In this lab, we will be securing our application using HTTPS to enable secure communication when; complete please upload a text file with a link to your secured pipeline here.



Navigate to the directory where your private key for your ec2 instance is and run the command: ssh -i "devops-ec-access.pem" [ubuntu@ec2-18-212-241-106.compute-1.amazonaws.com](mailto:ubuntu@ec2-18-212-241-106.compute-1.amazonaws.com)

1. Ensure you replace the devops-ec-access.pem with your pem file
2. Ensure your replace [ubuntu@ec2-18-212-241-106.compute-1.amazonaws.com](mailto:ubuntu@ec2-18-212-241-106.compute-1.amazonaws.com) with the host address of your machine; this can be found in your   
   AWS EC2 console

A screenshot of a computer

Description automatically generated with medium confidence

First run the command: sudo apt-get install openssl

This will install openssl on your machine.

Then run the command:

openssl req -x509 -newkey rsa:4096 -keyout privatekey.pem -out server.crt -days 10000 -nodes -subj "/C=IE/ST=Leinster/L=Dublin/O=National College of Ireland/OU=School of Computing/CN=ncirl.ie"

This openssl command is going to generate a self-signed PKCS #10 certificate (instead of a certificate request). This certificate is going to be used by our browser when our application is loaded to allow our application to work over https.

However, self-signed certificates are not validated with by any third party so chrome will show us a warning when using our application.

For future reference if we did not want this warning we would need a certification authority (CA) to issue us a certificate. A CA acts as a trusted third party—trusted both by the subject (owner) of the certificate and by the party (end user) relying upon the certificate.

Let's Encrypt a nonprofit Certificate Authority which provides TLS certificates to 260 million websites recommends certbot for issuing these certificates (however a domain name is required for this).

<https://letsencrypt.org/getting-started/>

<https://certbot.eff.org/>

After running this command we will have our server.crt (our certificate) and a privatekey.pem which uses RSA algorithm to generate the certificate and private key. The 4096 is the amount of bits which is generated for your private key.

The -days specifies how many days until our certificate expires in this case it is 10000.

Graphical user interface

Description automatically generated

Run the command: cat privatekey.pem

To display the contents of the private key and copy the output of this command.

Now in CircleCI add a new environment variable called "PRIVATE\_KEY" and paste the copied private key for the value of the environment variable.

A computer screen with a blue background

Description automatically generated with low confidence

Run the command: cat server.crt

To display the contents of the certificate and copy the output of this command.

Now in CircleCI add a new environment variable called "SERVER" and paste the copied certificate for the value of the environment variable.

Text

Description automatically generated

Update the deploy.sh to ensure the private key and server key from CircleCI are passed to the application

Text

Description automatically generated

Update line 45 to ensure that the private key and and server certificate are passed to the EC2 instance using the export command this will create environment variables called SERVER and PRIVATE\_KEY in the EC2 instance.

var app = require('../app');

var debug = require('debug')('phishsense:server');

var http = require('http');

/\*\*

\* Normalize a port into a number, string, or false.

\*/

function normalizePort(val) {

var port = parseInt(val, 10);

if (isNaN(port)) {

// named pipe

return val;

}

if (port >= 0) {

// port number

return port;

}

return false;

}

/\*\*

\* Event listener for HTTP server "error" event.

\*/

function onError(error) {

if (error.syscall !== 'listen') {

throw error;

}

var bind = typeof port === 'string'

? 'Pipe ' + port

: 'Port ' + port;

// handle specific listen errors with friendly messages

switch (error.code) {

case 'EACCES':

console.error(bind + ' requires elevated privileges');

process.exit(1);

break;

case 'EADDRINUSE':

console.error(bind + ' is already in use');

process.exit(1);

break;

default:

throw error;

}

}

/\*\*

\* Event listener for HTTP server "listening" event.

