Cyber Security Fellowship by ByteWise powered by Secure Pwn.

Who we are?

At Secure Pwn, we're passionate about all things tech. From cloud engineering to cybersecurity, we're dedicated to helping our clients stay ahead of the curve in a rapidly evolving digital landscape. Whether you're a student looking to launch your career, an established professional seeking to enhance your skills, or a business in need of cutting-edge services, we've got you covered.

We specialize in organizing technology events that bring together the brightest minds in the industry, providing top-notch training to help you master the latest tools and techniques, and delivering customized services tailored to meet your unique needs. With our team of experienced professionals and our commitment to excellence, we're confident that we can help you achieve your goals and succeed in today's fast-paced world of tech.

So, whether you're looking to stay on top of the latest trends in cloud engineering and cybersecurity, or you're ready to take your business to the next level, trust Secure Pwn to be your partner in success. Contact us today to learn more about how we can help you thrive in the digital age.

Collaboration with ByteWise

We are glad to collaborate with the ByteWise to provide on the course content for Cyber Security. ByteWise has been working extraordinary for providing the students a great learning pattern and practical approach. I am hopeful you will have a great journey learning Cyber Security.

Follow us on LinkedIn

Linkedin: https://www.linkedin.com/company/secure-pwn

Author's profile: https://www.linkedin.com/in/syedalizain033

Author's Email: Syedalizain03@gmail.com

Author's social media: **@syedalizain033**

Good luck,

Contents

What we expect you already know?

- Python (Basics)
- Internet browsing
- Understanding of IT (Basics)
- Quick learning of IT and computer science

This course is completely for beginners. For learning cyber security, there is only one rule: "**DON'T BE A SCRIPT KIDDIE**".

Flask Applications

Flask is a popular and lightweight web framework for building web applications in Python. It is known for its simplicity, flexibility, and ease of use. Flask provides the basic tools and features necessary for creating a web application, while also allowing for customization and extensibility.

To get started with Flask, you'll first need to install it using pip:

```
pip install Flask
```

Once installed, you can create a basic Flask application using the following code:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

This code creates a new Flask application, and defines a single route that responds to requests to the root URL (/) by returning a "Hello, World!" message.

To run this application, you can save the code in a file named app.py, and then run the following command from the command line:

```
flask run
```

or run

```
Python app.py
```

This will start a development server and run the Flask application.

In addition to defining routes, Flask also provides a number of other features and tools for building web applications. For example, you can use Flask's built-in templating engine to generate dynamic HTML pages, and you can use Flask's support for handling forms and file uploads to build more complex web applications.

One of the most common uses of Flask is to build APIs (Application Programming Interfaces) for web services. Flask makes it easy to define routes that respond to HTTP requests with JSON data, which can be used to build client-side applications or integrate with other web services.

Here is an example of a basic Flask API that returns JSON data:

```
from flask import Flask, jsonify
app = Flask(__name__)
@app.route('/api')
def api():
    data = {'message': 'Hello, World!'}
    return jsonify(data)
```

This code defines a new route at /api that returns a JSON object containing a "message" field with the value "Hello, World!". With Flask, you can also define custom error pages, add authentication and authorization to your application, and even integrate with other Python libraries and frameworks to extend its functionality.

Overall, Flask is a powerful and versatile web framework that makes it easy to build web applications and APIs in Python.

Building API Application

We shall quickly build an API application where we shall send a POST data to the Application and get back as the API response. Python code is given below.

```
from flask import Flask, jsonify, request,render_template
app = Flask(__name__)
@app.route('/',methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        name = request.form['name']
        return jsonify({'name': name})
    else:
        return render_template('home.html')
if __name__ == '__main__':
    app.run(debug=True)
```

