5 Steps to Set Up a Fast, Secure NGINX Reverse Proxy Server

Written by Dan Ford at https://www.dlford.io/nginx-naxsi-http2-tls13-pagespeed/ but copying here for local reference
Learn how to set up a fast and secure NGINX reverse proxy server with NAXSI and PageSpeed plugins using secure TLSv1.3 and HTTP2 protocols

Update (October 6th, 2019): Added Brotli compression.

This guide assumes you are using Ubuntu 18.04, if you are not you will need to hunt down some package names.

In this tutorial, you will install NGINX with NAXSI web application firewall, and the PageSpeed and HTTP2 plugins, set up Email alerts when new versions of NGINX or NAXSI are released, and configure it as a fast and secure reverse proxy server for hosting and optimizing multiple web sites behind a single IP address.

What is NAXSI

NAXSI is a web application firewall for protecting web endpoints from malicious requests, it employs a whitelist what you need and block the rest methodology. Security should be addressed as a layered approach, NAXSI is a solid defensive layer against XSS, SQL injection, and some other nasty things bad actors can attempt, but be advised it cannot defend against every vulnerability, no single tool really can.

> NAXSI means Nginx Anti XSS & SQL Injection.
>
> Technically, it is a third party nginx module, available as a package for many UNIX-like platforms. This module, by default, reads a small subset of simple (and readable) rules containing 99% of known patterns involved in website

> vulnerabilities. For example, <, | or drop are not supposed to be part of a URI.
>
> Being very simple, those patterns may match legitimate queries, it is the Naxsi's administrator duty to add specific rules that will whitelist legitimate behaviours. The administrator can either add whitelists manually by analyzing nginx's error log, or (recommended) start the project with an intensive auto-learning phase that will automatically generate whitelisting rules regarding a website's behaviour.
>
> In short, Naxsi behaves like a DROP-by-default firewall, the only task is to add required ACCEPT rules for the target website to work properly. (Source: https://github.com/nbs-system/naxsi)

Why build from source

The packaged version doesn't include PageSpeed plugin or NAXSI WAF, if you don't mind foregoing the little speed boost you get from the PageSpeed plugin optimizations and the security layer you get from NAXSI, there's nothing wrong with using the packaged version. Just grab the NGINX configuration file and template files below, and remove any reference to NAXSI or PageSpeed and you will have a relatively secure reverse proxy server with HTTP2, *I'm not sure if you will be able to use TLSv1.3 with the packaged version or not.*

Step 1 - Build NGINX from source

First let's install a slew of dependancies for building from source and the PageSpeed plugin, and certbot for obtaining free SSL certificates from LetsEncrypt.

```
apt install -y libpcre3 libpcre3-dev libssl-dev \
unzip make libgoogle-perftools-dev google-perftools \
jq gcc git zlib1g zlib1g-dev build-essential uuid-dev \
certbot git
```

You may need to run the command `apt-add-repository universe` *if you get errors here, some versions of Ubuntu do not include the* `universe` *repository by default and that is where* `jq` *and* `certbot` *are sourced.*

Then install the packaged version of NGINX, and tell `apt` it is not to install any updates, since that would be a downgrade from our source build. Installing the packaged version first will set up the service in `systemd` so we don't have to worry about starting NGINX at boot time, and sets up a few

other default files.

```
apt install -y nginx
apt-mark hold nginx-core
apt-mark hold nginx-common
```

Check the installed version with the command `nginx -v`, in my case I had version `1.14.0` installed.

We have to remove all of the configuration files since they are likely not compatible with the newer version of NGINX.

```
rm -rf /etc/nginx/*
```

Now create the build script to install NGINX from source with all the extra goodies.

```
touch /usr/local/bin/build-nginx-naxsi.sh
chmod +x /usr/local/bin/build-nginx-naxsi.sh
```

First we need to check check the OpenSSL version on your system, if you are on Ubuntu 18.04 you should have 1.1.1 already, TLSv1.3 is only available with OpenSSL version 1.1.1 or higher.

```
openssl version
```

Copy and paste the appropriate script below into the new file depending on whether or not you have OpenSSL version 1.1.1 or greater, or lower than 1.1.1. If you don't have version 1.1.1 or greater, the second script will download the source for OpenSSL and reference that when building NGINX.

This script will check for the newest versions and re-build every time it is run, so you'll want to keep it on the system for future updates.

