».
make[2] : on quitte le répertoire « /tmp/libdaq »
make[1] : on quitte le répertoire « /tmp/libdaq »
root@Debian:/tmp/libdaq#
```

Une fois la compilation effectuée, nous allons pouvoir installer notre programme avec la commande : **sudo make install** :

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

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

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

Maintenant, nous allons devoir exécuter ldconfig (**sudo ldconfig**). ldconfig met à jour la carte des bibliothèques partagées de votre système Linux. Ce qui est nécessaire après l'installation de DAQ :

```
root@Debian:/tmp/libdaq# sudo ldconfig
root@Debian:/tmp/libdaq#
```

3. Installation Snort3

Une fois ceci fait, nous allons pouvoir passer à l'installation de Snort 3. Pour cela même principe que DAQ, nous allons nous rendre dans le dossier /tmp, puis nous

allons cloner avec git nos fichiers Snort, puis nous allons nous rendre dans le dossier snort3 :

```
root@Debian:/tmp/libdaq# cd /tmp
root@Debian:/tmp# git clone https://github.com/snort3/snort3.git
Clonage dans 'snort3'...
remote: Enumerating objects: 126868, done.
remote: Counting objects: 100% (15492/15492), done.
remote: Compressing objects: 100% (3902/3902), done.
remote: Total 126868 (delta 12668), reused 11941 (delta 11590), pack-reused 111376 (from 3)
Réception d'objets: 100% (126868/126868), 92.21 Mio | 15.51 Mio/s, fait.
Résolution des deltas: 100% (107174/107174), fait.
root@Debian:/tmp# cd snort3/
root@Debian:/tmp/snort3#
```

Maintenant, nous allons lancer le script configure_cmake.sh avec comme argument, le chemin de notre snort (**./configure_cmake.sh --prefix=/usr/local/snort**) :

```
-------------------------------------------------------
snort version 3.11.1.0

Install options:
    prefix:     /usr/local/snort
    includes:   /usr/local/snort/include/snort
    plugins:    /usr/local/snort/lib/snort

Compiler options:
    CC:             /usr/bin/cc
    CXX:            /usr/bin/c++
    CFLAGS:            -fvisibility=hidden   -DNDEBUG -g -ggdb    -O2 -g -DNDEBUG
    CXXFLAGS:          -fvisibility=hidden   -DNDEBUG -g -ggdb    -O2 -g -DNDEBUG
    EXE_LDFLAGS:
    MODULE_LDFLAGS:

Feature options:
    DAQ Modules:    Static (afpacket;bpf;dump;fst;gwlb;nfq;pcap;savefile;trace)
    libatomic:      System-provided
    Hyperscan:      OFF
    ICONV:          ON
    Libunwind:      OFF
    LZMA:           ON
    RPC DB:         Built-in
    SafeC:          OFF
    TCMalloc:       OFF
    JEMalloc:       OFF
    UUID:           ON
    NUMA:           ON
    LibML:          OFF
-------------------------------------------------------

-- Configuring done (1.2s)
-- Generating done (0.1s)
-- Build files have been written to: /tmp/snort3/build
root@Debian:/tmp/snort3#
```

Une fois nos options prises en compte, les fichiers de compilations ont été placé dans le dossier « build ». Pour lancer la compilation, il suffit donc de se rendre dans ce dossier, et de lancer un **make** comme précédemment :

```
root@Debian:/tmp/snort3# cd build/
root@Debian:/tmp/snort3/build# make -j$(nproc)
```

```
[ 94%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_bo.cc.o
[ 94%] Building CXX object tools/snort2lua/keyword_states/CMakeFiles/keyword_states.dir/keywords_api.cc.o
[ 94%] Built target keyword_states
[ 94%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_dcerpc.cc.o
[ 94%] Building CXX object tools/show_flows/CMakeFiles/show_flows.dir/show_flows.cc.o
[ 95%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_dcerpc_server.cc.o
[ 95%] Linking CXX executable show_flows
[ 95%] Built target show_flows
[ 95%] Building C object daqs/CMakeFiles/daq_file.dir/daq_file.c.o
[ 95%] Linking C shared module daq_file.so
[ 95%] Built target daq_file
[ 95%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_dnp3.cc.o
[ 95%] Building C object daqs/CMakeFiles/daq_hext.dir/daq_hext.c.o
[ 95%] Linking C shared module daq_hext.so
[ 95%] Built target daq_hext
[ 95%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_frag3_engine.cc.o
[ 96%] Building CXX object src/mp_transport/mp_unix_transport/CMakeFiles/mp_unix_transport.dir/mp_unix_transport.cc.o
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_frag3_global.cc.o
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_ftp_telnet.cc.o
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_ftp_telnet_protocol.cc.o
[ 96%] Building CXX object src/mp_transport/mp_unix_transport/CMakeFiles/mp_unix_transport.dir/mp_unix_transport_module.cc.o
[ 96%] Built target mp_unix_transport
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_gtp.cc.o
[ 96%] Building CXX object src/CMakeFiles/snort.dir/main.cc.o
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_http_inspect.cc.o
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_http_inspect_server.cc.o
[ 96%] Linking CXX executable snort
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_normalizers.cc.o
[ 96%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_perfmonitor.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_reputation.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_rpc_decode.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_sdf.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_sip.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_ssh.cc.o
[ 97%] Built target snort
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_ssl.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_dns.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_pop.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_imap.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_modbus.cc.o
[ 97%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_rna.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_smtp.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_sfportscan.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_stream5_ip.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_stream5_global.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_stream5_tcp.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_stream5_udp.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_stream5_ha.cc.o
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/preprocessor_api.cc.o
[ 98%] Built target preprocessor_states
[100%] Building CXX object tools/snort2lua/CMakeFiles/snort2lua.dir/init_state.cc.o
[100%] Building CXX object tools/snort2lua/CMakeFiles/snort2lua.dir/snort2lua.cc.o
[100%] Linking CXX executable snort2lua
[100%] Built target snort2lua
root@Debian:~/snort3/build#
```

Une fois notre compilation terminée, nous pouvons l'installer avec **sudo make install** :

