bunkerized-nginx

Release v1.2.6

bunkerity

Jun 14, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

nginx Docker image secure by default.

Avoid the hassle of following security best practices "by hand" each time you need a web server or reverse proxy. Bunkerized-nginx provides generic security configs, settings and tools so you don't need to do it yourself.

Non-exhaustive list of features :

- HTTPS support with transparent Let's Encrypt automation
- State-of-the-art web security : HTTP security headers, prevent leaks, TLS hardening, ...
- Integrated ModSecurity WAF with the OWASP Core Rule Set
- Automatic ban of strange behaviors
- Antibot challenge through cookie, javascript, captcha or recaptcha v3
- Block TOR, proxies, bad user-agents, countries, ...
- Block known bad IP with DNSBL and CrowdSec
- Prevent bruteforce attacks with rate limiting
- Plugins system for external security checks (e.g. : ClamAV)
- Easy to configure with environment variables or web UI
- Automatic configuration with container labels
- Docker Swarm support

Fooling automated tools/scanners :

You can find a live demo at https://demo-nginx.bunkerity.com, feel free to do some security tests.

CHAPTER

TWO

QUICKSTART GUIDE

2.1 Run HTTP server with default settings

```
docker run -p 80:8080 -v /path/to/web/files:/www:ro bunkerity/bunkerized-nginx
```

Web files are stored in the /www directory, the container will serve files from there. Please note that *bunkerized-nginx* doesn't run as root but as an unprivileged user with UID/GID 101 therefore you should set the rights of */path/to/web/files* accordingly.

2.2 In combination with PHP

```
docker network create mynet
```

```
docker run --network mynet \
           -p 80:8080 \
           -v /path/to/web/files:/www:ro \
           -e REMOTE_PHP=myphp \
           -e REMOTE_PHP_PATH=/app \
           bunkerity/bunkerized-nginx
```

```
docker run --network mynet \
           --name myphp \
           -v /path/to/web/files:/app \
           php:fpm
```

The `REMOTE_PHP` environment variable lets you define the address of a remote PHP-FPM instance that will execute the .php files. `REMOTE_PHP_PATH` must be set to the directory where the PHP container will find the files.

2.3 Run HTTPS server with automated Let's Encrypt

```
docker run -p 80:8080 \
           -p 443:8443 \
           -v /path/to/web/files:/www:ro \
           -v /where/to/save/certificates:/etc/letsencrypt \
           -e SERVER_NAME=www.yourdomain.com \
           -e AUTO_LETS_ENCRYPT=yes \
           -e REDIRECT_HTTP_TO_HTTPS=yes \
           bunkerity/bunkerized-nginx
```

Certificates are stored in the /etc/letsencrypt directory, you should save it on your local drive. Please note that *bunkerized-nginx* doesn't run as root but as an unprivileged user with UID/GID 101 therefore you should set the rights of */where/to/save/certificates* accordingly.

If you don't want your webserver to listen on HTTP add the environment variable `LISTEN_HTTP` with a *no* value (e.g. HTTPS only). But Let's Encrypt needs the port 80 to be opened so redirecting the port is mandatory.

Here you have three environment variables :

- `SERVER_NAME` : define the FQDN of your webserver, this is mandatory for Let's Encrypt (www.yourdomain.com should point to your IP address)
- `AUTO_LETS_ENCRYPT` : enable automatic Let's Encrypt creation and renewal of certificates
- `REDIRECT_HTTP_TO_HTTPS` : enable HTTP to HTTPS redirection

2.4 As a reverse proxy

```
docker run -p 80:8080 \
           -e USE_REVERSE_PROXY=yes \
           -e REVERSE_PROXY_URL=/ \
           -e REVERSE_PROXY_HOST=http://myserver:8080 \
           bunkerity/bunkerized-nginx
```

This is a simple reverse proxy to a unique application. If you have more than one application you can add more REVERSE_PROXY_URL/REVERSE_PROXY_HOST by appending a suffix number like this :

```
docker run -p 80:8080 \
           -e USE_REVERSE_PROXY=yes \
           -e REVERSE_PROXY_URL_1=/app1/ \
           -e REVERSE_PROXY_HOST_1=http://myapp1:3000/ \
           -e REVERSE_PROXY_URL_2=/app2/ \
           -e REVERSE_PROXY_HOST_2=http://myapp2:3000/ \
           bunkerity/bunkerized-nginx
```

2.5 Behind a reverse proxy

```
docker run -p 80:8080 \
           -v /path/to/web/files:/www \
           -e PROXY_REAL_IP=yes \
           bunkerity/bunkerized-nginx
```

The PROXY_REAL_IP environment variable, when set to *yes*, activates the ngx_http_realip_module to get the real client IP from the reverse proxy.

See this section if you need to tweak some values (trusted ip/network, header, ...).

2.6 Multisite

By default, bunkerized-nginx will only create one server block. When setting the MULTISITE environment variable to *yes*, one server block will be created for each host defined in the SERVER_NAME environment variable.You can set/override values for a specific server by prefixing the environment variable with one of the server name previously defined.

```
docker run -p 80:8080 \
           -p 443:8443 \
           -v /where/to/save/certificates:/etc/letsencrypt \
           -e SERVER_NAME=app1.domain.com app2.domain.com \
           -e MULTISITE=yes \
           -e AUTO_LETS_ENCRYPT=yes \
           -e REDIRECT_HTTP_TO_HTTPS=yes \
           -e USE_REVERSE_PROXY=yes \
           -e app1.domain.com_REVERSE_PROXY_URL=/ \
           -e app1.domain.com_REVERSE_PROXY_HOST=http://myapp1:8000 \
           -e app2.domain.com_REVERSE_PROXY_URL=/ \
           -e app2.domain.com_REVERSE_PROXY_HOST=http://myapp2:8000 \
           bunkerity/bunkerized-nginx
```

The USE_REVERSE_PROXY is a *global* variable that will be applied to each server block. Whereas the app1.domain.com_* and app2.domain.com_* will only be applied to the app1.domain.com and app2.domain.com server block respectively.

When serving files, the web root directory should contains subdirectories named as the servers defined in the SERVER_NAME environment variable. Here is an example :

```
docker run -p 80:8080 \
           -p 443:8443 \
           -v /where/to/save/certificates:/etc/letsencrypt \
           -v /where/are/web/files:/www:ro \
           -e SERVER_NAME=app1.domain.com app2.domain.com \
           -e MULTISITE=yes \
           -e AUTO_LETS_ENCRYPT=yes \
           -e REDIRECT_HTTP_TO_HTTPS=yes \
           -e app1.domain.com_REMOTE_PHP=php1 \
           -e app1.domain.com_REMOTE_PHP_PATH=/app \
           -e app2.domain.com_REMOTE_PHP=php2 \
```

(continues on next page)

(continued from previous page)

```
            -e app2.domain.com_REMOTE_PHP_PATH=/app \
            bunkerity/bunkerized-nginx
```

The */where/are/web/files* directory should have a structure like this :

```
/where/are/web/files
├── app1.domain.com
│   └── index.php
│   └── ...
└── app2.domain.com
    └── index.php
    └── ...
```

2.7 Automatic configuration

The downside of using environment variables is that you need to recreate a new container each time you want to add or remove a web service. An alternative is to use the *bunkerized-nginx-autoconf* image which listens for Docker events and "automagically" generates the configuration.

First we need a volume that will store the configurations :

```
docker volume create nginx_conf
```

Then we run bunkerized-nginx with the `bunkerized-nginx.AUTOCONF` label, mount the created volume at /etc/nginx and set some default configurations for our services (e.g. : automatic Let's Encrypt and HTTP to HTTPS redirect) :

```
docker network create mynet

docker run -p 80:8080 \
           -p 443:8443 \
           --network mynet \
           -v /where/to/save/certificates:/etc/letsencrypt \
           -v /where/are/web/files:/www:ro \
           -v nginx_conf:/etc/nginx \
           -e SERVER_NAME= \
           -e MULTISITE=yes \
           -e AUTO_LETS_ENCRYPT=yes \
           -e REDIRECT_HTTP_TO_HTTPS=yes \
           -l bunkerized.nginx.AUTOCONF \
           bunkerity/bunkerized-nginx
```

When setting `SERVER_NAME` to nothing bunkerized-nginx won't create any server block (in case we only want automatic configuration).

Once bunkerized-nginx is created, let's setup the autoconf container :

```
docker run -v /var/run/docker.sock:/var/run/docker.sock:ro \
           -v nginx_conf:/etc/nginx \
           bunkerity/bunkerized-nginx-autoconf
```

We can now create a new container and use labels to dynamically configure bunkerized-nginx. Labels for automatic configuration are the same as environment variables but with the "bunkerized-nginx." prefix.