/usr/local/bin/build-nginx-naxsi.sh (OpenSSL 1.1.1 or greater)

```
#!/usr/bin/env bash


# move previous build to backup if it exists, overwriting previous backup
if [[ -d "/usr/local/src/nginx" ]]; then
  if [[ -d "/usr/local/src/nginx-old" ]]; then
    rm -rf /usr/local/src/nginx-old
```

```
  fi
  mv /usr/local/src/nginx /usr/local/src/nginx-old
fi
# delete previous nginx config backup if it exists
if [[ -d "/usr/local/src/nginx-conf" ]]; then
  rm -rf /usr/local/src/nginx-conf
fi
# backup nginx config just in case
mkdir /usr/local/src/nginx-conf
cp -a /etc/nginx/* /usr/local/src/nginx-conf/
# create new build directory and cd to it
mkdir /usr/local/src/nginx
cd /usr/local/src/nginx
# get versions
latestNginx=$(curl -s http://hg.nginx.org/nginx/atom-tags |
grep "<title>release-" | sort --version-sort | tail -1 |
sed 's/<title>release-//g' | sed 's/<\/title>//g' | sed 's/^ *//g')
latestNaxsi=$(curl -s https://api.github.com/repos/nbs-system/naxsi/releases |
  jq -r .[].tag_name | grep -v rc | head -1)
latestPagespeed=$(curl -s https://api.github.com/repos/apache/incubator-pagespeed-ngx/tags |
  jq -r .[].name | grep stable | head -1)
# get source files for pagespeed nginx, naxsi, and brotli
wget http://nginx.org/download/nginx-`echo $latestNginx`.tar.gz
wget https://github.com/nbs-system/naxsi/archive/${latestNaxsi}.tar.gz
wget https://github.com/apache/incubator-pagespeed-ngx/archive/${latestPagespeed}.tar.gz
git clone --recursive https://github.com/google/ngx_brotli.git
tar xzf nginx-`echo $latestNginx`.tar.gz
tar xzf ${latestNaxsi}.tar.gz
tar xzf ${latestPagespeed}.tar.gz
# prepare pagespeed
nps_dir=$(find . -name "*pagespeed-ngx-*" -type d)
cd "$nps_dir"
NPS_RELEASE_NUMBER=${latestPagespeed/beta/}
NPS_RELEASE_NUMBER=${latestPagespeed/stable/}
psol_url=https://dl.google.com/dl/page-speed/psol/${NPS_RELEASE_NUMBER}.tar.gz
[ -e scripts/format_binary_url.sh ] && psol_url=$(scripts/format_binary_url.sh PSOL_BINARY_URL)
wget ${psol_url}
tar xzf $(basename ${psol_url})
# build and install
```

```
cd /usr/local/src/nginx
cd nginx-`echo $latestNginx`
./configure --conf-path=/etc/nginx/nginx.conf \
  --add-module=../naxsi-${latestNaxsi}/naxsi_src/ \
  --add-module=../$nps_dir \
  --add-module=../ngx_brotli \
  --error-log-path=/var/log/nginx/error.log \
  --http-client-body-temp-path=/var/lib/nginx/body \
  --http-fastcgi-temp-path=/var/lib/nginx/fastcgi \
  --http-log-path=/var/log/nginx/access.log \
  --http-proxy-temp-path=/var/lib/nginx/proxy \
  --lock-path=/var/lock/nginx.lock \
  --pid-path=/var/run/nginx.pid \
  --with-http_ssl_module \
  --with-http_v2_module \
  --with-stream \
  --with-stream_realip_module \
  --with-stream_ssl_module \
  --without-mail_pop3_module \
  --without-mail_smtp_module \
  --without-mail_imap_module \
  --without-http_uwsgi_module \
  --without-http_scgi_module \
  --prefix=/usr
make
make install
# backup naxsi core rules and download latest core rules
cd /etc/nginx
mv /etc/nginx/naxsi_core.rules /etc/nginx/naxsi_core.rules.bak
wget -q https://raw.githubusercontent.com/nbs-system/naxsi/master/naxsi_config/naxsi_core.rules
# do nginx config test and restart nginx if passed
check="$(/usr/sbin/nginx -t 2>&1 | grep success | sed 's/.*conf //g')"
if [[ $check == "test is successful" ]];then
  systemctl restart nginx
  sleep 5
  systemctl status nginx
else
  echo "nginx config test failed!!!"
fi
```

```
exit 0
```

