Web Application : Astral
By Abdelrahman Ashraf 37-11102
T15

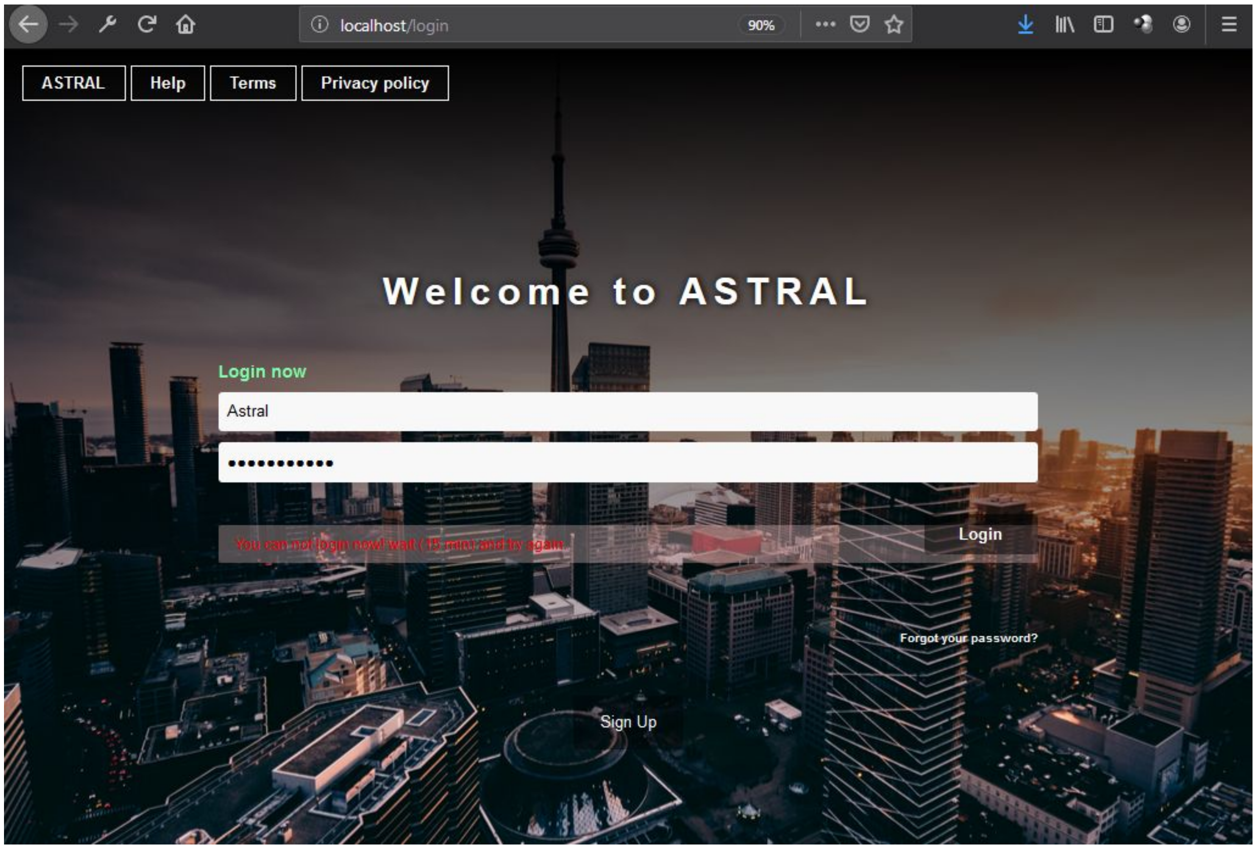
Summary of Project :
The Project is a secure twitter-like social network that runs using PHP, Javascript, AJAX, Bootstrap, HTML, CSS, Apache, MySQL. The Web app is a simple social network where clients can re-tweet, share, post, follow, like, Trending topics, functional search and so on…
The Motivation was to make a social network where the flow of information between users, inter and intra communities is controlled by authenticity and security principles yet, everything still feels like a typical social network to the end user. I implemented the features by trying to make the app feel more secure through securing traffic and defending against possible attacks.

