IERG4210
Web Programming and Security

Report on Assignment

Secure E-Commerce Website

Name: KEI Yat-long

Phase 1: Layout

The appearance of a website plays a big role in attracting visitors.

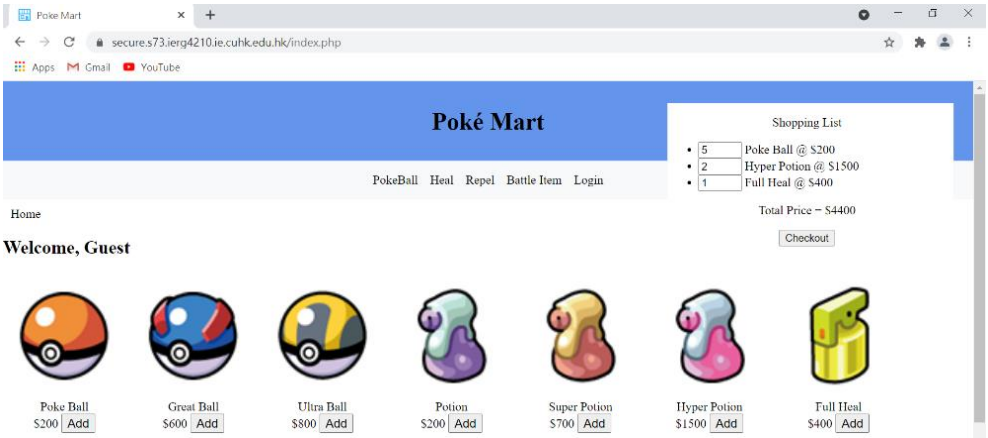
In this phase, I created a dummy shopping website from scratch by hardcoding the basic elements.

1. HTML: Make good use of semantic HTML

2. CSS: Clean separation of HTML, CSS and JS code and files

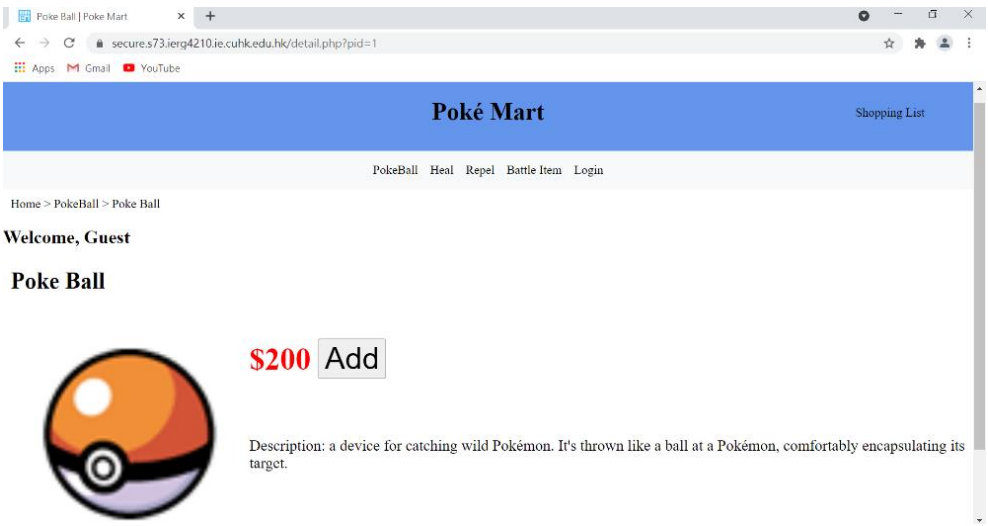
```
<html>
▼<head>
    <title>Poke Mart</title>
    <meta charset="utf-8">
    <meta name="description" content="The main page displaying all available products in PokeMart">
    <meta name="author" content="Andes KEI">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="favicon.ico" rel="icon" type="image/x-icon">
    <link rel="stylesheet" href="./css/style-min.css">
  </head>
▼<body onload="refreshCart()">
  ▶<header>…</header>
  ▶<nav>…</nav>
  ▶<main>…</main>
  ▶<footer>…</footer>
    <script type="text/javascript" src="./lib/myLib-min.js"></script>
    <script type="text/javascript" src="./js/manageCart-min.js"></script>
  </body>
</html>
```

3. Main page demonstrates the use of CSS table-less product list and CSS hover shopping list



4. Product page provides product details

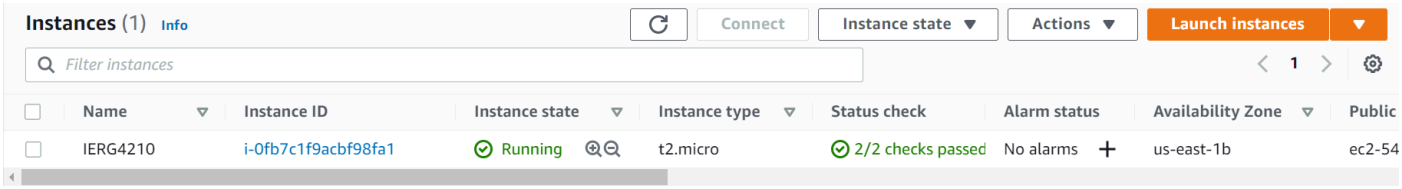
5. Include a hierarchical navigation menu for main page and product page



Phase 2: Secure Server Setup, Data Presentation and Management

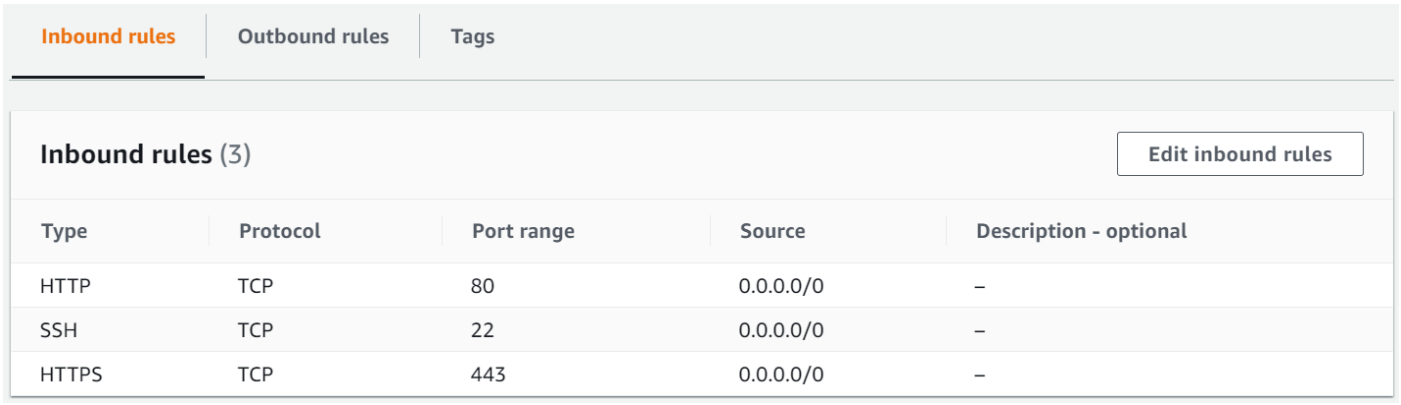
In this phase, I set up a secure server for later development and implemented the core functions of the website mainly with PHP and SQL.