/usr/local/bin/build-nginx-naxsi.sh (OpenSSL version lower than 1.1.1)

```
#!/usr/bin/env bash


# move previous build to backup if it exists, overwriting previous backup
if [[ -d "/usr/local/src/nginx" ]]; then
  if [[ -d "/usr/local/src/nginx-old" ]]; then
    rm -rf /usr/local/src/nginx-old
  fi
  mv /usr/local/src/nginx /usr/local/src/nginx-old
fi
# delete previous nginx config backup if it exists
if [[ -d "/usr/local/src/nginx-conf" ]]; then
  rm -rf /usr/local/src/nginx-conf
fi
# backup nginx config just in case
mkdir /usr/local/src/nginx-conf
cp -a /etc/nginx/* /usr/local/src/nginx-conf/
# create new build directory and cd to it
mkdir /usr/local/src/nginx
cd /usr/local/src/nginx
# # get source for openssl 1.1.1 (tls 1.3 compatibility)
git clone https://github.com/openssl/openssl.git
cd openssl
git checkout OpenSSL_1_1_1-stable
cd /usr/local/src/nginx
# get versions
latestNginx=$(curl -s http://hg.nginx.org/nginx/atom-tags |
grep "<title>release-" | sort --version-sort | tail -1 |
sed 's/<title>release-//g' | sed 's/<\/title>//g' | sed 's/^ *//g')
latestNaxsi=$(curl -s https://api.github.com/repos/nbs-system/naxsi/releases |
  jq -r .[].tag_name | grep -v rc | head -1)
latestPagespeed=$(curl -s https://api.github.com/repos/apache/incubator-pagespeed-ngx/tags |
  jq -r .[].name | grep stable | head -1)
# get source files for pagespeed nginx, naxsi, and brotli
```

```
wget http://nginx.org/download/nginx-`echo $latestNginx`.tar.gz
wget https://github.com/nbs-system/naxsi/archive/${latestNaxsi}.tar.gz
wget https://github.com/apache/incubator-pagespeed-ngx/archive/${latestPagespeed}.tar.gz
git clone --recursive https://github.com/google/ngx_brotli.git
tar xzf nginx-`echo $latestNginx`.tar.gz
tar xzf ${latestNaxsi}.tar.gz
tar xzf ${latestPagespeed}.tar.gz
# prepare pagespeed
nps_dir=$(find . -name "*pagespeed-ngx-*" -type d)
cd "$nps_dir"
NPS_RELEASE_NUMBER=${latestPagespeed/beta/}
NPS_RELEASE_NUMBER=${latestPagespeed/stable/}
psol_url=https://dl.google.com/dl/page-speed/psol/${NPS_RELEASE_NUMBER}.tar.gz
[ -e scripts/format_binary_url.sh ] && psol_url=$(scripts/format_binary_url.sh PSOL_BINARY_URL)
wget ${psol_url}
tar xzf $(basename ${psol_url})
# build and install
cd /usr/local/src/nginx
cd nginx-`echo $latestNginx`
./configure --conf-path=/etc/nginx/nginx.conf \
  --add-module=../naxsi-${latestNaxsi}/naxsi_src/ \
  --add-module=../$nps_dir \
  --add-module=../ngx_brotli \
  --error-log-path=/var/log/nginx/error.log \
  --http-client-body-temp-path=/var/lib/nginx/body \
  --http-fastcgi-temp-path=/var/lib/nginx/fastcgi \
  --http-log-path=/var/log/nginx/access.log \
  --http-proxy-temp-path=/var/lib/nginx/proxy \
  --lock-path=/var/lock/nginx.lock \
  --pid-path=/var/run/nginx.pid \
  --with-http_ssl_module \
  --with-http_v2_module \
  --with-stream \
  --with-stream_realip_module \
  --with-stream_ssl_module \
  --without-mail_pop3_module \
  --without-mail_smtp_module \
  --without-mail_imap_module \
  --without-http_uwsgi_module \
  --without-http_scgi_module \
```

```
  --prefix=/usr \
  --with-openssl=/usr/local/src/nginx/openssl
make
make install
# backup naxsi core rules and download latest core rules
cd /etc/nginx
mv /etc/nginx/naxsi_core.rules /etc/nginx/naxsi_core.rules.bak
wget -q https://raw.githubusercontent.com/nbs-system/naxsi/master/naxsi_config/naxsi_core.rules
# do nginx config test and restart nginx if passed
check="$(/usr/sbin/nginx -t 2>&1 | grep success | sed 's/.*conf //g')"
if [[ $check == "test is successful" ]];then
  systemctl restart nginx
  sleep 5
  systemctl status nginx
else
  echo "nginx config test failed!!!"
fi
exit 0
```

Now run the script (`/usr/local/bin/build-nginx-naxsi.sh`), and watch it go. You can re-check the version now and it should be newer with the same command `nginx -v`, in my case I'm now running `1.17.1`.

