

# AI for Cyber Security With IBM Qrador (AI for Web Security)

## ASSIGNMENT-1

NAME : OMKAR SANJAY NARKAR

Reg. no. : 21BCE8412



## OWASP 2021 Report's Top 10

Although the list is updated periodically to reflect the ever evolving threat landscape, the last issued report for web applications was in 2021 and in that report, the vulnerabilities were:

1. A01:2021-Broken Access Control
2. A02:2021-Cryptographic Failures
3. A03:2021-Injection
4. A04:2021-Insecure Design
5. A05:2021-Security Misconfiguration
6. A06:2021-Vulnerable and Outdated Components
7. A07:2021-Identification and Authentication Failures
8. A08:2021-Software and Data Integrity Failures
9. A09:2021-Security Logging and Monitoring Failures
  
10. A10:2021-Server-Side Request Forgery

## Vulnerability Documentation

### 1. Broken Access Control

Broken access control is a security vulnerability that occurs when a system's access control mechanisms, which are designed to restrict unauthorized users from accessing certain resources or performing specific actions, are not properly implemented or enforced. Access control is a fundamental aspect of information security and is crucial for protecting sensitive data and maintaining the integrity of systems.

### **Impact:**

When access control is broken, it can lead to unauthorized users gaining access to resources or performing actions they should not be allowed to. This can have serious consequences, such as data breaches, unauthorized modifications, data leaks, and more.

### **List of Mapped CWEs:**

CWE Code	Name
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-23	Relative Path Traversal
CWE-35	Path Traversal: '.../...//'
CWE-59	Improper Link Resolution Before File Access ('Link Following')
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor
CWE-201	Exposure of Sensitive Information Through Sent Data
CWE-219	Storage of File with Sensitive Data Under Web Root
CWE-264	Permissions, Privileges, and Access Controls (should no longer be used)
CWE-275	Permission Issues
CWE-276	Incorrect Default Permissions
CWE-284	Improper Access Control
CWE-285	Improper Authorization
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-359	Exposure of Private Personal Information to an Unauthorized Actor
CWE-377	Insecure Temporary File
CWE-402	Transmission of Private Resources into a New Sphere ('Resource Leak')
CWE-425	Direct Request ('Forced Browsing')
CWE-441	Unintended Proxy or Intermediary ('Confused Deputy')
CWE-497	Exposure of Sensitive System Information to an Unauthorized Control Sphere
CWE-538	Insertion of Sensitive Information into Externally-Accessible File or Directory
CWE-540	Inclusion of Sensitive Information in Source Code

CWE-548	Exposure of Information Through Directory Listing
CWE-552	Files or Directories Accessible to External Parties
CWE-566	Authorization Bypass Through User-Controlled SQL Primary Key
CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
CWE-639	Authorization Bypass Through User-Controlled Key
CWE-651	Exposure of WSDL File Containing Sensitive Information
CWE-668	Exposure of Resource to Wrong Sphere
CWE-706	Use of Incorrectly-Resolved Name or Reference
CWE-862	Missing Authorization
CWE-863	Incorrect Authorization
CWE-913	Improper Control of Dynamically-Managed Code Resources
CWE-922	Insecure Storage of Sensitive Information
CWE-1275	Sensitive Cookie with Improper SameSite Attribute

### **Example: CWE-201 (Exposure of Sensitive Information Through Sent Data)**

Picked from the MITRE Corporation's Web Definitions for CWES

#### **Description**

The code transmits data to another actor, but a portion of the data includes sensitive information that should not be accessible to that actor.

Sensitive information could include data that is sensitive in and of itself (such as credentials or private messages), or otherwise useful in the further exploitation of the system (such as internal file system structure).

## **Business Impact**

Sending data that contains sensitive information to unauthorized actors can result in severe confidentiality breaches. This can lead to unauthorized access, data leaks, identity theft, and other malicious activities, undermining user trust, violating compliance regulations, and causing reputational damage. Additionally, the exploitation of system internals can provide attackers insights into the system's architecture, potentially aiding in further attacks and exploitation.

## **Overview of the target:**

The target website is FastFoodHackings. It is a website meant for bug bounty practice so I am authorized to hack into the website and/or to try to exploit common vulnerabilities.