```
-- Installing: /usr/local/snort/share/doc/snort/building.txt
-- Installing: /usr/local/snort/share/doc/snort/builtin_stubs.txt
-- Installing: /usr/local/snort/share/doc/snort/enviro.txt
-- Installing: /usr/local/snort/share/doc/snort/snort_reference.txt
-- Installing: /usr/local/snort/share/doc/snort/snort_reference.text
-- Installing: /usr/local/snort/share/doc/snort/differences.txt
-- Up-to-date: /usr/local/snort/share/doc/snort/overview.txt
-- Installing: /usr/local/snort/share/doc/snort/snort2lua.txt
-- Installing: /usr/local/snort/share/doc/snort/snort_upgrade.txt
-- Installing: /usr/local/snort/share/doc/snort/config_changes.txt
-- Installing: /usr/local/snort/share/doc/snort/snort_upgrade.text
-- Installing: /usr/local/snort/share/doc/snort/snort_devel.txt
-- Installing: /usr/local/snort/share/doc/snort/extending.txt
-- Installing: /usr/local/snort/share/doc/snort/style.txt
-- Installing: /usr/local/snort/share/doc/snort/versions.txt
root@Debian:~/snort3/build#
```

Et comme fait précédemment, nous pouvons faire un « sudo ldconfig » pour mettre à jour la carte des bibliothèques partagées :

```
root@Debian:~/snort3/build# sudo ldconfig
root@Debian:~/snort3/build#
```

4. Vérification de l'installation

Maintenant que Snort est installé, nous allons vérifier si ce dernier répond à la commande -V (pour voir sa version actuel).

Pour cela, il suffit d'exécuter la commande suivante : **/usr/local/snort/bin/snort -V** :

```
root@Debian:~/snort3/build# /usr/local/snort/bin/snort -V

   ,,_     -*> Snort++ <*-
  o"  )~   Version 3.11.1.0
   ''''    By Martin Roesch & The Snort Team
           http://snort.org/contact#team
           Copyright (C) 2014-2026 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using DAQ version 3.0.25
           Using libpcap version 1.10.5 (with TPACKET_V3)
           Using LuaJIT version 2.1.1737090214
           Using LZMA version 5.8.1
           Using OpenSSL 3.5.4 30 Sep 2025
           Using PCRE2 version 10.46 2025-08-27
           Using ZLIB version 1.3.1

root@Debian:~/snort3/build#
```

La version 3.11.1.0 de Snort a bien été installée ! On peut observer que les dépendances ainsi que LibDAQ est bien installé aussi car Snort détecte la version de ces derniers.

```
root@Debian:~/snort3/build# /usr/local/snort/bin/snort -V

   ,,_     -*> Snort++ <*-
  o"  )~   Version 3.11.1.0
   ''''    By Martin Roesch & The Snort Team
           http://snort.org/contact#team
           Copyright (C) 2014-2026 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using DAQ version 3.0.25
           Using libpcap version 1.10.5 (with TPACKET_V3)
           Using LuaJIT version 2.1.1737090214
           Using LZMA version 5.8.1
           Using OpenSSL 3.5.4 30 Sep 2025
           Using PCRE2 version 10.46 2025-08-27
           Using ZLIB version 1.3.1

root@Debian:~/snort3/build#
```

III. Configuration

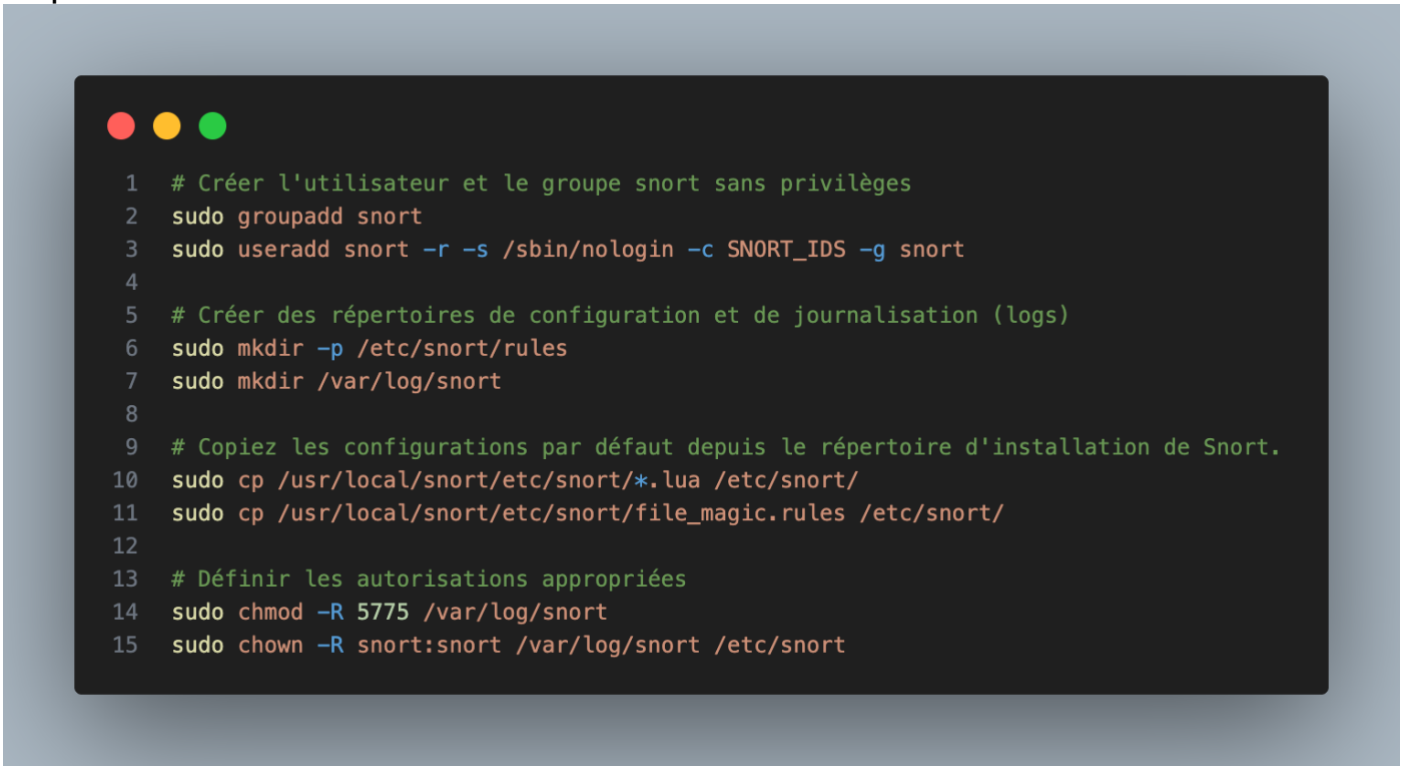
1. Configuration par défaut

Maintenant que nous avons installé Snort, nous allons devoir le configurer avant de définir des règles et de l'utiliser.

Nous allons devoir :

- Créer un utilisateur et un groupe pour Snort (sans privilèges).
- Créer des répertoires pour les règles et les logs.
- Copier les fichiers de configuration par défaut dans le dossier d'installation de Snort.
- Définir les autorisations appropriées pour notre nouvel utilisateur et groupe.

Voici donc une proposition de script bash faisant toutes les tâches de post installation :