Potential Build Error and Solution

If you run into errors like the following:

```
../naxsi-0.56/naxsi_src/naxsi_runtime.c:728:5: error: 'strncat' specified bound 1 equals source length [-
Werror=stringop-overflow=]
     strncat((char*)tmp_hashname.data, "#", 1);
     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
../naxsi-0.56/naxsi_src/naxsi_runtime.c:731:3: error: 'strncat' specified bound 1 equals source length [-
Werror=stringop-overflow=]
   strncat((char*)tmp_hashname.data, "#", 1);
   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cc1: all warnings being treated as errors
```

This is caused by warnings being treated as errors in the GCC compiler, I beleive this behavior change was added in GCC version 8 (use the command `gcc -v` to check). You can disable this behavior when building NGINX by adding the line `CFLAGS="-Wno-stringop-truncation -Wno-stringop-overflow -Wno-size-of-pointer-memaccess" \` right before the `./configure ...` lines, it should look something

like this:

```
...
# build and install
cd nginx-`echo $latestNginx`
CFLAGS="-Wno-stringop-truncation -Wno-stringop-overflow -Wno-size-of-pointer-memaccess" \
./configure --conf-path=/etc/nginx/nginx.conf \
  --add-module=../naxsi-${latestNaxsi}/naxsi_src \
...
```

Step 2 - Software update Email alerts

It is critically important that you stay on top of updates being as we are running the latest stable version, most OS packages are a few versions behind to ensure all the issues have been worked out. If a vulnerability is found, you'll want to patch the day of! Before proceeding you should set up Postfix on your server to relay email alerts to you, and install `mailutils` *for Email submission.*

If you don't wish to do this just skip to the next step, I do recommend you sign up for the NGINX Announce mailing list instead so you can still re-run the installer script when new versions are released.

Let's create a script to check the latest versions of NGINX and NAXSI, and send out an Email if they are newer than what is installed.

```
touch /usr/local/bin/nginx-update-notifier.sh
chmod +x /usr/local/bin/nginx-update-notifier.sh
```

Copy and paste the script below into the file

don't forget to change the Email address at the `emailTo:` *line in the script.*

/usr/local/bin/nginx-update-notifier.sh

```
#!/bin/bash


emailTo="you@yourdomain.com"
```

```
latestNginx=$(curl -s http://hg.nginx.org/nginx/atom-tags |
grep "<title>release-" | sort --version-sort | tail -1 |
sed 's/<title>release-//g' | sed 's/<\/title>//g' | sed 's/^ *//g')
latestNaxsi=$(curl -s https://api.github.com/repos/nbs-system/naxsi/releases | \
  jq -r .[].tag_name | grep -v rc | head -1)
currentNginx="$(/usr/sbin/nginx -v 2>&1 | sed 's/.*nginx\///g')"
currentNaxsi="$(/usr/sbin/nginx -V 2>&1 | grep 'naxsi' | \
  sed 's/.*naxsi-//g' | sed 's/\/.*//g')"
versionCheck="$(printf "$latestNginx\n$currentNginx" | sort -V | tail -n1)"
if [[ "$versionCheck" != "$currentNginx" ]] ||
  [[ "$latestNaxsi" != "$currentNaxsi" ]]; then
  echo -e \
  "Use build-nginx-naxsi.sh to update:
  NGINX: $currentNginx -> $latestNginx
  NAXSI: $currentNaxsi -> $latestNaxsi" | \
  mail -s "NGINX/NAXSI Update Available" $emailTo
else
  echo -e \
  "NGINX and NAXSI are up to date:
  NGINX: $currentNginx -> $latestNginx
  NAXSI: $currentNaxsi -> $latestNaxsi"
fi

exit 0
```

Now create the two following files to schedule this script as a `systemd` timer, or use cron if you prefer.

/etc/systemd/system/nginx-update-notifier.service

```
[Unit]
Description=Check for nginx and naxsi updates

[Service]
Type=oneshot
ExecStart=/usr/local/bin/nginx-update-notifier.sh
```

/etc/systemd/system/nginx-update-notifier.timer

```
[Unit]
Description=Check for nginx and naxsi updates

[Timer]
# Check daily at 5am
OnCalendar=*-*-* 05:00:00
Persistent=true
Unit=nginx-update-notifier.service

[Install]
WantedBy=timers.target
```

To enable the timer, we just need to run a few commands.

```
systemctl enable nginx-update-notifier.timer
systemctl start nginx-update-notifier.timer
```

Step 3 - Boilerplate configuration

We need a cache directory for the PageSpeed plugin to use, and some directories for site configurations and NAXSI configurations. I really think Apache had it right with the `sites-available` and `sites-enabled` format, so we will set that up.

```
mkdir /etc/nginx/sites-available
mkdir /etc/nginx/sites-enabled
mkdir /etc/nginx/snippets
mkdir /etc/nginx/naxsi-rules
mkdir -p /var/cache/pagespeedcache
```

The PageSpeed plugin works best when the cache directory is on tmpfs (stored in RAM instead of hard disk), so we will make `/var/cache/pagespeedcache` a tmpfs directory by declaring it as such in `/etc/fstab`, add the following to the bottom of the file.

