# PSP0201 Week 3 Writeup

Group Name: Undecided

Members

| ID | Name | Role |
|---|---|---|
| 1211101390 | Aslamia Najwa Binti Ahmad Khadri | Leader |
| 1211100431 | Mohammad Omar Torofder | Member |
| 1211103388 | Vishnu Karmegam | Member |
| 1211103092 | Farryn Aisha binti Muhd Firdaus | Member |

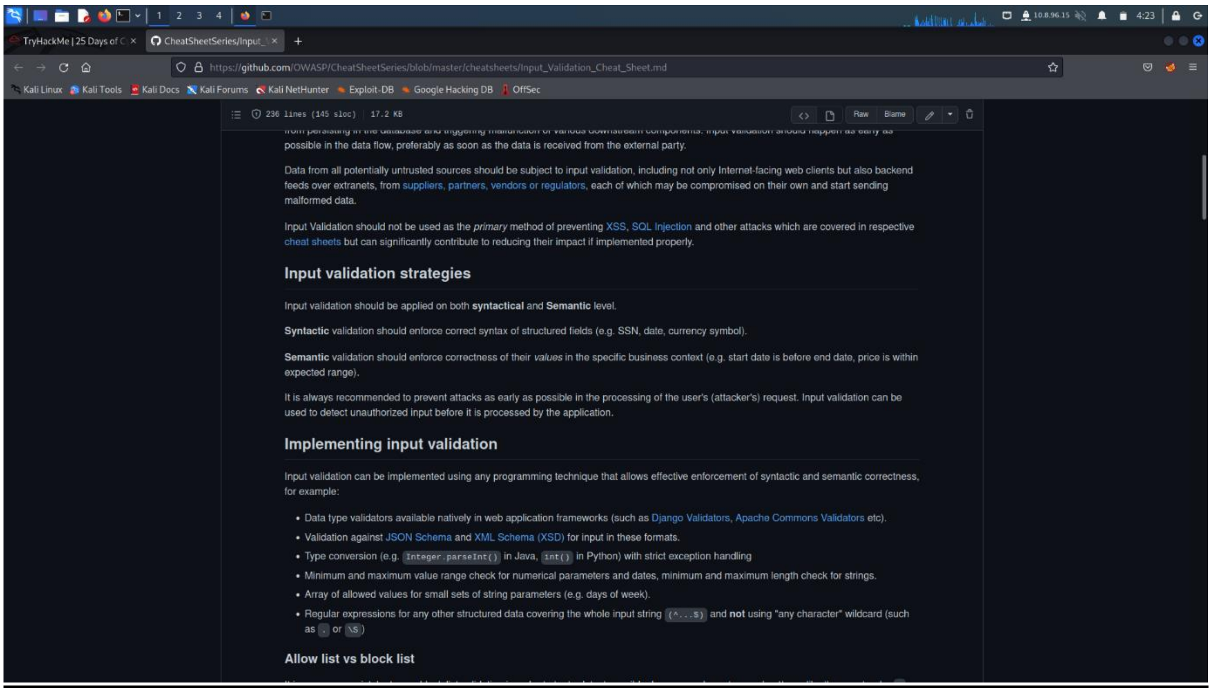
## Day 6: Web Exploitation – Be careful with what you wish on a Christmas night

**Tools used**: Kali Linux, Firefox, OWASP ZAP

**Solution/walkthrough**:

### Question 1

Open the link provided in the room that leads to the OWASP cheat sheet. Examine the content.


https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input_Validation_Cheat_Sheet.md

236 lines (145 sloc) 17.2 KB

possible in the data flow, preferably as soon as the data is received from the external party.

Data from all potentially untrusted sources should be subject to input validation, including not only Internet-facing web clients but also backend feeds over extranets, from suppliers, partners, vendors or regulators, each of which may be compromised on their own and start sending malformed data.

Input Validation should not be used as the *primary* method of preventing XSS, SQL Injection and other attacks which are covered in respective cheat sheets but can significantly contribute to reducing their impact if implemented properly.

**Input validation strategies**

Input validation should be applied on both **syntactical** and **Semantic** level.

**Syntactic** validation should enforce correct syntax of structured fields (e.g. SSN, date, currency symbol).

**Semantic** validation should enforce correctness of their *values* in the specific business context (e.g. start date is before end date, price is within expected range).

It is always recommended to prevent attacks as early as possible in the processing of the user's (attacker's) request. Input validation can be used to detect unauthorized input before it is processed by the application.