```
# Créer l'utilisateur et le groupe snort sans privilèges
sudo groupadd snort
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snort

# Créer des répertoires de configuration et de journalisation (logs)
sudo mkdir -p /etc/snort/rules
sudo mkdir /var/log/snort

# Copiez les configurations par défaut depuis le répertoire d'installation de Snort.
sudo cp /usr/local/snort/etc/snort/*.lua /etc/snort/
sudo cp /usr/local/snort/etc/snort/file_magic.rules /etc/snort/

# Définir les autorisations appropriées
sudo chmod -R 5775 /var/log/snort
sudo chown -R snort:snort /var/log/snort /etc/snort
```

Nous allons donc créer un fichier sur notre machine avec notre script, puis nous allons l'exécuter :

```
root@Debian:~# nano postinstall.sh
root@Debian:~# chmod +x postinstall.sh
root@Debian:~# ./postinstall.sh
root@Debian:~#
```

2. Vérification de la configuration

Une fois ceci fait, nous allons tester notre configuration avec le fichier de configuration **/etc/snort/snort.lua**. Pour cela, nous allons exécuter la commande **/usr/local/snort/bin/snort -c /etc/snort/snort.lua --warn-all** qui va nous permettre de valider la configuration actuel :

```
Finished /etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
--------------------------------------------------
ips policies rule stats
               id  loaded  shared enabled    file
                0     219       0     219    /etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 219
               text rules: 219
             option chains: 219
             chain headers: 1
--------------------------------------------------
service rule counts          to-srv  to-cli
                   file_id:      219     219
                     total:      219     219
--------------------------------------------------
fast pattern groups
                 to_server: 1
                 to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                 instances: 2
                  patterns: 438
              pattern chars: 2602
                 num states: 1832
           num match states: 392
               memory scale: KB
               total memory: 71.2812
             pattern memory: 19.6484
          match list memory: 28.4375
          transition memory: 22.9453
appid: MaxRss diff: 3072
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to passive.

Snort successfully validated the configuration (with 1 warnings).
o")~   Snort exiting
```

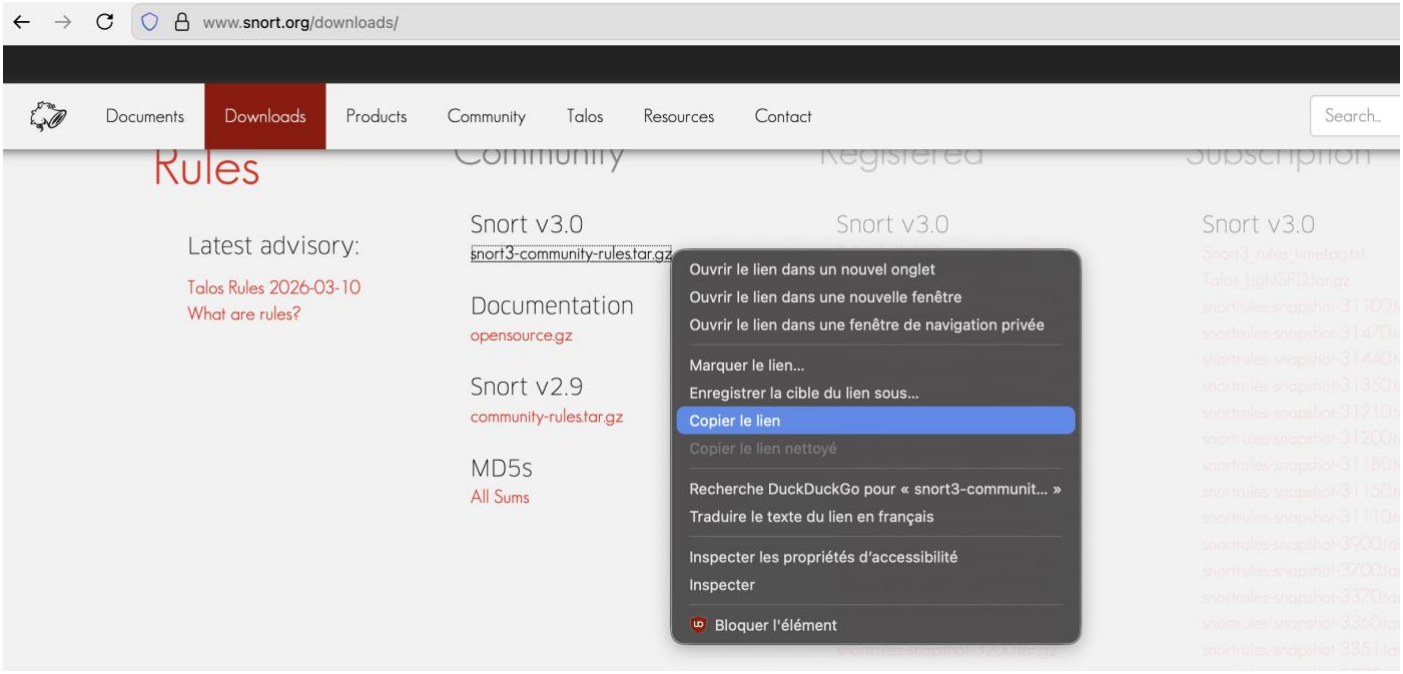
Une fois le test effectué, nous avons le message « Snort successfully validated the configuration » signifiant que notre fichier de configuration est actuellement valide et prêt à être utilisé.

IV. Règles et Snort Live

1. Mise en place des Snort Community Rules

Avant de lancer Snort, nous allons devoir y installer des règles. Il est possible d'en créer sois même bien sûr (nous le verrons plus tard dans ce TP), mais aussi de télécharger des règles existantes d'attaques connues. Ces dernières sont gratuites et disponible sur le site de Snort.

Pour cela, nous allons nous rendre dans /tmp, puis nous allons télécharger les community rules via le site de Snort (commande wget) :