/etc/fstab

Be mindful of the amount of space you give here at `size=512M`*, I would say no more than 1/4 of your available RAM, my server has 2G of RAM so I used 512M. If you can't allocate at least 64M to the tmpfs, just skip this step and leave it as a regular directory on the hard disk.*

```
# tmpfs ramdisk for PageSpeed cache
tmpfs /var/cache/pagespeedcache tmpfs defaults,noatime,nosuid,nodev,noexec,mode=1777,size=512M 0 0
```

No we need to mount the tmpfs and set permissions.

```
mount -a
chown www-data. /var/cache/pagespeedcache
```

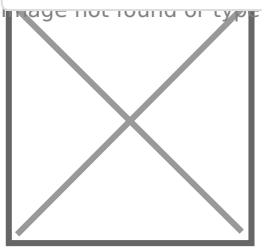
Let's make a quick and easy 403 page, this is what will be shown for requests that are blocked by NAXSI, you can spruce this up however you like later.

/var/www/html/403.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>403</title>
  </head>
  <body>
    Error 403: Access Denied
  </body>
</html>
```

We also need to generate Diffie-Hellman parameters, without getting into too much jargon, these are used in key exchanges and should be unique to your specific server for security, you can read more here if you are interested.

```
openssl dhparam -out /etc/ssl/certs/dhparam.pem 4096
```



Further Reading: This process will take a few minutes, while you are waiting I recommend taking a look at this book because it gives a very in-depth view of all the advanced features and functionality that NGINX has to offer: NGINX Cookbook, by Tim Butler

NGINX Cookbook: Over 70 recipes for real-world configuration, deployment, and performance

$48.99 on Amazon

And a dummy self-signed certificate that we will temporarily use to get signed certificates from LetsEncrypt.

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365000 -nodes
mv key.pem /etc/ssl/private/ssl-cert-snakeoil.key
mv cert.pem /etc/ssl/certs/ssl-cert-snakeoil.pem
```

Now let's put together some configuration files for common settings, create the following files.

/etc/nginx/snippets/limit-request-methods.conf

```
# Only allow common method : GET, POST and HEAD
# (HEAD is implicitly allowed with GET)
limit_except GET POST {
    deny all;
}
```

/etc/nginx/snippets/pagespeed.conf

```
# Needs to exist and be writable by nginx.
pagespeed FileCachePath /var/ngx_pagespeed_cache;

# Lazyload images
pagespeed EnableFilters lazyload_images;

# Auto convert images
pagespeed EnableFilters rewrite_images;
pagespeed EnableFilters convert_png_to_jpeg;
pagespeed EnableFilters convert_jpeg_to_webp;
```

```
pagespeed EnableFilters convert_to_webp_lossless;

# Ensure requests for pagespeed optimized resources go to the pagespeed handler
# and no extraneous headers get set.
location ~ "\.pagespeed\.([a-z]\.)?[a-z]{2}\.[^.]{10}\.[^.]+" {
  add_header "" "";
}
location ~ "^/pagespeed_static/" { }
location ~ "^/ngx_pagespeed_beacon$" { }
```

/etc/nginx/snippets/ssl-params.conf

Be warned that the `Strict-Transport-Security` header will cause you major problems if you remove SSL/TLS from any proxied site later, you should probably comment that line out for now and enable it later after you are sure everything is running smoothly.

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers 'TLS13-AES-256-GCM-SHA384:TLS-CHACHA20-POLY1305-SHA256:TLS-AES-256-GCM-SHA384:TLS-AES-128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256';
ssl_prefer_server_ciphers on;
ssl_ecdh_curve secp384r1;
ssl_session_cache shared:SSL:50m;
ssl_session_timeout 1d;
ssl_session_tickets off;
ssl_stapling on;
ssl_stapling_verify on;
resolver 8.8.8.8 8.8.4.4 valid=300s;
resolver_timeout 5s;
add_header Strict-Transport-Security "max-age=31536000; includeSubdomains" always;
add_header X-Frame-Options SAMEORIGIN;
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection "1; mode=block";
ssl_dhparam /etc/ssl/certs/dhparam.pem;
```

Note: For a more compatible set of SSL protocols while maintaining a decent level of security, replace the first two lines of the `ssl-params.conf` file with the following.

```
ssl_protocols SSLv3 TLSv1.1 TLSv1.2 TLSv1.3;
ssl_ciphers 'TLS13-AES-256-GCM-SHA384:TLS-CHACHA20-POLY1305-SHA256:TLS-AES-256-GCM-SHA384:TLS-AES-128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH+aRSA+RC4:EECDH:EDH+aRSA:RC4:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!CAMELLIA';
```

/etc/nginx/naxsi-rules/template

```
LearningMode;
SecRulesEnabled;
DeniedUrl "/403.html";

## check rules
CheckRule "$SQL >= 4" BLOCK;
CheckRule "$RFI >= 4" BLOCK;
CheckRule "$TRAVERSAL >= 2" BLOCK;
CheckRule "$EVADE >= 2" BLOCK;
CheckRule "$XSS >= 4" BLOCK;