1. Instantiate a free virtual cloud machine

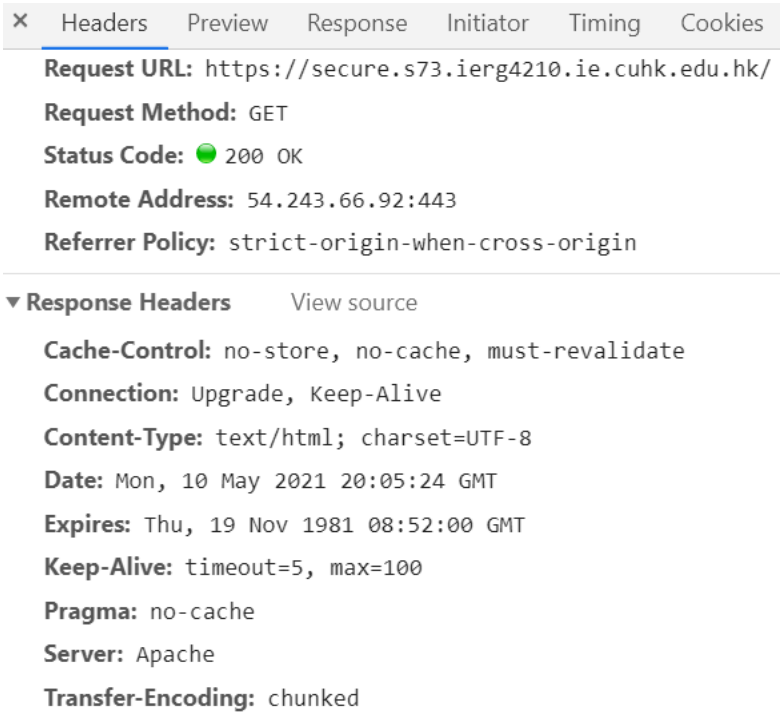


2. Apply necessary security configurations

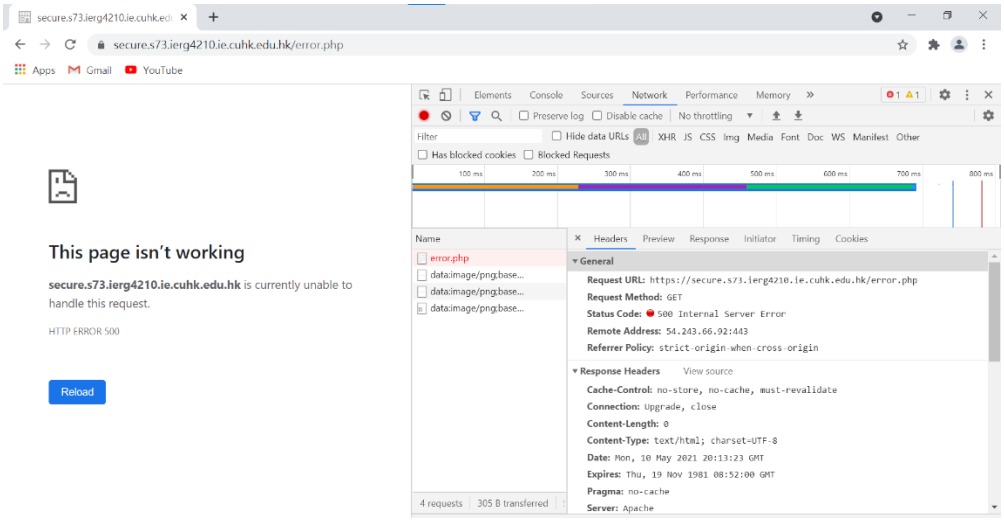
- Apply proper firewall settings to my VM: block all ports except 22, 80 and 443 only



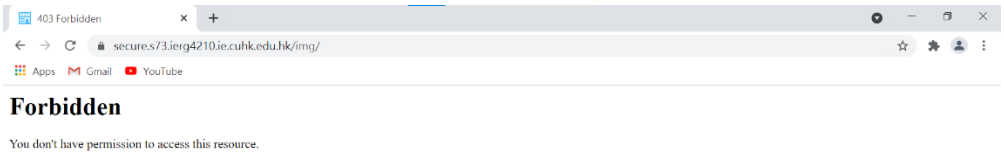
- Hide the versions of OS, Apache, and PHP in HTTP response headers



- Do not display any PHP warnings and errors to the end users



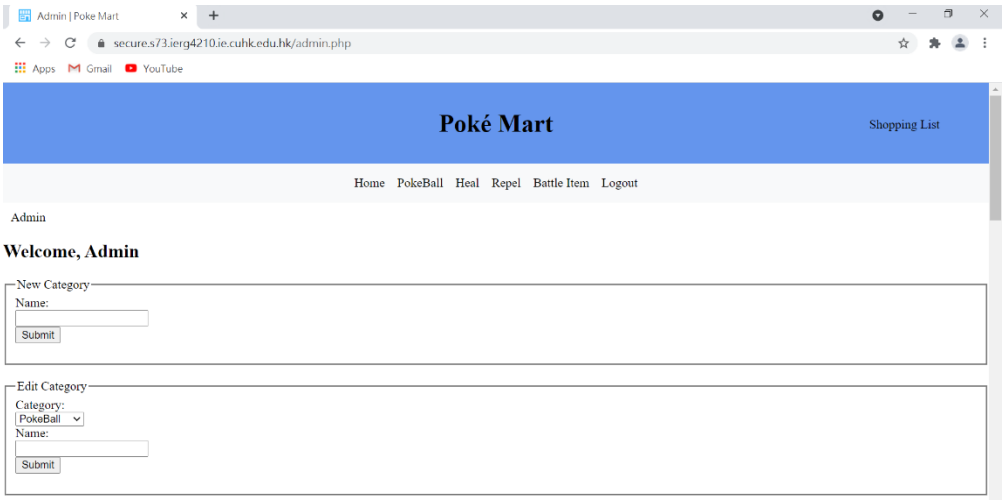
- Disable directory index in Apache



3. SQL: Create a table for categories and a table for products

```
[ec2-user@ip-172-31-31-36 ~]$ sudo sqlite3 /var/www/cart.db
SQLite version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE categories (
  catid INTEGER PRIMARY KEY,
  name TEXT
);
CREATE TABLE products (
  pid INTEGER PRIMARY KEY,
  catid INTEGER,
  name TEXT,
  price INTEGER,
  description TEXT,
  FOREIGN KEY(catid) REFERENCES categories(catid)
);
```