Here is a PHP example :

```
docker run --network mynet \
           --name myapp \
           -v /where/are/web/files/app.domain.com:/app \
           -l bunkerized-nginx.SERVER_NAME=app.domain.com \
           -l bunkerized-nginx.REMOTE_PHP=myapp \
           -l bunkerized-nginx.REMOTE_PHP_PATH=/app \
           php:fpm
```

And a reverse proxy example :

```
docker run --network mynet \
           --name anotherapp \
           -l bunkerized-nginx.SERVER_NAME=app2.domain.com \
           -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
           -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
           -l bunkerized-nginx.REVERSE_PROXY_HOST=http://anotherapp \
           tutum/hello-world
```

2.8 Swarm mode

Automatic configuration through labels is also supported in swarm mode. The *bunkerized-nginx-autoconf* is used to listen for Swarm events (e.g. service create/rm) and "automagically" edit configurations files and reload nginx.

As a use case we will assume the following :

- Some managers are also workers (they will only run the *autoconf* container for obvious security reasons)
- The bunkerized-nginx service will be deployed on all workers (global mode) so clients can connect to each of them (e.g. load balancing, CDN, edge proxy, ...)
- There is a shared folder mounted on managers and workers (e.g. NFS, GlusterFS, CephFS, ...)

Let's start by creating the network to allow communications between our services :

```
docker network create -d overlay mynet
```

We can now create the *autoconf* service that will listen to swarm events :

```
docker service create --name autoconf \
                      --network mynet \
                      --mount type=bind,source=/var/run/docker.sock,destination=/var/run/
↪docker.sock,ro \
                      --mount type=bind,source=/shared/confs,destination=/etc/nginx \
                      --mount type=bind,source=/shared/letsencrypt,destination=/etc/
↪letsencrypt \
                      --mount type=bind,source=/shared/acme-challenge,destination=/acme-
↪challenge \
                      -e SWARM_MODE=yes \
                      -e API_URI=/ChangeMeToSomethingHardToGuess \
                      --replicas 1 \
                      --constraint node.role==manager \
                      bunkerity/bunkerized-nginx-autoconf
```

You need to change `API_URI` to something hard to guess since there is no other security mechanism to protect the API at the moment.

When *autoconf* is created, it's time for the *bunkerized-nginx* service to be up :

```
docker service create --name nginx \
                      --network mynet \
                      -p published=80,target=8080,mode=host \
                      -p published=443,target=8443,mode=host \
                      --mount type=bind,source=/shared/confs,destination=/etc/nginx \
                      --mount type=bind,source=/shared/letsencrypt,destination=/etc/
↪letsencrypt,ro \
                      --mount type=bind,source=/shared/acme-challenge,destination=/acme-
↪challenge,ro \
                      --mount type=bind,source=/shared/www,destination=/www,ro \
                      -e SWARM_MODE=yes \
                      -e USE_API=yes \
                      -e API_URI=/ChangeMeToSomethingHardToGuess \
                      -e MULTISITE=yes \
                      -e SERVER_NAME= \
                      -e AUTO_LETS_ENCRYPT=yes \
                      -e REDIRECT_HTTP_TO_HTTPS=yes \
                      -l bunkerized-nginx.AUTOCONF \
                      --mode global \
                      --constraint node.role==worker \
                      bunkerity/bunkerized-nginx
```

The `API_URI` value must be the same as the one specified for the *autoconf* service.

We can now create a new service and use labels to dynamically configure bunkerized-nginx. Labels for automatic configuration are the same as environment variables but with the "bunkerized-nginx." prefix.

Here is a PHP example :

```
docker service create --name myapp \
                      --network mynet \
                      --mount type=bind,source=/shared/www/app.domain.com,destination=/
↪app \
                      -l bunkerized-nginx.SERVER_NAME=app.domain.com \
                      -l bunkerized-nginx.REMOTE_PHP=myapp \
                      -l bunkerized-nginx.REMOTE_PHP_PATH=/app \
                      --constraint node.role==worker \
                      php:fpm
```

And a reverse proxy example :

```
docker service create --name anotherapp \
                      --network mynet \
                      -l bunkerized-nginx.SERVER_NAME=app2.domain.com \
                      -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
                      -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
                      -l bunkerized-nginx.REVERSE_PROXY_HOST=http://anotherapp \
                      --constraint node.role==worker \
                      tutum/hello-world
```

2.9 Web UI

A dedicated image, *bunkerized-nginx-ui*, lets you manage bunkerized-nginx instances and services configurations through a web user interface. This feature is still in beta, feel free to open a new issue if you find a bug and/or you have an idea to improve it.

First we need a volume that will store the configurations and a network because bunkerized-nginx will be used as a reverse proxy for the web UI :

```
docker volume create nginx_conf
docker network create mynet
```

Let's create the bunkerized-nginx-ui container that will host the web UI behind bunkerized-nginx :

```
docker run --network mynet \
           --name myui \
           -v /var/run/docker.sock:/var/run/docker.sock:ro \
           -v nginx_conf:/etc/nginx \
           -e ABSOLUTE_URI=https://admin.domain.com/webui/ \
           bunkerity/bunkerized-nginx-ui
```

You will need to edit the `ABSOLUTE_URI` environment variable to reflect your actual URI of the web UI.

We can now setup the bunkerized-nginx instance with the `bunkerized-nginx.UI` label and a reverse proxy configuration for our web UI :

```
docker network create mynet

docker run -p 80:8080 \
           -p 443:8443 \
           --network mynet \
           -v nginx_conf:/etc/nginx \
           -v /where/are/web/files:/www:ro \
           -v /where/to/save/certificates:/etc/letsencrypt \
           -e SERVER_NAME=admin.domain.com \
           -e MULTISITE=yes \
           -e AUTO_LETS_ENCRYPT=yes \
           -e REDIRECT_HTTP_TO_HTTPS=yes \
           -e DISABLE_DEFAULT_SERVER=yes \
           -e admin.domain.com_USE_MODSECURITY=no \
           -e admin.domain.com_SERVE_FILES=no \
           -e admin.domain.com_USE_AUTH_BASIC=yes \
           -e admin.domain.com_AUTH_BASIC_USER=admin \
           -e admin.domain.com_AUTH_BASIC_PASSWORD=password \
           -e admin.domain.com_USE_REVERSE_PROXY=yes \
           -e admin.domain.com_REVERSE_PROXY_URL=/webui/ \
           -e admin.domain.com_REVERSE_PROXY_HOST=http://myui:5000/ \
           -l bunkerized-nginx.UI \
           bunkerity/bunkerized-nginx
```

The `AUTH_BASIC` environment variables let you define a login/password that must be provided before accessing to the web UI. At the moment, there is no authentication mechanism integrated into bunkerized-nginx-ui so **using auth basic with a strong password coupled with a "hard to guess" URI is strongly recommended**.

Web UI should now be accessible from https://admin.domain.com/webui/.

CHAPTER

THREE

SECURITY TUNING

bunkerized-nginx comes with a set of predefined security settings that you can (and you should) tune to meet your own use case.

3.1 Miscellaneous

Here is a list of miscellaneous environment variables related more or less to security :

- `MAX_CLIENT_SIZE=10m` : maximum size of client body
- `ALLOWED_METHODS=GET|POST|HEAD` : list of HTTP methos that clients are allowed to use
- `DISABLE_DEFAULT_SERVER=no` : enable/disable the default server (i.e. : should your server respond to unknown Host header ?)
- `SERVER_TOKENS=off` : enable/disable sending the version number of nginx

3.2 HTTPS

3.2.1 Settings

Here is a list of environment variables and the corresponding default value related to HTTPS :

- `LISTEN_HTTP=yes` : you can set it to `no` if you want to disable HTTP access
- `REDIRECT_HTTP_TO_HTTPS=no` : enable/disable HTTP to HTTPS redirection
- `HTTPS_PROTOCOLS=TLSv1.2 TLSv1.3` : list of TLS versions to use
- `HTTP2=yes` : enable/disable HTTP2 when HTTPS is enabled
- `COOKIE_AUTO_SECURE_FLAG=yes` : enable/disable adding Secure flag when HTTPS is enabled
- `STRICT_TRANSPORT_SECURITY=max-age=31536000` : force users to visit the website in HTTPS (more info here)

3.2.2 Let's Encrypt

Using Let's Encrypt with the `AUTO_LETS_ENCRYPT=yes` environment variable is the easiest way to add HTTPS supports to your web services if they are connected to internet and you have public DNS A record(s).

You can also set the `EMAIL_LETS_ENCRYPT` environment variable if you want to receive notifications from Let's Encrypt (e.g. : expiration).