## **Methodology/Procedure to Exploit:**

I found the above bug while monitoring the data exchange happening while I book an appointment in BurpSuite. I was exploring the website in a Chromium browser proxied thru BurpSuite. Even for anything that doesn't require proxying or intercepting, I usually browse the target here itself since it picks up HTTP and WebSocket history that I can later analyse, or it will pick up stuff on the dashboard (A feature I'm not too comfortable with).

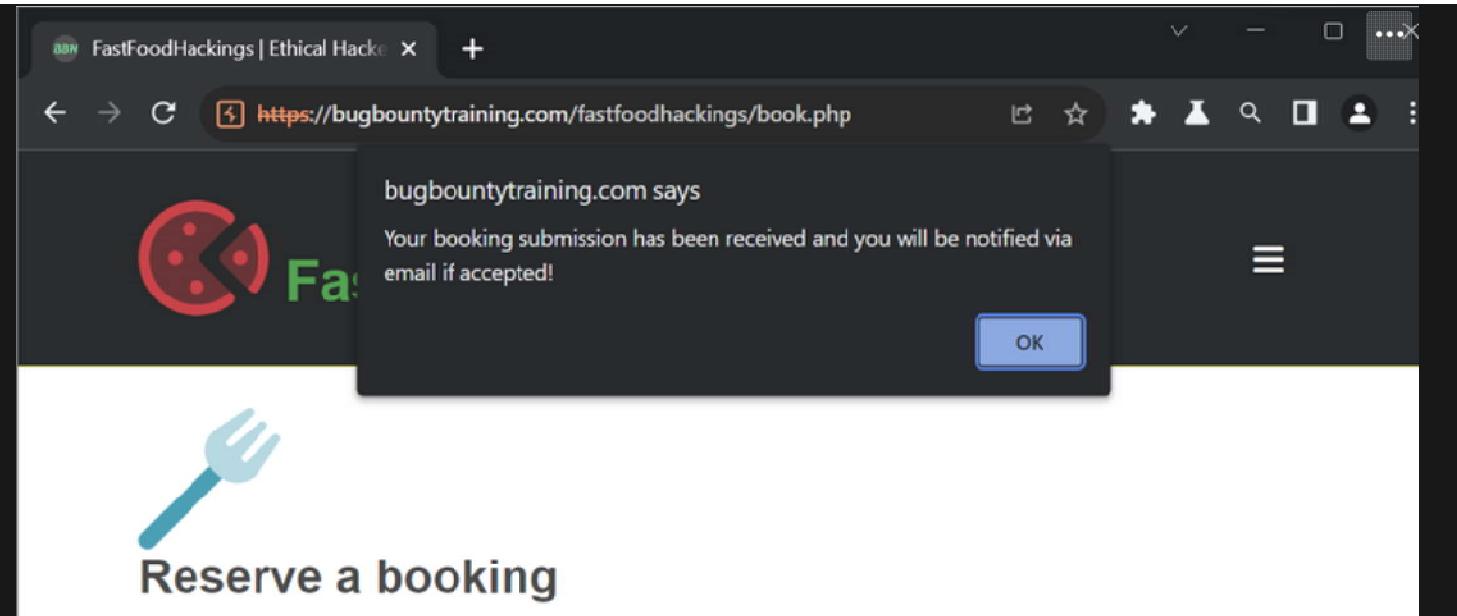
The booking page normally works as follows:

The screenshot shows a web browser window for 'FastFoodHackings' with a booking form. The form includes fields for 'Full Name' (John Doe), 'Email Address' (test@test.com), and 'Date' (08-08-2023). A green 'RESERVE BOOKING' button is visible. Below the form is a section titled 'BROWSE OUR MENU' with images of pizzas.