4. HTML, PHP & SQL: Create an admin panel

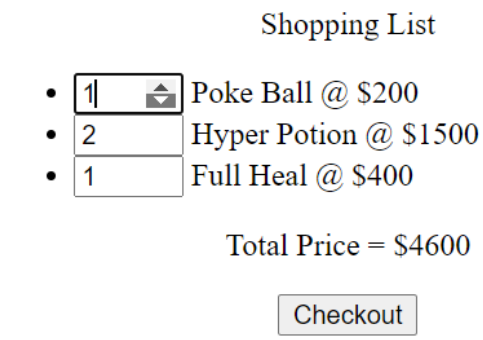


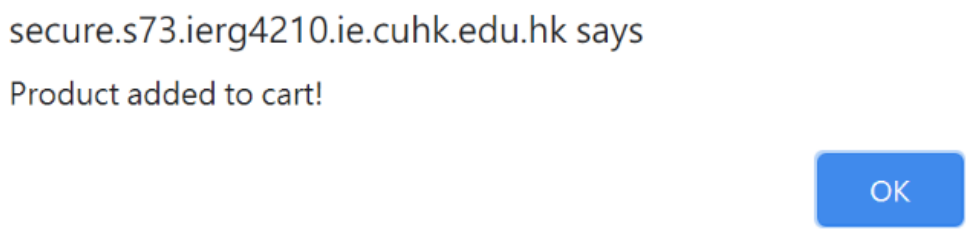
Phase 3: AJAX Shopping List

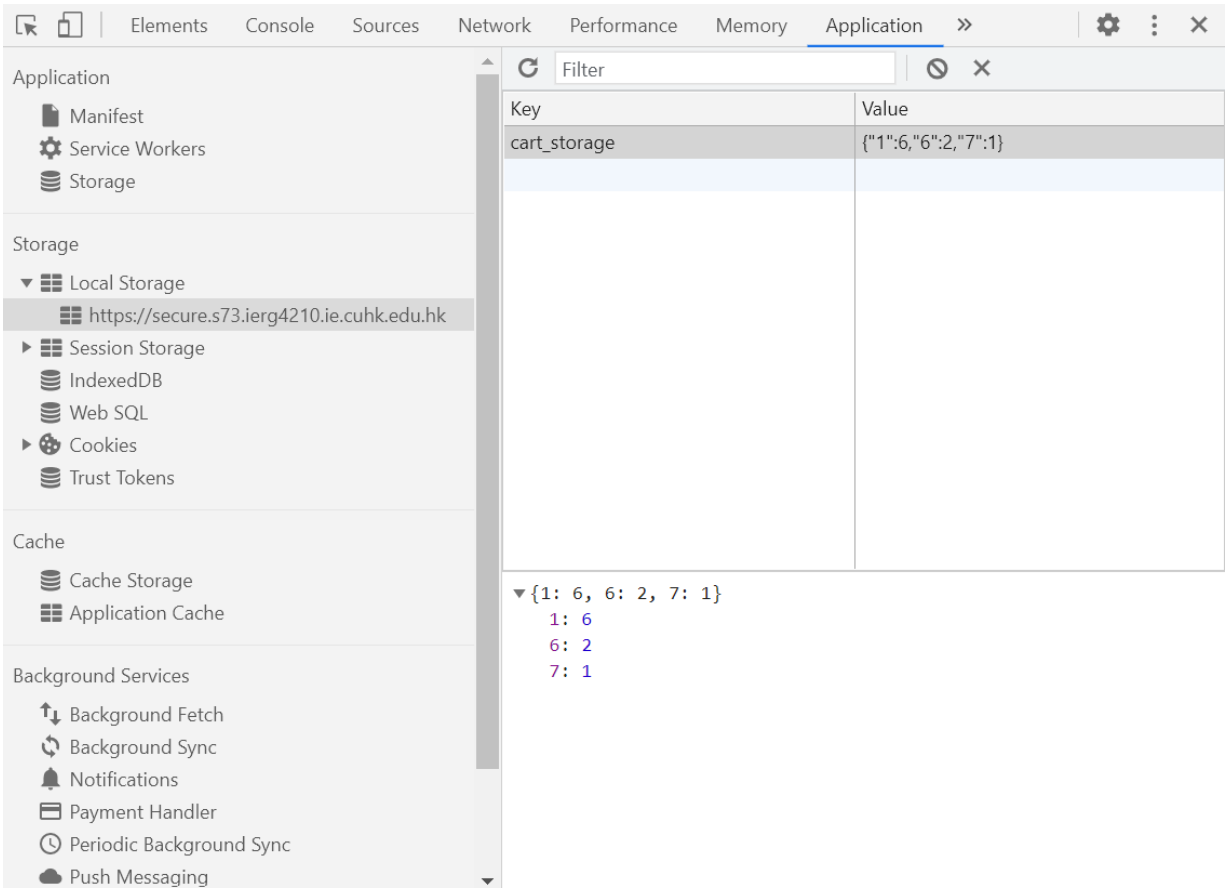
In this phase, I implemented the shopping list, which allows users to shop around my products.

1. JS: Dynamically update the shopping list

- When the addToCart button of a product is clicked, add it to the shopping list
- Users are allowed to update the quantity of a product in cart and delete it with a number input
- Store its pid and quantity in the browser's localStorage
- Get the name and price over AJAX (with pid as input)
- Calculate and display the total amount at the client-side
- Once the page is reloaded, the shopping list is restored





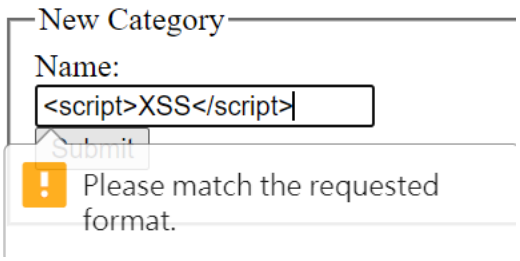


Phase 4: Securing the Website

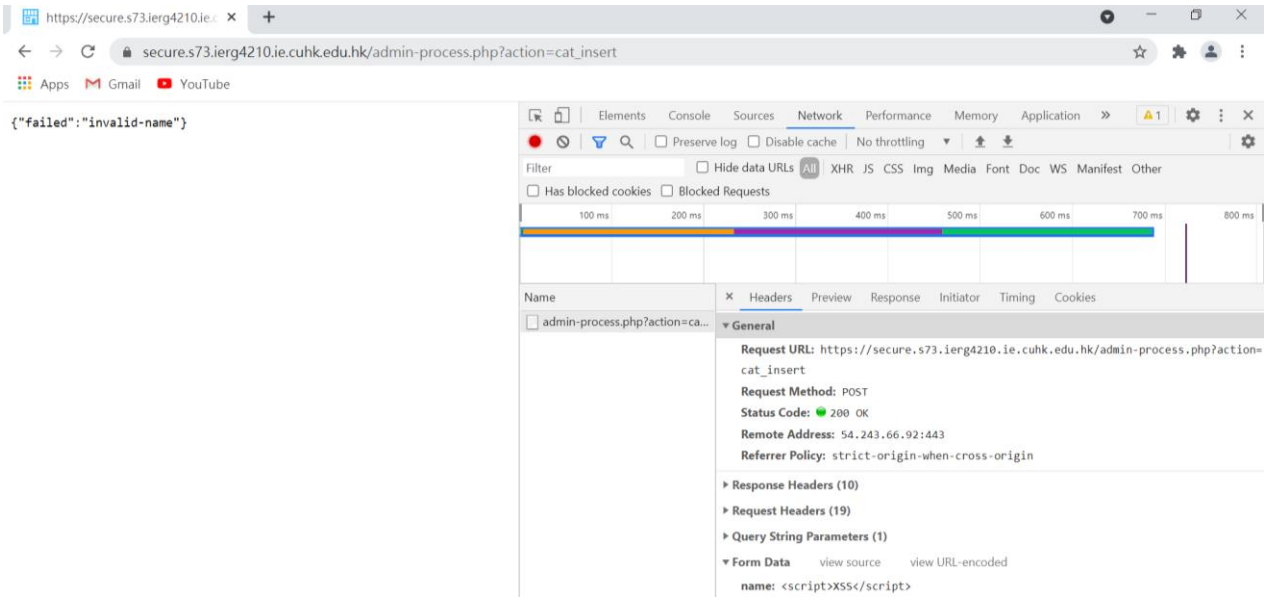
In this phase, I protected my website against some popular web application security threats.