**Implementing input validation**

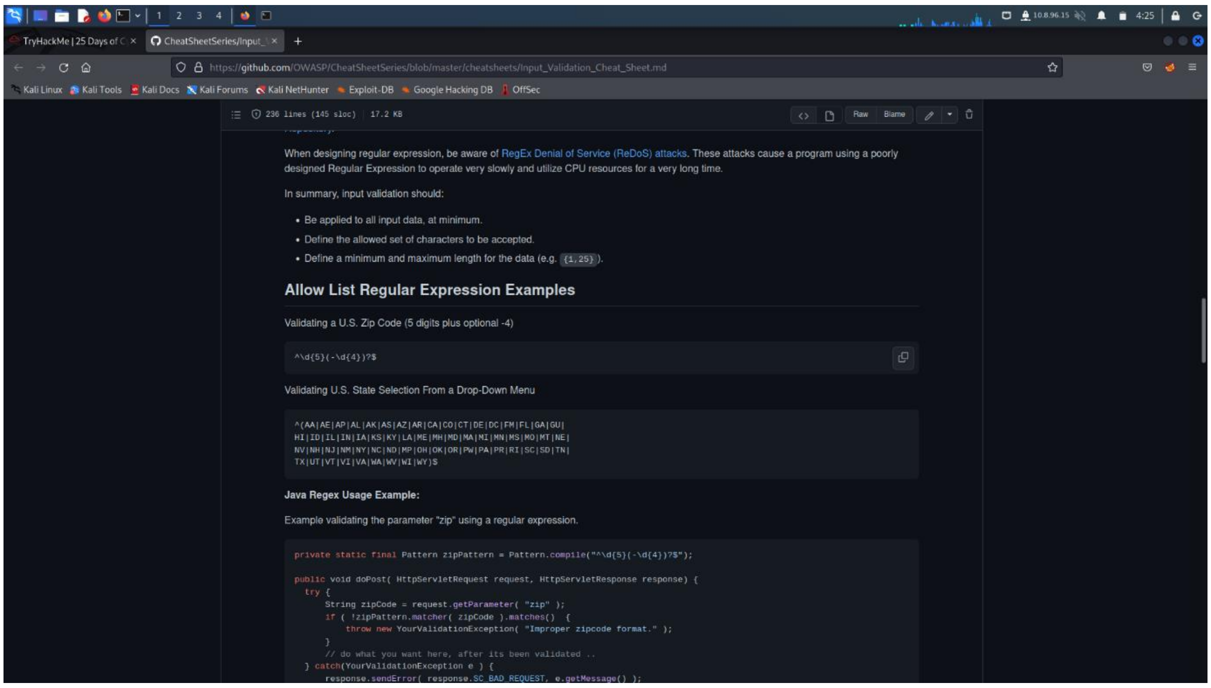
Input validation can be implemented using any programming technique that allows effective enforcement of syntactic and semantic correctness, for example:

- Data type validators available natively in web application frameworks (such as Django Validators, Apache Commons Validators etc).
- Validation against JSON Schema and XML Schema (XSD) for input in these formats.
- Type conversion (e.g. `Integer.parseInt()` in Java, `int()` in Python) with strict exception handling
- Minimum and maximum value range check for numerical parameters and dates, minimum and maximum length check for strings.
- Array of allowed values for small sets of string parameters (e.g. days of week).
- Regular expressions for any other structured data covering the whole input string `(^...$)` and **not** using "any character" wildcard (such as `.` or `\S`)

**Allow list vs block list**

### Question 2

Open the link provided in the room that leads to the OWASP cheat sheet. Examine the content.


https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input_Validation_Cheat_Sheet.md

236 lines (145 sloc) | 17.2 KB

When designing regular expression, be aware of RegEx Denial of Service (ReDoS) attacks. These attacks cause a program using a poorly designed Regular Expression to operate very slowly and utilize CPU resources for a very long time.

In summary, input validation should:

- Be applied to all input data, at minimum.
- Define the allowed set of characters to be accepted.
- Define a minimum and maximum length for the data (e.g. `{1,25}`).

### Allow List Regular Expression Examples

Validating a U.S. Zip Code (5 digits plus optional -4)

```
^\d{5}(-\d{4})?$
```

Validating U.S. State Selection From a Drop-Down Menu

```
^(AA|AE|AP|AL|AK|AS|AZ|AR|CA|CO|CT|DE|DC|FM|FL|GA|GU|
HI|ID|IL|IN|IA|KS|KY|LA|ME|MH|MD|MA|MI|MN|MS|MO|MT|NE|
NV|NH|NJ|NM|NY|NC|ND|MP|OH|OK|OR|PW|PA|PR|RI|SC|SD|TN|
TX|UT|VT|VI|VA|WA|WV|WI|WY)$
```

**Java Regex Usage Example:**

Example validating the parameter "zip" using a regular expression.