## white list
# e.g. BasicRule wl:1007 "mz:$URL:/index.html|$BODY_VAR:message|URL";
```

/etc/nginx/sites-available/template

```
# redirect http traffic to https
server {
    listen 80;
    server_name SERVERNAME;
    return 301 https://SERVERNAME$request_uri;
}

# https reverse proxy
server {
```

```
    listen 443 ssl http2;
    server_name SERVERNAME;
    ssl_certificate /etc/ssl/certs/ssl-cert-snakeoil.pem;
    ssl_certificate_key /etc/ssl/private/ssl-cert-snakeoil.key;
    # ssl_certificate /etc/letsencrypt/live/SERVERNAME/fullchain.pem;
    # ssl_certificate_key /etc/letsencrypt/live/SERVERNAME/privkey.pem;
    include /etc/nginx/snippets/ssl-params.conf;
    include /etc/nginx/snippets/pagespeed.conf;
    location / {
      include /etc/nginx/snippets/limit-request-methods.conf;
      include /etc/nginx/naxsi-rules/SERVERNAME.rules;
      proxy_hide_header X-Powered-By;
      proxy_pass_header Authorization;
      proxy_pass http://SERVERIPADDRESS;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Host $host;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_http_version 1.1;
      proxy_set_header Connection "";
      proxy_buffering off;
      client_max_body_size 0;
      proxy_read_timeout 36000s;
      proxy_redirect off;
    }

    # letsencrypt validation
    location '/.well-known/acme-challenge' {
        default_type "text/plain";
        root /var/www/html;
    }

    # 403 redirect for NAXSI rejections
    location '/403.html' {
        default_type "text/html";
        root /var/www/html;
    }
}
```

/etc/nginx/sites-available/default

```
server {
    listen 80 default_server;
    server_name _;
    return 444;
}

server {
    listen 443 ssl default_server http2;
    server_name _;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'TLS13-AES-256-GCM-SHA384:TLS-CHACHA20-POLY1305-SHA256:TLS-AES-256-GCM-SHA384:TLS-AES-128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256';
    ssl_prefer_server_ciphers on;
    ssl_ecdh_curve secp384r1;
    ssl_session_cache shared:SSL:50m;
    ssl_session_timeout 1d;
    ssl_session_tickets off;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains" always;
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
    ssl_dhparam /etc/ssl/certs/dhparam.pem;
    ssl_certificate /etc/ssl/certs/ssl-cert-snakeoil.pem;
    ssl_certificate_key /etc/ssl/private/ssl-cert-snakeoil.key;
    return 444;
}
```

Go ahead and get rid of the main configuration file with `rm /etc/nginx/nginx.conf`, and create it again with the following configuration.

/etc/nginx/nginx.conf

```
user www-data;
worker_processes auto;
```

```
pid /run/nginx.pid;

events {
	worker_connections 768;
	use epoll;
	multi_accept on;
}

http {

	##
	# Basic Settings
	##

	include /etc/nginx/naxsi_core.rules;

	sendfile on;
	tcp_nopush on;
	tcp_nodelay on;
	keepalive_timeout 65;
	keepalive_requests 100000;
	types_hash_max_size 2048;
	server_tokens off;
	include /etc/nginx/mime.types;
	default_type application/octet-stream;

	##
	# Logging Settings
	##

	# Log to file
	#access_log /var/log/nginx/access.log;
	#error_log /var/log/nginx/error.log;
	# Log to syslog
	error_log syslog:server=unix:/dev/log,facility=local7,tag=nginx,severity=error;
	access_log syslog:server=unix:/dev/log,facility=local7,tag=nginx,severity=info;

	##
	# Brotli Settings
	##
```

```
        brotli on;
        brotli_comp_level 6;
        brotli_static on;
        brotli_types text/xml image/svg+xml application/x-font-ttf image/vnd.microsoft.icon application/x-font-opentype application/json font/eot application/vnd.ms-fontobject application/javascript font/otf application/xml application/xhtml+xml text/javascript  application/x-javascript text/plain application/x-font-truetype application/xml+rss image/x-icon font/opentype text/css image/x-win-bitmap;

        ##
        # Gzip Settings
        ##

        gzip on;
        gzip_disable "msie6";
        gzip_vary on;
        gzip_proxied any;
        gzip_comp_level 6;
        gzip_buffers 16 8k;
        gzip_http_version 1.1;
        gzip_types text/plain text/css text/xml application/json application/javascript application/xml+rss application/atom+xml image/svg+xml;

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;

        ##
        # Caching
        ##

        open_file_cache max=1000 inactive=20s;
        open_file_cache_valid 30s;
        open_file_cache_min_uses 5;
        open_file_cache_errors off;
}
```

Lastly, let’s enable the default server, this will ignore all requests to the NGINX server by it’s IP address, so if you just type the public IP address into your browser it will not respond, it will only respond to requests with a valid hostname. We will enable it by making a symlink from the file in `sites-available` to the `sites-enabled` directory.