3.2.3 Custom certificate(s)

If you have security constraints (e.g : local network, custom PKI, ...) you can use custom certificates of your choice and tell bunkerized-nginx to use them with the following environment variables :

- `USE_CUSTOM_HTTPS=yes`
- `CUSTOM_HTTPS_CERT=/path/inside/container/to/cert.pem`
- `CUSTOM_HTTPS_KEY=/path/inside/container/to/key.pem`

Here is a an example on how to use custom certificates :

```
$ ls /etc/ssl/my-web-app
cert.pem key.pem

$ docker run -p 80:8080 \
             -p 443:8443 \
             -v /etc/ssl/my-web-app:/certs:ro \
             -e USE_CUSTOM_HTTPS=yes \
             -e CUSTOM_HTTPS_CERT=/certs/cert.pem \
             -e CUSTOM_HTTPS_KEY=/certs/key.pem \
             ...
             bunkerity/bunkerized-nginx
```

Please note that if you have one or more intermediate certificate(s) in your chain of trust, you will need to provide the bundle to `CUSTOM_HTTPS_CERT` (more info here).

You can reload the certificate(s) (e.g. : in case of a renewal) by sending the SIGHUP/HUP signal to the container bunkerized-nginx will catch the signal and send a reload order to nginx :

```
docker kill --signal=SIGHUP my-container
```

3.2.4 Self-signed certificate

This method is not recommended in production but can be used to quickly deploy HTTPS for testing purposes. Just use the `GENERATE_SELF_SIGNED_SSL=yes` environment variable and bunkerized-nginx will generate a self-signed certificate for you :

```
$ docker run -p 80:8080 \
             -p 443:8443 \
             -e GENERATE_SELF_SIGNED_SSL=yes \
             ...
             bunkerity/bunkerized-nginx
```

3.3 Headers

Some important HTTP headers related to client security are sent with a default value. Sometimes it can break a web application or can be tuned to provide even more security. The complete list is available here.

You can also remove headers (e.g. : too verbose ones) by using the `REMOVE_HEADERS` environment variable which takes a list of header name separated with space (default value = `Server X-Powered-By X-AspNet-Version X-AspNetMvc-Version`).

3.4 ModSecurity

ModSecurity is integrated and enabled by default alongside the OWASP Core Rule Set within bunkerized-nginx. To change this behaviour you can use the `USE_MODSECURITY=no` or `USE_MODSECURITY_CRS=no` environment variables.

We strongly recommend to keep both ModSecurity and the OWASP Core Rule Set enabled. The only downsides are the false positives that may occur. But they can be fixed easily and the CRS team maintains a list of exclusions for common application (e.g : wordpress, nextcloud, drupal, cpanel, ...).

Tuning the CRS with bunkerized-nginx is pretty simple : you can add configuration before (i.e. : exclusions) and after (i.e. : exceptions/tuning) the rules are loaded. You just need to mount your .conf files into the /modsec-crs-confs (before CRS is loaded) and /modsec-confs (after CRS is loaded).

Here is an example to illustrate it :

```
$ cat /data/exclusions-crs/wordpress.conf
SecAction \
 "id:900130,\
  phase:1,\
  nolog,\
  pass,\
  t:none,\
  setvar:tx.crs_exclusions_wordpress=1"

$ cat /data/tuning-crs/remove-false-positives.conf
SecRule REQUEST_FILENAME "/wp-admin/admin-ajax.php" "id:1,ctl:ruleRemoveByTag=attack-xss,
↪ctl:ruleRemoveByTag=attack-rce"
SecRule REQUEST_FILENAME "/wp-admin/options.php" "id:2,ctl:ruleRemoveByTag=attack-xss"
SecRule REQUEST_FILENAME "^/wp-json/yoast" "id:3,ctl:ruleRemoveById=930120"

$ docker run -p 80:8080 \
             -p 443:8443 \
             -v /data/exclusions-crs:/modsec-crs-confs:ro \
             -v /data/tuning-crs:/modsec-confs:ro \
             ...
             bunkerity/bunkerized-nginx
```

3.5 Bad behaviors detection

When attackers search for and/or exploit vulnerabilities they might generate some suspicious HTTP status codes that a "regular" user won't generate within a period of time. If we detect that kind of behavior we can ban the offending IP address and force the attacker to come with a new one.

That kind of security measure is implemented and enabled by default in bunkerized-nginx. Here is the list of the related environment variables and their default value :

- `USE_BAD_BEHAVIOR=yes` : enable/disable "bad behavior" detection and automatic ban of IP
- `BAD_BEHAVIOR_STATUS_CODES=400 401 403 404 405 429 444` : the list of HTTP status codes considered as "suspicious"
- `BAD_BEHAVIOR_THRESHOLD=10` : the number of "suspicious" HTTP status codes required before we ban the corresponding IP address
- `BAD_BEHAVIOR_BAN_TIME=86400` : the duration time (in seconds) of the ban
- `BAD_BEHAVIOR_COUNT_TIME=60` : the duration time (in seconds) to wait before resetting the counter of "suspicious" HTTP status codes for a given IP

3.6 Antibot challenge

Attackers will certainly use automated tools to exploit/find some vulnerabilities on your web service. One countermeasure is to challenge the users to detect if it looks like a bot. It might be effective against script kiddies or "lazy" attackers.

You can use the `USE_ANTIBOT` environment variable to add that kind of checks whenever a new client is connecting. The available challenges are : `cookie`, `javascript`, `captcha` and `recaptcha`. More info here.

3.7 External blacklists

3.7.1 DNSBL

Automatic checks on external DNS BlackLists are enabled by default with the `USE_DNSBL=yes` environment variable. The list of DNSBL zones is also configurable, you just need to edit the `DNSBL_LIST` environment variable which contains the following value by default `bl.blocklist.de problems.dnsbl.sorbs.net sbl.spamhaus.org xbl.spamhaus.org`.

3.7.2 CrowdSec

CrowdSec is not enabled by default because it's more than an external blacklists and needs some extra work to get it working. But bunkerized-nginx is fully working with CrowdSec, here are the related environment variables :

- `USE_CROWDSEC=no` : enable/disable CrowdSec checks before we authorize a client
- `CROWDSEC_HOST=` : full URL to your CrowdSec instance API
- `CROWDSEC_KEY=` : bouncer key given from **cscli bouncer add MyBouncer**

You will also need to share the logs generated by bunkerized-nginx with your CrowdSec instance. One approach is to send the logs to a syslog server which is writing the logs to the file system and then CrowdSec can easily read the logs. If you want to give it a try, you have a concrete example on how to use CrowdSec with bunkerized-nginx here.

3.7.3 User-Agents

Sometimes script kiddies or lazy attackers don't put a "legitimate" value inside the **User-Agent** HTTP header so we can block them. This is controlled with the `BLOCK_USER_AGENT=yes` environment variable. The blacklist is composed of two files from here and here.

If a legitimate User-Agent is blacklisted, you can use the `WHITELIST_USER_AGENT` while still keeping the `BLOCK_USER_AGENT=yes` (more info here).

3.7.4 TOR exit nodes

Blocking TOR exit nodes might not be a good decision depending on your use case. We decided to enable it by default with the `BLOCK_TOR_EXIT_NODE=yes` environment variable. If privacy is a concern for you and/or your clients, you can override the default value (i.e : `BLOCK_TOR_EXIT_NODE=no`).

Please note that you have a concrete example on how to use bunkerized-nginx with a .onion hidden service here.

3.7.5 Proxies

This list contains IP addresses and networks known to be open proxies (downloaded from here). Unless privacy is important for you and/or your clients, you should keep the default environment variable `BLOCK_PROXIES=yes`.

3.7.6 Abusers

This list contains IP addresses and networks known to be abusing (downloaded from here). You can control this feature with the `BLOCK_ABUSERS` environment variable (default : `yes`).

3.7.7 Referrers

This list contains bad referrers domains known for spamming (downloaded from here). If one value is found inside the **Referer** HTTP header, request will be blocked. You can control this feature with the `BLOCK_REFERRER` environment variable (default = `yes`).

3.8 Limiting

3.8.1 Requests

To limit bruteforce attacks we decided to use the rate limiting feature in nginx so attackers will be limited to X request(s)/s for the same resource. That kind of protection might be useful against other attacks too (e.g. : blind SQL injection).

Here is the list of related environment variables and their default value :

- `USE_LIMIT_REQ=yes` : enable/disable request limiting
- `LIMIT_REQ_RATE=1r/s` : the rate to apply for the same resource
- `LIMIT_REQ_BURST=2` : the number of request tu put in a queue before effectively rejecting requests

3.8.2 Connections

Opening too many connections from the same IP address might be considered as suspicious (unless it's a shared IP and everyone is sending requests to your web service). It can be a dos/ddos attempt too. Bunkerized-nginx levarages the ngx_http_conn_module from nginx to prevent users opening too many connections.

Here is the list of related environment variables and their default value :

- `USE_LIMIT_CONN=yes` : enable disable connection limiting
- `LIMIT_CONN_MAX=50` : maximum number of connections per IP

3.9 Country

If the location of your clients is known, you may want to add another security layer by whitelisting or blacklisting some countries. You can use the `BLACKLIST_COUNTRY` or `WHITELIST_COUNTRY` environment variables depending on your approach. They both take a list of 2 letters country code separated with space.