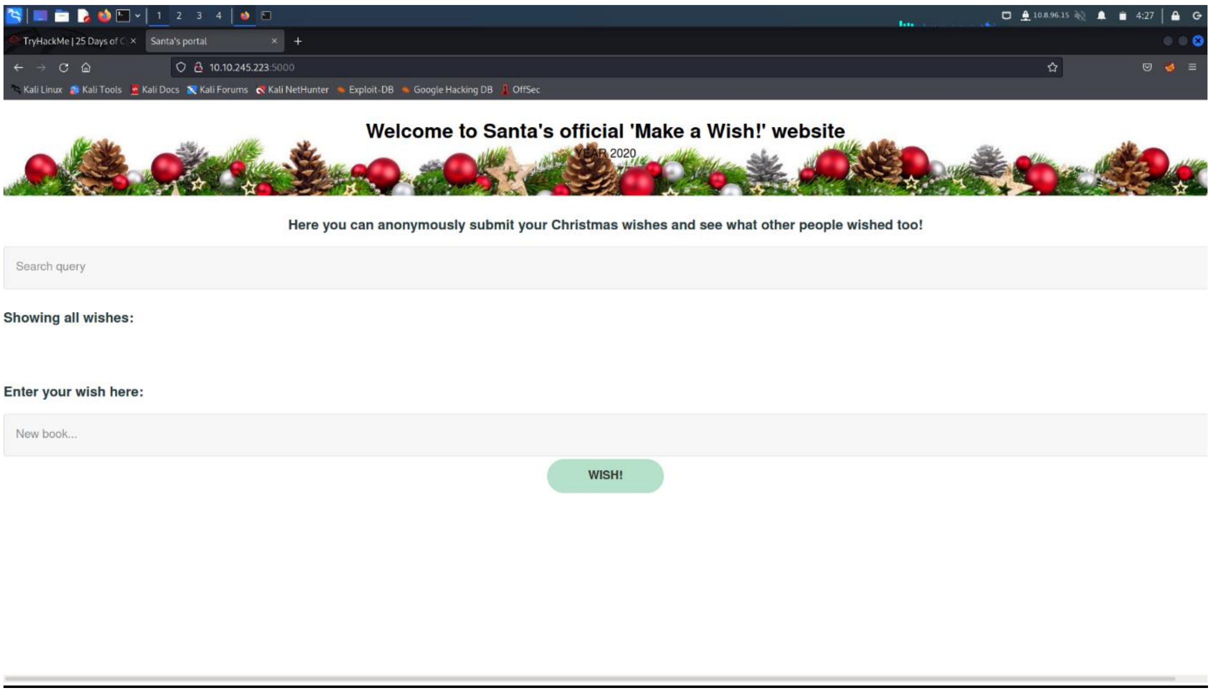
```
private static final Pattern zipPattern = Pattern.compile("^\d{5}(-\d{4})?$");

public void doPost( HttpServletRequest request, HttpServletResponse response) {
  try {
      String zipCode = request.getParameter( "zip" );
      if ( !zipPattern.matcher( zipCode ).matches()  {
          throw new YourValidationException( "Improper zipcode format." );
      }
      // do what you want here, after its been validated ..
  } catch(YourValidationException e ) {
      response.sendError( response.SC_BAD_REQUEST, e.getMessage() );
```

## Question 3

Stored XSS was used to exploit the application since it works when a certain malicious Javascript is submitted and stored directly on the website. In this case, we submitted the malicious Javascript in the wish text box.

## Question 4 (What query string can be abused to craft a reflected XSS?)

Enter the IP address provided by the machine in tryhackme.com into a browser with the port 8000. The URL will lead to a ‘Make a Wish’ website.


Welcome to Santa's official 'Make a Wish!' website

YEAR 2020

Here you can anonymously submit your Christmas wishes and see what other people wished too!

Search query

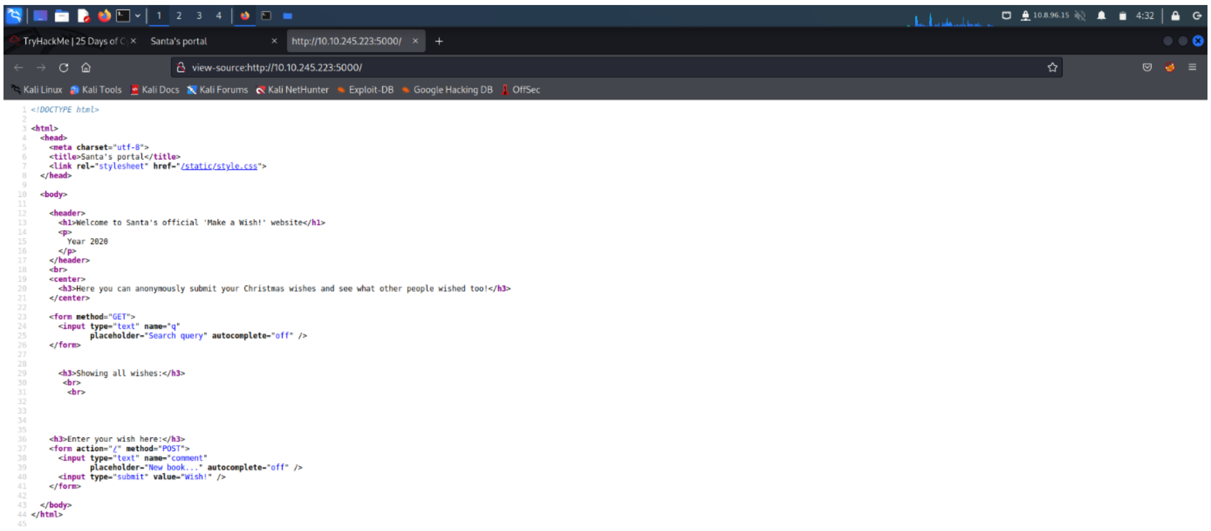
Showing all wishes:

Enter your wish here:

New book...

WISH!