In this code we explained a route “/” means localhost/ or 127.0.0.1/. If we write (‘/api/’) instead, we shall have to request the 127.0.0.1/api in order to get that page. We have to create a folder of templates as well so we could put our HTML files in it. We are putting home.html file in it with below code.

```
<!DOCTYPE html>
<html>
<head>
	<title>Flask API Example</title>
</head>
<body>
	<form id="my-form">
		<label for="name">Name:</label>
		<input type="text" name="name" id="name-input"><br>
		<button type="submit">Submit</button>
	</form>

	<div id="output"></div>

	<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
	<script>
		$(document).ready(function() {
			$("#my-form").submit(function(event) {
				event.preventDefault();
				var name = $("#name-input").val();

				$.post("/", {name: name})
					.done(function(data) {
						$("#output").html("Hello, " + data.name + "!");
					})
					.fail(function() {
						console.log("Error occurred");
					});
			});
		});
	</script>
</body>
</html>
```

This code contains some classic Javascript to request the “/” endpoint and on post request completion it would insert the name variable in a tag which contains “id” equals to “output” but we won’t go deeper into JavaScript for now but above example explains how API works.

As advance topics, the variable in the API gets saved to the database and is retrieved from the database which must be a self-learning tasks as track would help both the developers and the testers.

Content reflections

If we notice we see content being reflected to the web page that the user inputs. Now imagine if we see content being reflected is becoming as the part of HTML code or JS but still the HTML code. If we observe it as a hacker's perspective, hacker is controlling a part of HTML code coming from the server. What does the browser do? Browser's job is to render the HTML code that comes in response for us and provide JavaScript an engine to execute its functionalities as compiler. The HTML code with any user's input gets executed by the browser tricking the user.

Cross-Side Scripting

Cross-Site Scripting (XSS) is a type of web security vulnerability that allows an attacker to inject malicious scripts into a web page viewed by other users. It occurs when an application does not properly validate user input before displaying it back to the user.

XSS can be divided into two categories: reflected and stored. Reflected XSS occurs when the malicious script is reflected off the web server, such as in the search results page, and is immediately executed by the user's browser. Stored XSS occurs when the script is stored in a database or on the web server and is executed whenever the user requests the page.

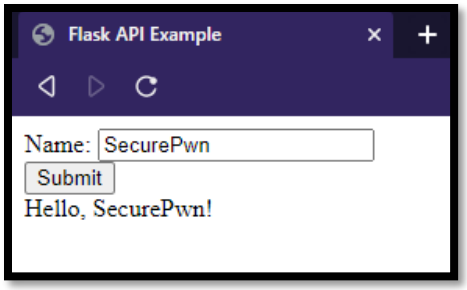
To demonstrate the danger of XSS, consider the following example. Suppose there is a search bar on a web page that allows users to search for products. The search query is displayed back to the user in a sentence, like "Your search for 'iphone' returned 10 results." If the input is not properly sanitized, an attacker could inject a script into the search query, like <script>alert('XSS Attack');</script>. When the user submits the search query, the script is executed in their browser, resulting in an alert box displaying "XSS Attack."

To prevent XSS attacks, web developers should sanitize all user input, whether it is from a form field, URL parameter, or cookie. This can be done using various techniques such as encoding special characters, validating input against a whitelist of allowed characters, and using content security policies. If we observe the above example, do we find any sanitization being implemented? Well, we can test it.

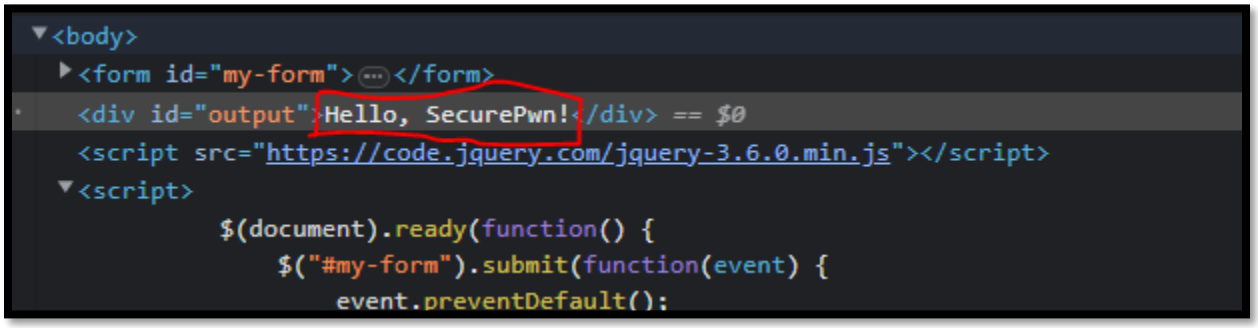
XSS Testing methodology

We shall start with certain steps.