The Design choices of cryptography:
For encryption I used a SSL localhost certificate because javascript can oftenly breakdown sometimes due to SSL certificate when the web application is deployed. So the localhost certificate provides security during testing the app and makes sure the code will still operate and function normally upon deployment. I also used hashing for password security, specifically a bcrypt hash. I chose the B-crypt hash because it is generally less common than md5 and just in case I used the default php hash. I might not be able to guarantee if its size/length would grow beyond my initial value in the database, because the default hash is dependent on the password’s length and the version of the php that means its size can exceed 200 bits. I’ve tried to implement csrf tokens to the app, and it worked and was functional. It was a randomly generated 64 bits encoded in a Base64 encoding. I’ve also filtered all the input fields by sanitizing and using other methods of filtering them in order for them to make it harder for anyone trying to use scripts or commands. As for the user ids, they get a randomly generated number between 0 and 99999 with the current timestamp appended on it. I’ve also added a simple security measure to delay cases of any attempts of brute force, if the user enters incorrect password for more than a defined number of times. His cookie gets set to prevent him from entering any input for 15 mins which are refreshed upon any attempt after that.

For Hashing : I used B-crypt hashing which generates a 64 bit hash. It is rather slower than md5 and sha1 ;however, it is more secure since brute force can easily get through md5 and sha1. Not to mention that fact that they are some of the most used hashing methods which makes them a bit more vulnerable to attacks. There is also Argon2 which is equally good as B-crypt ; however, I was not sure if it was supported by PHP, however, it is support since php 7.2 and is likely to be the next default hashing method. I would have rather prefered to use a Secret key encryption which works by creating a random 64 bits and a random nonce, by passing them through sodium_crypto_secretbox() which is an encryption; however, it had no documentation on the php site. Then creating a ciphertext made using the sodium_crypto_secretbox(), the nonce and 64 bit key, then encoding the ciphertext in a base64 encoding. The result key would look like this 'v6KhzRACVfUCyJKCGQF4VNoPXYfeFY+/pyRZcixz4x/0jLJOo+RbeGBTiZudMLEO7aRvg44HRecC' of 74 bits. And decoding it would use mb_substr() to remove the nonce since we already know the nounce's size, then decoding it then using sodium_crypto_secretbox_open() which seemingly opens the encryption box.