3.10 Authentication

You can quickly protect sensitive resources (e.g. : admin panels) by requiring HTTP authentication. Here is the list of related environment variables and their default value :

- `USE_AUTH_BASIC=no` : enable/disable auth basic
- `AUTH_BASIC_LOCATION=sitewide` : location of the sensitive resource (e.g. `/admin`) or `sitewide` to force authentication on the whole service
- `AUTH_BASIC_USER=changeme` : the username required
- `AUTH_BASIC_PASSWORD=changeme` : the password required
- `AUTH_BASIC_TEXT=Restricted area` : the text that will be displayed to the user

3.11 Whitelisting

Adding extra security can sometimes trigger false positives. Also, it might be not useful to do the security checks for specific clients because we decided to trust them. Bunkerized-nginx supports two types of whitelist : by IP address and by reverse DNS.

Here is the list of related environment variables and their default value :

- `USE_WHITELIST_IP=yes` : enable/disable whitelisting by IP address
- `WHITELIST_IP_LIST=23.21.227.69 40.88.21.235 50.16.241.113 50.16.241.114 50.16.241.117 50.16.247.234 52.204.97.54 52.5.190.19 54.197.234.188 54.208.100.253 54.208.102.37 107.21.1.8` : list of IP addresses and/or network CIDR blocks to whitelist (default contains the IP addresses of the DuckDuckGo crawler)
- `USE_WHITELIST_REVERSE=yes` : enable/disable whitelisting by reverse DNS
- `WHITELIST_REVERSE_LIST=.googlebot.com .google.com .search.msn.com .crawl.yahoot.net .crawl.baidu.jp .crawl.baidu.com .yandex.com .yandex.ru .yandex.net` : the list of reverse DNS suffixes to trust (default contains the list of major search engines crawlers)

3.12 Blacklisting

Sometimes it isn't necessary to spend some resources for a particular client because we know for sure that he is malicious. Bunkerized-nginx nginx supports two types of blacklisting : by IP address and by reverse DNS.

Here is the list of related environment variables and their default value :

- `USE_BLACKLIST_IP=yes` : enable/disable blacklisting by IP address
- `BLACKLIST_IP_LIST=` : list of IP addresses and/or network CIDR blocks to blacklist
- `USE_BLACKLIST_REVERSE=yes` : enable/disable blacklisting by reverse DNS
- `BLACKLIST_REVERSE_LIST=.shodan.io` : the list of reverse DNS suffixes to never trust

3.13 Web UI

Mounting the docker socket in a container which is facing the network, like we do with the web UI, is not a good security practice. In case of a vulnerability inside the application, attackers can freely use the Docker socket and the whole host can be compromised.

A possible workaround is to use the tecnativa/docker-socket-proxy image which acts as a reverse proxy between the application and the Docker socket. It can allow/deny the requests made to the Docker API.

Before starting the web UI, you need to fire up the docker-socket-proxy (we also need a network because of inter-container communication) :

```
docker network create mynet
```

```
docker run --name mysocketproxy \
           --network mynet \
           -v /var/run/docker.sock:/var/run/docker.sock:ro \
           -e POST=1 \
           -e CONTAINERS=1 \
           tecnativa/docker-socket-proxy
```

You can now start the web UI container and use the `DOCKER_HOST` environment variable to define the Docker API endpoint :

```
docker run --network mynet \
           -v autoconf:/etc/nginx \
           -e ABSOLUTE_URI=https://my.webapp.com/admin/ \
           -e DOCKER_HOST=tcp://mysocketproxy:2375 \
           bunkerity/bunkerized-nginx-ui
```

3.14 Plugins

Some security features can be added through the plugins system (e.g. : ClamAV). You will find more info in the plugins section.

3.15 Container hardening

You will find a ready to use docker-compose.yml file focused on container hardening here.

3.15.1 Drop capabilities

By default, *bunkerized-nginx* runs as non-root user inside the container and should not use any of the default capabilities allowed by Docker. You can safely remove all capabilities to harden the container :

```
docker run ... --drop-cap=all ... bunkerity/bunkerized-nginx
```

3.15.2 No new privileges

Bunkerized-nginx should never tries to gain additional privileges through setuid/setgid executables. You can safely add the **no-new-privileges** security configuration when creating the container :

```
docker run ... --security-opt no-new-privileges ... bunkerity/bunkerized-nginx
```

3.15.3 Read-only

Since the locations where bunkerized-nginx needs to write are known, we can run the container with a read-only root file system and only allow writes to specific locations by adding volumes and a tmpfs mount :

```
docker run ... --read-only --tmpfs /tmp -v cache-vol:/cache -v conf-vol:/etc/nginx -v /
↪path/to/web/files:/www:ro -v /where/to/store/certificates:/etc/letsencrypt bunkerity/
↪bunkerized-nginx
```

3.15.4 User namespace remap

Another hardening trick is user namespace remapping : it allows you to map the UID/GID of users inside a container to another UID/GID on the host. For example, you can map the user nginx with UID/GID 101 inside the container to a non-existent user with UID/GID 100101 on the host.

Let's assume you have the /etc/subuid and /etc/subgid files like this :

```
user:100000:65536
```

It means that everything done inside the container will be remapped to UID/GID 100101 (100000 + 101) on the host.

Please note that you must set the rights on the volumes (e.g. : /etc/letsencrypt, /www, ...) according to the remapped UID/GID :

```
$ chown root:100101 /path/to/letsencrypt
$ chmod 770 /path/to/letsencrypt
$ docker run ... -v /path/to/letsencrypt:/etc/letsencrypt ... bunkerity/bunkerized-nginx
```

CHAPTER

FOUR

TROUBLESHOOTING

4.1 Logs

When troubleshooting, the logs are your best friends. We try our best to provide user-friendly logs to help you understand what happened. Please note that we don't store the logs inside the container, they are all displayed on stdout/stderr so Docker can capture them. They can be displayed using the docker logs command.

You can edit the `LOG_LEVEL` environment variable to increase or decrease the verbosity of logs with the following values : `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert` or `emerg` (with `debug` being the most verbose level).

4.2 Permissions

Don't forget that bunkerized-nginx runs as an unprivileged user with UID/GID 101. Double check the permissions of files and folders for each volumes (see the volumes list).

4.3 ModSecurity

The OWASP Core Rule Set can sometimes leads to false positives. Here is what you can do :

- Check if your application has exclusions rules (e.g : wordpress, nextcloud, drupal, ...)
- Edit the matched rules to exclude some parameters, URIs, ...
- Remove the matched rules if editing it is too much a hassle

Some additional resources :

- Wordpress example
- Handling false positive
- Adding exceptions and tuning

4.4 Bad behavior

The bad behavior feature comes with a set of status codes considered as "suspicious". You may need to tweak the corresponding list to avoid false positives within your application.

4.5 Whitelisting

It's a common case that a bot gets flagged as suspicious and can't access your website. Instead of disabling the corresponding security feature(s) we recommend a whitelisting approach. Here is a list of environment variables you can use :

- `WHITELIST_IP_LIST`
- `WHITELIST_REVERSE_LIST`
- `WHITELIST_URI`
- `WHITELIST_USER_AGENT`

More information here.

CHAPTER

FIVE

VOLUMES LIST

Please note that bunkerized-nginx run as an unprivileged user inside the container (UID/GID = 101) and you should set the rights on the host accordingly (e.g. : chmod 101:101 ...) to the files and folders on your host.

5.1 Web files

Mountpoint : `/www`

Description :If `MULTISITE=no`, the web files are directly stored inside the `/www` folder. When `MULTISITE=yes`, you need to create subdirectories named as the servers defined in the `SERVER_NAME` environment variable.

Examples : basic and multisite

Read-only : yes

5.2 Let's Encrypt

Mountpoint : `/etc/letsencrypt`

Description :When `AUTO_LETS_ENCRYPT=yes`, certbot will save configurations, certificates and keys inside the `/etc/letsencrypt` folder. It's a common practise to save it so you can remount it in case of a container restart and certbot won't generate new certificate(s).

Examples : here

Read-only : no

5.3 Custom nginx configurations

5.3.1 http context

Mountpoint : `/http-confs`

Description :If you need to add custom configurations at http context, you can create **.conf** files and mount them to the `/http-confs` folder.

Examples : load balancer

Read-only : yes

5.3.2 server context

Mountpoint : `/server-confs`

Description :If `MULTISITE=no`, you can create **.conf** files and mount them to the `/server-confs` folder. When `MULTISITE=yes`, you need to create subdirectories named as the servers defined in the `SERVER_NAME` environment variable.