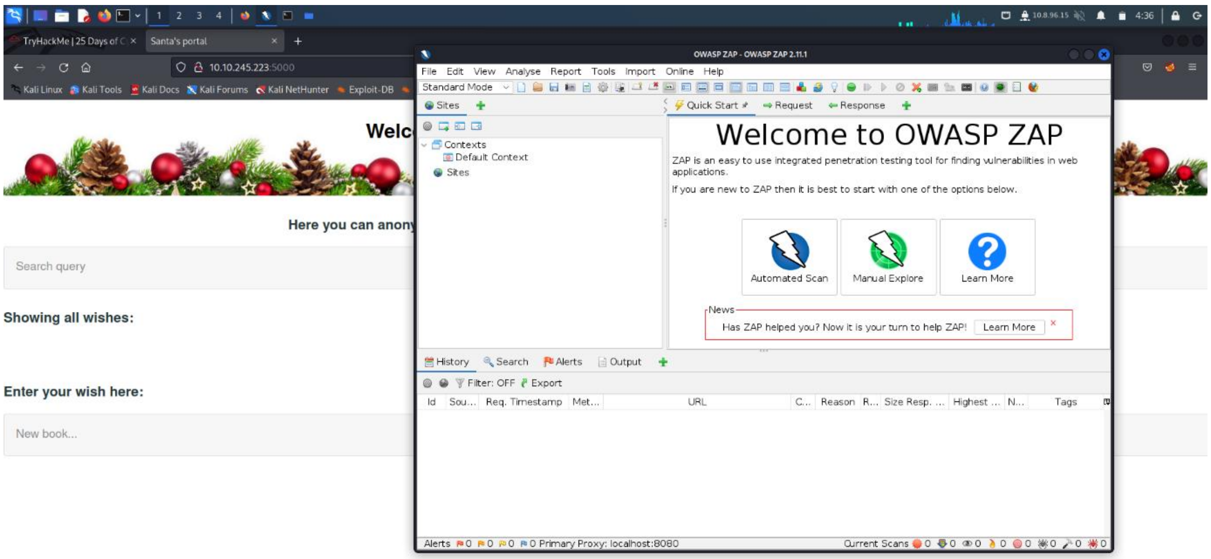
Right click on the mouse and click ‘View Page Source’. This option directs us to the page source where it display the query string that can be abused which is ‘q’.


view-source:http://10.10.245.223:5000/

```html
<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8">
    <title>Santa's portal</title>
    <link rel="stylesheet" href="/static/style.css">
  </head>

  <body>

    <header>
      <h1>Welcome to Santa's official 'Make a Wish!' website</h1>
      <p>
        Year 2020
      </p>
    </header>
    <br>
    <center>
      <h3>Here you can anonymously submit your Christmas wishes and see what other people wished too!</h3>
    </center>

    <form method="GET">
      <input type="text" name="q"
             placeholder="Search query" autocomplete="off" />
    </form>


      <h3>Showing all wishes:</h3>
       <br>
        <br>




    <h3>Enter your wish here:</h3>
    <form action="/" method="POST">
      <input type="text" name="comment"
             placeholder="New book..." autocomplete="off" />
      <input type="submit" value="Wish!" />
    </form>

  </body>
</html>
```

## Question 5

Open application and run the OWASP ZAP. If it is not available in your machine (which happen in my case), download the application by simply searching it in the browser. Once the application is available, click ‘Automated Scan’ to detect any XSS vulnerabilities.

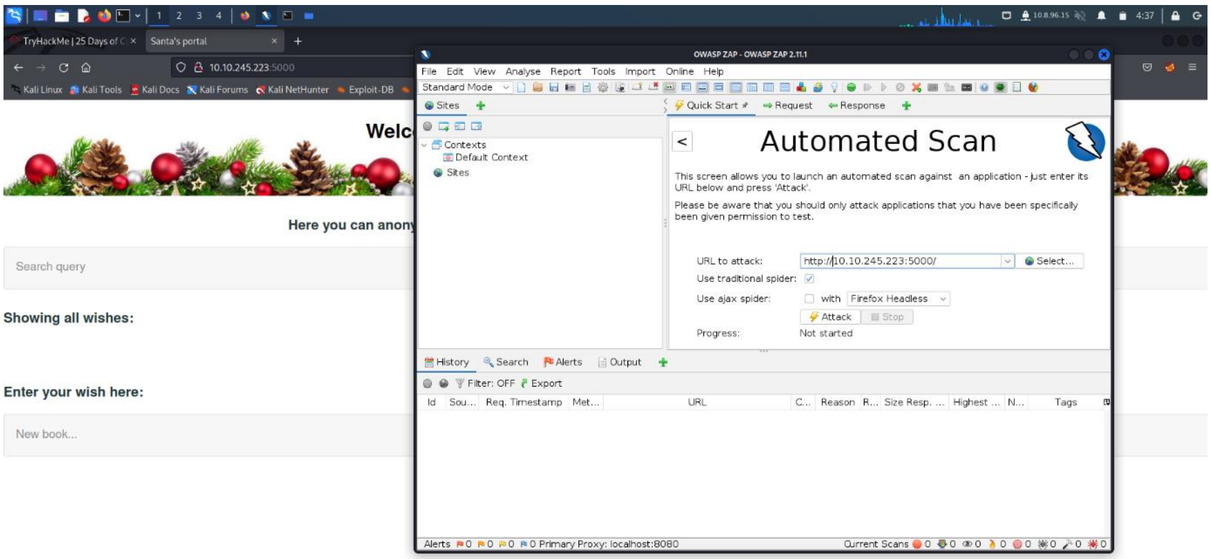


ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

If you are new to ZAP then it is best to start with one of the options below.

Has ZAP helped you? Now it is your turn to help ZAP!

Make sure to enter the URL of the website that we want to exploit and click ‘Attack’.