1. No XSS injection and parameter tampering vulnerabilities

- Proper and vigorous client-side input restrictions



- Proper and vigorous server-side input sanitizations and validations

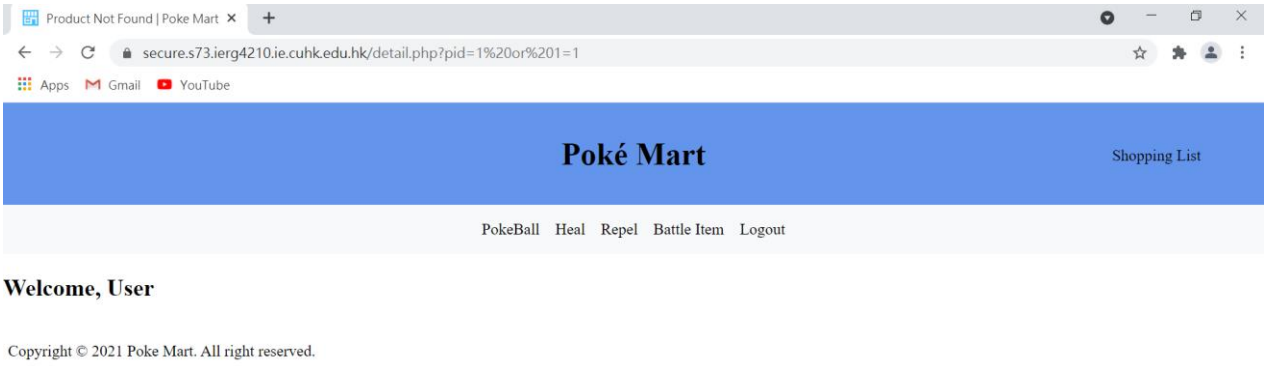


- Proper and vigorous context-dependent output sanitizations

```
// generate all products
foreach($prod_res as $value){
    $prod_list .= '<li class="list-inline-item center"><a href="detail.php?pid='.urlencode($value["pid"]).
                  '"><img class="thumbnail" src="img/'.urlencode($value["pid"]).'.png" /><br/>'.htmlspecialchars($value["name"]).
                  '</a><br/>$'.htmlspecialchars($value["price"]).' <button type="button" onclick="addtocart('.htmlspecialchars($value["pid"]).
                  ')">Add</button></li>';
}
```

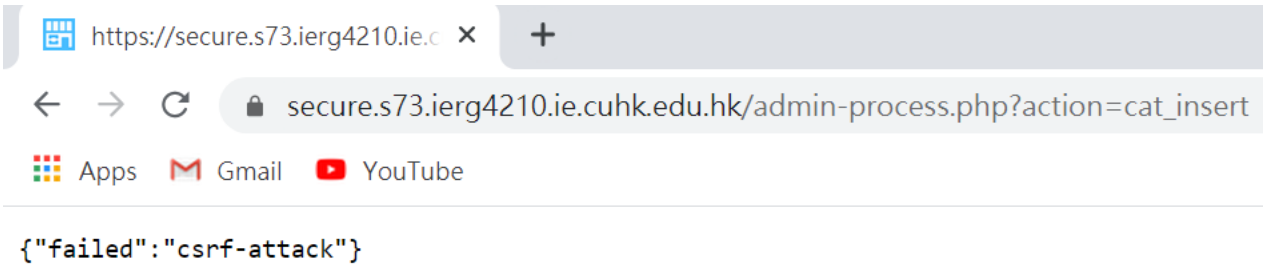
2. Mitigate SQL injection vulnerabilities

- Apply parameterized SQL statements



3. Mitigate CSRF vulnerabilities

- Apply and validate secret nonces for every form
- All generated session IDs and nonces are not guessable

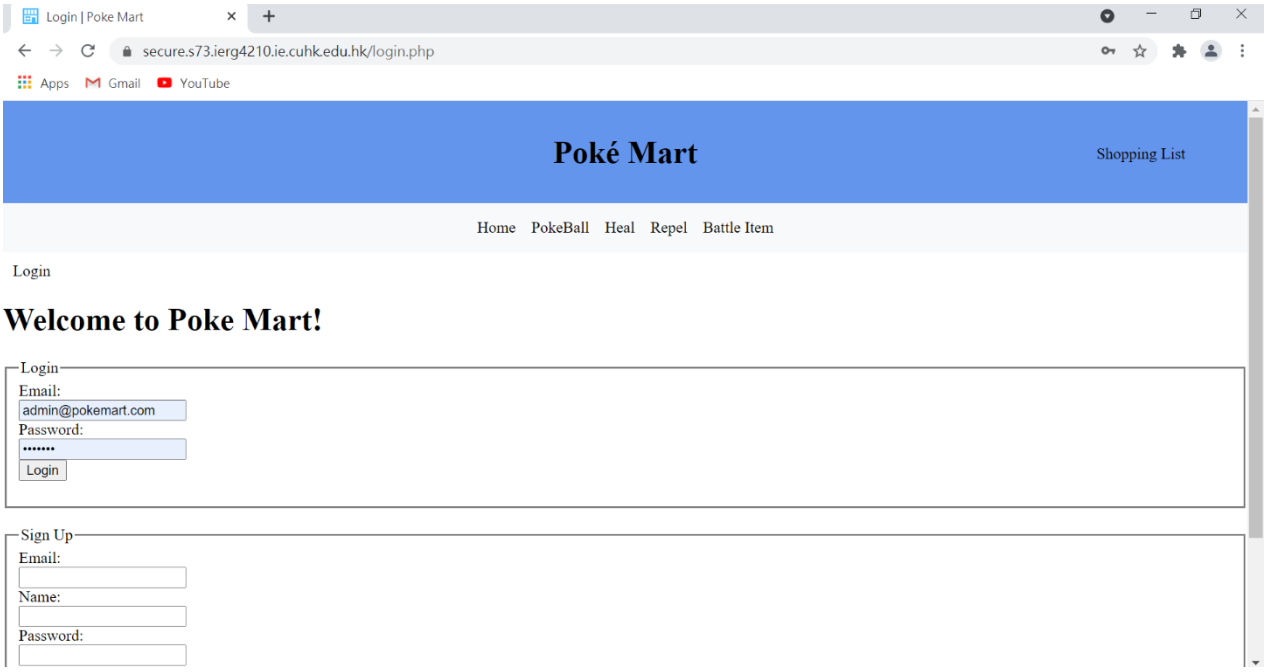


4. Authentication for admin panel

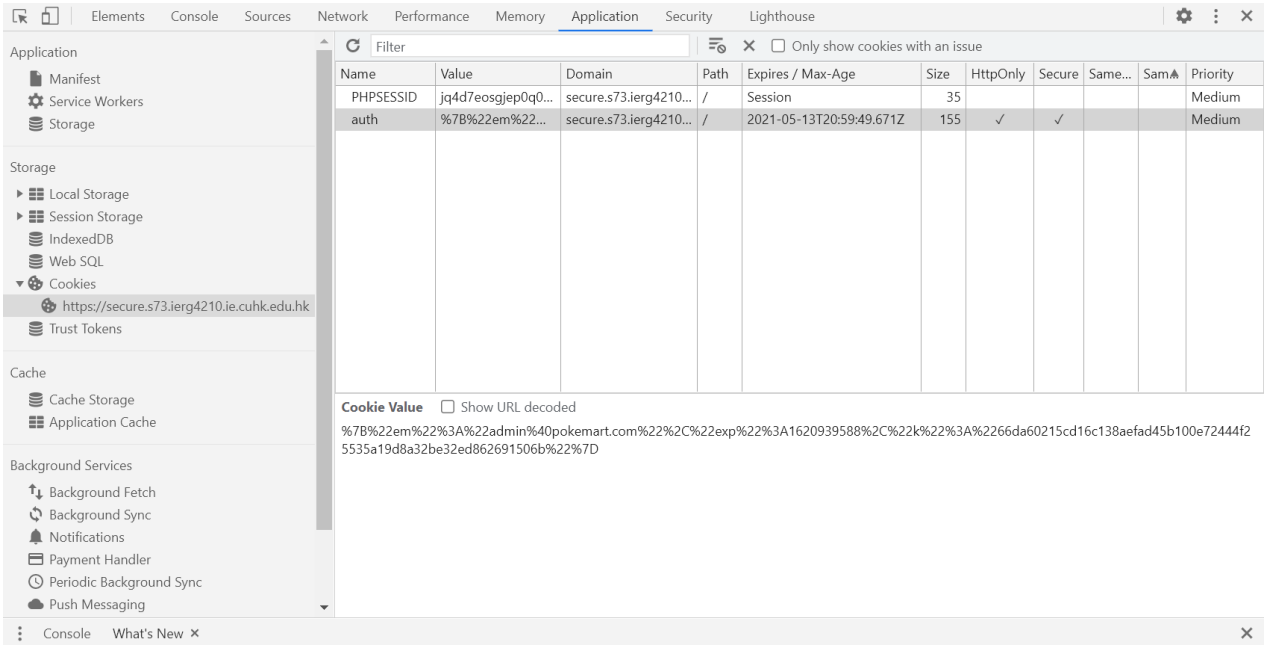
- Create a user table

```
[ec2-user@ip-172-31-31-36 ~]$ sudo sqlite3 /var/www/cart.db
SQLite version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE account(
  userid INTEGER PRIMARY KEY,
  email TEXT UNIQUE,
  name TEXT,
  salt TEXT,
  password INTEGER,
  admin INTEGER
);
```