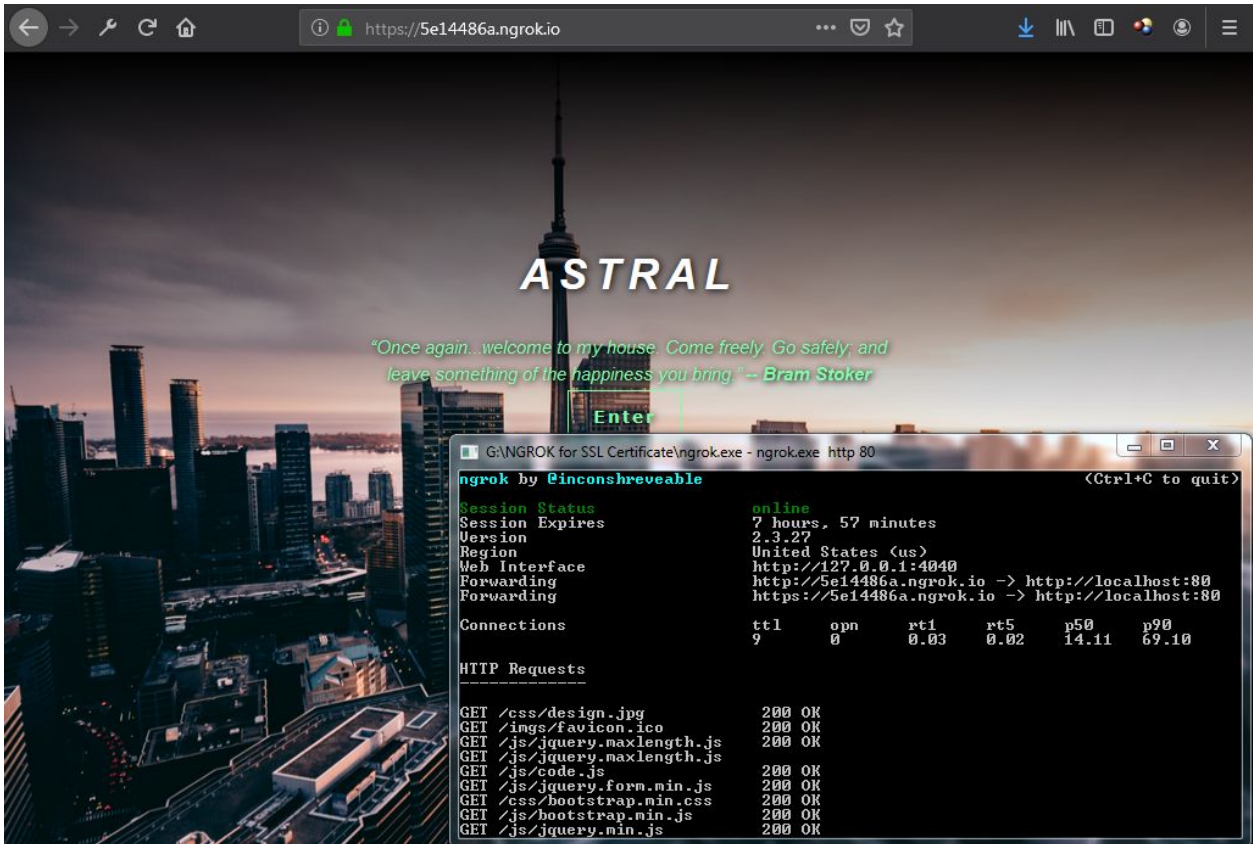
There was also Envelope Encryption, It provides a safer encryption since it stores the key in a different place than the data. It uses Google cloud key management which is a cloud storage hosted by google to store cryptographic keys and allows for different features to securely preserve your key such as automatic key rotation and delayed key destruction ; it is downside is that it takes only up to 64 bits per key and some people wouldn't want to trust their valuable data/keys to a third-party organization even if it is google. It works by letting you encrypt your data, and after you generate the key, you send it to the cloud which returns a KEK then you destroy your key, upon decryption you request the key from the cloud by sending the KEK which was stored/appended with your data and upon receiving the KEK, the cloud sends the key, and after you use the key, the key is destroyed … again. It is similar to secret key encryption except it uses encryption over multiple times. One of the reasons why I did not choose to use is because I had no experience in it prior to this, the main reason why I implemented neither was because I lacked time and I was working alone.

One of the possible attacks that like I mentioned earlier is brute force, and I slowed it down by forcing the end user to wait a significant amount of time before re-attempting to enter the password upon failing to input a valid password several times