This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

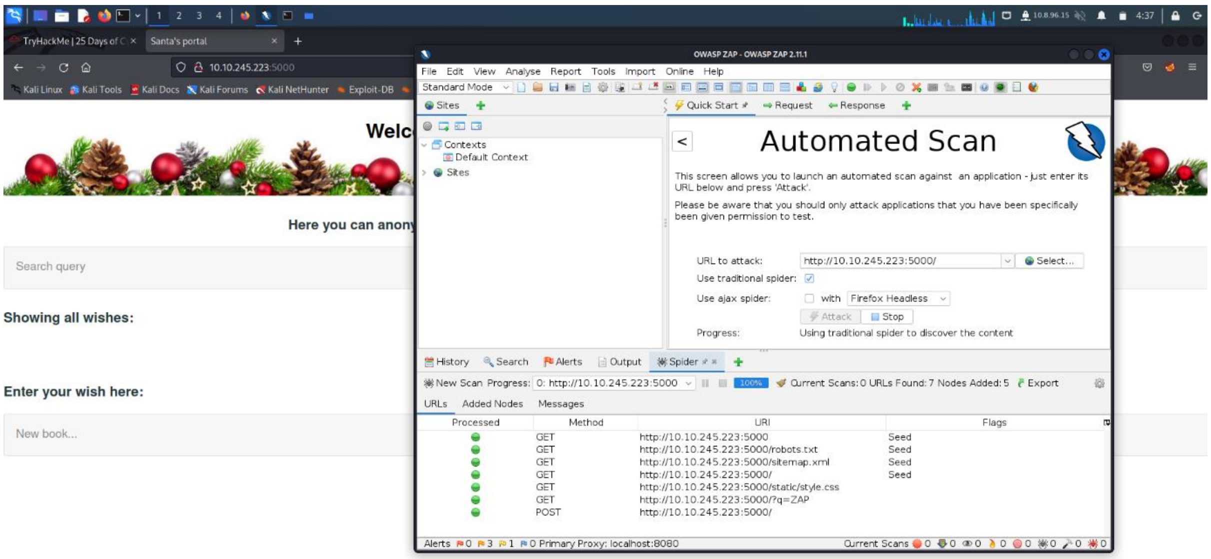
URL to attack: http://10.10.245.223:5000/

Use traditional spider: ✓

Use ajax spider: with Firefox Headless

Progress: Not started

It will take some time for the application to completely scan the website for vulnerabilities.



OWASP ZAP - OWASP ZAP 2.11.1

**Automated Scan**

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

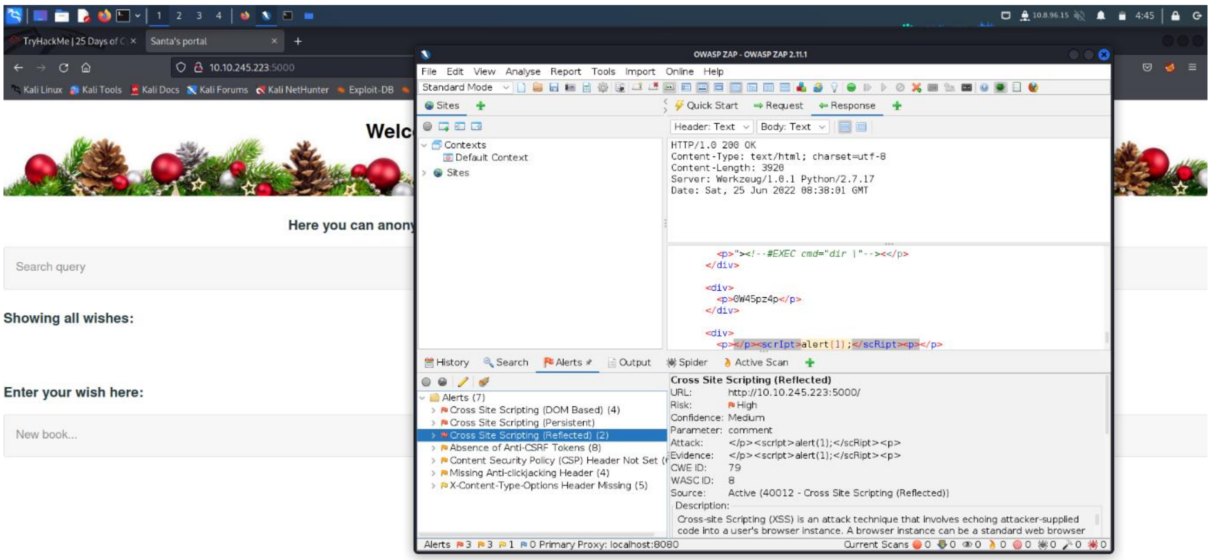
URL to attack: http://10.10.245.223:5000/
Use traditional spider: ✓
Use ajax spider: ☐ with Firefox Headless
Progress: Using traditional spider to discover the content

```
New Scan Progress: 0: http://10.10.245.223:5000  100%  Current Scans: 0 URLs Found: 7 Nodes Added: 5  Export
```

| Processed | Method | URI | Flags |
|---|---|---|---|
| ● | GET | http://10.10.245.223:5000 | Seed |
| ● | GET | http://10.10.245.223:5000/robots.txt | Seed |
| ● | GET | http://10.10.245.223:5000/sitemap.xml | Seed |
| ● | GET | http://10.10.245.223:5000/ | Seed |
| ● | GET | http://10.10.245.223:5000/static/style.css | |
| ● | GET | http://10.10.245.223:5000/?q=ZAP | |
| ● | POST | http://10.10.245.223:5000/ | |

```
Alerts 0 3 1 0 Primary Proxy: localhost:8080
```

Once the scan is complete, the application showed us the vulnerabilities that the website has but due to our Kali's different version, our answer is different than the actual answer. Hence why we included another photo with what AttackBox's Zap 2.0.0 or Kali's 2.11.1 version answer would be.