1) We shall test for the input and observe the reflection. Try to use a unique string.



2) Observe in the source code to find the right context where it is being reflected.



The context I see is Hello, SecurePwn! so we need to think which code would look real to the browser if we modify the code.

3) We shall put the right context code so it gets executed well and in right context by the browser.

The above code example shows the best case that how the simple H1 tag is getting executed by the browser because we did not know if it was well sanitized or not.

Question is what could go wrong? We know HTML is not harmful but can we do something harmful? Let's study the JavaScript and Cookie affair.

JavaScript and Cookie

JavaScript is a high-level programming language that is commonly used for developing web applications. It is a client-side scripting language that runs on the user's web browser and allows for dynamic interactions between the user and the web application. JavaScript is used for a wide range of functions, including form validation, manipulating the DOM, and creating interactive user interfaces.

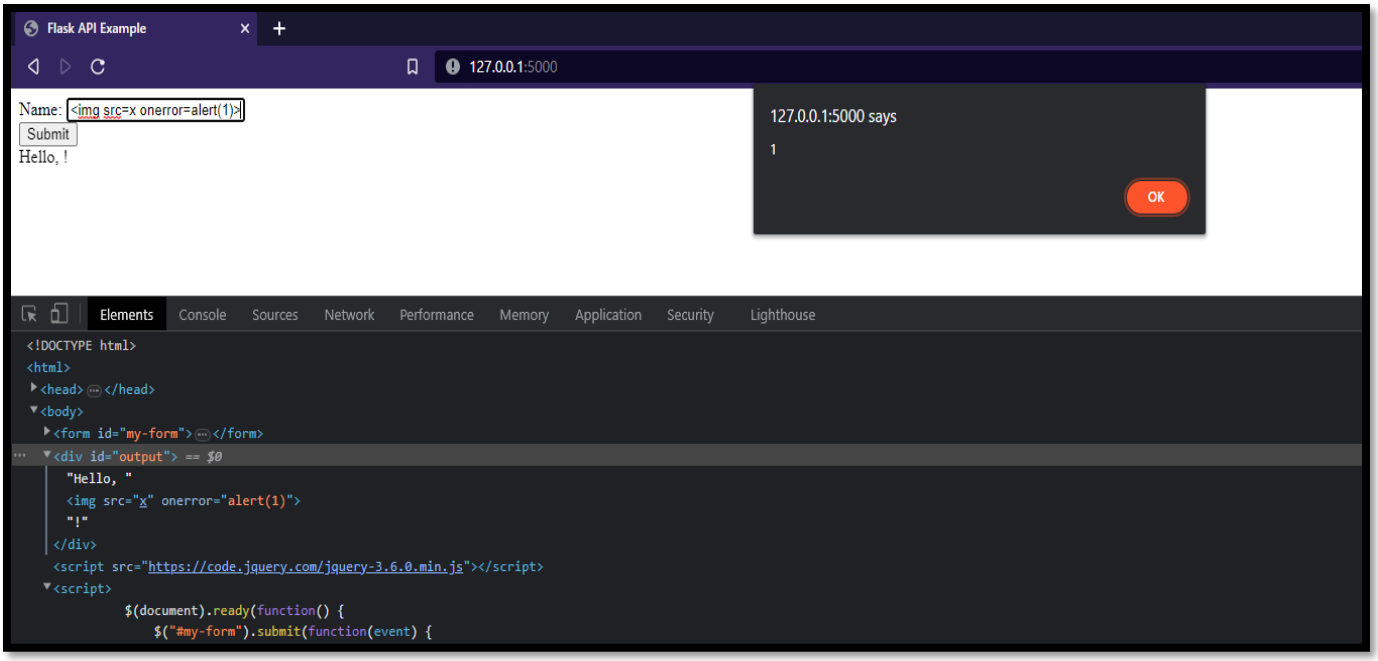
Session cookies are small pieces of data that are stored on the user's browser by the web server. They are used to maintain the state of the user's session on the website. Session cookies are unique to each user session and are deleted when the user closes the browser window. They are commonly used for authentication and tracking user behavior on the website.