Examples : nextcloud and multisite

Read-only : yes

5.4 ModSecurity

5.4.1 Rules and before CRS

Mountpoint : `/modsec-confs`

Description :Use this volume if you need to add custom ModSecurity rules and/or OWASP Core Rule Set configurations before the rules are loaded (e.g. : exclusions).If `MULTISITE=no` you can create **.conf** files and mount them to the `/modsec-confs` folder. When `MULTISITE=yes`, you need to create subdirectories named as the servers defined in the `SERVER_NAME` environment variable. You can also apply global configuration to all servers by putting **.conf** files directly on the root folder.

Examples : wordpress and multisite

Read-only : yes

5.4.2 After CRS

Mountpoint : `/modsec-crs-confs`

Description :Use this volume to tweak OWASP Core Rule Set (e.g. : tweak rules to avoid false positives). Your files are loaded after the rules. If `MULTISITE=no` you can create **.conf** files and mount them to the `/modsec-crs-confs` folder. When `MULTISITE=yes`, you need to create subdirectories named as the servers defined in the `SERVER_NAME` environment variable. You can also apply global configuration to all servers by putting **.conf** files directly on the root folder.

Examples : wordpress and multisite

Read-only : yes

5.5 Cache

Mountpoint : `/cache`

Description :Depending of the settings you use, bunkerized-nginx may download external content (e.g. : blacklists, GeoIP DB, ...). To avoid downloading it again in case of a container restart, you can save the data on the host.

Read-only : no

5.6 Plugins

Mountpoint : `/plugins`

Description :This volume is used to extend bunkerized-nginx with additional plugins. Please note that you will need to have a subdirectory for each plugin you want to enable.

Read-only : yes

CHAPTER

SIX

LIST OF ENVIRONMENT VARIABLES

6.1 nginx

6.1.1 Misc

`MULTISITE`Values : *yes* | *no*Default value : *no*Context : *global*When set to *no*, only one server block will be generated. Otherwise one server per host defined in the `SERVER_NAME` environment variable will be generated.Any environment variable tagged as *multisite* context can be used for a specific server block with the following format : *host_VARIABLE=value*. If the variable is used without the host prefix it will be applied to all the server blocks (but still can be overriden).

`SERVER_NAME`Values : <*first name*> <*second name*> ...Default value : *www.bunkerity.com*Context : *global*, *multisite*Sets the host names of the webserver separated with spaces. This must match the Host header sent by clients.Useful when used with `MULTISITE=yes` and/or `AUTO_LETSENCRYPT=yes` and/or `DISABLE_DEFAULT_SERVER=yes`.

`MAX_CLIENT_SIZE`Values : *0* | *Xm*Default value : *10m*Context : *global*, *multisite*Sets the maximum body size before nginx returns a 413 error code.Setting to 0 means "infinite" body size.

`ALLOWED_METHODS`Values : *allowed HTTP methods separated with* | *char*Default value : *GET|POST|HEAD*Context : *global*, *multisite*Only the HTTP methods listed here will be accepted by nginx. If not listed, nginx will close the connection.

`DISABLE_DEFAULT_SERVER`Values : *yes* | *no*Default value : *no*Context : *global*If set to yes, nginx will only respond to HTTP request when the Host header match a FQDN specified in the `SERVER_NAME` environment variable.For example, it will close the connection if a bot access the site with direct ip.

`SERVE_FILES`Values : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, nginx will serve files from /www directory within the container.A use case to not serving files is when you setup bunkerized-nginx as a reverse proxy.

`DNS_RESOLVERS`Values : <*two IP addresses separated with a space*>Default value : *127.0.0.11*Context : *global*The IP addresses of the DNS resolvers to use when performing DNS lookups.

`ROOT_FOLDER`Values : <*any valid path to web files*>Default value : */www*Context : *global*The default folder where nginx will search for web files. Don't change it unless you know what you are doing.

`ROOT_SITE_SUBFOLDER`Values : <*any valid directory name*>Default value :Context : *global*, *multisite*The subfolder where nginx will search for site web files.

`LOG_FORMAT`Values : <*any values accepted by the log_format directive*>Default value : $host remote_addr - remote_user time_local] "request" status body_bytes_sent "http_referer" "$http_user_agent"$Context : *global*The log format used by nginx to generate logs. More info here.

`LOG_LEVEL`Values : *debug, info, notice, warn, error, crit, alert, or emerg*Default value : *info*Context : *global*The level of logging : *debug* means more logs and *emerg* means less logs. More info here.

HTTP_PORTValues : *<any valid port greater than 1024>*Default value : *8080*Context : *global*The HTTP port number used by nginx inside the container.

HTTPS_PORTValues : *<any valid port greater than 1024>*Default value : *8443*Context : *global*The HTTPS port number used by nginx inside the container.

WORKER_CONNECTIONS Values : *<any positive integer>*Default value : 1024Context : *global*Sets the value of the worker_connections directive.

WORKER_RLIMIT_NOFILE Values : *<any positive integer>*Default value : 2048Context : *global*Sets the value of the worker_rlimit_nofile directive.

6.1.2 Information leak

SERVER_TOKENSValues : *on* | *off*Default value : *off*Context : *global*If set to on, nginx will display server version in Server header and default error pages.

REMOVE_HEADERSValues : *<list of headers separated with space>*Default value : *Server X-Powered-By X-AspNet-Version X-AspNetMvc-Version*Context : *global*, *multisite*List of header to remove when sending responses to clients.

6.1.3 Custom error pages

ERRORSValues : *<error1=/page1 error2=/page2>*Default value :Context : *global*, *multisite*Use this kind of environment variable to define custom error page depending on the HTTP error code. Replace errorX with HTTP code.Example : `ERRORS=404=/404.html 403=/403.html` the /404.html page will be displayed when 404 code is generated (same for 403 and /403.html page). The path is relative to the root web folder.

6.1.4 HTTP basic authentication

USE_AUTH_BASICValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*If set to yes, enables HTTP basic authentication at the location AUTH_BASIC_LOCATION with user AUTH_BASIC_USER and password AUTH_BASIC_PASSWORD.

AUTH_BASIC_LOCATIONValues : *sitewide* | */somedir* | *<any valid location>*Default value : *sitewide*Context : *global*, *multisite*The location to restrict when USE_AUTH_BASIC is set to *yes*. If the special value *sitewide* is used then auth basic will be set at server level outside any location context.

AUTH_BASIC_USERValues : *<any valid username>*Default value : *changeme*Context : *global*, *multisite*The username allowed to access AUTH_BASIC_LOCATION when USE_AUTH_BASIC is set to yes.

AUTH_BASIC_PASSWORDValues : *<any valid password>*Default value : *changeme*Context : *global*, *multisite*The password of AUTH_BASIC_USER when USE_AUTH_BASIC is set to yes.

AUTH_BASIC_TEXTValues : *<any valid text>*Default value : *Restricted area*Context : *global*, *multisite*The text displayed inside the login prompt when USE_AUTH_BASIC is set to yes.

6.1.5 Reverse proxy

USE_REVERSE_PROXYValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*Set this environment variable to *yes* if you want to use bunkerized-nginx as a reverse proxy.

REVERSE_PROXY_URLValues : <*any valid location path*>Default value :Context : *global*, *multisite*Only valid when USE_REVERSE_PROXY is set to *yes*. Let's you define the location path to match when acting as a reverse proxy.You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_URL_1, REVERSE_PROXY_URL_2, REVERSE_PROXY_URL_3, ...

REVERSE_PROXY_HOSTValues : <*any valid proxy_pass value*>Default value :Context : *global*, *multisite*Only valid when USE_REVERSE_PROXY is set to *yes*. Let's you define the proxy_pass destination to use when acting as a reverse proxy.You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_HOST_1, REVERSE_PROXY_HOST_2, REVERSE_PROXY_HOST_3, ...

REVERSE_PROXY_WSValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*Only valid when USE_REVERSE_PROXY is set to *yes*. Set it to *yes* when the corresponding REVERSE_PROXY_HOST is a WebSocket server.You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_WS_1, REVERSE_PROXY_WS_2, REVERSE_PROXY_WS_3, ...

REVERSE_PROXY_HEADERSValues : <*list of custom headers separated with a semicolon like this : header1 value1;header2 value2...*> Default value :Context : *global*, *multisite*Only valid when USE_REVERSE_PROXY is set to *yes*.You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_HEADERS_1, REVERSE_PROXY_HEADERS_2, REVERSE_PROXY_HEADERS_3, ...

PROXY_REAL_IPValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*Set this environment variable to *yes* if you're using bunkerized-nginx behind a reverse proxy. This means you will see the real client address instead of the proxy one inside your logs. Ssecurity tools will also then work correctly.

PROXY_REAL_IP_FROMValues : <*list of trusted IP addresses and/or networks separated with spaces*>Default value : *192.168.0.0/16 172.16.0.0/12 10.0.0.0/8*Context : *global*, *multisite*When PROXY_REAL_IP is set to *yes*, lets you define the trusted IPs/networks allowed to send the correct client address.

