SECURING YOUR NGINX API GATEWAY

Yossi Koren

3SCALE INC. 450 Townsend Street, San Francisco, CA

Table of Contents

Steps to Improve & Secure your Nginx API gateway

After installing Nginx, you should gain a good understanding of Nginx's configuration settings which are found in nginx.conf. This is the main configuration file for Nginx and therefore most of the security checks will be done using this file. By default nginx.conf can be found in [Nginx Installation Directory]/conf on Windows systems, and in the /etc/nginx or the /usr/local/etc/nginx directories on Linux systems.

1. Disable any unwanted Nginx modules

Nginx modules are automatically included during installation of Nginx and no run-time selection of modules is currently supported, therefore disabling certain modules would require re-compilation of Nginx. It is recommended to disable any modules which are not required as this will minimize the risk of any potential attacks by limiting the operations allowed by the web server. To do this, you would need to disable these modules with the configure option during installation. The example below disables the auto index module, which generates automatic directory listings and recompiles Nginx.

```
# ./configure --without-http_autoindex_module
# make
# make install
```

2. Disable nginx server_tokens

By default the server_tokens directive in Nginx displays the Nginx version number in all automatically generated error pages. This could lead to unnecessary information disclosure where an unauthorized user would be able to gain knowledge about the version of Nginx that is being used. The server_tokens directive should be disabled from the Nginx configuration file by setting – server_tokens off.

3. Control Buffer Overflow Attacks

Buffer overflow attacks are made possible by writing data to a buffer and exceeding that buffers' boundary and overwriting memory fragments of a process. To prevent this in Nginx we can set buffer size limitations for all clients. This can be done through the Nginx configuration file using the following directives:

- **client_body_buffer_size** – Use this directive to specify the client request body buffer size. The default value is 8k or 16k but it is recommended to set this as low as 1k as follows: client_body_buffer_size 1k
- **client_header_buffer_size** – Use this directive to specify the header buffer size for the client request header. A buffer size of 1k is adequate for the majority of requests.
- **client_max_body_size** – Use this directive to specify the maximum accepted body size for a client request. A 1kdirective should be sufficient, however this needs to be increased if you are receiving file uploads via the POST method.
- **large_client_header_buffers** – Use this directive to specify the maximum number and size of buffers to be used to read large client request headers. A large_client_header_buffers 2 1k directive sets the maximum number of buffers to 2, each with a maximum size of 1k. This directive will accept 2kB data URI.

4. Disable any unwanted HTTP methods

It is suggested to disable any HTTP methods which are not going to be utilized and which are not required to be implemented on the web server. The below condition, which is added under the 'server' section in the Nginx configuration file will only allow GET, HEAD, and POST methods and will filter out methods such as DELETE and TRACE by issuing a 444 No Response status code.

```
if ($request_method !~ ^(GET|HEAD|POST)$ )
{
        return 444;
}
```

5. Make use of ModSecurity

ModSecurity is an open-source module that works as a web application firewall. Different functionalities include filtering, server identity masking, and null byte attack prevention. Real-time traffic monitoring is also allowed through this module. Therefore it is recommended to follow the ModSecurity manual to install the mod_security module in order to strengthen your security options.

6. Set up and configure Nginx access and error logs

Nginx access and error logs are enabled by default and are located at logs/error.log for error logs and at logs/access.log for access logs. The error_log directive in the Nginx configuration file will allow you to set the directory where the error logs will be saved as well as specify which logs will be recorded according to their severity level. For example, a 'crit' severity level will log important problems that need to be addressed and any other issues which have a higher severity level than 'crit'. To set the severity level of error logs to 'crit' the error_log directive needs to be set up as follows – error_log logs/error.log crit; A complete list of error_log severity levels can be found in the official nginx documentation available here.

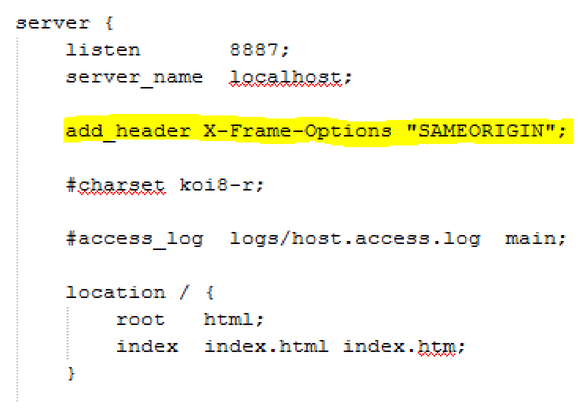
Alternatively, the access_log directive can be modified from the Nginx configuration file to specify a location where the access logs will be saved (other than the default location). Also the log_format directive can be used to configure the format of the logged messages as explained here.

7. Monitor Nginx access and error logs

Continuous monitoring and management of the Nginx log files will give a better understanding of requests made to your web server and also list any errors that were encountered. This will help to expose any attempted attacks made against the server as well as identify any optimizations that need to be carried out to improve the server's performance. Log management tools, such as logrotate, can be used to rotate and compress old logs in order to free up disk space. Also the ngx_http_stub_status_module module provides access to basic status information, and nginx Plus, the commercial version of Nginx, provides real-time activity monitoring of traffic, load and other performance metrics.