```
ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/
```

Step 4 - Setting up backend sites

Now for the easy part! Any time you wish to host a new site, just run through this checklist (remember to restart NGINX after each one):

1. Copy the template in the template in `naxsi-rules` to `nameofyoursite.rules` and the template in `sites-available` to `nameofyoursite.conf` and edit it accordingly, and symlink it to the `sites-enabled` directory
2. Get a free SSL certificate from LetsEncrypt and update the site configuration file to use it
3. Configure NAXSI rules

Let’s set up the site `example.com` with a backend at IP Address 10.5.5.5.

```
cd /etc/nginx/naxsi-rules
cp template example.com.rules
cd /etc/nginx/sites-available
cp template example.com.conf
```

Below are the lines I edited, I’ve omitted the lines that weren’t changed, note that the SSL certificate paths are still commented out, we need to use the snakeoil (self-signed) certificate for the moment.

The default settings are very restrictive, you may need to open them up a bit, you could for example modify the limit-request-methods.conf file if all sites need changing, or make a new snippet file and modify that for example `limit-request-methods-wordpress-sites.conf` and import that snippet in the main site configuration file, or just copy it’s contents to the main site configuration file instead of including the snippet file and modify them there if it’s just one site.

/etc/nginx/sites-available/example.com.conf

```
# redirect http traffic to https
server {
    ...
    server_name example.com www.example.com;
    return 301 https://www.example.com$request_uri;
}

# https reverse proxy
server {
    ...
    server_name example.com www.example.com;
    ...
    # ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    # ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
    ...
    location / {
      ...
      proxy_pass http://10.5.5.5;
      ...
    }

    ...
}
```

Now symlink the configuration file to the `sites-enabled` directory.

```
ln -s /etc/nginx/sites-available/example.com.conf /etc/nginx/sites-enabled/
```

If you ever need to take the site down, you can just delete the symlink and leave the configuration file in `sites-available` for later use or reference (e.g. `rm /etc/nginx/sites-enabled/example.com.conf`).

Great, now test the configuration with the command `nginx -t` and then restart NGINX with the command `service nginx restart` if the test passed. The NGINX server needs to be accessible from the outside internet, and have the domain name `example.com` resolve to it's public IP address before proceeding. LetsEncrypt needs to verify we own the domain before it will hand out a certificate, so when we run the next command, the `certbot` program will create a file with a unique identifier in it, and the remote LetsEncrypt server will request that file from `example.com/.well-known/acme-challenge`, if the file matches it will issue the certificate. `certbot` should automatically set up a service timer in `systemd` when it is installed to renew certificates before they expire, so you shouldn't need to worry about that.

```
certbot certonly --webroot -w /var/www/html -d example.com -d www.example.com
```

You will be asked to provide your Email address for expiration alerts, and to accept the terms of service. The renewal alerts are handy because if something ever goes wrong and renewal fails, you'll be informed before there is a real problem.

You should now have a valid SSL certificate for `example.com`, we just need to enable it in the sites configuration file by commenting out the snakeoil certificate paths and uncommenting the LetsEncrypt certificate paths.

/etc/nginx/sites-available/example.com.conf (Again)

```
# ssl_certificate /etc/ssl/certs/ssl-cert-snakeoil.pem;
# ssl_certificate_key /etc/ssl/private/ssl-cert-snakeoil.key;
ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
```

After testing and restarting NGINX again, your site should be live. Right now the NAXSI web application firewall is in learning mode, so it will not block any requests. You should visit your site, click all the buttons, fill out all the forms, and generally do all of the things you would expect a normal user to do while NAXSI collects logs about which requests it would normally block.

All that's left is to add exclusions for the NAXSI rules that triggered in learning mode, and disable learning mode. Look in your NGINX error log for lines with the text `NAXSI_FMT`, if you used the NGINX configuration file above the logs will be in your `syslog` file. There will likely be a lot, but I'll work through one example with you, I've chosen a pretty complex one for your benefit.

This will look very intimidating the first time you see it, but I promise once you work through a few it will become second nature pretty quickly.

`cat /var/log/syslog | grep 'NAXSI_FMT' | less`