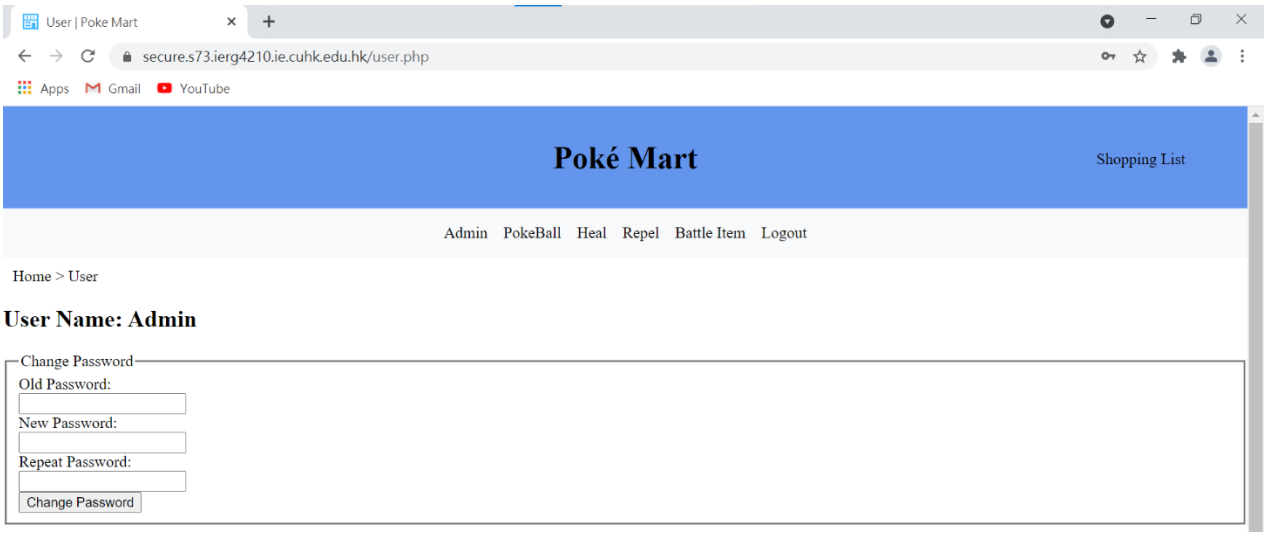
- Build a login page login.php that requests for email and password
- Upon validated and authenticated, redirect the user to the admin panel or main page



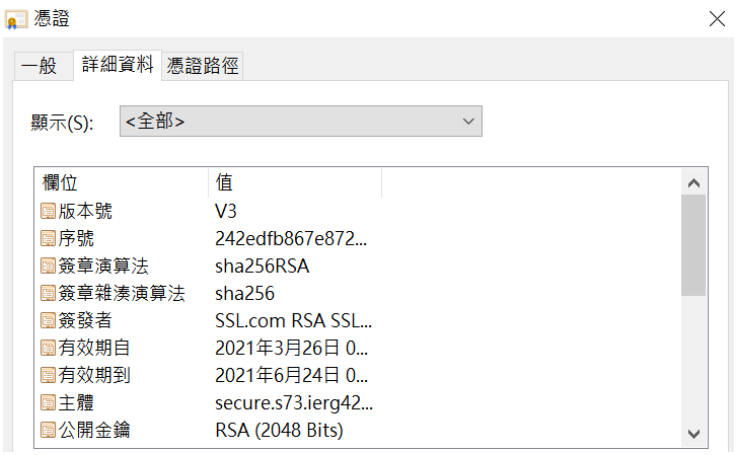
- Maintain an authentication token using Cookies
- Validate the authentication token before revealing and executing admin feature



- Supporting change of password



5. Apply SSL certificate and serve the website over HTTPS



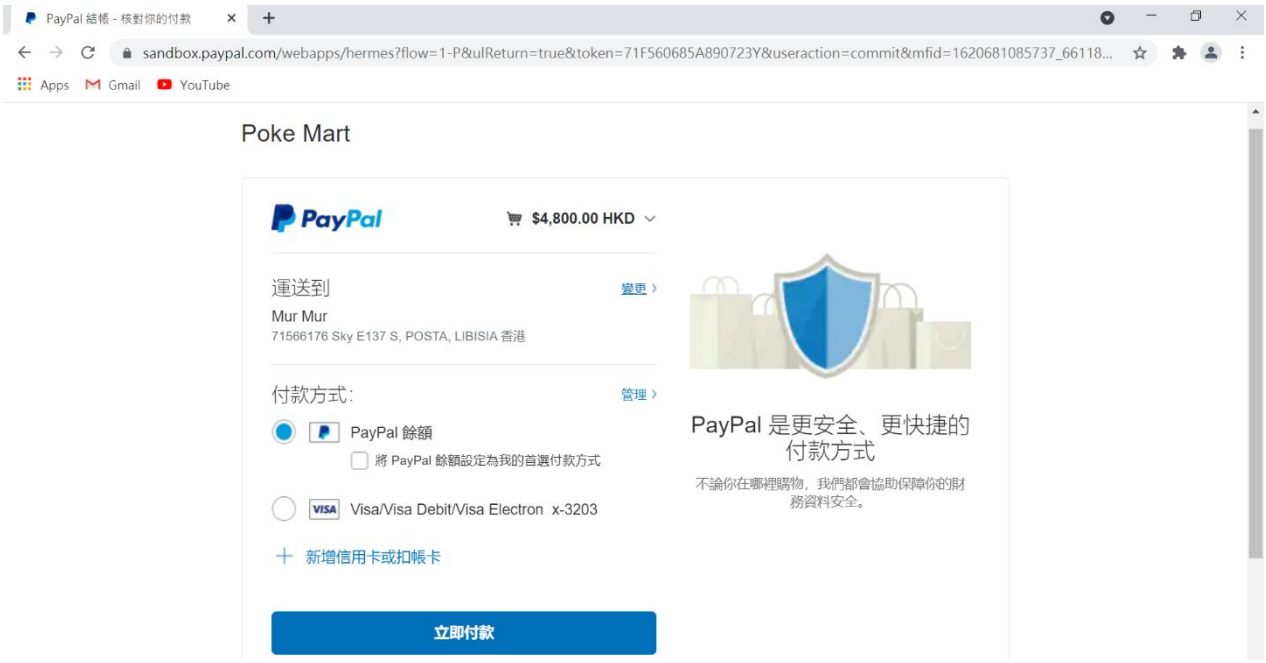
Phase 5: Secure Checkout Flow

In this phase, I completed the secure checkout flow with PayPal.