You can not login now! wait (15 min) and try again

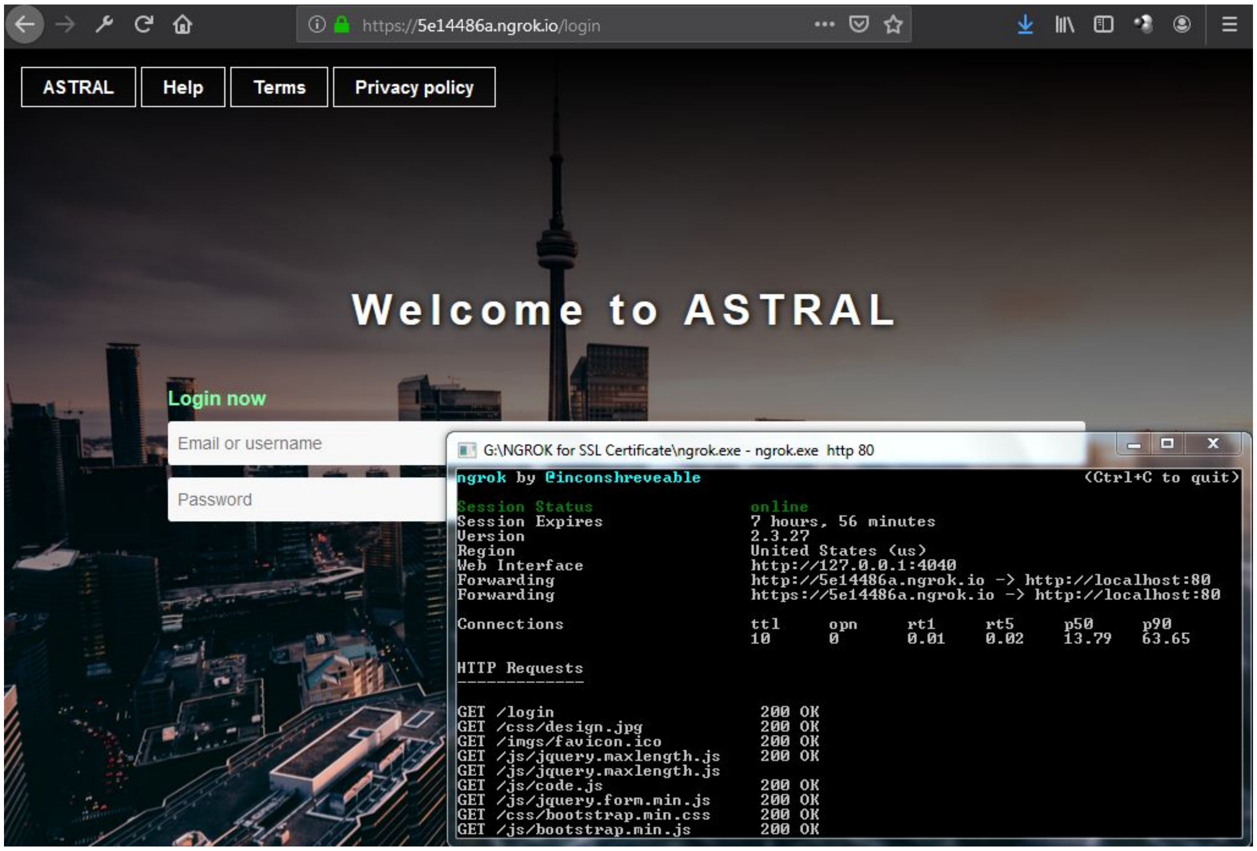
Another type of attack would be trying intercept the data sent and modifying it. I solved this by implementing a localhost SSL certificate which gives my web app and https and makes it more secure.


https://5e14486a.ngrok.io
ASTRAL
"Once again...welcome to my house. Come freely. Go safely; and leave something of the happiness you bring." -- Bram Stoker
Enter
G:\NGROK for SSL Certificate\ngrok.exe - ngrok.exe http 80

```
ngrok by @inconshreveable                                        (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 57 minutes
Version                       2.3.27
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://5e14486a.ngrok.io -> http://localhost:80
Forwarding                    https://5e14486a.ngrok.io -> http://localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              9       0       0.03    0.02    14.11   69.10

HTTP Requests
-------------

GET /css/design.jpg            200 OK
GET /imgs/favicon.ico          200 OK
GET /js/jquery.maxlength.js    200 OK
GET /js/jquery.maxlength.js
GET /js/code.js                200 OK
GET /js/jquery.form.min.js     200 OK
GET /css/bootstrap.min.css     200 OK
GET /js/bootstrap.min.js       200 OK
GET /js/jquery.min.js          200 OK
```

