Setting Up ModSecurity + OWASP Core Rule Set + Nginx On AWS EC2

10 July, 2022. It was a Sunday

Posted under security web

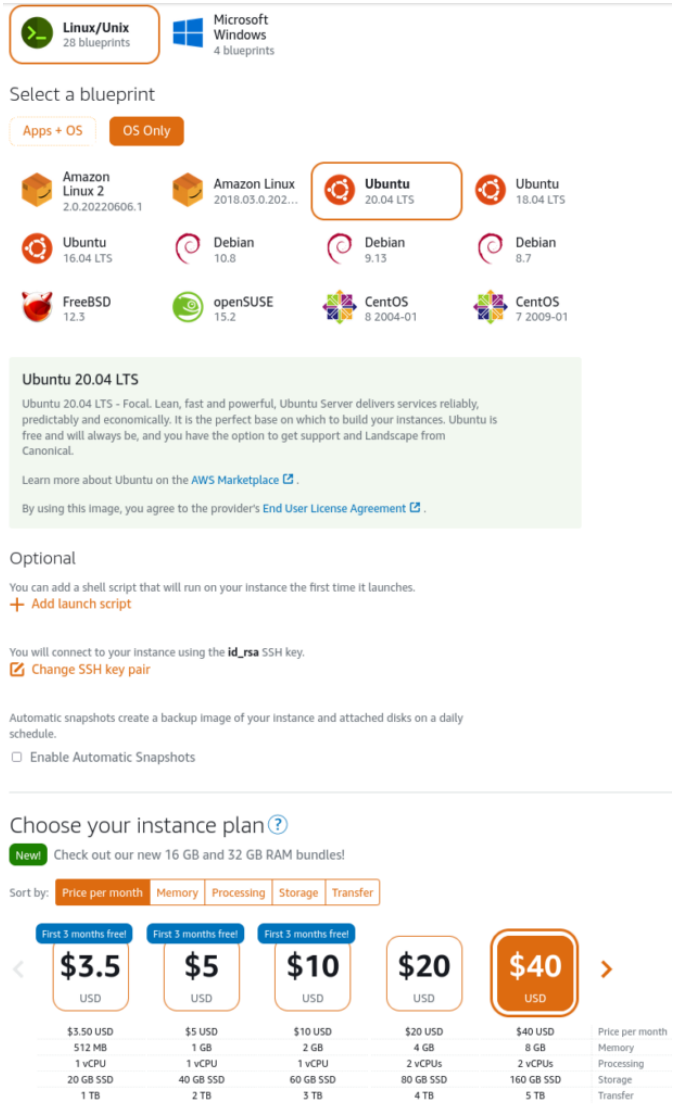
ModSecurity is a web application firewall. It can protect your web application from preying eyes of vulnerability scanners and attackers. It is extremely customizable, and when paired with OWASP's Core Rule Set, covers quite a lot of web technologies and frameworks.

In this article, we'll set up ModSecurity on an AWS EC2 Server running Nginx web server.

Table of contents

Set up Nginx on Ubuntu server

For this tutorial, we're using AWS LightSail's Ubuntu image. Choose any instance size depending on your requirements. I'll choose a 40$ / Month instance with 8GB RAM and 2vCPUs just so that the compilation of ModSecurity is faster.



Once the instance is created, log into the instance with SSH and update packages

```
$ apt update && apt upgrade -y
```

Install Nginx

```
$ sudo apt install nginx
```

Check what version of Nginx did we get from our package manager. This will be used when compiling Nginx later.

```
$ nginx -v
```

I got the following output:

```
nginx version: nginx/1.18.0 (Ubuntu)
```

To make sure the webserver is successfully installed and running, simply visit the IP address of the server. It should look something similar to this:

Set up ModSecurity

First we'll need to install compilation and other dependencies.

```
$ sudo apt-get install -y apt-utils autoconf automake build-essential git
```

Next we'll clone the ModSecurity repository into the /opt directory

```
$ cd /opt && sudo git clone --recursive https://github.com/SpiderLabs/ModS
```

Next we run the build script

```
$ sudo ./build.sh
```

Next we'll run the compile script that will fetch all the dependencies for the compilation

```
$ sudo ./configure
```

It is possible that this command fails and reports you of any dependencies that are still missing. You can simply google them with "install XYZ on Ubuntu" and run the configure command again. Ideally it will just exit without any errors.
Next we start with the actual compilation of ModSecurity

```
$ sudo make
```

A reason why I didn't go with the smallest server was that this step is resource intensive and could take 15 minutes or more depending on your server's CPU and memory.

If all went through, we can now install ModSecurity

```
$ sudo make install
```

If all went through without any errors, we have ModSecurity installed.