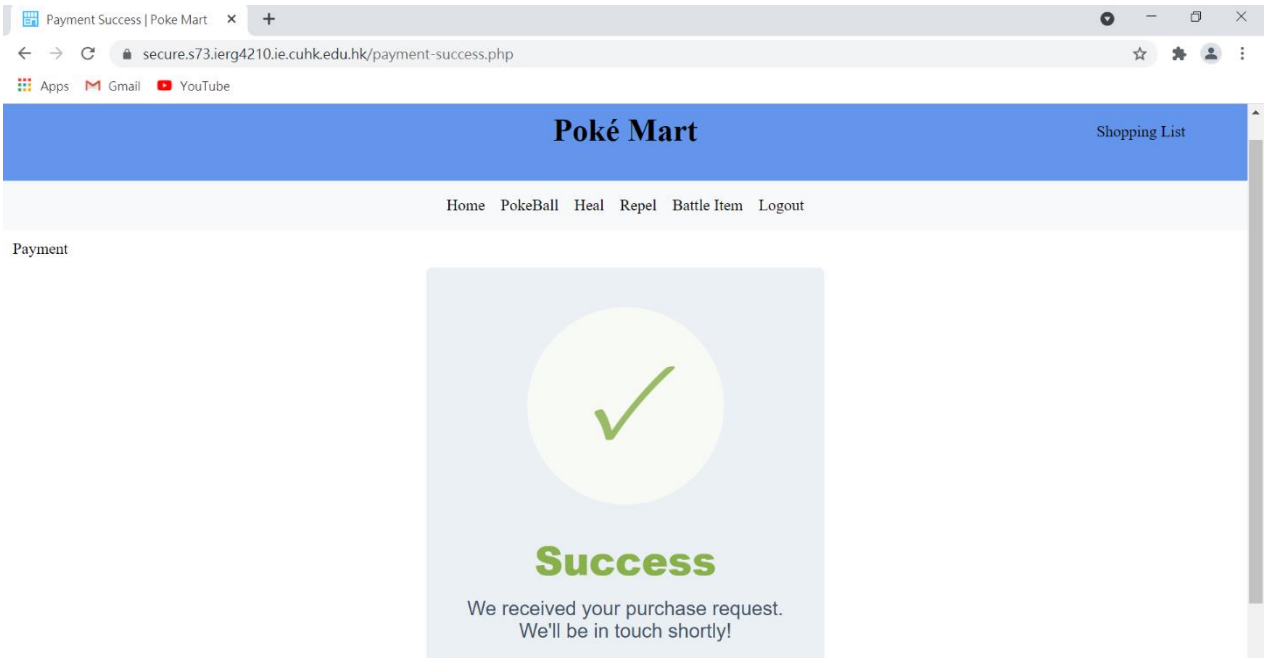
1. Enclose the shopping cart with a <form> element

```
<form method="POST" id="cart" action="https://www.sandbox.paypal.com/cgi-bin/webscr" onsubmit="cartSubmit(); event.preventDefault();">
  <input type="hidden" name="cmd" value="_cart">
  <input type="hidden" name="upload" value="1">
  <input type="hidden" name="business" value="admin@pokemart.com">
  <input type="hidden" name="currency_code" value="HKD">
  <input type="hidden" name="charset" value="utf-8">
  <input type="hidden" name="custom" value="0">
  <input type="hidden" name="invoice" value="0">
  <input type="submit" value="Checkout">
</form>
```

2. Submit the form to PayPal after various procedures

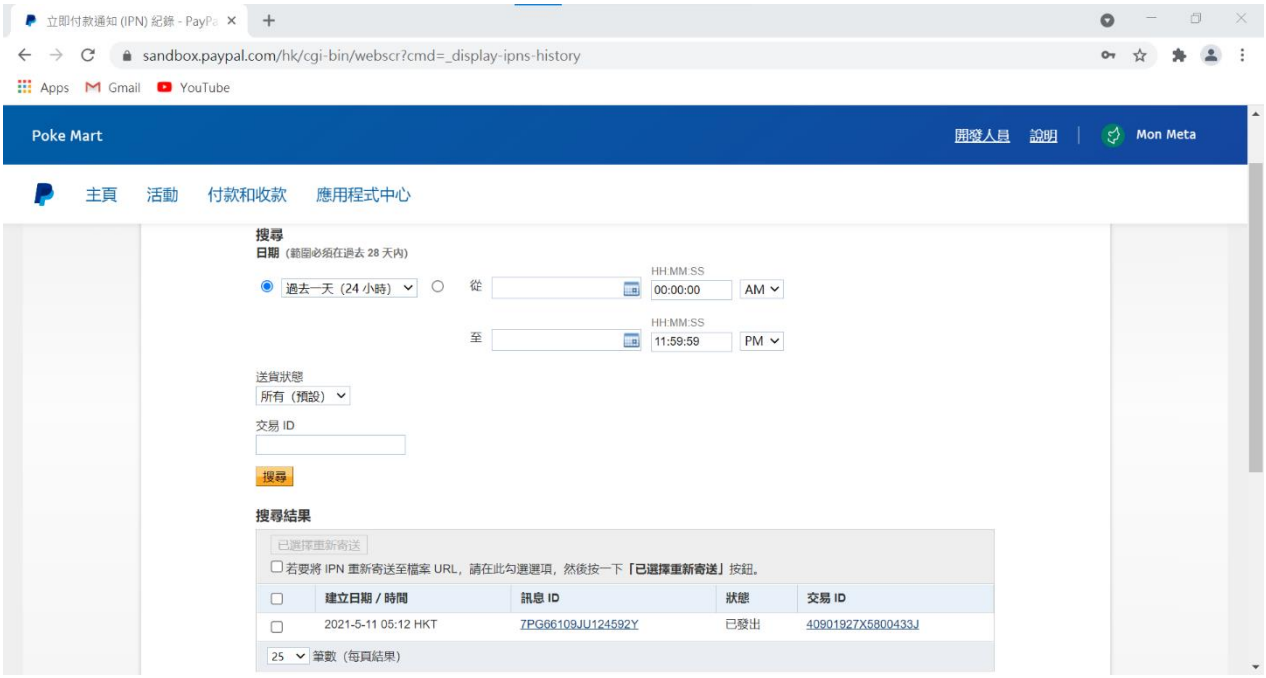


3. After the buyer has finished paying with PayPal, auto-redirect the buyer back to my shop



4. Setup an Instant Payment Notification (IPN) page to get notified once payment is completed

- Validate the authenticity of data, Save the txn_id and product list after validation



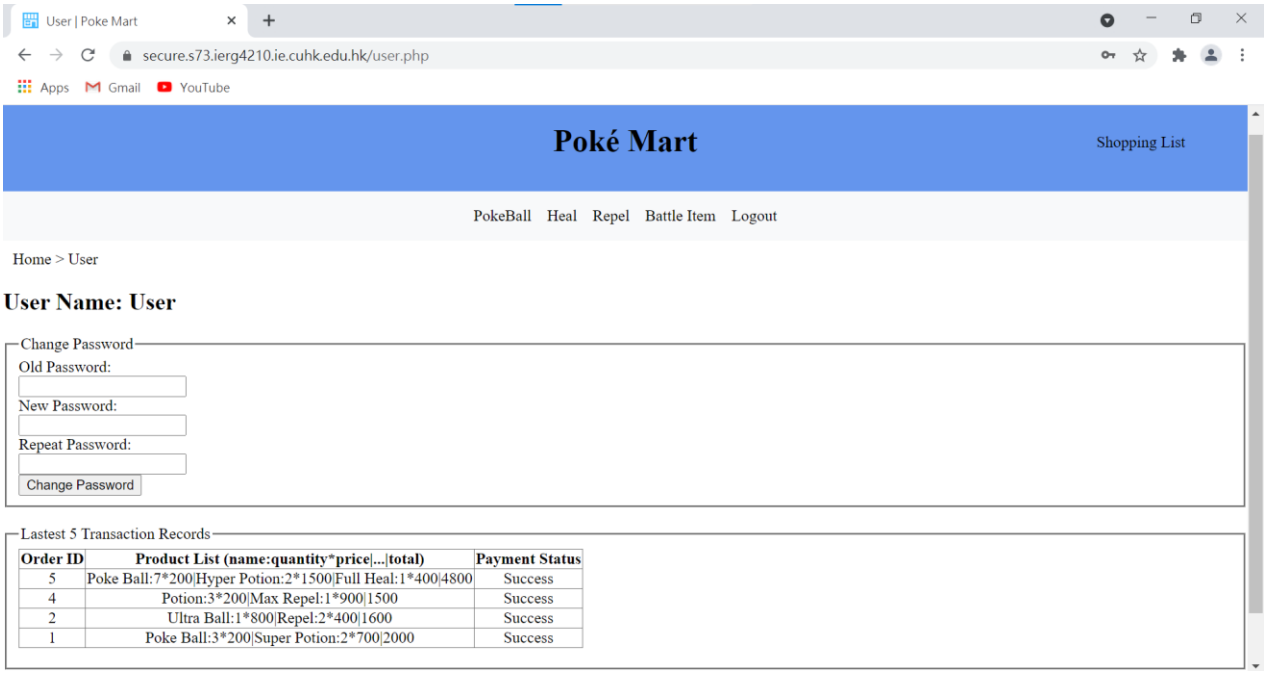
5. Display the DB orders table in the admin panel