PROXY_REAL_IP_HEADERValues : *X-Forwarded-For* | *X-Real-IP* | *custom header*Default value : *X-Forwarded-For*Context : *global*, *multisite*When PROXY_REAL_IP is set to *yes*, lets you define the header that contains the real client IP address.

PROXY_REAL_IP_RECURSIVEValues : *on* | *off*Default value : *on*Context : *global*, *multisite*When PROXY_REAL_IP is set to *yes*, setting this to *on* avoid spoofing attacks using the header defined in PROXY_REAL_IP_HEADER.

6.1.6 Compression

USE_GZIPValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*When set to *yes*, nginx will use the gzip algorithm to compress responses sent to clients.

GZIP_COMP_LEVELValues : <*any integer between 1 and 9*>Default value : *5*Context : *global*, *multisite*The gzip compression level to use when USE_GZIP is set to *yes*.

GZIP_MIN_LENGTHValues : <*any positive integer*>Default value : *1000*Context : *global*, *multisite*The minimum size (in bytes) of a response required to compress when USE_GZIP is set to *yes*.

GZIP_TYPESValues : <*list of mime types separated with space*>Default value : *application/atom+xml application/javascript application/json application/rss+xml application/vnd.ms-fontobject application/x-font-opentype application/x-font-truetype application/x-font-ttf application/x-javascript application/xhtml+xml application/xml font/eot font/opentype font/otf font/truetype image/svg+xml image/vnd.microsoft.icon image/x-icon image/x-win-bitmap text/css text/javascript text/plain text/xml*Context : *global*, *multisite*List of response MIME type required to compress when USE_GZIP is set to *yes*.

USE_BROTLIValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*When set to *yes*, nginx will use the brotli algorithm to compress responses sent to clients.

BROTLI_COMP_LEVELValues : <*any integer between 1 and 9*>Default value : *5*Context : *global*, *multisite*The brotli compression level to use when USE_BROTLI is set to *yes*.

BROTLI_MIN_LENGTHValues : <*any positive integer*>Default value : *1000*Context : *global*, *multisite*The minimum size (in bytes) of a response required to compress when USE_BROTLI is set to *yes*.

BROTLI_TYPESValues : <*list of mime types separated with space*>Default value : *application/atom+xml application/javascript application/json application/rss+xml application/vnd.ms-fontobject application/x-font-opentype application/x-font-truetype application/x-font-ttf application/x-javascript application/xhtml+xml application/xml font/eot font/opentype font/otf font/truetype image/svg+xml image/vnd.microsoft.icon image/x-icon image/x-win-bitmap text/css text/javascript text/plain text/xml*Context : *global*, *multisite*List of response MIME type required to compress when USE_BROTLI is set to *yes*.

6.1.7 Cache

USE_CLIENT_CACHEValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*When set to *yes*, clients will be told to cache some files locally.

CLIENT_CACHE_EXTENSIONSValues : <*list of extensions separated with* |>Default value : *jpg|jpeg|png|bmp|ico|svg|tif|css|js|otf|ttf|eot|woff|woff2*Context : *global*, *multisite*List of file extensions that clients should cache when USE_CLIENT_CACHE is set to *yes*.

CLIENT_CACHE_CONTROLValues : <*Cache-Control header value*>Default value : *public, max-age=15552000*Context : *global*, *multisite*Content of the Cache-Control header to send when USE_CLIENT_CACHE is set to *yes*.

CLIENT_CACHE_ETAGValues : *on* | *off*Default value : *on*Context : *global*, *multisite*Whether or not nginx will send the ETag header when USE_CLIENT_CACHE is set to *yes*.

USE_OPEN_FILE_CACHEValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*When set to *yes*, nginx will cache open fd, existence of directories, ... See open_file_cache.

OPEN_FILE_CACHEValues : <*any valid open_file_cache parameters*>Default value : *max=1000 inactive=20s*Context : *global*, *multisite*Parameters to use with open_file_cache when USE_OPEN_FILE_CACHE is set to *yes*.

OPEN_FILE_CACHE_ERRORSValues : *on* | *off*Default value : *on*Context : *global*, *multisite*Whether or not nginx should cache file lookup errors when USE_OPEN_FILE_CACHE is set to *yes*.

OPEN_FILE_CACHE_MIN_USESValues : <*any valid integer *>Default value : *2*Context : *global*, *multisite*The minimum number of file accesses required to cache the fd when USE_OPEN_FILE_CACHE is set to *yes*.

OPEN_FILE_CACHE_VALIDValues : <*any time value like Xs, Xm, Xh, ...*>Default value : *30s*Context : *global*, *multisite*The time after which cached elements should be validated when USE_OPEN_FILE_CACHE is set to *yes*.

USE_PROXY_CACHEValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*When set to *yes*, nginx will cache responses from proxied applications. See proxy_cache.

PROXY_CACHE_PATH_ZONE_SIZEValues : <*any valid size like Xk, Xm, Xg, ...*>Default value : *10m*Context : *global*, *multisite*Maximum size of cached metadata when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_PATH_PARAMSValues : <*any valid parameters to proxy_cache_path directive*>Default value : *max_size=100m*Context : *global*, *multisite*Parameters to use for proxy_cache_path directive when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_METHODSValues : <*list of HTTP methods separated with space*>Default value : *GET HEAD*Context : *global*, *multisite*The HTTP methods that should trigger a cache operation when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_MIN_USESValues : <*any positive integer*>Default value : *2*Context : *global*, *multisite*The minimum number of requests before the response is cached when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_KEYValues : <*list of variables*>Default value : $scheme$host$request_uriContext : *global*, *multisite*The key used to uniquely identify a cached response when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_VALIDValues : <*status=time list separated with space*>Default value : *200=10m 301=10m 302=1h*Context : *global*, *multisite*Define the caching time depending on the HTTP status code (list of status=time separated with space) when USE_PROXY_CACHE is set to *yes*.

PROXY_NO_CACHEValues : <*list of variables*>Default value : *$http_authorization*Context : *global*, *multisite*Conditions that must be met to disable caching of the response when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_BYPASSValues : <*list of variables*> Default value : *$http_authorization*Context : *global*, *multisite* Conditions that must be met to bypass the cache when USE_PROXY_CACHE is set to *yes*.

6.2 HTTPS

6.2.1 Let's Encrypt

AUTO_LETS_ENCRYPTValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*If set to yes, automatic certificate generation and renewal will be setup through Let's Encrypt. This will enable HTTPS on your website for free.You will need to redirect the 80 port to 8080 port inside container and also set the SERVER_NAME environment variable.

EMAIL_LETS_ENCRYPTValues : *contact@yourdomain.com*Default value : *contact@first-domain-in-server-name*Context : *global*, *multisite*Define the contact email address declare in the certificate.

6.2.2 HTTP

LISTEN_HTTPValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to no, nginx will not in listen on HTTP (port 80).Useful if you only want HTTPS access to your website.

REDIRECT_HTTP_TO_HTTPSValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*If set to yes, nginx will redirect all HTTP requests to HTTPS.

6.2.3 Custom certificate

USE_CUSTOM_HTTPSValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*If set to yes, HTTPS will be enabled with certificate/key of your choice.

CUSTOM_HTTPS_CERTValues : <*any valid path inside the container*>Default value :Context : *global*, *multisite*Full path of the certificate or bundle file to use when USE_CUSTOM_HTTPS is set to yes. If your chain of trust contains one or more intermediate certificate(s), you will need to bundle them into a single file (more info here).

CUSTOM_HTTPS_KEYValues : <*any valid path inside the container*>Default value :Context : *global*, *multisite*Full path of the key file to use when USE_CUSTOM_HTTPS is set to yes.

6.2.4 Self-signed certificate

`GENERATE_SELF_SIGNED_SSL`Values : *yes* | *no*Default value : *no*Context : *global*, *multisite*If set to yes, HTTPS will be enabled with a container generated self-signed certificate.

`SELF_SIGNED_SSL_EXPIRY`Values : *integer*Default value : *365* (1 year)Context : *global*, *multisite*Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the expiry date for the self generated certificate.

`SELF_SIGNED_SSL_COUNTRY`Values : *text*Default value : *Switzerland*Context : *global*, *multisite*Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the country for the self generated certificate.

`SELF_SIGNED_SSL_STATE`Values : *text*, *multisite*Default value : *Switzerland*Context : *global*, *multisite*Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the state for the self generated certificate.

`SELF_SIGNED_SSL_CITY`Values : *text*Default value : *Bern*Context : *global*, *multisite*Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the city for the self generated certificate.

`SELF_SIGNED_SSL_ORG`Values : *text*Default value : *AcmeInc*Context : *global*, *multisite*Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the organisation name for the self generated certificate.

`SELF_SIGNED_SSL_OU`Values : *text*Default value : *IT*Context : *global*, *multisite*Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the organisitional unit for the self generated certificate.

