Introduction

Nginx is a widely used Powerful **web server** software. It has the ability to manage traffic even with the small resources.

The web server software is the **entry point** to a **server**, unless and until a server has the web server, it can't be accessed from a browser using the IP.

The Web Server becomes the main **source for hackers** to steal your data When you do not take care of **security measures** of your web server.

I know how difficult it is for you to prevent the **hacking activity** and **secure nginx server**

security standards and **Nginx security hardening process**.

Here in this article you will learn,

- How to prevent information disclosure
- How to secure Nginx Server
- Protecting Server with SSL
- Create IP Restriction
- How to conduct the Security Audit
- Additional Security Measures

Let us see the Nginx security best practices to protect your site and server.

Prerequisites

- Ubuntu 14.04 Server with **Sudo non-root user**.
- You should have installed and **configured Nginx** on your server.
- A **domain pointing to the server**.

1) Ubuntu Update package

The first and necessary step to prevent the loopholes in **software** is **updating** them.

Whenever the software developer finds security threat and loopholes, they will **fix the issue** and release an updated version or **release update** for that security problem.

So you have to **frequently update the software** whenever you get the new version is released.

But:

Before Going for any software update, first, **check** whether the **update** will **create an issue or not** and then install.

However, Most of the time, you won't face any problem.

To update the packages and software all at once, use the below command.

```
$ sudo apt-get update && sudo apt-get upgrade
```

Instead of that, you can just **update the Nginx** in the Ubuntu Repository. It will update the Nginx packages and dependencies of Nginx.

```
$ sudo apt-get upgrade nginx
```

2) Preventing Information Disclosure

To **prevent** the web server **information** from being **disclosed** to all, we have to make some changes in the **Nginx Configuration**.

The information is leaked from HTTP header and application error reporting. You can see the **Nginx version** and name using the following command.

```
$ curl -I http://localhost
```

The Output will look like this.

```
Output of curl -I http://localhost
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
...
```

Look at the output of the command.

The output contains the information about server **operating system and Nginx Version**. This may not be a serious problem.

But it can't be always like that.

So, we have to always **keep safe** of our **data**.

First, Open the **Nginx configuration file** using **nano editor**.

```
$ sudo nano /etc/nginx/nginx.conf
```

Look for ***http*** configuration section and add the line *server_tokens off* as mentioned below.

```
/etc/nginx/nginx.conf

http {

    ##
    # Basic Settings
    ##
    server_tokens off;
```

...

After that, Save and Exit the file.

Now:

Reload the Nginx to see the effect.

```
$ sudo service nginx reload
```

Let us try to look at the **web server information**.

```
$ curl -I http://localhost
```

This time, you can only see less information.

```
Server: nginx
...
```

The Server **Operating system and Nginx version** are not displayed in the above output.

Still, you might be thinking of re**moving the information** about the **server**.

But:

This is not easy as it does not have any configuration option to hide. You can still do that by **recompiling the Nginx** from source.

It is not easy task and also it is not worth to do.

There is one more **header** which contains more **sensitive information** and it displays the **version of PHP**, **Tomcat or other engines** behind the Nginx.

In case you run the **Nginx with PHP**, The output of the *curl* will look like this.

```
Output of curl -I http://localhost on nginx with php
HTTP/1.1 200 OK
Server: nginx
...
X-Powered-By: PHP/5.5.9-1ubuntu4.14
...
```

The ***X-powered-By header*** shows the PHP version and the operating system version.

If the hacker knows the version, he will try to use the existing **vulnerability** of the software and **operating system** to gain access.

So, it is very important to hide this information.

You **cannot hide this information** using the Nginx configuration file.

Because there is **no configuration option** for that.

The **default option** will be set to "**on**".

The next step is to change the **4XX(error) pages**. Those pages can become the resource for the hackers.

The **Unauthorized 401 and Forbidden 403 error pages** need not be shown to the regular user unless you are debugging.

If you still want to know about this errors, you can find them in the **Nginx error log** ***(/var/log/nginx/error.log)***.

Open the configuration file and go to server block.

```
$ sudo nano /etc/nginx/sites-enabled/default
```

Specify the error in the server configuration part.

```
/etc/nginx/sites-enabled/default

server {
...
   error_page 401 403 404 /404.html;
...
```

Save and exit the file.

Then **reload the Nginx** for the **new settings** to load and work.

```
sudo service nginx reload
```

This is how we have to hide the small information as much as possible.

The more you hide, the higher the server security will be. We have to always **restrict the amount of information** to **minimal**.

When we go for higher **security essentials**, we can't ignore the **SSL**.

They are the main thing which **encrypts and decrypts** data at **client and server side**.

So, the **traffic is entirely encrypted**.

It is always important to **secure the traffic** on the internet as hackers will try to breach the server protection through the traffic.

You can get **free SSL for your Nginx server** and that will help you in basic level protection.

But:

It is still not enough to **secure the server**. The hackers can **decrypt your data** with a little effort.