Set up ModSecurity <-> Nginx connector

We start off by downloading ModSecurity-Nginx and Nginx source code. **Note that the version of Nginx in the next command must match the version installed on our system**. For me, that's 1.18.0 but it could be different for you.

```
$ cd /opt && git clone https://github.com/SpiderLabs/ModSecurity-nginx.git
$ cd /opt && sudo wget http://nginx.org/download/nginx-1.18.0.tar.gz
```

Untar the Nginx source. Replace the Nginx version in the next command if needed.

```
$ sudo tar -xvf nginx-1.18.0.tar.gz
```

Next, we need to grab configure arguments. For that, run the nginx command with a capital 'V' flag.

```
$ nginx -V
nginx version: nginx/1.18.0 (Ubuntu)
built with OpenSSL 1.1.1f  31 Mar 2020
TLS SNI support enabled
configure arguments: --with-cc-opt='-g -O2 -fdebug-prefix-map=/build/ngin
```

Note the "configure arguments" in the command's output. We need to build modsecurity with these arguments.

```
$ sudo ./configure --add-dynamic-module=../ModSecurity-nginx <paste config
```

For example, here's what I'll run:

```
$ sudo ./configure --add-dynamic-module=../ModSecurity-nginx --with-cc-opt
```

Don't copy the above command. You must use the configure arguments supplied by your installation of Nginx.

Next we build the modules

```
$ sudo make modules
```

This is a compilation step and may take a little while (a minute or so) to complete. The final step here is to copy the compiled modules to a place from where we can reference them from our Nginx config.

```
$ sudo mkdir /etc/nginx/modules
$ sudo cp objs/ngx_http_modsecurity_module.so /etc/nginx/modules
```

Loading ModSecurity module in Nginx

Simply add the following line to the nginx config file at `/etc/nginx/nginx.conf` outside any block.

```
load_module /etc/nginx/modules/ngx_http_modsecurity_module.so;
```

Set up OWASP Core Rule Set

OWASP's Core Rule Set is a set of rules that cover most common frameworks and technologies as well as cover signatures for common web application attack payload. It is a good place to start if you don't want to write custom rules for many common attacks.

First, we'll clone the modsecurity-crs repository

```
$ sudo git clone https://github.com/coreruleset/coreruleset /opt/corerules
```

Then we'll rename the crs-setup config file

```
$ sudo mv /opt/coreruleset/crs-setup.conf.example /opt/coreruleset/crs-set
```

Activate the default exclusion rule file.

```
sudo mv /opt/coreruleset/rules/REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf
```

Next we create a place for coreruleset to live within `/etc/nginx` and copy some additional config files.

```
$ sudo mkdir -p /etc/nginx/modsec
$ sudo cp /opt/ModSecurity/unicode.mapping /etc/nginx/modsec
$ sudo cp /opt/ModSecurity/modsecurity.conf-recommended /etc/nginx/modsec/
```

Next we create a config file that will include our main ModSecurity config file and CRS setup files.

```
$ sudo touch /etc/nginx/modsec/main.conf
```

And add the following lines to the config file.

```
Include /etc/nginx/modsec/modsecurity.conf
Include /opt/coreruleset/crs-setup.conf
Include /opt/coreruleset/rules/*.conf
```

Finally, we reference this main.conf file from our Nginx config.

```
$ sudo vim /etc/nginx/sites-available/default
```

And add the following line within the server block.

```
modsecurity on;
modsecurity_rules_file /etc/nginx/modsec/main.conf;
```

Turn ModSecurity "on" and test XSS payload

So far, we've configured everything but if we restart Nginx now, it won't filter attacks but only detect them since the default operating mode of ModSecurity is to only log malicious requests. To change that, let's open the file `/etc/nginx/modsec/modsecurity.conf` and change the line

```
SecRuleEngine DetectionOnly
```

to

```
SecRuleEngine On
```

For our changes to go live, we'll need to restart Nginx.

```
$ sudo systemctl restart nginx
```

Let's test our ModSecurity installation. Open your browser and send a sample payload in the GET parameter. It doesn't have to be a real parameter, but just something that can trigger an XSS filter.

```
http://[server-ip]/index.html?xss-payload=<script>alert(1)</script>
```

That's ideal. ModSecurity is working and blocking seemingly malicious requests to our web server. Now any application that sits behind our web server will be protected against many generic web application attacks, even OWASP Top 10 thanks to OWASP's CoreRuleSet.

In conclusion

It isn't the most straightforward of installations, but it isn't very difficult either. The hard part, however, starts here and it is to get rid of all false positives and tweak the installation such that it fits the needs of your specific web application. Depending on how complex an application you're trying to protect, it can be fairly time consuming.

I'll write an article on how to tweak the parameters of ModSecurity and make it fit our needs in the future in a separate article.

That's it for this article, thank you for reading!

Unrelated

- Trek To Bramhatal (Uttarakhand)
- Privacy - How I Converted
- Jagriti Yatra 2017
- Private Cloud Part 2 | Encrypted Storage With NextCloud
- ELI5 - How HTTPS Works

« WordPress Security Checklist: How To Secure Your WordPress Website

Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF) And Server Side Request Forgery (SSRF) »