```
root@Debian:~# cd /tmp
root@Debian:/tmp# wget https://www.snort.org/downloads/community/snort3-community-rules.tar.gz
--2026-03-11 09:59:07--  https://www.snort.org/downloads/community/snort3-community-rules.tar.gz
Résolution de www.snort.org (www.snort.org)… 104.16.92.19, 104.16.91.19
Connexion à www.snort.org (www.snort.org)|104.16.92.19|:443… connecté.
requête HTTP transmise, en attente de la réponse… 302 Found
Emplacement : https://snort-org-site.s3.amazonaws.com/production/release_files/files/000/053/574/original/snort3-community-rules.tar.gz?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAU7AK5ITMMFKW2CPY%2F20
260311%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20260311T085908Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=3ca176b9db5dca70cda5ec2fafad25bf78a84617a0479600a1a4f73ffe0195e7 [suivant]
--2026-03-11 09:59:08--  https://snort-org-site.s3.amazonaws.com/production/release_files/files/000/053/574/original/snort3-community-rules.tar.gz?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAU7AK5ITMMF
KW2CPY%2F20260311%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20260311T085908Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=3ca176b9db5dca70cda5ec2fafad25bf78a84617a0479600a1a4f73ffe0195e7
Résolution de snort-org-site.s3.amazonaws.com (snort-org-site.s3.amazonaws.com)… 52.217.166.17, 52.217.168.201, 52.217.203.201, ...
Connexion à snort-org-site.s3.amazonaws.com (snort-org-site.s3.amazonaws.com)|52.217.166.17|:443… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 331476 (324K) [application/octet-stream]
Sauvegarde en : « snort3-community-rules.tar.gz »

snort3-community-rules.tar.gz                      100%[===================================================================================================>] 323,71K   922KB/s    ds 0,4s

2026-03-11 09:59:09 (922 KB/s) — « snort3-community-rules.tar.gz » sauvegardé [331476/331476]

root@Debian:/tmp#
```

Nous pouvons donc maintenant décompresser ces dernières (avec la commande **tar**) et puis copier le fichier **snort3-community.rules** dans le dossier que nous avions créé pour les règles (**/etc/snort/rules/**) :

```
root@Debian:/tmp# tar -xvzf snort3-community-rules.tar.gz
snort3-community-rules/
snort3-community-rules/snort3-community.rules
snort3-community-rules/VRT-License.txt
snort3-community-rules/LICENSE
snort3-community-rules/AUTHORS
snort3-community-rules/sid-msg.map
root@Debian:/tmp# sudo cp snort3-community-rules/snort3-community.rules /etc/snort/rules/
root@Debian:/tmp#
```

2. Création de règles local (test ICMP)

Maintenant, nous allons devoir inclure notre règle dans le fichier de configuration (**snort.lua**). Pour cela, nous allons ouvrir le fichier, et puis nous rendre à la ligne « ips = » :

```
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
    -- (see also related path vars at the top of snort_defaults.lua)

    variables = default_variables
}
```

Ici, nous allons devoir enlever les tirets avant le « **enable_builtin_rules = true** », et nous allons devoir ajouter cette ligne : « **include = '/etc/snort/rules/snort3-community.rules')** » pour que Snort prenne en compte les règles que nous venons de télécharger :

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

    -- use include for rules files; be sure to set your path
    include = '/etc/snort/rules/snort3-community.rules',
    -- note that rules files can include other rules files
    -- (see also related path vars at the top of snort_defaults.lua)

    variables = default_variables
}
```

Maintenant que nous avons ajouté nos community rules, nous allons créer notre propre règle local permettant de détecter si une machine sur le réseau envoie des pings (requêtes ICMP) sur cette notre machine Debian. Pour cela, nous allons créer un fichier **/etc/snort/rules/local.rules** avec comme contenu le suivant :

```
 GNU nano 8.4                                                    /etc/snort/rules/local.rules *
alert icmp any any -> any any (msg:"TEST: Quelqu'un ping cette machine !"; sid:1000002; rev:1;)
```

Ici, nous demandons à Snort de surveiller toute les requêtes ICMP (ping) arrivant sur notre machine. Nous allons donc sauvegarder ce fichier.

Il faut aussi ne pas oublier de rajouter le fichier que nous venons de créer dans notre fichier de configuration **snort.lua** :

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

    -- use include for rules files; be sure to set your path
    include = '/etc/snort/rules/snort3-community.rules',
    include = '/etc/snort/rules/local.rules',
    -- note that rules files can include other rules files
    -- (see also related path vars at the top of snort_defaults.lua)

    variables = default_variables
}
```

3. Exécuter Snort Live

Une fois que tout est configuré, nous allons pouvoir Snort en direct pour que ce dernier puisse surveiller les requêtes qu'on lui envoie.

Avant de commencer, nous allons devoir récupérer le nom de notre carte réseau, pour récupérer cette dernière, il suffit de faire un **ip a** dans notre terminal et de copier le nom de notre carte :