So, we are going to **create a strong SSL** with **stronger SSL encryption algorithms**.

The **SSL** will be compliant with **security standards and practices**.

Now:

Let us see **how to create SSL certificate for Nginx using SHA256**.

First, **create a directory** for **SSL**.

```
$ sudo mkdir /etc/nginx/ssl/
```

We have to **create an SSL certificate** with the strong signature algorithm.

For our testing purpose, we are going to **create a self-signed SSL certificate** and ignore the warning about SSL certificate.

Here is the command to **generate SSL certificate**.

The certificate will ask you few pieces of information about your business. Enter the information.

After that, the command will create a **2048 bit RSA encrypted key** at ***/etc/nginx/ssl/nginx.key*** and ***SHA256 certificate*** at /etc/nginx/ssl/nginx.crt.

Then, you have to **create a 4086 bit DH parameters** using the following command. Creating dhparam 4096 is a time-consuming process and you have to wait for some time.

Based on the server capacity, the time will vary. Use the Nginx ssl_dhparam like below.

```
$ sudo openssl dhparam -out /etc/nginx/ssl/dhparam.pem 4096
```

Let us **configure the SSL** in our server. Open the Nginx default configuration file.

```
$ sudo nano /etc/nginx/sites-enabled/default
```

Go to the server section and Look for the ***server_name*** directive.

In that add the following lines after ***server_name***.

```
/etc/nginx/sites-enabled/default

server {
...
   server_name localhost;

    ### SSL Part
    listen 443 ssl;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH';
    ssl_prefer_server_ciphers on;
    ssl_dhparam /etc/nginx/ssl/dhparam.pem;
    ssl_certificate /etc/nginx/ssl/nginx.crt;
```

Let me explain the directives we used in the above configuration.

Listen: This directive enables the SSL listeners on HTTPS port (Port 443).

ssl_protocols: It enables three secure protocols, they are TLSv1, TLSv1.1, TLSv1.2.

ssl_ciphers: Enables the three secure Nginx SSL ciphers, they are **EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH**. The Nginx SSL ciphers highly protects the server.

ssl_prefer_server_ciphers: Ensures that the client agree with the Server's cipher choice.

ssl_dhparam: It uses the DH parameters we have generated.

ssl_certificate: It uses the self signed certificate generated by us. You have to change it if you are using another SSL certificate.

ssl_certificate_key: It uses the SSL private key.

Reload the Nginx to make the changes to work.

```
$ sudo service nginx reload
```

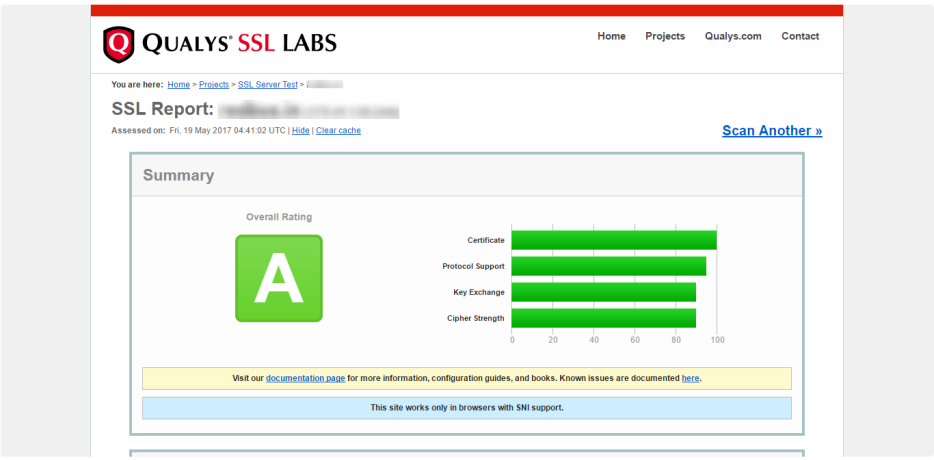
We have to check our **SSL configuration**. We can use the external tools.

The **SSL LABS** is one of the website where you can **check the SSL**.

When you input your domain name and test it for SSL, it will show warning that the SSL Certificate is not trusted.

It is because we are using the **Self Signed SSL certificate** and you can just ignore the warning.

The result will be **"T"**. If the trust issues(Since it is a self signed certificate") are ignored, then the Grade will be **"A"**.



This is how the result would be if you are using a SSL certificate issued by trusted provider.

You may want to **remove the Self signed SSL certificate** to get the Grade **"A"**.

To do this, you can use **Let's encrypt SSL certificates**. It is free and let you set the **RSA key size upto 4096**.

You can also use **SSL** from third parties and ensure that you choose **SHA256 Certificate**.

4) Restrict the IPs from the Access

Security is always not about setting strong password. You have to **keep sensitive areas** of your website **away from the attackers**.

The sensitive area such as **PHPMyadmin, CPanel** and some other pages can be accessed easily. Attacker can leverage this opportunity to gain access to your website.