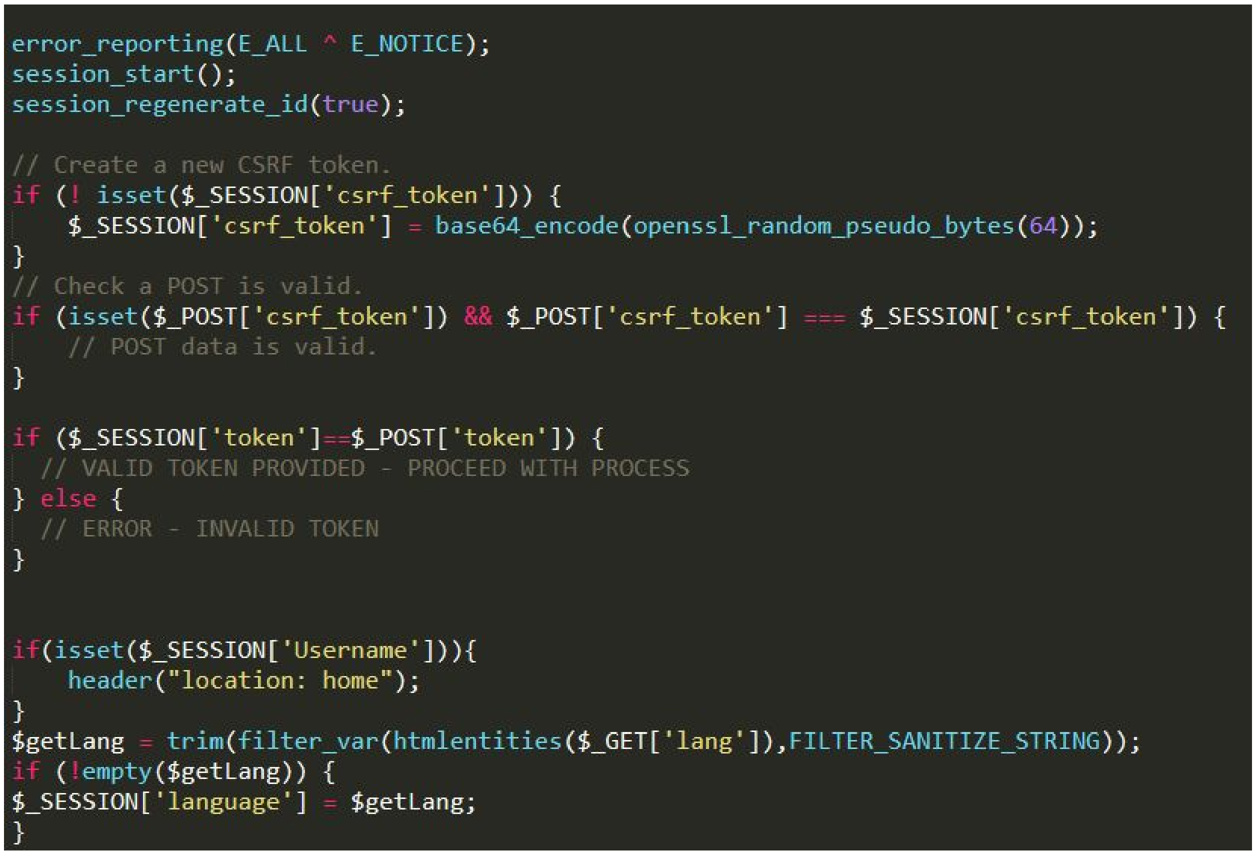
As you can see the web app url starts with https and the NGROK tool reads 200 OK on HTTP GET Requests. Sometimes it occasionally reads .JPG images as 206 Partial Content ; However, it is still safe and it doesn't affect any functionalities.

And it works for all pages


https://5e14486a.ngrok.io/login
ASTRAL
Help
Terms
Privacy policy
Welcome to ASTRAL
Login now
Email or username
Password
G:\NGROK for SSL Certificate\ngrok.exe - ngrok.exe http 80