```
root@Debian:/tmp# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:1c:42:60:a1:00 brd ff:ff:ff:ff:ff:ff
    altname enx001c4260a100
    inet 10.211.55.9/24 brd 10.211.55.255 scope global dynamic noprefixroute enp0s5
       valid_lft 1153sec preferred_lft 1153sec
    inet6 fdb2:2c26:f4e4:0:8e5f:6753:d43:6b2b/64 scope global temporary dynamic
       valid_lft 598867sec preferred_lft 80024sec
    inet6 fdb2:2c26:f4e4:0:21c:42ff:fe60:a100/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 2591905sec preferred_lft 604705sec
    inet6 fe80::21c:42ff:fe60:a100/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
root@Debian:/tmp#
```

Une fois le nom de notre carte récupéré, nous allons pouvoir l'insérer dans la commande suivante afin de lancer Snort sur notre interface :



-c : Fichier de configuration

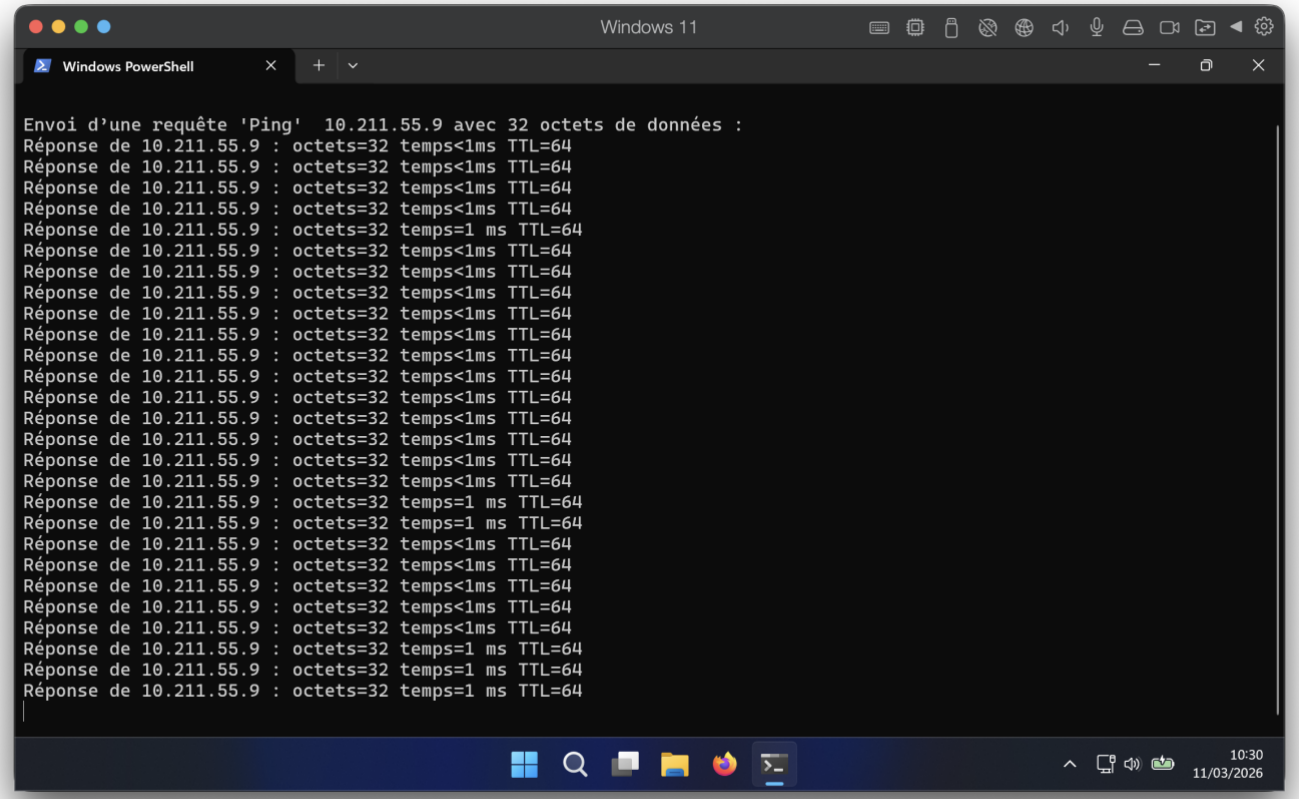
-i : Interface réseau

-A : Le mode d'alertes (afficher uniquement ce qui est nécessaire)

-k : le mode somme de contrôle (checksum en none sur une VM)

```
Finished /etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading /etc/snort/rules/local.rules:
Finished /etc/snort/rules/local.rules:
-------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     865       0     865    /etc/snort/snort.lua
-------------------------------------------------
rule counts
       total rules loaded: 865
               text rules: 220
             builtin rules: 645
             option chains: 865
             chain headers: 3
-------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any     645       0       1       0
   total     645       0       1       0
-------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
-------------------------------------------------
fast pattern groups
                to_server: 1
                to_client: 1
-------------------------------------------------
search engine (ac_bnfa)
                 instances: 2
                  patterns: 438
             pattern chars: 2602
                num states: 1832
          num match states: 392
              memory scale: KB
              total memory: 71.2812
            pattern memory: 19.6484
         match list memory: 28.4375
         transition memory: 22.9453
appid: MaxRss diff: 2816
appid: patterns loaded: 300
-------------------------------------------------
pcap DAQ configured to passive.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] enp0s5
```

Snort est maintenant en train de fonctionner, nous pouvons tenter sur une machine distante de faire un ping pour voir si Snort le détecte :