JavaScript can be used to interact with session cookies on the user's browser. For example, JavaScript can read, write, and delete cookies. This can be used to store user preferences or to maintain the user's session state across multiple pages on the website. However, the use of JavaScript with session cookies can also create security risks, particularly with cross-site scripting (XSS) attacks. XSS attacks involve injecting malicious code into a web page viewed by other users. Attackers can use JavaScript to read or modify the content of session cookies, steal sensitive user data, or impersonate the user.

In **above example** we can exploit the vulnerability with the execution of JavaScript. There are several ways we can execute the JavaScript here so my favorite way is using **IMG** tag. Below is the payload

```
<img src=x onerror=alert(1)>
```

This code tries to access the x path for image and as it does not exist it moves to "onerror" which executes the JavaScript. Alert is used for a demonstration as it visualizes the output very well. There are several functions we use like "confirm()","prompt()". Let's put this as input and observe the output.



We can clearly see that the HTML code got executed and the JavaScript is being executed. There are several other cases as the XSS is a very wide range topic. We shall dive deep into it.

Patch

We shall work on a server-side patch on this where we properly sanitize the data before we send it back as response to the API. There are several ways to secure website from the XSS but its always hard to beat the XSS but should be based on the context.

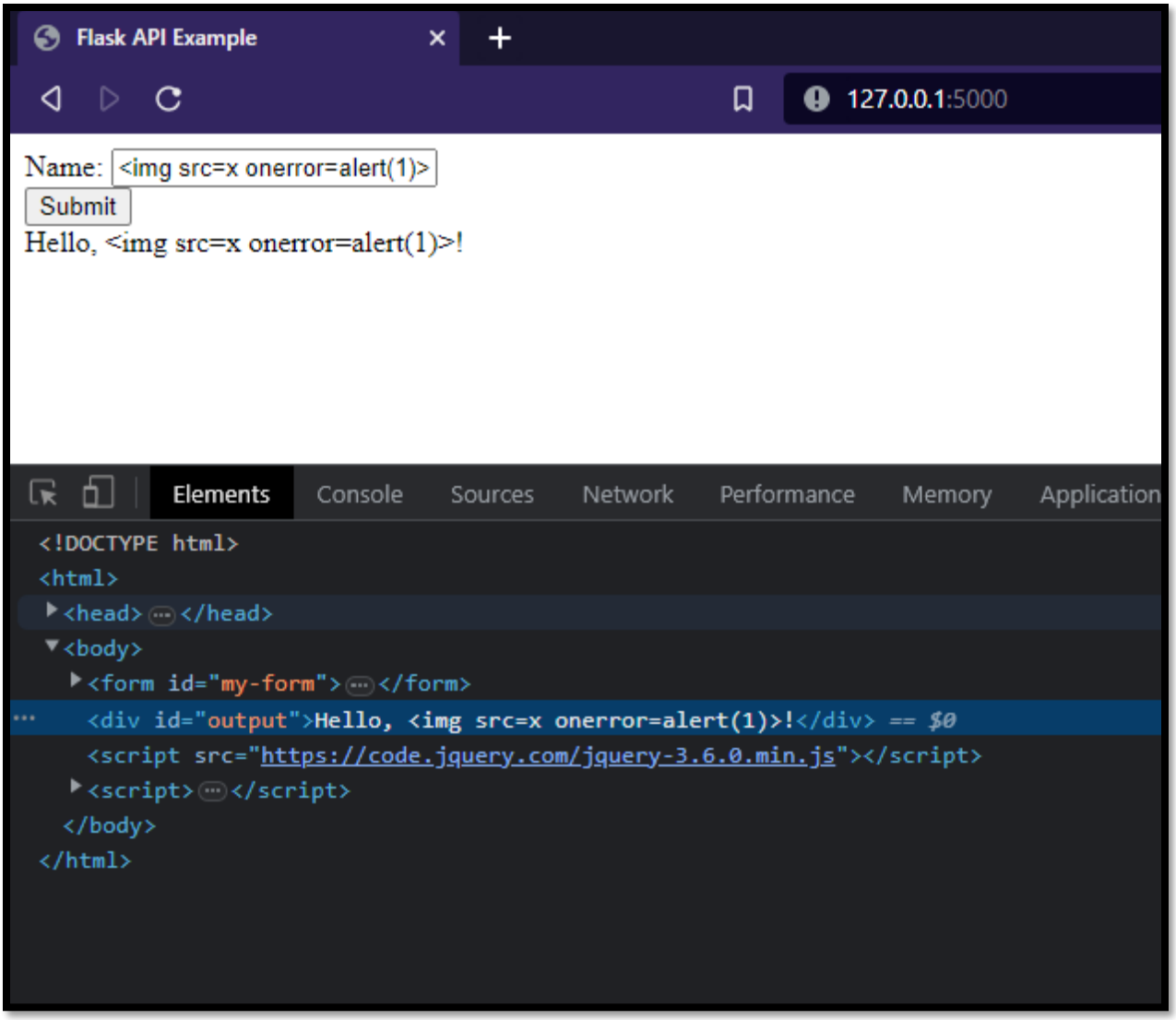
In this context, we can secure using the following functions.

```
def sanitize_input(input_str):
    sanitized_str = input_str.replace('<', '&lt;').replace('>', '&gt;')
    return sanitized_str
```