If you have a wordpress site, the sensitive page like ***/wp-admin*** can be accessed from anywhere else.

If you have **weak password**, then the attacker can easily gain the access to your site.

All other **IPs** should be blocked from the access.

You can **enable the IP restriction** through the Nginx configuration.

Open the Nginx Configuration.

```
sudo nano /etc/nginx/sites-enabled/default
```

Look at the **server configuration** part and add the following line.

```
 /etc/nginx/sites-enabled/default
 server {
 ...
location /wp-admin/ {
    allow 192.168.1.1/24;
 allow 10.0.0.1/24;
    deny  all;
}
...
...
```

Make sure to **change the IP address** above mentioned with your **IP address**.

You can also **enable access to specific port** by changing the **Network mask**. Also you can allow the access for other **IP address**.

Reload the Nginx.

```
$ sudo service nginx reload
```

If you try to access the ***/wp-admin*** page from a IP address which is not enabled to access the page, you will get **402 forbidden error**.

You can also see the error in the error logs at Nginx.

```
$ sudo tail /var/log/nginx/error.log
```

The **402(access forbidden) error** will look like this.

```
Output of sudo tail -f /var/log/nginx/error.log
...
2016/01/02 04:16:12 [error] 4767#0: *13 access forbidden by rule, client: X.
```

It is always good to have more than one year security precautions such as **changing error pages** as well as **restricting IP** will improve the security.

When you restrict the IP address for wordpress admin page, the **attacker or automated tool** will see the **Page not found error**. This will discourage them from trying further.

5) Security Audit

Our next step is to conduct security audit.

We can secure the server with **initial security precautions.**

But:

It is not possible to find all the security threats manually.

So, we are going to use a third party **web vulnerability checking tool** which will give us more details.

Let us use the **wapiti** a open source web vulnerability checking tool.

It is free but it has limited options when compared to some advanced tools.

Install Wapiti using the following command.

Using wapiti website vulnerability scanner is very easy. Execute the below command.

```
$ wapiti http://example.org -n 10 -b folder
```

you have to replace the example.org with your domain name.

Here we have mentioned two arguments. They are **-n 10 and -b**. Here the **-n** sets the number of same pattern URL to be checked to 10.

So, there will be no endless checking due to the loop.

The **-b** sets the motive to check only the given **domain name**.

The processed result will be saved in the directory called ***generated_report***.

The directory from where you have started the scanning will have the report.

For better view, you can download the file.

The website security scan is necessary for all websites. If you are not going for website security test, then you are leaving the opportunity for the hackers.

Open the index.html file in the browser.

In that Report, you can **see the vulnerabilities** under **10 categories** and their **severity level** will be displayed.

These 10 categories are

- SQL Injection
- Blind SQL Injection
- File Handling
- Cross Site Scripting
- CRLF

- Htaccess Bypass
- Backup file
- Potentially dangerous file.

If you see any vulnerability, you can expand the corresponding section and look for more details.

Run this type of **scan frequently** using different tools to find out the vulnerabilities and fix them.

This way you can **secure your Nginx and websites**.

Additional Security Measures

6) Installing and configuring Naxsi

Naxsi is a web application firewall for Nginx. It protects the web server from the large range of vulnerabilities.

Naxsi is powerful and complex to configure. It will take much time and effort to configure it.

But:

There are some configuration comes with the Firewall which you can use and customize further if you require.

7) Protect server with Fail2Ban

Fail2Ban is a great **security tools** which protects the web server against attacks.

We have already **blocked IP ranges** and some part of our website to prevent the public to access them.

This tool will **ban attackers** for some **time period** when it **find** that the user perform some **malicious activities**.

Monitoring is always **important** since it can give you information about system activity.

We can do monitoring on Ubuntu server with **Monit** which supports Nginx.

The tool not only shows the **Track record of Malicious activity** but also shows the **CPU Load and Memory usage**.

This will help you to identify the suspicious activity on your website.

You can set customer alerts for some security events.

When someone tries to **access** your **website's sensitive area**, you will receive **notification**.

You also need to **setup firewalls using IP tables**. You have to set the **https port(tcp 443)** to allow the incoming connections.

Installing Aide improves the security of the server. This tool notify you if any changes made in the system file and directory.

If a file is **added or deleted from your server**, you will get notification for that.

Conclusion:

In this article you have learned the following

- **updating the software packages**.
- Preventing the **server Information** being **disclosed**.
- Configuring and testing the **SSL**.
- Creating **Restriction** for certain **IPs**.
- **Performing security audit** with **wapiti**
  and taking some additional steps to **secure the site**.

Securing a server is hard task and when you make some **initial level server security setup**, you can almost prevent a lot of security attacks.

So always be **aware and update your security precautions**. If you still have any problem in securing your server, let us know the difficulty in command, we will help you.

UBUNTU

NGINX

AUTHOR
Selvakumar

I am an Online Marketer and technology lover. I like to learn new things and share that with people.