```
03/11-10:30:40.955552 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:40.955580 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:42.016636 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:42.016688 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:42.512067 [**] [112:1:1] "(arp_spoof) unicast ARP request" [**] [Priority: 3] {ARP}  ->
03/11-10:30:43.030255 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:43.030322 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:44.043215 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:44.043273 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:45.065014 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:45.065042 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:46.080486 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:46.080562 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:47.096443 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:47.096502 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:48.112728 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:48.112783 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:49.129669 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:49.129712 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:50.155810 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:50.155874 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:51.226888 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:51.226948 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:52.281381 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:52.281467 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:53.308141 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:53.308219 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:54.367439 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:54.367489 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:55.381852 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:55.381918 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:56.396476 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:56.396522 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:57.437182 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:57.437244 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:58.453889 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:58.453953 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:30:59.530379 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:30:59.530444 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:31:00.576019 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:31:00.576072 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:31:01.624081 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:31:01.624158 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:31:02.619799 [**] [112:1:1] "(arp_spoof) unicast ARP request" [**] [Priority: 3] {ARP}  ->
03/11-10:31:02.646574 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:31:02.646674 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:31:03.681991 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:31:03.682076 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:31:04.688138 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:31:04.688188 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:31:05.714837 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:31:05.714929 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
03/11-10:31:06.775311 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.4 -> 10.211.55.9
03/11-10:31:06.775383 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} 10.211.55.9 -> 10.211.55.4
```

Nous observons que Snort détecte bien les requêtes ICMP (ping) !

4. Création d'une règle SSH

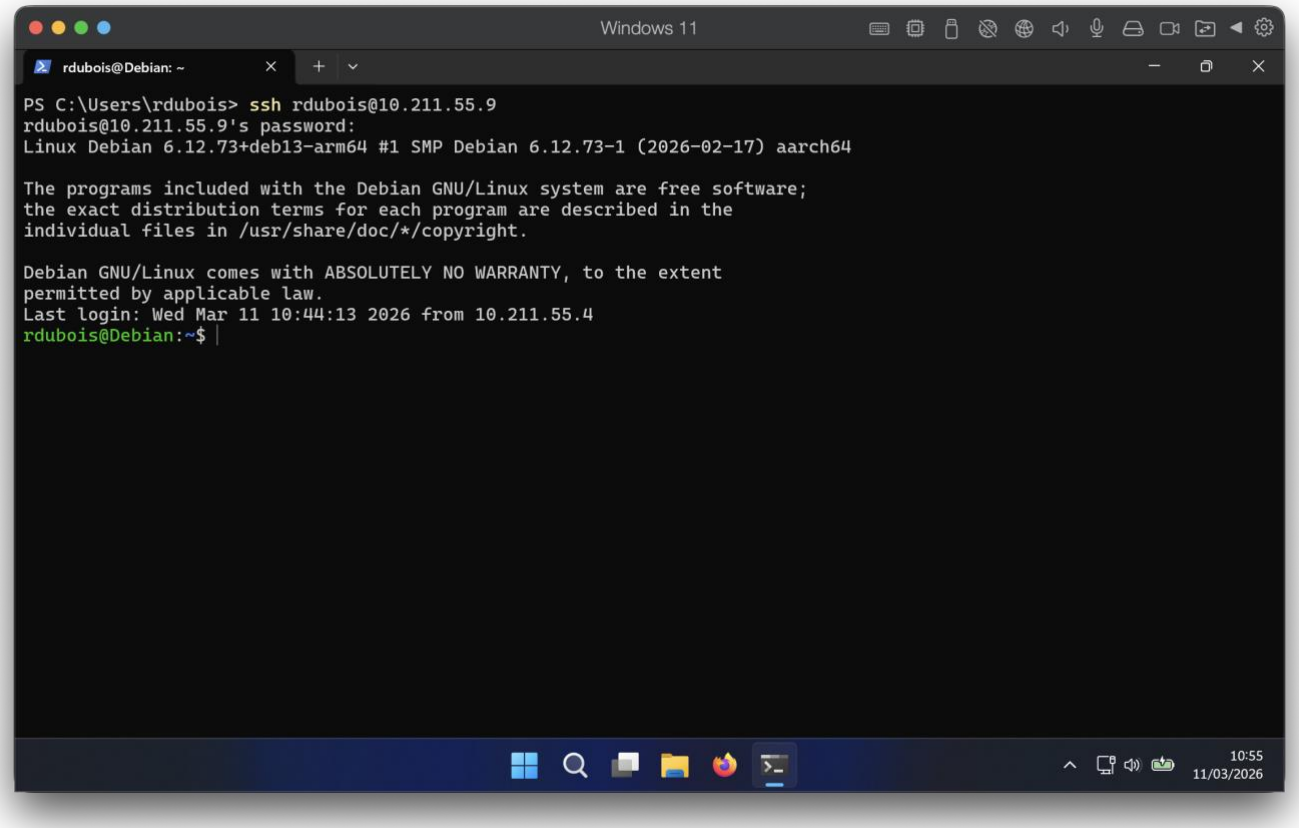
Nous allons maintenant créer une nouvelle règle détectant les connexions SSH entrante, voici la règle qui a été créer en amont :

```
alert tcp any any -> any 22 (msg:"TEST: Quelqu'un tente de se connecter en SSH !"; sid:1000003; rev:1;)
```

Ici, nous allons regarder les requêtes TCP arrivant sur le port 22 (SSH)

⚠ Le SID doit être modifié pour ne pas être le même que la règle précédente !
Sinon Snort ne pourra pas démarrer.

Maintenant nous allons relancer Snort et tenter de se connecter en SSH à notre VM :



```
++ [0] enp0s5
03/11-10:55:30.688465 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.688933 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.690562 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.696024 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.697742 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.702721 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.744054 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.744364 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:30.785695 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:33.735778 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:33.819484 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:33.824320 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:33.886019 [**] [1:1000003:1] "TEST: Quelqu'un tente de se connecter en SSH !" [**] [Priority: 0] {TCP} 10.211.55.4:55550 -> 10.211.55.9:22
03/11-10:55:41.718721 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} fe80::21c:42ff:fe00:18 -> ff02::1
03/11-10:55:41.731936 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} fe80::21c:42ff:fe60:a100 -> ff02::16
03/11-10:55:42.363792 [**] [1:1000002:1] "TEST: Quelqu'un ping cette machine !" [**] [Priority: 0] {ICMP} fe80::21c:42ff:fe60:a100 -> ff02::16
```

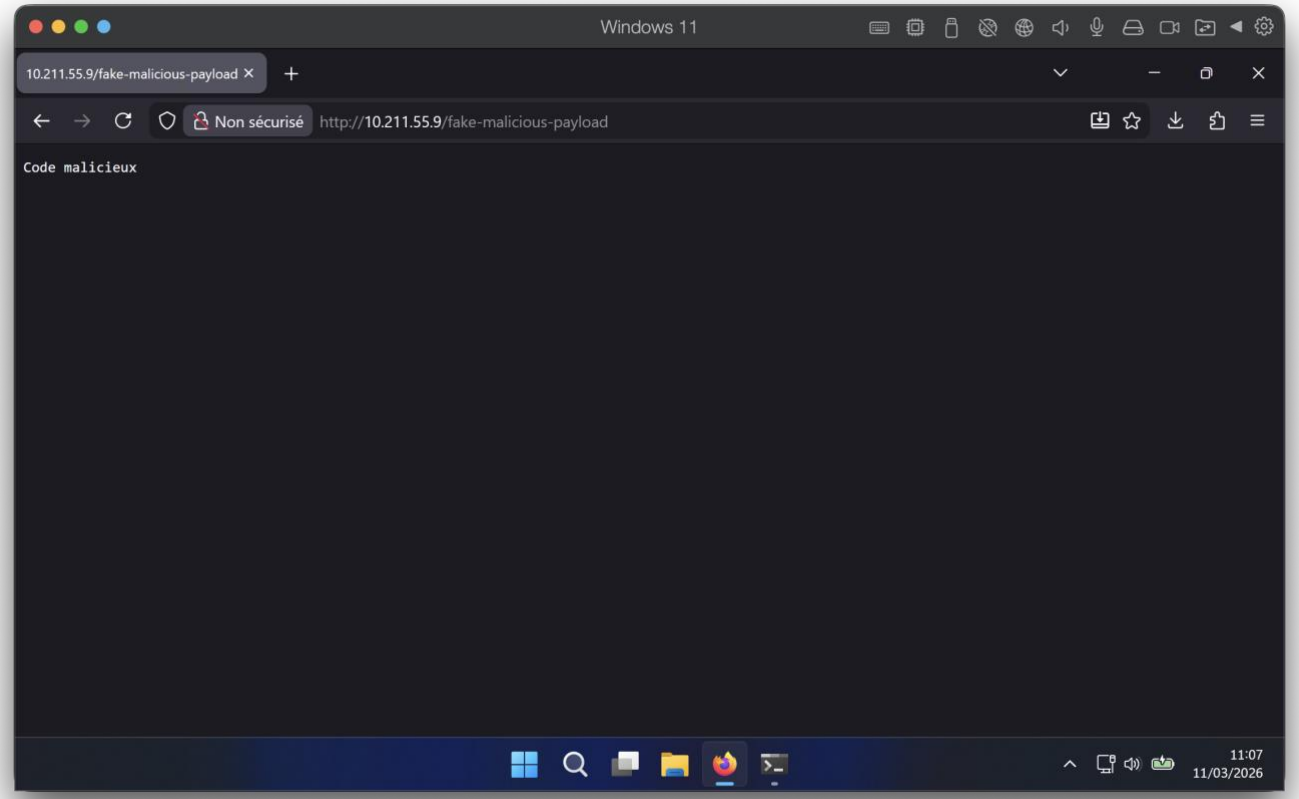
Snort arrive donc bien à détecter les connexions SSH entrante !

5. Création d'une règle HTTP

Maintenant, nous allons créer une règle qui détecte si un utilisateur se rend sur la page « fake-malicious-payload » de notre serveur web. Voici la règle qui a été créé en amont :