Lastest 50 Transaction Records

Order ID	User Account	Product List (name:quantity*price|...|total)	Digest	Salt	Payment Status
5	user@pokemart.com	Poke Ball:7*200|Hyper Potion:2*1500|Full Heal:1*400|4800	060a9a47a1c421f7a70b4413d3b7214df08b735e5d5f07af8a22ee19f6da3197	14717743642090534974	40901927X5800433J
4	user@pokemart.com	Potion:3*200|Max Repel:1*900|1500	902a6b74b8bfb1a0eb28a5d0c3ff3f1d35ebc9c3d1cb5a467aceb4ea5aca322b	420613747381506825	96D41241E1260845K
3	Guest	Poke Ball:1*200|Dire Hit:1*1000|1200	1d808c73a6b3ac92b6ef45a76fe9599ebaba2710f558190c2d3b971f4a20dc25	9693158425595734	43U96699XW831882L
2	user@pokemart.com	Ultra Ball:1*800|Repel:2*400|1600	ebb3d092153f9e8ecabae7941b170482bce5792b1fe35034f0c8b5b6e1a068c5	4918351101355057508	3V226151385499437
1	user@pokemart.com	Poke Ball:3*200|Super Potion:2*700|2000	7e31b36731be744a4e2e477a570d39ac504615ed9196f7992953fab0ea7f954d	2384445381760045916	9T180134VG989161N

Remark: The payment status field will show the corresponding transaction ID if the payment succeed.

6. Let members check what they have purchased in the most recent five orders



Final Phase: Peer Hacking

In this phase, I tried to improve the security of my website by the use of automated vulnerability scanner and performed ethical hacking for the shop of my classmates.

1. Practice with the use of any automated vulnerability scanner

   To discover any possible vulnerability of my website, I used an online automated vulnerability scanner called Pentest-tools.com, link: Website Scanner Online - Find Vulns Fast | Pentest-Tools.com.

Findings

Insecure cookie setting: missing HttpOnly flag

Cookie Name	URL	Evidence
PHPSESSID	https://secure.s73.ierg4210.ie.cuhk.edu.hk/	Set-Cookie: PHPSESSID=s84fuebto5v3ahhuuehbfuev88; path=/

Insecure cookie setting: missing Secure flag

Cookie Name	URL	Evidence
PHPSESSID	https://secure.s73.ierg4210.ie.cuhk.edu.hk/	Set-Cookie: PHPSESSID=s84fuebto5v3ahhuuehbfuev88; path=/

From the above, we can see that there is an insecure cookie setting for PHPSESSID cookie in my website. If this kind of vulnerability is ignored, an attacker may be able to gain unauthorized access to the victim's web session. Therefore, I defended my website by setting the HttpOnly and secure flag for that cookie.

```
// start PHP session
session_set_cookie_params(0, '', '', true, true);
session_start();
```

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure
PHPSESSID	9s8bipqj1oqv5nkcop...	secure.s73.ierg4210.ie.cuhk.edu.hk	/	Session	35	✓	✓
auth	%7B%22em%22%3A...	secure.s73.ierg4210.ie.cuhk.edu.hk	/	2021-05-18T08:35:10.400Z	155	✓	✓

Also, according to the advice from the vulnerability scanner, I included some security headers in HTTP response to defense on some common attacks:

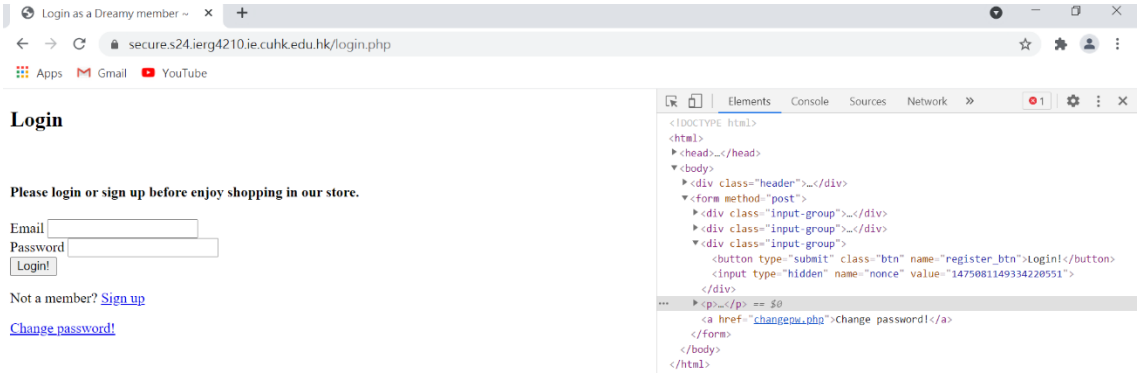
```
// include security headers
header('Referrer-Policy: no-referrer');
header('Strict-Transport-Security: max-age=7776000')
header('X-Content-Type-Options: nosniff');
header('X-Frame-Options: DENY');
header('X-XSS-Protection: 1; mode=block');
```

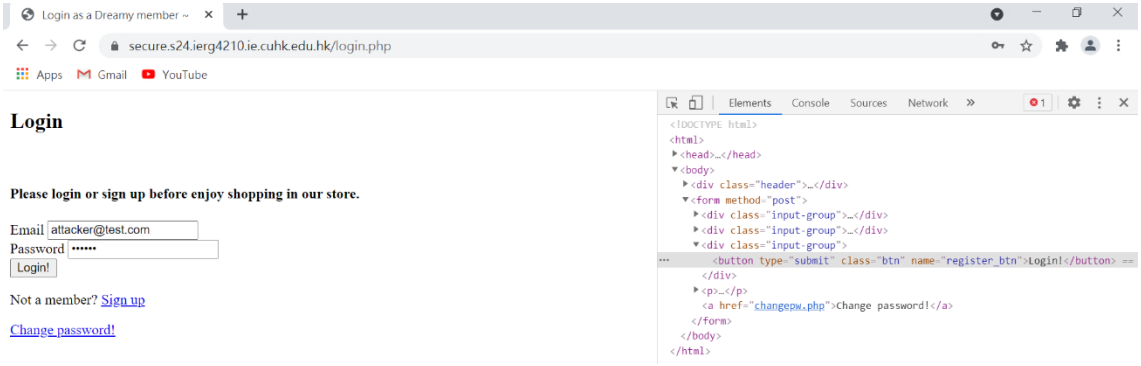
- Referrer-Policy: no-referrer => avoid user tracking and unwanted data leakage
- Strict-Transport-Security: max-age=7776000 => defend against session hijacking
- X-Content-Type-Options: nosniff => defend against file-based XSS injection
- X-Frame-Options: DENY => defend against clickjacking attack
- X-XSS-Protection: 1; mode=block => defend against reflected XSS attack