OWASP ZAP - OWASP ZAP 2.11.1

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 3920
Server: Werkzeug/1.0.1 Python/2.7.17
Date: Sat, 25 Jun 2022 08:38:01 GMT
```

```
    <p>"><!--#EXEC cmd="dir |"--><</p>
  </div>

  <div>
    <p>0W45pz4p</p>
  </div>

  <div>
    <p></p><scrIpt>alert(1);</scRipt><p></p>
```

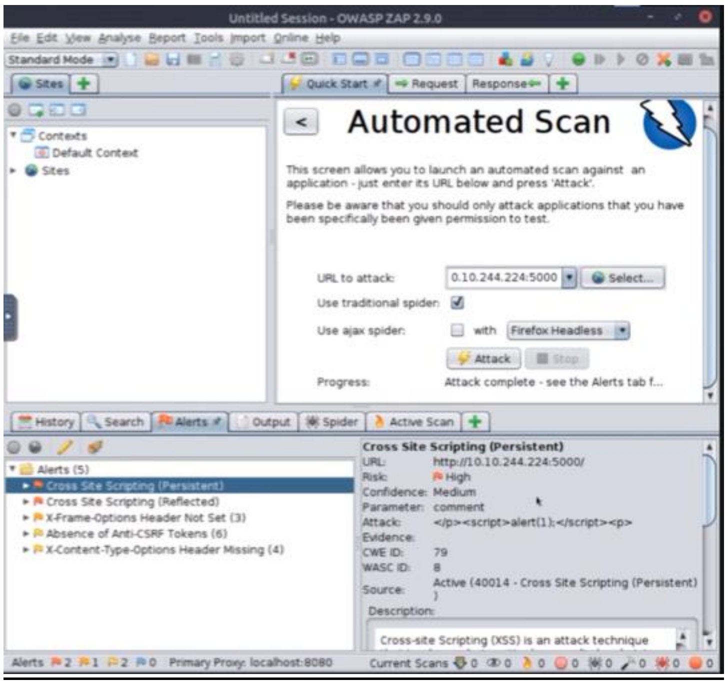
Alerts (7)
- Cross Site Scripting (DOM Based) (4)
- Cross Site Scripting (Persistent)
- Cross Site Scripting (Reflected) (2)
- Absence of Anti-CSRF Tokens (8)
- Content Security Policy (CSP) Header Not Set (
- Missing Anti-clickjacking Header (4)
- X-Content-Type-Options Header Missing (5)

**Cross Site Scripting (Reflected)**

| | |
|---|---|
| URL: | http://10.10.245.223:5000/ |
| Risk: | High |
| Confidence: | Medium |
| Parameter: | comment |
| Attack: | </p><script>alert(1);</scRipt><p> |
| Evidence: | </p><script>alert(1);</scRipt><p> |
| CWE ID: | 79 |
| WASC ID: | 8 |
| Source: | Active (40012 - Cross Site Scripting (Reflected)) |

Description:
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser

```
Alerts 3 3 1 0 Primary Proxy: localhost:8080
```


Untitled Session - OWASP ZAP 2.9.0

Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: 0.10.244.224:5000
Use traditional spider: ✓
Use ajax spider: with Firefox Headless
Progress: Attack complete - see the Alerts tab f...

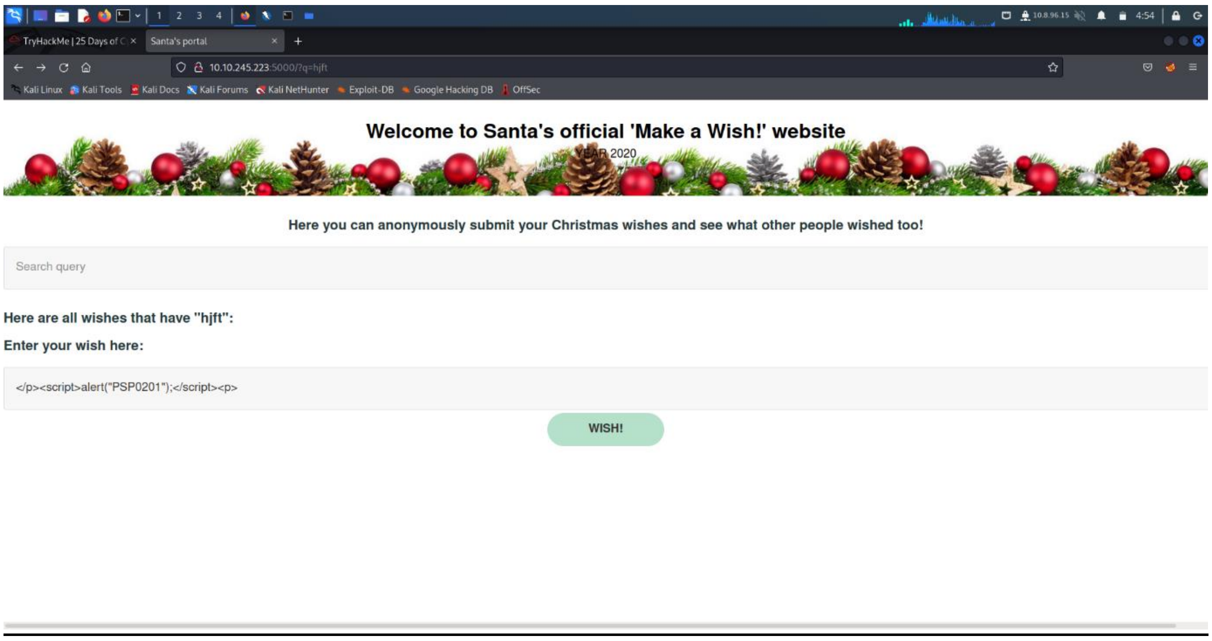
Alerts (5)
- Cross Site Scripting (Persistent)
- Cross Site Scripting (Reflected)
- X-Frame-Options Header Not Set (3)
- Absence of Anti-CSRF Tokens (6)
- X-Content-Type-Options Header Missing (4)

Cross Site Scripting (Persistent)
URL: http://10.10.244.224:5000/
Risk: High
Confidence: Medium
Parameter: comment
Attack: </p><script>alert(1);</script><p>
Evidence:
CWE ID: 79
WASC ID: 8
Source: Active (40014 - Cross Site Scripting (Persistent))
Description:
Cross-site Scripting (XSS) is an attack technique

Alerts 2 1 2 0 Primary Proxy: localhost:8080 Current Scans 0 0 0 0 0 0 0 0

## Question 6

Using the vulnerabilities that we have found using OWASP Zap, we can now exploit the XSS vulnerabilities. Under the '<p></p>' tags, we enter a '<script></script>' tags to execute the 'alert('PSP0201')'. This created an alert when the website is accessed.


10.10.245.223:5000/?q=hjft

Welcome to Santa's official 'Make a Wish!' website

YEAR 2020

Here you can anonymously submit your Christmas wishes and see what other people wished too!

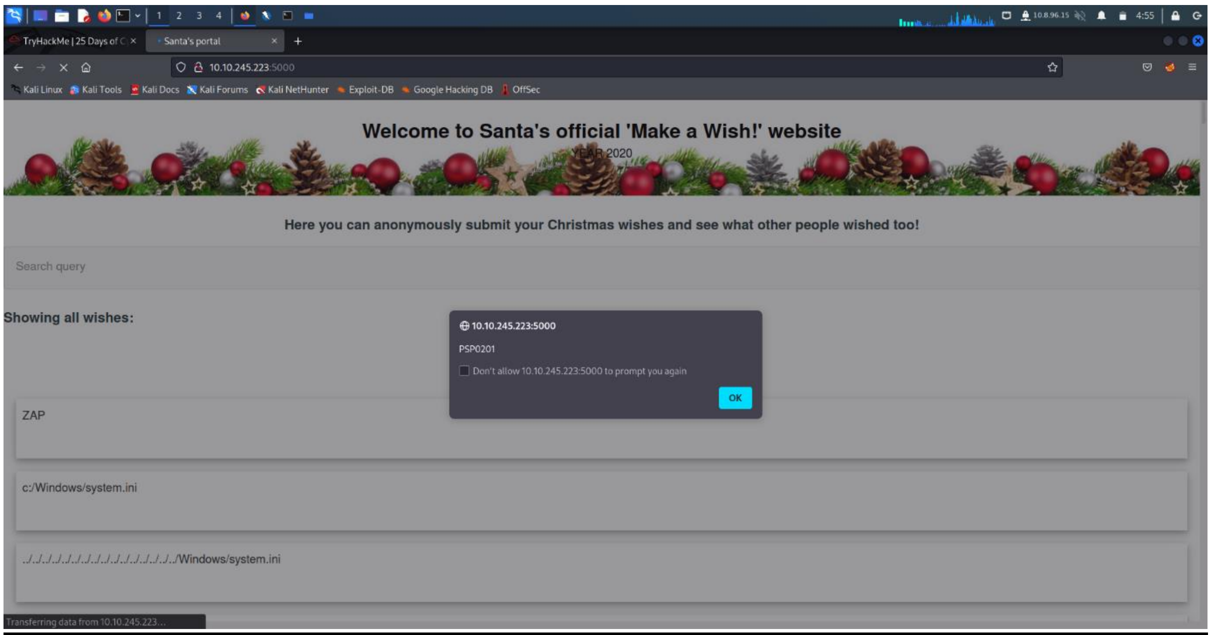
Search query

Here are all wishes that have "hjft":

Enter your wish here:

</p><script>alert("PSP0201");</script><p>

WISH!



Welcome to Santa's official 'Make a Wish!' website

YEAR 2020

Here you can anonymously submit your Christmas wishes and see what other people wished too!

Search query

Showing all wishes:

10.10.245.223:5000

PSP0201

Don't allow 10.10.245.223:5000 to prompt you again

OK

ZAP

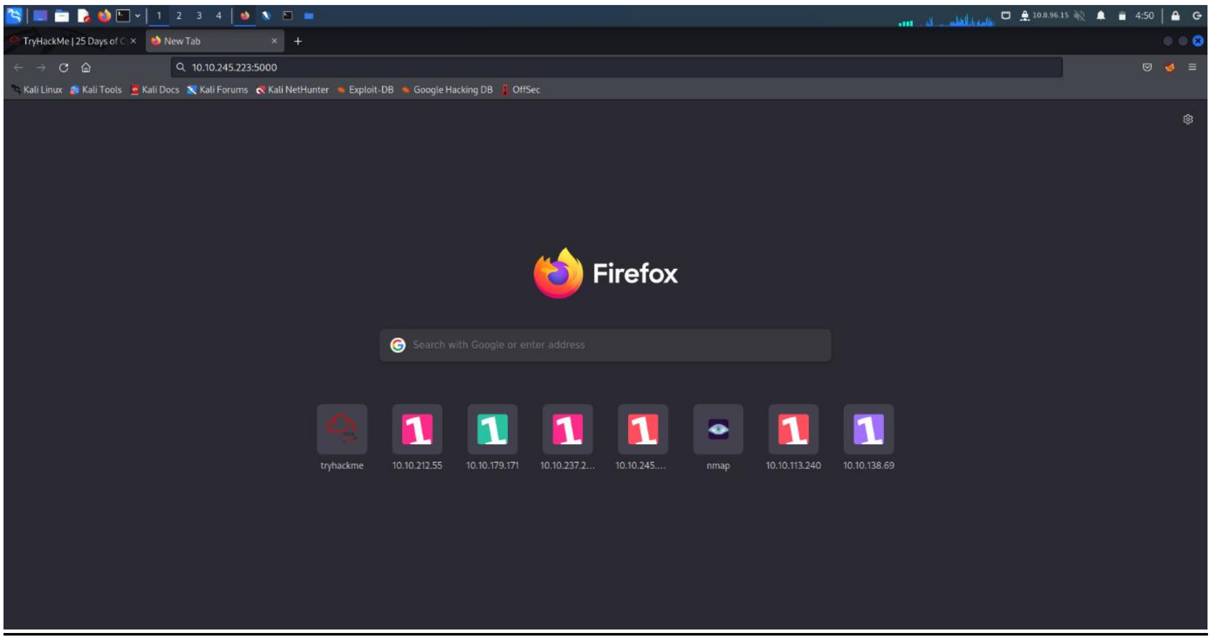
c:/Windows/system.ini