We can utilize in code like this.

```
name = sanitize_input( request.form['name'])
```

Ultimately, we see that after this code the XSS does not gets triggered.



This is one case of XSS. There are several other cases of XSS where the reflection occurs in different context of the HTML or CSS or JS. Based on the context you have to break the context and trigger JS. Let's review a few examples of XSS in 2 different contexts with string "SecurePwn" as reflection. If you can not understand them, you probably need to learn HTML, CSS, JS from W3schools to understand the structure.

```
<input type="text" value="SecurePwn">
Payload: "><img src=x onerror="alert()"><"
After payload: <input type="text" value="SecurePwn"><img src=x onerror="alert()"><"">
Payload2: " onmouseover=alert() "
After payload 2: <input type="text" value="SecurePwn" onmouseover=alert() "">
```

```
var x="Welcome SecurePwn"
Payload: "-alert(1)-"
After payload: var x="Welcome SecurePwn"-alert(1)-""
```

There are several other ways as well which you need to study from Ashar Javed's video on XSS. Read it from here.

Once you understand everything about the XSS, time to jump into practical from PortSwigger. Also visit some XSS labs for context from here. Once you are good with the XSS, you can make further research on bypassing and different sanitizations both from Developer's perspective and Tester's perspective.

Template Injection

XSS is one simple way where the input is reflecting so we trigger the JavaScript. It's not always the case and several other technologies are also working the same way with different technology stack having different behavior where the Template Injection comes in.

What is template

A template is a predefined format that acts as a base for creating something new. In the context of web development, templates are used to create web pages or web applications that display data dynamically. Templates allow developers to separate the presentation layer from the application logic and data layer. By doing so, templates make it easy to maintain and modify the appearance of the application without having to change the underlying code. Templates typically contain placeholders or variables that can be filled in with data dynamically at runtime. The final output is generated by combining the template with the data, resulting in a complete HTML page or other output format. Templates are used in almost every web technology stack, from Flask to Django, from Angular to React, and beyond.