\*/

function onListening() {

var addr = server.address();

var bind = typeof addr === 'string'

? 'pipe ' + addr

: 'port ' + addr.port;

debug('Listening on ' + bind);

}

// Add HTTPS Section

var fs = require('fs');

var https = require('https');

var port = normalizePort(process.env.PORT || '8080');

var https\_port = process.env.PORT\_HTTPS || 8443;

var options = {}

if(process.env.ENV !== "DEV") {

var privatekey = fs.readFileSync('privatekey.pem', "utf8")

console.log(typeof privatekey)

var cert = fs.readFileSync('server.crt', "utf8")

var header = "-----BEGIN PRIVATE KEY-----"

var footer = "-----END PRIVATE KEY-----"

console.log(privatekey.split(header))

privatekey= privatekey.split(header)[1]

privatekey = privatekey.split(footer)[0]

privatekey = header + "\n" + privatekey.replace(/ /g, "\n") + footer+"\n"

console.log(privatekey)

var header = "-----BEGIN CERTIFICATE-----"

var footer = "-----END CERTIFICATE-----"

cert= cert.split(header)[1]

cert = cert.split(footer)[0]

cert = header + "\n" + cert.replace(/ /g, "\n") + footer+"\n"

var options = {

key : privatekey,

cert : cert

};

app.set("port",https\_port);

/\*

° Create HTTPS server.

\*/

var server = https.createServer(options, app).listen(https\_port, function () {

console.log('Magic happens on port ' + https\_port);

});

/\*\*

\* Listen on provided port, on all network interfaces.

\*/

server.on('error', onError);

server.on('listening', onListening);

// Redirect from http port to https

http.createServer(function (req, res) {

res.writeHead(301, { "Location": "https://" + req.headers['host'].replace(port,https\_port) + req.url });

console.log("http requet, will go to >> ");

console.log("https://" + req.headers['host'].replace(port,https\_port) + req.url );

res.end();

}).listen(port);

} else {

var server = http.createServer(app);

/\*\*

\* Listen on provided port, on all network interfaces.

\*/

server.listen(port);

server.on('error', onError);

server.on('listening', onListening);

Update your bin/www file in the folder of your application, the application will now be hosted on port 8443 using https. This file will also be provided on Moodle and will be called www

Text

Description automatically generated

Text

Description automatically generated

Now navigate to your application in your local and let's update our deployed instance of the application run the commands:

* git status
* git add .
* git commit -m"updated pipeline to be secured with https"
* git push –set-upstream origin add\_home\_link

This will push our changes to the add\_home\_link branch

Graphical user interface, text, application, email

Description automatically generated

Let's now merge our changes into our master branch click on pull requests

Graphical user interface, text, application, email, Teams

Description automatically generated

Click on new pull request

Graphical user interface, text, application

Description automatically generated

Now select the branch to be merged into to be master and the branch merged from to be add\_home\_link and create the pull request. As our application will redeploy when pushed up to main.

Graphical user interface, text, application, email

Description automatically generated

We can see the branches are able to merge without conflicts now click the create pull request button

Graphical user interface, text, application, email

Description automatically generated

Click merge pull request.

Text

Description automatically generated

Let's now update our local machines main branch with the most recent changes.

To switch back to our main branch run the command: git checkout main

Text

Description automatically generated

Now to receive the most up to date changes run the command: git pull

Graphical user interface, application

Description automatically generated

Now remove the previous inbound traffic rules we had.

Add the following inbound for Custom TCP on port 8443:

* For 0.0.0.0/0 (ipv4 addresses)
* For ::/0 (ipv6 addresses)

Then click safe rules in a production environment we should never leave ports open that don't have any use.

A principal of securing systems is the Principle of Least Privilege which states that a subject should be given only those privileges needed for it to complete its task. If a subject does not need an access right, the subject should not have that right.

In our case our application does not need these ports open on the production web server and thus should not have these ports open as these could pose a security risk.

Graphical user interface, application, Teams

Description automatically generated

When we navigate to our application in this case <https://18.212.241.106:8443/math/add> (note your URL will be different as your application is hosted on another AWS instance) we are faced with a warning screen this is as we are using a self-signed certificate. To access our application we can click advanced.

Graphical user interface, text, application, Teams

Description automatically generated

Then click proceed.

Graphical user interface

Description automatically generated

Fantastic our application is now using https! We can now use the application the same as before