`SELF_SIGNED_SSL_CN`Values : *text*Default value : *bunkerity-nginx*Context : *global*, *multisite*Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the CN server name for the self generated certificate.

6.2.5 Misc

`HTTP2`Values : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, nginx will use HTTP2 protocol when HTTPS is enabled.

`HTTPS_PROTOCOLS`Values : *TLSv1.2* | *TLSv1.3* | *TLSv1.2 TLSv1.3*Default value : *TLSv1.2 TLSv1.3*Context : *global*, *multisite*The supported version of TLS. We recommend the default value *TLSv1.2 TLSv1.3* for compatibility reasons.

6.3 ModSecurity

`USE_MODSECURITY`Values : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, the ModSecurity WAF will be enabled.You can include custom rules by adding .conf files into the /modsec-confs/ directory inside the container (i.e : through a volume).

`USE_MODSECURITY_CRS`Values : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, the OWASP ModSecurity Core Rule Set will be used. It provides generic rules to detect common web attacks.You can customize the CRS (i.e. : add WordPress exclusions) by adding custom .conf files into the /modsec-crs-confs/ directory inside the container (i.e : through a volume). Files inside this directory are included before the CRS rules. If you need to tweak (i.e. : SecRuleUpdateTargetById) put .conf files inside the /modsec-confs/ which is included after the CRS rules.

`MODSECURITY_SEC_AUDIT_ENGINE`Values : *On* | *Off* | *RelevantOnly*Default value : *RelevantOnly*Context : *global*, *multisite*Sets the value of the SecAuditEngine directive of ModSecurity.

6.4 Security headers

`X_FRAME_OPTIONS`Values : *DENY* | *SAMEORIGIN* | *ALLOW-FROM https://www.website.net* Default value : *DENY*Context : *global*, *multisite*Policy to be used when the site is displayed through iframe. Can be used to mitigate clickjacking attacks. More info here.

`X_XSS_PROTECTION`Values : *0* | *1* | *1; mode=block*Default value : *1; mode=block*Context : *global*, *multisite*Policy to be used when XSS is detected by the browser. Only works with Internet Explorer.More info here.

`X_CONTENT_TYPE_OPTIONS`Values : *nosniff*Default value : *nosniff*Context : *global*, *multisite*Tells the browser to be strict about MIME type.More info here.

`REFERRER_POLICY`Values : *no-referrer* | *no-referrer-when-downgrade* | *origin* | *origin-when-cross-origin* | *same-origin* | *strict-origin* | *strict-origin-when-cross-origin* | *unsafe-url*Default value : *no-referrer*Context : *global*, *multisite*Policy to be used for the Referer header.More info here.

`FEATURE_POLICY`Values : *<directive> <allow list>*Default value : *accelerometer 'none'; ambient-light-sensor 'none'; autoplay 'none'; camera 'none'; display-capture 'none'; document-domain 'none'; encrypted-media 'none'; fullscreen 'none'; geolocation 'none'; gyroscope 'none'; magnetometer 'none'; microphone 'none'; midi 'none'; payment 'none'; picture-in-picture 'none'; speaker 'none'; sync-xhr 'none'; usb 'none'; vibrate 'none'; vr 'none'*Context : *global*, *multisite*Tells the browser which features can be used on the website.More info here.

`PERMISSIONS_POLICY`Values : *feature=(allow list)*Default value : accelerometer=(), ambient-light-sensor=(), autoplay=(), camera=(), display-capture=(), document-domain=(), encrypted-media=(), fullscreen=(), geolocation=(), gyroscope=(), magnetometer=(), microphone=(), midi=(), payment=(), picture-in-picture=(), speaker=(), sync-xhr=(), usb=(), vibrate=(), vr=()Context : *global*, *multisite*Tells the browser which features can be used on the website.More info here.

`COOKIE_FLAGS`Values : * *HttpOnly* | *MyCookie secure SameSite=Lax* | ...Default value : * *HttpOnly SameSite=Lax*Context : *global*, *multisite*Adds some security to the cookies set by the server.Accepted value can be found here.

`COOKIE_AUTO_SECURE_FLAG`Values : *yes* | *no*Default value : *yes*Context : *global*, *multisite*When set to *yes*, the *secure* will be automatically added to cookies when using HTTPS.

`STRICT_TRANSPORT_SECURITY`Values : *max-age=expireTime [; includeSubDomains] [; preload]*Default value : *max-age=31536000*Context : *global*, *multisite*Tells the browser to use exclusively HTTPS instead of HTTP when communicating with the server.More info here.

`CONTENT_SECURITY_POLICY`Values : *<directive 1>; <directive 2>; ...*Default value : *object-src 'none'; frame-ancestors 'self'; form-action 'self'; block-all-mixed-content; sandbox allow-forms allow-same-origin allow-scripts allow-popups allow-downloads; base-uri 'self';*Context : *global*, *multisite*Policy to be used when loading resources (scripts, forms, frames, ...).More info here.

6.5 Blocking

6.5.1 Antibot

`USE_ANTIBOT`Values : *no* | *cookie* | *javascript* | *captcha* | *recaptcha*Default value : *no*Context : *global*, *multisite*If set to another allowed value than *no*, users must complete a "challenge" before accessing the pages on your website :

- *cookie* : asks the users to set a cookie
- *javascript* : users must execute a javascript code
- *captcha* : a text captcha must be resolved by the users
- *recaptcha* : use Google reCAPTCHA v3 score to allow/deny users

ANTIBOT_URIValues : *<any valid uri>*Default value : */challenge*Context : *global*, *multisite*A valid and unused URI to redirect users when USE_ANTIBOT is used. Be sure that it doesn't exist on your website.

ANTIBOT_SESSION_SECRETValues : *random* | *<32 chars of your choice>*Default value : *random*Context : *global*, *multisite*A secret used to generate sessions when USE_ANTIBOT is set. Using the special *random* value will generate a random one. Be sure to use the same value when you are in a multi-server environment (so sessions are valid in all the servers).

ANTIBOT_RECAPTCHA_SCOREValues : *<0.0 to 1.0>*Default value : *0.7*Context : *global*, *multisite*The minimum score required when USE_ANTIBOT is set to *recaptcha*.

ANTIBOT_RECAPTCHA_SITEKEYValues : *<public key given by Google>*Default value :Context : *global*, *multisite*The sitekey given by Google when USE_ANTIBOT is set to *recaptcha*.

ANTIBOT_RECAPTCHA_SECRETValues : *<private key given by Google>*Default value :Context : *global*, *multisite*The secret given by Google when USE_ANTIBOT is set to *recaptcha*.

6.5.2 External blacklists

BLOCK_USER_AGENTValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, block clients with "bad" user agent.Blacklist can be found here and here.

BLOCK_TOR_EXIT_NODEValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*Is set to yes, will block known TOR exit nodes.Blacklist can be found here.

BLOCK_PROXIESValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*Is set to yes, will block known proxies.Blacklist can be found here.

BLOCK_ABUSERSValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*Is set to yes, will block known abusers.Blacklist can be found here.

BLOCK_REFERRERValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*Is set to yes, will block known bad referrer header.Blacklist can be found here.

6.5.3 DNSBL

USE_DNSBLValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to *yes*, DNSBL checks will be performed to the servers specified in the DNSBL_LIST environment variable.

DNSBL_LISTValues : *<list of DNS zones separated with spaces>*Default value : *bl.blocklist.de problems.dnsbl.sorbs.net sbl.spamhaus.org xbl.spamhaus.org*Context : *global*, *multisite*The list of DNSBL zones to query when USE_DNSBL is set to *yes*.

6.5.4 CrowdSec

USE_CROWDSECValues : *yes* | *no*Default value : *no*Context : *global*, *multisite*If set to *yes*, CrowdSec will be enabled. Please note that you need a CrowdSec instance running see example here.

CROWDSEC_HOSTValues : *<full URL to the CrowdSec instance API>*Default value :Context : *global*The full URL to the CrowdSec API.

CROWDSEC_KEYValues : *<CrowdSec bouncer key>*Default value :Context : *global*The CrowdSec key given by *cscli bouncer add BouncerName*.

6.5.5 Custom whitelisting

USE_WHITELIST_IPValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to *yes*, lets you define custom IP addresses to be whitelisted through the WHITELIST_IP_LIST environment variable.

WHITELIST_IP_LISTValues : *<list of IP addresses and/or network CIDR blocks separated with spaces>*Default value : *23.21.227.69 40.88.21.235 50.16.241.113 50.16.241.114 50.16.241.117 50.16.247.234 52.204.97.54 52.5.190.19 54.197.234.188 54.208.100.253 54.208.102.37 107.21.1.8*Context : *global*, *multisite*The list of IP addresses and/or network CIDR blocks to whitelist when USE_WHITELIST_IP is set to *yes*. The default list contains IP addresses of the DuckDuckGo crawler.