1. Flask (Python):

Templates in Flask are written in Jinja2. The syntax is similar to Django templates. The templates are HTML files, but with variables, control structures, and expressions. Here's an example of a Flask template:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <h1>Hello, {{ name }}!</h1>
  </body>
</html>
```

In this template, the **{{ title }}** and **{{ name }}** are variables that will be replaced with actual values at runtime. To render this template, you can use the **render_template()** function provided by Flask.

2. Ruby on Rails:

Templates in Rails are called ERB templates, which stands for Embedded Ruby. They are essentially HTML files with embedded Ruby code. Here's an example:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= @title %></title>
  </head>
  <body>
    <h1>Hello, <%= @name %>!</h1>
  </body>
</html>
```

In this template, the **<%= @title %>** and **<%= @name %>** are Ruby code that will be evaluated at runtime and replaced with actual values. To render this template, you can use the render method provided by Rails.

3. Angular (JavaScript):

Templates in Angular are written in HTML, but with Angular-specific syntax. The syntax is based on curly braces and directives. Here's an example:

```
<div>
    <h1>{{ title }}</h1>
    <p *ngFor="let name of names">{{ name }}</p>
  </div>
```

In this template, the **{{ title }}** is a variable that will be replaced with an actual value at runtime, and the ***ngFor** directive is used to loop through an array of names. To render this template, you can use the **Component** decorator provided by Angular.

4. Vue.js (JavaScript):

Templates in Vue.js are similar to Angular templates, but with a slightly different syntax. Here's an example:

```
<div>
    <h1>{{ title }}</h1>
    <ul>
      <li v-for="name in names">{{ name }}</li>
    </ul>
  </div>
```

In this template, the **{{ title }}** is a variable that will be replaced with an actual value at runtime, and the **v-for** directive is used to loop through an array of names. To render this template, you can use the **Vue** object provided by Vue.js.

Exploiting

1. Jinja2:

Jinja2 is a popular templating engine for Python. In Jinja2, template injection can be exploited by injecting code into variables that are used in template tags such as {% for %} and {% if %}. For example, injecting the following code: {{ 7*'7' }} can cause the template to render 7777777.

2. Handlebars.js:

Handlebars.js is a popular templating engine for JavaScript. In Handlebars.js, template injection can be exploited by injecting code into variables that are used in template expressions such as {{ }}. For example, injecting the following code: {{#with "7"}}{{#each $root}}{{this}}{{/each}}{{/with}} can cause the template to render 7777777.

3. AngularJS:

AngularJS is a popular JavaScript framework. In AngularJS, template injection can be exploited by injecting code into variables that are used in template expressions such as {{ }}. For example, injecting the following code: {{constructor.constructor('alert(1)')()}} can cause the template to execute the alert() function.

4. Ruby on Rails:

Ruby on Rails is a popular web framework for Ruby. In Ruby on Rails, template injection can be exploited by injecting code into variables that are used in ERB tags such as <% %> and <%= %>. For example, injecting the following code: <%= 7 * 7 %> can cause the template to render 49.

5. Vue.js:

Vue.js is a popular JavaScript framework. In Vue.js, template injection can be exploited by injecting code into variables that are used in template expressions such as {{ }}. For example, injecting the following code: {{7*7}} can cause the template to render 49.

There is much more about Template Injection. Read more on PortSwigger and try out it's labs as well. Try to make grip on it because next we are going to solve HackTheBox.

Application Exploitation

Its not just limited to the detecting it through the mathematical operations. This vulnerability is much beyond it. There are several payloads that you can study and build while exploitation. All the payloads can be found on the GitHub repo of PayloadAllTheThings.

HackTheBox time

Its time to make some impact on the community as well by showcasing your skills by solving a web challenge on Hackthebox. Usually, we never know which vulnerability is on the Hackthebox but I am giving you a hint of this vulnerability that you have to exploit on your own. The more you put payloads from **Paylaodallthethings**, the more you will get idea about how is it working. Make sure to utilize **BurpSuite** for this. Your goal is to execute system commands like `ls` to see the files and there would be a file with name **flag** or **flag.txt** which you have to read.

Get challenge link from here. Solve it on your own.