```
alert tcp any any -> any any (msg:"TEST: Attaque sur le serveur web !"; content:"GET /fake-malicious-payload"; sid:1000004; rev:1;)
```

Cette fois ci, nous allons chercher les paquets avec comme contenu « GET /fake-malicious-payload », ce qui nous permettra de détecter si quelqu'un tente de s'y rendre (la page **fake-malicious-payload** doit être créer sur le serveur Apache pour que cela fonctionne) :




```
++ [0] enp0s5
03/11-11:01:27.010748 [**] [116:6:1] "(ipv4) IPv4 datagram length > captured length" [**] [Priority: 3] {eth}  ->
03/11-11:01:27.039234 [**] [116:6:1] "(ipv4) IPv4 datagram length > captured length" [**] [Priority: 3] {eth}  ->
03/11-11:01:34.827484 [**] [1:1000004:1] "TEST: Attaque sur le serveur web !" [**] [Priority: 0] {TCP} 10.211.55.4:55571 -> 10.211.55.9:80
03/11-11:02:27.282574 [**] [1:1000004:1] "TEST: Attaque sur le serveur web !" [**] [Priority: 0] {TCP} 10.211.55.4:55580 -> 10.211.55.9:80
03/11-11:03:29.604034 [**] [1:1000004:1] "TEST: Attaque sur le serveur web !" [**] [Priority: 0] {TCP} 10.211.55.4:55582 -> 10.211.55.9:80
03/11-11:03:36.502556 [**] [1:1000004:1] "TEST: Attaque sur le serveur web !" [**] [Priority: 0] {TCP} 10.211.55.4:55583 -> 10.211.55.9:80
```

Comme nous pouvons l'observer, notre fausse attaque fonctionne et est détectée par Snort !

6. Blocage d'une requête après une tentative de spam ICMP

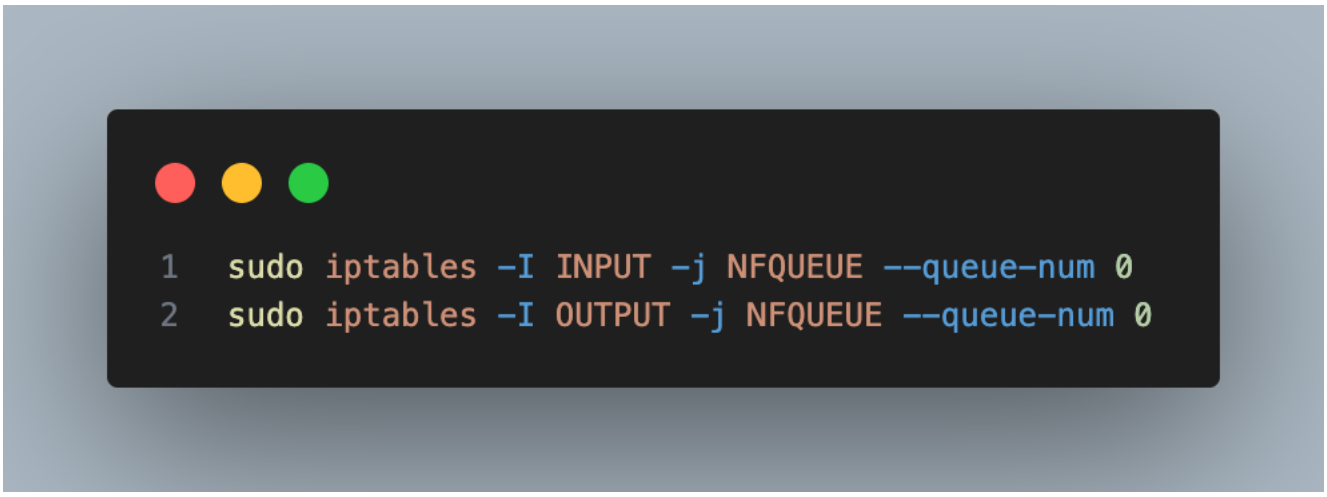
A présent que nous savons faire des alertes dans Snort, nous allons faire en sorte que Snort passe du statut d'**IDS** (système de détection d'intrusion) à celui d'**IPS** (système de prévention d'intrusion).

Pour cela, nous allons devoir rediriger le trafic réseau vers Snort grâce à iptables (un pare feu disponible sur Linux).

Pour cela, nous allons installer ce dernier avec la commande **sudo apt install iptables** :

```
root@Debian:~# sudo apt install iptables
Installation de :
  iptables

Installation de dépendances :
  libip4tc2  libip6tc2

Paquets suggérés :
  firewalld

Sommaire :
  Mise à niveau de : 0. Installation de : 3Supprimé : 0. Non mis à jour : 0
Taille du téléchargement : 393 kB
  Espace nécessaire : 8 841 kB / 53,3 GB disponible

