Linnéuniversitetet
Kalmar Växjö

Report

Exam Assignment 3

Github Webhook App

Författare: Helena Tevar
Termin: HT-19
Ämne: Server-based Web Programming
Kurskod: 1DV523

Innehållsförteckning

Address

https://cscloud603.lnu.se/

Application security

The application includes security solutions as:

- The server runs in a reversed proxy (Nginx) that acts as a fasade from our Node server.
- The proxy runs in HTTPS
- The fetching of issues from Github is done by Octonode, using a Github token.
- The post request from Github is checked using the x-hub-signature.
- The client side javascript is minimum, has not access to any credential nor Github data or Github API.
- The client does not save any kind of information. If there are no cookies, the monsters will not try to eat them.

Describe your application

Reversed Proxy

The server uses Nginx as reverse proxy. It creates a facade in front of the Node server that hides its origin and provides a secure later TLS. Nginx also listens to HTTP and re-route the calls to the HTTPS port. Nginx then calls the localhost server that is our actual application server in Node Express.

```
server {
        listen 80;
        server_name cscloud603.lnu.se;
        return 301 https://$server_name$request_uri;
}

server {
    listen 443;
    listen [::]:443;
    ssl on;
    ssl_certificate /var/www/githook/config/sslcerts/cert.pem;
    ssl_certificate_key /var/www/githook/config/sslcerts/key.pem;
    ssl_ciphers EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH;
    ssl_prefer_server_ciphers on;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    server_name cscloud603.lnu.se;
    location / {
        proxy_pass http://localhost:3000/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
```

```
        proxy_set_header Connection 'upgrade';
    }
}
```

Process Manager

The application uses PM2 as Process Manager That helps to manage servers in production, like reloading, monitoring, or handling many servers at the same time. In this case, it is used to reload if the application crashes. However, the application is so simple that it is very robust, so PM2 was mainly used to monitor the websocket during the final implementation.

TLS Certificates

Nginx uses an HTTPS layer by using a self signed certificate. This certificate adds the connection a layer of security. It makes the connections secure, encrypts the data and makes it difficult to get the information transmitted. It would be better to have a certificate from a major company, but for this assignment I used a self-signed certificate created in the server by using a script.

```
#!/bin/bash

echo "Generating self-signed certificates..."
mkdir -p ./config/sslcerts
openssl genrsa -out ./config/sslcerts/key.pem 4096
openssl req -new -key ./config/sslcerts/key.pem -out
./config/sslcerts/csr.pem
openssl x509 -req -days 365 -in ./config/sslcerts/csr.pem
-signkey ./config/sslcerts/key.pem -out
./config/sslcerts/cert.pem
rm ./config/sslcerts/csr.pem
chmod 600 ./config/sslcerts/key.pem ./config/sslcerts/cert.pem
```

Development vs Production

I tried my best to apply environmental variables and git branches for development and production. During Development I would use specific code that would test, for instance, the WebSocket emissions and events or using modules like Morgan as loggers to test in localhost. When deploying, the code would be omitted and the development modules not used. When deploying, the server would only download modules that are required for development only.

Extra Modules

I made this assignment with the basic modules from Express and Handlebars. Apart from these, I used:

- **Octonode**: A module to manage the Github API with ease.

- **Crypto**: A module to perform a cryptographic test on the x-hub-signature from the post request from Github
- **Socket IO**: A module to manage the WebSocket from server and client side.

Extra Features

Dreams and Hopes

Time and Holydays were a high impediment for this assignment. I began having some expectations for the application that I had to downgrade to the most basics. Anyways, the expectations I had with this application were:

- Have a Github OAuth that would check that I was actually the owner of the repo to improve privacy.
    - I would check saved data in my credentials with the one provided by Github OAuth to return the authorization result
- A Guard service that would allow me to see the issues from my repo, and only me, kicking out of the app everybody not authorised.
- Add functionalities:
    - CRUD comments / issues with Octonode
    - Create hooks for ALL repos with Octonode
    - Improve the front-end by adding a drop list in the navbar with all the repos and be able to see the issues from the repo selected.
- Add even more functionalities using Octonode:
    - Apply the step before for ANY user logged using Github OAuth.
        - I would get the Github Token from OAuth and use it with Octonode to create Webhooks for all the repos.
        - Even way better, create a form to pick which repos the user wants to hook with.