USE_WHITELIST_REVERSEValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to *yes*, lets you define custom reverse DNS suffixes to be whitelisted through the WHITELIST_REVERSE_LIST environment variable.

WHITELIST_REVERSE_LISTValues : *<list of reverse DNS suffixes separated with spaces>*Default value : *.google-bot.com .google.com .search.msn.com .crawl.yahoot.net .crawl.baidu.jp .crawl.baidu.com .yandex.com .yandex.ru .yandex.net*Context : *global*, *multisite*The list of reverse DNS suffixes to whitelist when USE_WHITELIST_REVERSE is set to *yes*. The default list contains suffixes of major search engines.

WHITELIST_USER_AGENTValues : *<list of regexes separated with spaces>*Default value :Context : *global*, *multisite*Whitelist user agent from being blocked by BLOCK_USER_AGENT.

WHITELIST_URIValues : *<list of URI separated with spaces>*Default value :Context : *global*, *multisite*URI listed here have security checks like bad user-agents, bad IP, ... disabled. Useful when using callbacks for example.

6.5.6 Custom blacklisting

USE_BLACKLIST_IPValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to *yes*, lets you define custom IP addresses to be blacklisted through the BLACKLIST_IP_LIST environment variable.

BLACKLIST_IP_LISTValues : *<list of IP addresses and/or network CIDR blocks separated with spaces>*Default value :Context : *global*, *multisite*The list of IP addresses and/or network CIDR blocks to blacklist when USE_BLACKLIST_IP is set to *yes*.

USE_BLACKLIST_REVERSEValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to *yes*, lets you define custom reverse DNS suffixes to be blacklisted through the BLACKLIST_REVERSE_LIST environment variable.

BLACKLIST_REVERSE_LISTValues : *<list of reverse DNS suffixes separated with spaces>*Default value : *.shodan.io*Context : *global*, *multisite*The list of reverse DNS suffixes to blacklist when USE_BLACKLIST_REVERSE is set to *yes*.

6.5.7 Requests limiting

USE_LIMIT_REQValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, the amount of HTTP requests made by a user for a given resource will be limited during a period of time.More info rate limiting here (the key used is $binary_remote_addr$uri).

LIMIT_REQ_RATEValues : *Xr/s* | *Xr/m*Default value : *1r/s*Context : *global*, *multisite*The rate limit to apply when USE_LIMIT_REQ is set to *yes*. Default is 1 request to the same URI and from the same IP per second.

LIMIT_REQ_BURSTValues : *<any valid integer>*Default value : *2*Context : *global*, *multisite*The number of requests to put in queue before rejecting requests.

LIMIT_REQ_CACHEValues : *Xm* | *Xk*Default value : *10m*Context : *global*The size of the cache to store information about request limiting.

6.5.8 Connections limiting

USE_LIMIT_CONNValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, the number of connections made by an ip will be limited during a period of time. (ie. very small/weak ddos protection)More info connections limiting here.

LIMIT_CONN_MAXValues : <*any valid integer*>Default value : *50*Context : *global*, *multisite*The maximum number of connections per ip to put in queue before rejecting requests.

LIMIT_CONN_CACHEValues : *Xm* | *Xk*Default value : *10m*Context : *global*The size of the cache to store information about connection limiting.

6.5.9 Countries

BLACKLIST_COUNTRYValues : <*country code 1*> <*country code 2*> ... Default value :Context : *global*, *multisite*Block some countries from accessing your website. Use 2 letters country code separated with space.

WHITELIST_COUNTRYValues : <*country code 1*> <*country code 2*> ... Default value :Context : *global*, *multisite*Only allow specific countries accessing your website. Use 2 letters country code separated with space.

6.6 PHP

REMOTE_PHPValues : <*any valid IP/hostname*>Default value :Context : *global*, *multisite*Set the IP/hostname address of a remote PHP-FPM to execute .php files.

REMOTE_PHP_PATHValues : <*any valid absolute path*>Default value : */app*Context : *global*, *multisite*The path where the PHP files are located inside the server specified in REMOTE_PHP.

6.7 Bad behavior

USE_BAD_BEHAVIORValues : *yes* | *no*Default value : *yes*Context : *global*, *multisite*If set to yes, bunkerized-nginx will block users getting too much "suspicious" HTTP codes in a period of time.

BAD_BEHAVIOR_STATUS_CODESValues : <*HTTP status codes separated with space*>Default value : *400 401 403 404 405 429 444*Context : *global*, *multisite*List of HTTP status codes considered as "suspicious".

BAD_BEHAVIOR_THRESHOLDValues : Default value : *10*Context : *global*, *multisite*The number of "suspicious" HTTP status code before the corresponding IP is banned.

BAD_BEHAVIOR_BAN_TIMEValues : Default value : *86400*Context : *global*, *multisite*The duration time (in seconds) of a ban when the corresponding IP has reached the BAD_BEHAVIOR_THRESHOLD.

BAD_BEHAVIOR_COUNT_TIMEValues : Default value : *60*Context : *global*, *multisite*The duration time (in seconds) before the counter of "suspicious" HTTP is reset.

6.8 misc

`SWARM_MODE`Values : *yes* | *no*Default value : *no*Context : *global*Only set to *yes* when you use *bunkerized-nginx* with *autoconf* feature in swarm mode. More info *here*.

`USE_API`Values : *yes* | *no*Default value : *no*Context : *global*Only set to *yes* when you use *bunkerized-nginx* with *autoconf* feature in swarm mode. More info *here*.

`API_URI`Values : *random* | <*any valid URI path*>Default value : *random*Context : *global*Set it to a random path when you use *bunkerized-nginx* with *autoconf* feature in swarm mode. More info *here*.

`API_WHITELIST_IP`Values : <*list of IP/CIDR separated with space*>Default value : *192.168.0.0/16 172.16.0.0/12 10.0.0.0/8*Context : *global*List of IP/CIDR block allowed to send API order using the `API_URI` uri.

CHAPTER

SEVEN

PLUGINS

Bunkerized-nginx comes with a plugin system that lets you extend the core with extra security features. To add a plugin you will need to download it, edit its settings and mount it to the `/plugins` volume.

7.1 Official plugins

- ClamAV : automatically scan uploaded files and deny access if a virus is detected

7.2 Community plugins

If you have made a plugin and want it to be listed here, feel free to create a pull request and edit that section.

7.3 Use a plugin

The generic way of using a plugin consists of :

- Download it to a folder (e.g. : myplugin/)
- Edit the settings inside the plugin.json files (e.g. : myplugin/plugin.json)
- Mount the plugin folder to the /plugins/plugin-id inside the container (e.g. : /where/is/myplugin:/plugins/myplugin)

To check if the plugin is loaded you should see log entries like that :

```
2021/06/05 09:19:47 [error] 104#104: [PLUGINS] *NOT AN ERROR* plugin MyPlugin/1.0 has␣
↪been loaded
```

7.4 Write a plugin

A plugin is composed of a plugin.json which contains metadata (e.g. : name, settings, ...) and a set of LUA files for the plugin code.

7.4.1 plugin.json

```
{
        "id": "myplugin",
        "name": "My Plugin",
        "description": "Short description of my plugin.",
        "version": "1.0",
        "settings": {
                "MY_SETTING": "value1",
                "ANOTHER_SETTING": "value2",
        }
}
```

The `id` value is really important because it must match the subfolder name inside the `plugins` volume. Choose one which isn't already used to avoid conflicts.

Settings names and default values can be choosen freely. There will be no conflict when you retrieve them because they will be prefixed with your plugin id (e.g. : `myplugin_MY_SETTING`).

7.4.2 Main code

```
local M                = {}
local logger        = require "logger"

-- this function will be called for each request
-- the name MUST be check without any argument
function M.check ()

        -- the logger.log function lets you write into the logs
        logger.log(ngx.NOTICE, "MyPlugin", "check called")

        -- here is how to retrieve a setting
        local my_setting = ngx.shared.plugins_data:get("pluginid_MY_SETTING")

        -- a dummy example to show how to block a request
        if my_setting == "block" then
                ngx.exit(ngx.HTTP_FORBIDDEN)
        end

end

return M
```

That file must have the same name as the `id` defined in the plugin.json with a .lua suffix (e.g. : `myplugin.lua`).

Under the hood, bunkerized-nginx uses the lua nginx module therefore you should be able to access to the whole **ngx.*** functions.

7.4.3 Dependencies

Since the core already uses some external libraries you can use it in your own plugins too (see the compile.sh file and the core lua files).

In case you need to add dependencies, you can do it by placing the corresponding files into the same folder of your main plugin code. Here is an example with a file named **dependency.lua** :

```
local M = {}

function M.my_function ()
        return "42"
end

return M
```

To include it from you main code you will need to prefix it with your plugin id like that :

```
...
local my_dependency = require "pluginid.dependency"

function M.check ()
        ...
        local my_value = my_dependency.my_function()
        ...
end
...
```