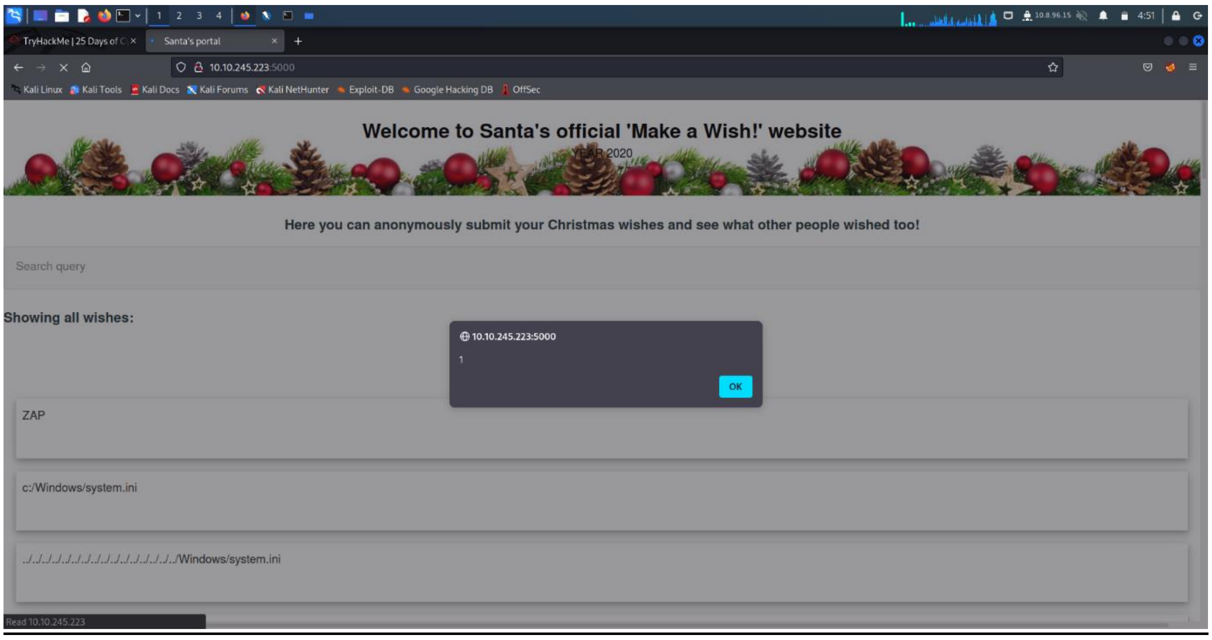
../../../../../../../../../../../../../../../../Windows/system.ini

Transferring data from 10.10.245.223...

## Question 7

We close the browser and re-enter the URL. Once the website loaded, we found that the XSS attack persisted.





Welcome to Santa's official 'Make a Wish!' website

YEAR 2020

Here you can anonymously submit your Christmas wishes and see what other people wished too!

Showing all wishes:

10.10.245.223:5000

1

ZAP

c:/Windows/system.ini

../../../../../../../../../../../../../../Windows/system.ini

**Thought Process/Methodology:**

We open the link provided in the room that led us to the OWASP cheat sheet where there are many details explaining how OWASP works. We examine the content and find the input validation strategies. We continued examining the content and came across the regular expression of validating a U.S Zip Code. After reading and understanding the content explaining XSS exploit in tryhackme.com, we realized that the vulnerabilities that we will exploit is stored XSS. We enter the IP address provided by the machine with the port 8000. The URL then led us to a ‘Make a Wish’ website. To find the query string that we can abuse, we find the page source and discover the query string which is ‘q’. After downloading the OWASP Zap application, we run it and click ‘Automated Scan’ to detect any XSS vulnerabilities. We enter the URL of the website that we want to exploit and click ‘Attack. Once it finished scanning for vulnerabilities, we discover the website XSS vulnerabilities. With that knowledge, we enter a ‘<script></script>’ tags under the ‘<p></p>’ tags and execute the ‘alert(‘PSP0201’)’. This created an alert when the website is accessed. Lastly, we tried closing the browser and re-enter the URL. Upon accessing the web page, we found that the XSS attack persisted.