8. Configure Nginx to include an X-Frame-Options header

The X-Frame-Options HTTP response header is normally used to indicate if a browser should be allowed to render a page in a <frame> or an <iframe>. This could prevent clickjacking attacks and therefore it is recommended to enable the Nginx server to include the X-Frame-Options header. In order to do so the following parameter must be added to the nginx configuration file under the 'server' section – add_header X-Frame-Options "SAMEORIGIN";



```
server {
    listen       8887;
    server_name  localhost;

    add_header X-Frame-Options "SAMEORIGIN";

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location / {
        root   html;
        index  index.html index.htm;
    }
```

9. Protect Nginx from DDoS attacks

3scale's API Traffic Management agent doesn't provide out-of-the-box such threat protection features. Since the 3scale API proxy is built on NGINX open source there are modules available to perform DDOS prevention and to limit the number of requests forwarded to the back end service. Two modules in particular make this possible:

http://wiki.nginx.org/NginxHttpLimitReqModule

http://wiki.nginx.org/NginxHttpLimitZoneModule

10. Nginx block sql injection and file injection

Custom rules to block sql injection

Sql injection is referred to a code injection technique specifically made to attack data driven application in order to get some information from the database back to the attackers screen. This would only work if the application itself is not properly secured, eg. the input filter of the application does not filter properly the input data. In such cases the attack will abuse the security holes of the application and run the sql queries.

Currently I've been using the following custom rules to handle sql injection vulnerability:

```
server {
    [...]
    set $block_sql_status 0;
    if ($query_string ~ "union.*select.*\(") {
        set $block_sql_status 1;
    }
    if ($query_string ~ "union.*all.*select.*") {
        set $block_sql_status 1;
    }
    if ($query_string ~ "concat.*\(") {
        set $block_sql_status 1;
    }
    if ($block_sql_status = 1) {
        return 403;
    }
    [...]
}
```

```
    if ($block_sql_status = 1) {
        return 444;
    }
```

The above will return a 403 error, forbidden message, however you can use code 444 to reset connection without replying anything to the attacker.

Custom rules to block file injection

Same as with the sql injection code, only this time the attacker will try to inject a file, in most cases a shell script. In such cases we try to block again GET requests that contains the string “http://” or directory tree”../..”. Normally you should not use such codes in your links or images, but in certain cases, for example timthumb which uses the approach of using http:// to access the image and creates the thumb. We can create a whitelist for such cases of course, but we should really advise the user to not use such scripts.

The recommended code to block file injection is:

```
server {
    [...]
    set $block_file_status 0;
    if ($query_string ~ "[a-zA-Z0-9_]=http://") {
        set $block_file_status 1;
    }
    if ($query_string ~ "[a-zA-Z0-9_]=(\.\.//?)+") {
        set $block_file_status 1;
    }
    if ($query_string ~ "[a-zA-Z0-9_]=/([a-z0-9_.]//?)+") {
        set $block_file_status 1;
    }
    if ($block_file_status = 1) {
        return 403;
    }
    [...]
}
```

Again as before, we can use status code 444 instead of 403.

Common nginx block exploits

Using the same approach you can also block other requests that match "<script>" tag or "base64" strings in the query string, for eg:

```
    set $block_common_status 0;
    if ($query_string ~ "(<|%3C).*script.*(>|%3E)") {
        set $block_common_status 1;
    }
    if ($query_string ~ "base64_(en|de)code\(.*\)") {
        set $block_common_status 1;
    }
    if ($block_common_status = 1) {
        return 403;
    }
```

11. Add Extra Security to Nginx to Stop Clickjacking & XSS protection

Implementing Clickjacking within an Nginx instance is very straight forward. You can find number of ways to protect your Nginx from XSS. We can use Nginx to stop most of the attacks but for more sophisticated attacks recommend using Naxsi.

First we add the following lines in */etc/nginx/nginx.conf* so that site does not load in iframes. We need to add the following lines in http and add a default server which will return 444.

```
http {
    ...
    server {
            server_name  _;  #default
            return 444;
    }
}
```

If your site is a .com and the conf is in sites-enable folder. So add the following code at the top of */etc/nginx/sites-enable/a.com*

```
server {
    server_name      a.com;
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
    root             /var/www/a/;

    if ($host !~* ^(a.com)$ ) {
        return 444;
    }
    ...
}
```

This will stop the site being loading in iframe and give basic XSS protection. You can add following Javascript code to show black screen if any one try to load the site in iframe.

```
<script type="text/javascript">
    if (top != self) {
        window.document.write("<div style='background:black;opacity:0.5;
    }
</script>
```

12. Updates

As with any other server software, it is recommended that you always update your Nginx server to the latest stable version. These often contain fixes for vulnerabilities identified in previous versions, such as the directory traversal vulnerability that existed in Nginx versions prior to 0.7.63, and 0.8.x before 0.8.17. These updates frequently include new security features and improvements. Nginx security advisories can be found here and news about latest updates can be found here.