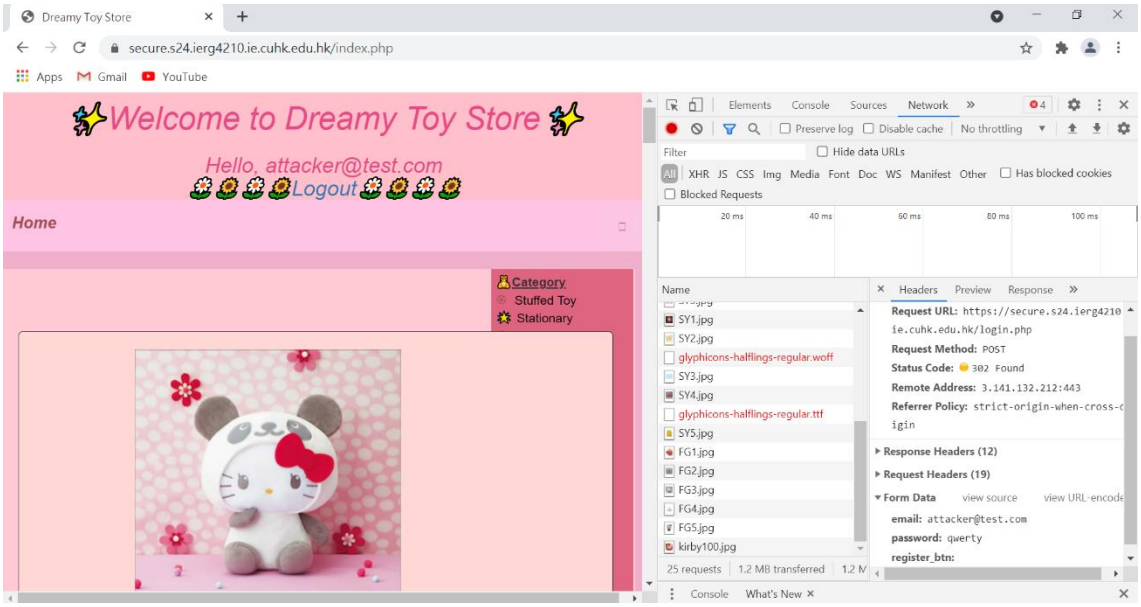
2. Perform manual ethical hacking for the shops of my classmate

Victim 1: https://secure.s24.ierg4210.ie.cuhk.edu.hk/

Attack Performed: [A5] CSRF



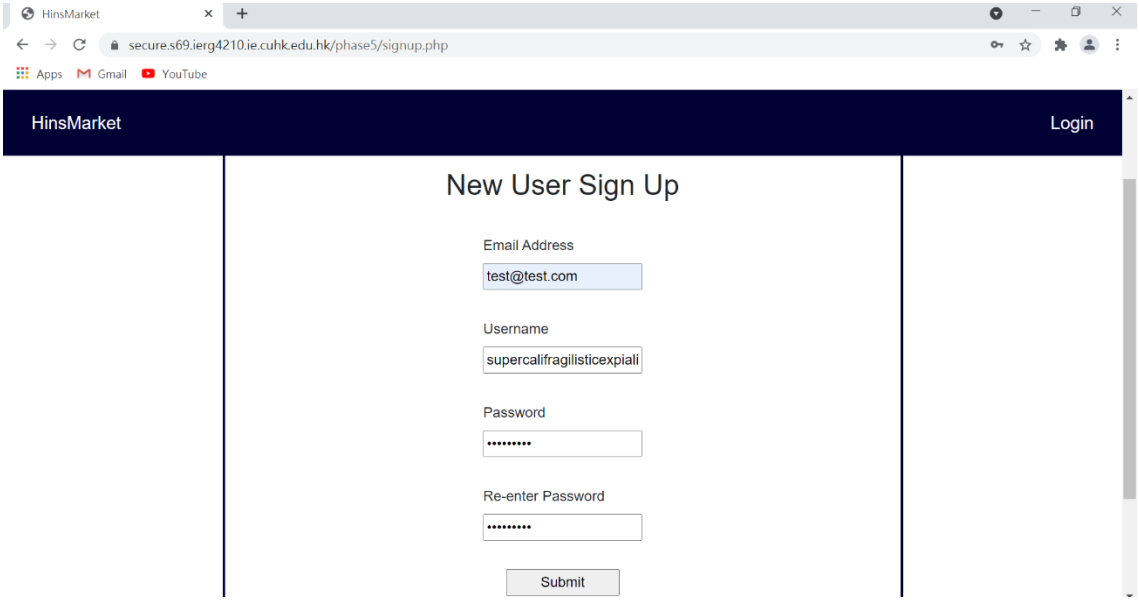


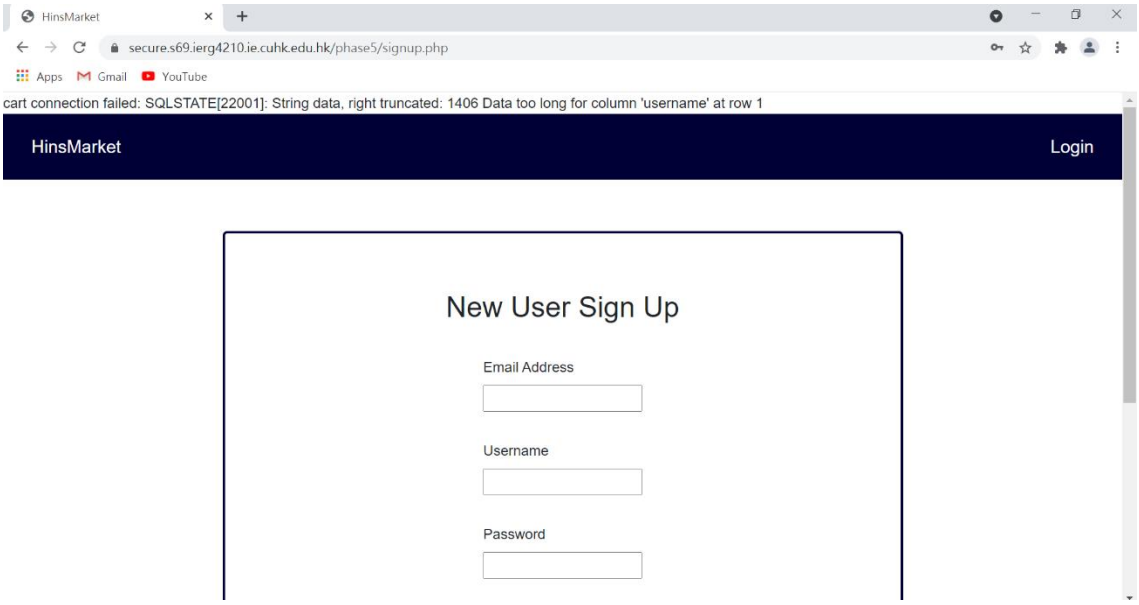


This website is vulnerable to CSRF because the browsers do not check whether a client actually initiates an HTTP request. Originally, this could be prevented by the use of nonce. However, from the figures above, we can still login successfully even we delete that hidden input “nonce”. Therefore, an attacker could perform a login CSRF attack to make a victim visits a page that automatically login the website using attacker’s credentials. To fix this problem, the website needs to validate the secret nonce for every form in the server side before performing further actions.

Victim 2: https://secure.s69.ierg4210.ie.cuhk.edu.hk

Attack Performed: [A6] Security Misconfiguration





From the figures above, we can see that when we input a very long username in the sign-up page, the user registration failed and an PHP error is displayed. This may provide additional information for attacker when he or she is attacking the server, which makes the website vulnerable. To fix this problem, the website needs to set display_errors = Off, display_startup_errors = Off in /etc/php.ini.