```
nginx: [error] 12267#0: *127 NAXSI_FMT:
ip=23.42.23.13&server=example.com&uri=/api/v1&learning=1&vers=0.56&total_processed=25&total_blocked
=1&block=1&cscore0=$SQL&score0=48&cscore1=$XSS&score1=40&cscore2=$TRAVERSAL&score2=24&zone
0=BODY&id0=1010&var_name0=query&zone1=BODY&id1=1011&var_name1=query&zone2=BODY&id2=1015
&var_name2=query&zone3=BODY&id3=1205&var_name3=query&zone4=BODY&id4=1310&var_name4=query
&zone5=BODY&id5=1311&var_name5=query, client: 23.42.23.13, server: example.com, request: "POST /api/v1
HTTP/2.0", host: "example.com", referrer: "https://example.com/"
```

There is an example of the rule syntax in the `naxsi-rules/template` file that will be in all of the site files for easy reference, it looks like this:

```
BasicRule wl:1007 "mz:$URL:/index.html|$BODY_VAR:message|URL";
```

To break that down some we have:

- `BasicRule` - Each rule starts with this
- `wl:1007` - Whitelist rule ID 1007
- `"mz:...";` - Match zone, or what zones to target for this whitelist
- `|` - Separate multiple targets, think of it as the word “and”
- `$URL:/index.html` - Target specific URL
- `$BODY_VAR:message` - Target the variable called ‘message’ in the request body
- `URL` - Target the URL path

All of this information is in the error log, since this is a complex event there are multiple zones, so I’m only looking at zone `0` right now, we have:

- Rule ID: `id0=1010`
- URL: `uri=/api/v1`
- Match Zone: `zone0=BODY`
- Variable: `var_name0=query`

We can build our rule with that information, and it looks like this:

```
BasicRule wl:1010 "mz:$URL:/api/v1|$BODY_VAR:query";
```

If `var_name0` in the error log was blank, we wouldn’t specify the variable name `query`, and it would look like this:

```
BasicRule wl:1010 "mz:$URL:/api/v1|BODY";
```

If you were seeing the same match for all URLs, you could omit the URL in the rule, and it would whitelist the variable for any URL, like this:

```
BasicRule wl:1010 "mz:$BODY_VAR:query";
```

If the variable name was constantly changing, for example `query_83jd82w` and `query_g8dj37s` and so on, you could use a regular expression like this:

```
BasicRule wl:1010 "mz:$URL:/api/v1|$BODY_VAR_X:^query_.*$";
```

You can also use regular expressions for the URL, and finally, if the match zone is a user input that you know for sure is sanitzed on the backend site, you could whitelist all rule IDs for that zone by specifying an ID of `0`, like this:

```
BasicRule wl:0 "mz:$URL:/api/v1|$BODY_VAR:query";
```

The goal is to be specific to let through as little exceptions as possible, so keep that in mind. Now to finish up, since all of these zones are matching the same variable, URL, and portion, only the rule ID is different between them, I can whitelist them all in one rule, like this:

```
BasicRule wl:1010,1011,1015,1205,1310,1311 "mz:$URL:/api/v1|$BODY_VAR:query";
```

A more detailed explaination of the whitelisting options and syntax can be found on the NAXSI GitHub page , I recommend skimming through the whole wiki, as there are many features and options available that I haven't covered here.

I usually write out my rules in a text editor for this part, each rule on a new line. Once that is done, copy and paste all the lines into the bottom of the file `/etc/nginx/naxsi-rules/example.com.conf`, while you are there comment out the line `LearningMode` to enable NAXSI rule enforcement.

/etc/nginx/naxsi-rules/example.com.conf

```
#LearningMode;
SecRulesEnabled;
DeniedUrl "/403.html";

## check rules
CheckRule "$SQL >= 4" BLOCK;
CheckRule "$RFI >= 4" BLOCK;
CheckRule "$TRAVERSAL >= 2" BLOCK;
CheckRule "$EVADE >= 2" BLOCK;
CheckRule "$XSS >= 4" BLOCK;

## white list
# e.g. BasicRule wl:1007 "mz:$URL:/index.html|$BODY_VAR:message|URL";
BasicRule wl:1010,1011,1015,1205,1310,1311 "mz:$URL:/api/v1|$BODY_VAR:query";
BasicRule wl:111 "mz:$URL:/somepath/|$ARGS_VAR:shoes";
BasicRule wl:1050 "mz:$URL:/someotherpath/|$ARGS_VAR:boots";
...
```

Nice, you should visit your website and do all the things again to make sure you didn't miss any rules. You can visit `https://example.com/delete` in your browser to test NAXSI, if you haven't whitelisted `/delete`, you should get the 403 page we made earlier.

Step 5 - Extra security

I *highly* recommended setting up Mitchell Krogza's "Ultimate Bad Bot and Referrer Blocker" for NGINX. It's free and open source, and works extremely well to add another layer of security by denying known bad actors across the internet. You can get it and the setup instructions on this GitHub page .

Revision #2
Created 2 February 2024 00:17:51 by Cleverness
Updated 2 February 2024 00:23:16 by Cleverness