Continuer ? [O/n]
Réception de : 1 http://deb.debian.org/debian trixie/main arm64 libip4tc2 arm64 1.8.11-2 [19,6 kB]
Réception de : 2 http://deb.debian.org/debian trixie/main arm64 libip6tc2 arm64 1.8.11-2 [19,8 kB]
Réception de : 3 http://deb.debian.org/debian trixie/main arm64 iptables arm64 1.8.11-2 [354 kB]
393 ko réceptionnés en 1s (262 ko/s)
Sélection du paquet libip4tc2:arm64 précédemment désélectionné.
(Lecture de la base de données... 158438 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de .../libip4tc2_1.8.11-2_arm64.deb ...
Dépaquetage de libip4tc2:arm64 (1.8.11-2) ...
Sélection du paquet libip6tc2:arm64 précédemment désélectionné.
Préparation du dépaquetage de .../libip6tc2_1.8.11-2_arm64.deb ...
Dépaquetage de libip6tc2:arm64 (1.8.11-2) ...
Sélection du paquet iptables précédemment désélectionné.
Préparation du dépaquetage de .../iptables_1.8.11-2_arm64.deb ...
Dépaquetage de iptables (1.8.11-2) ...
Paramétrage de libip4tc2:arm64 (1.8.11-2) ...
Paramétrage de libip6tc2:arm64 (1.8.11-2) ...
Paramétrage de iptables (1.8.11-2) ...
update-alternatives: utilisation de « /usr/sbin/iptables-legacy » pour fournir « /usr/sbin/iptables » (iptables) en mode automatique
update-alternatives: utilisation de « /usr/sbin/ip6tables-legacy » pour fournir « /usr/sbin/ip6tables » (ip6tables) en mode automatique
update-alternatives: utilisation de « /usr/sbin/iptables-nft » pour fournir « /usr/sbin/iptables » (iptables) en mode automatique
update-alternatives: utilisation de « /usr/sbin/ip6tables-nft » pour fournir « /usr/sbin/ip6tables » (ip6tables) en mode automatique
update-alternatives: utilisation de « /usr/sbin/arptables-nft » pour fournir « /usr/sbin/arptables » (arptables) en mode automatique
update-alternatives: utilisation de « /usr/sbin/ebtables-nft » pour fournir « /usr/sbin/ebtables » (ebtables) en mode automatique
Traitement des actions différées (« triggers ») pour man-db (2.13.1-1) ...
Traitement des actions différées (« triggers ») pour libc-bin (2.41-12+deb13u1) ...
root@Debian:~#
```

Une fois installé, nous allons ajouter une règle pour rediriger tout le trafic réseau vers Snort :



- **-I (Insert / Insérer)** : C'est crucial. Cela dit au pare-feu d'insérer cette règle tout en haut de la liste. Nous voulons que le pare-feu traite cette règle avant de regarder quoi que ce soit d'autre.
- **INPUT / OUTPUT (Entrée / Sortie)** : Cela indique au pare-feu quel flux de trafic surveiller. INPUT s'applique au trafic qui entre sur votre serveur (comme le ping d'un attaquant). OUTPUT s'applique au trafic qui sort de votre serveur.
- **-j NFQUEUE (Jump to Netfilter Queue / Sauter vers la file d'attente)** : C'est ici qu'est la magie. Au lieu de dire au videur d'ACCEPT (accepter) ou de DROP (jeter) le paquet, -j NFQUEUE lui dit : "Mets ce paquet en pause, place-le dans une salle d'attente, et attends qu'un programme externe (en l'occurrence Snort) te dise quoi en faire."
- **--queue-num 0** : Cela donne simplement un numéro à cette salle d'attente. Nous mettons le trafic réseau en pause dans la salle n°0.

Une fois cette commande entré dans notre firewall, nous n'avons plus de réseau, c'est normal étant donné que ce dernier attend Snort pour pouvoir effectuer des requêtes.

Avant de le relancer, nous allons ajouter une nouvelle règle, la voici :

```
drop icmp any any -> any any (msg:"TES: Attaque de ping bloquée !"; detection_filter:track by_src, count 30, seconds 60; sid:1000005; rev:1;)
```

La différence majeure comparé aux règles précédentes et le remplacement de « alert » par « drop » qui signifie abandonner. Au bout de 30 requêtes ICMP, Snort va abandonner les requêtes pendant 60 secondes.

Une fois notre règle ajoutée, nous allons démarrer Snort en rajoutant quelques arguments :



Voici les nouveaux arguments qui ont été ajoutée à notre commande :

- -Q (Mode Inline / En ligne) C'est ce qui transforme Snort d'un IDS (détecteur) en un IPS (protecteur).
  Ce qu'il fait : Il ordonne à Snort de s'exécuter en mode "Inline" (en ligne). Sans ce paramètre, Snort se contente de lire une copie du trafic réseau sans jamais interférer. Avec -Q, Snort se place physiquement au milieu du flux : chaque paquet réseau doit obligatoirement passer entre ses mains et recevoir son approbation avant de continuer son chemin. C'est ce qui lui donne le pouvoir d'exécuter l'action drop (bloquer).
- --daq nfq (Choix du module d'Acquisition de Données) DAQ signifie Data Acquisition (Acquisition de données). C'est la bibliothèque que Snort utilise pour savoir comment capturer le trafic.
  Ce qu'il fait : Lors de nos premiers tests, nous utilisions -i enp0s1, ce qui disait à Snort d'utiliser le module par défaut (pcap) pour écouter passivement la carte réseau. Ici, en utilisant --daq nfq, nous disons à Snort : "Arrête d'écouter la carte réseau. Utilise le module Netfilter Queue et va chercher les paquets directement dans le pare-feu Linux / **iptables** que nous avons installés"
- --daq-var queue=0 (Paramétrage de la file d'attente)
  C'est une variable (un réglage supplémentaire) que l'on donne spécifiquement au module DAQ que l'on vient de choisir.
  Ce qu'il fait : Puisque nous venons de dire à Snort d'utiliser nfq pour aller chercher les paquets dans le pare-feu, il doit savoir exactement où les trouver. --daq-var queue=0 lui indique le numéro de la "salle d'attente". Cela correspond exactement au --queue-num 0 que nous avions mis dans nos commandes **iptables**.

Maintenant nous pouvons lancer Snort avec la commande que nous avons faite, et puis faire nos ping vers la machine Snort :

Vidéo de démonstration :

https://drive.proton.me/urls/C5YSCM59C8#MARib6M3TZMO