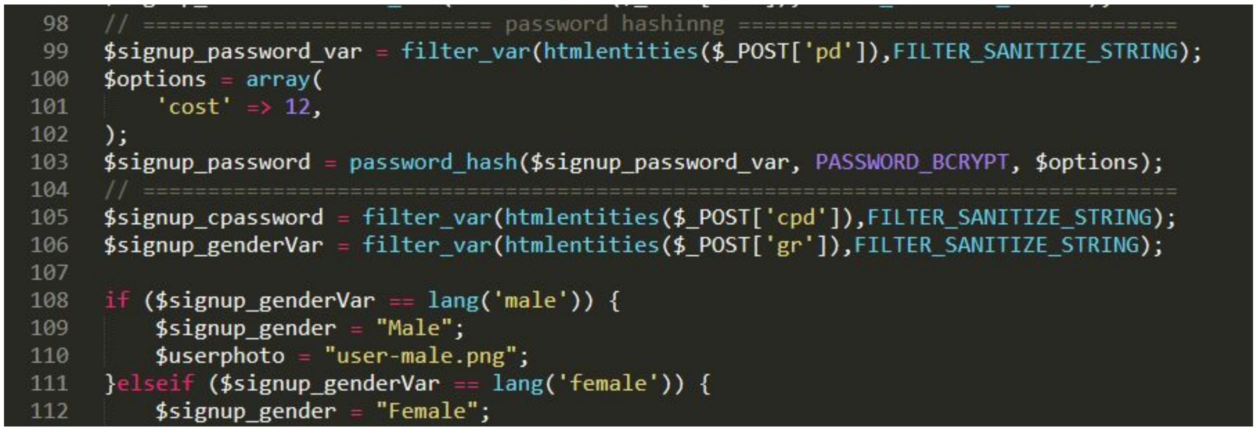
```
ngrok by @inconshreveable                                          (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 56 minutes
Version                       2.3.27
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://5e14486a.ngrok.io -> http://localhost:80
Forwarding                    https://5e14486a.ngrok.io -> http://localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              10      0       0.01    0.02    13.79   63.65

HTTP Requests
-------------

GET /login                     200 OK
GET /css/design.jpg            200 OK
GET /imgs/favicon.ico          200 OK
GET /js/jquery.maxlength.js    200 OK
GET /js/jquery.maxlength.js
GET /js/code.js                200 OK
GET /js/jquery.form.min.js     200 OK
GET /css/bootstrap.min.css     200 OK
GET /js/bootstrap.min.js       200 OK
```

For CSRF attacks I had to implement a token of 64 bits that was encoded in base64.



```php
error_reporting(E_ALL ^ E_NOTICE);
session_start();
session_regenerate_id(true);

// Create a new CSRF token.
if (! isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = base64_encode(openssl_random_pseudo_bytes(64));
}
// Check a POST is valid.
if (isset($_POST['csrf_token']) && $_POST['csrf_token'] === $_SESSION['csrf_token']) {
    // POST data is valid.
}

if ($_SESSION['token']==$_POST['token']) {
  // VALID TOKEN PROVIDED - PROCEED WITH PROCESS
} else {
  // ERROR - INVALID TOKEN
}


if(isset($_SESSION['Username'])){
    header("location: home");
}
$getLang = trim(filter_var(htmlentities($_GET['lang']),FILTER_SANITIZE_STRING));
if (!empty($getLang)) {
$_SESSION['language'] = $getLang;
}
```

I have also used a B-crypt hashing method for the password. That has 3 parameters, the password to be hashed, the hashing function and a cost of 12, the cost is basically the maximum algorithmic cost i set it as a value for the hashing function, the default is 10, increasing the value makes the hashing better trading off for the cost of the device hardware it is running on.



```php
// ========================== password hashinng ================================
$signup_password_var = filter_var(htmlentities($_POST['pd']),FILTER_SANITIZE_STRING);
$options = array(
    'cost' => 12,
);
$signup_password = password_hash($signup_password_var, PASSWORD_BCRYPT, $options);
// ==========================================================================
$signup_cpassword = filter_var(htmlentities($_POST['cpd']),FILTER_SANITIZE_STRING);
$signup_genderVar = filter_var(htmlentities($_POST['gr']),FILTER_SANITIZE_STRING);

if ($signup_genderVar == lang('male')) {
    $signup_gender = "Male";
    $userphoto = "user-male.png";
}elseif ($signup_genderVar == lang('female')) {
    $signup_gender = "Female";
```

Framework used is XAMPP which provides apache, and mysql, alongside with ngrok which generates a localhost ssl certificate, as well as wallstant for base web application.

References :

https://www.geeksforgeeks.org/php-crypt-password_hash-functions/
https://deliciousbrains.com/php-encryption-methods/
https://www.youtube.com/watch?v=8MksjpfDvRw&list=PLgt-G5tEOlKUJaQgCad0hbtVDWWRdmP0O&index=2
https://www.youtube.com/watch?v=3bGDe0rbImY&list=PLgt-G5tEOlKUJaQgCad0hbtVDWWRdmP0O&index=3
https://www.youtube.com/watch?v=U4LDsxSzZeI&list=PLgt-G5tEOlKUJaQgCad0hbtVDWWRdmP0O&index=4