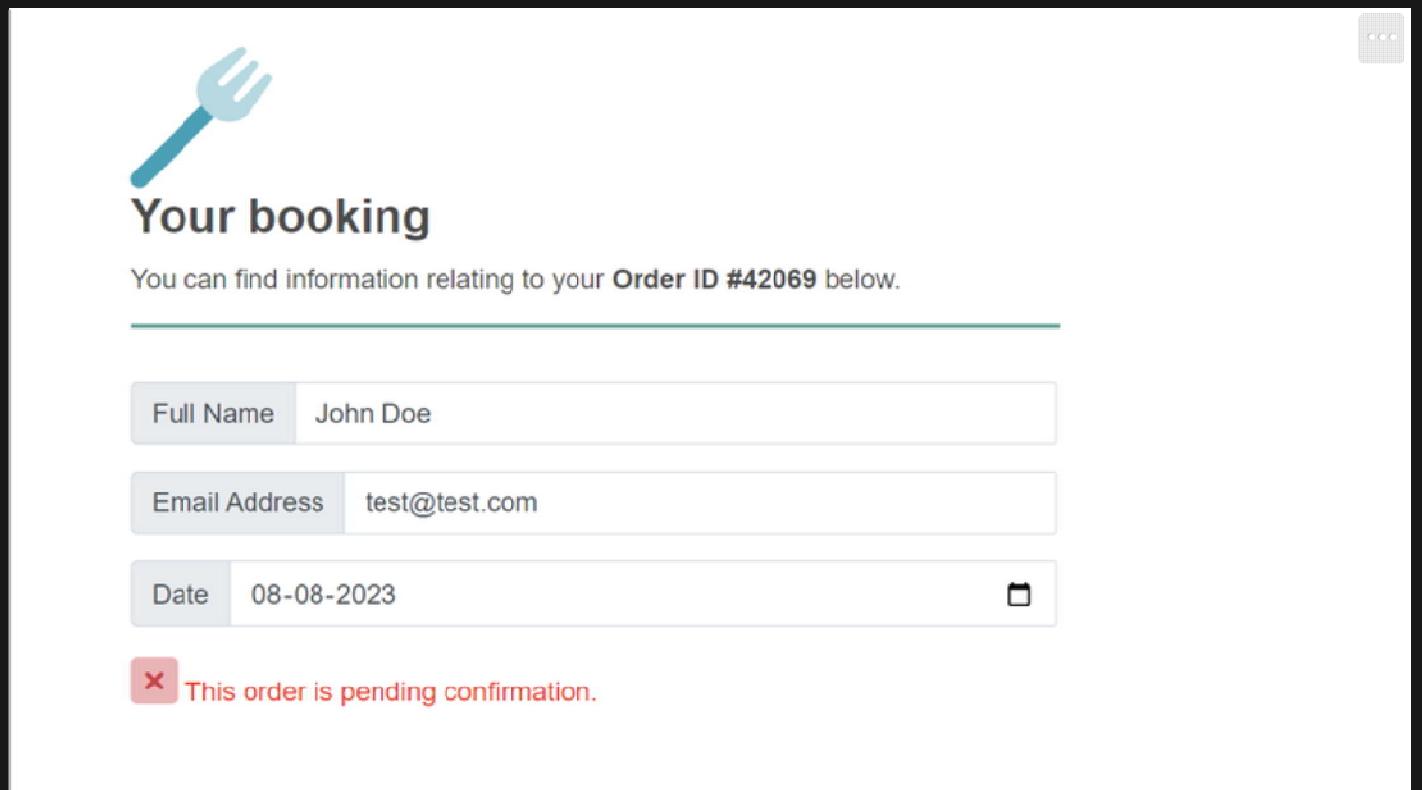
On the right side of the screen, the Burp Suite interface is open. The 'Proxy' tab is selected, showing a list of network requests. The table below lists the details of the captured requests:

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extens.
42	https://cdn.jsdelivr.net	GET	/jquerysmashrview/0.1/jquery-deliv...			200	2136	script	js
44	https://bugbountytraining.com	HEAD	/fastfoodhacking/index.php			404	138	HTML	php
45	https://bugbountytraining.com	GET	/fastfoodhacking/index.php			200	22404	HTML	php
47	https://bugbountytraining.com	GET	/fastfoodhacking/index.php			200	22404	HTML	php
48	https://bugbountytraining.com	HEAD	/fastfoodhacking/index.php			404	138	HTML	php
70	https://bugbountytraining.com	GET	/fastfoodhacking/index.php			200	9347	HTML	php
72	https://www.fonts.googleapis...	GET	/css?family=Open+Sans:400,700,100						✓
73	https://www.fonts.googleapis...	GET	/css?family=Rambda:400,700						✓
74	https://www.fonts.googleapis...	GET	/css?family=Calligraffiti						✓
75	https://www.fonts.googleapis...	GET	/css?family=Roboto+Slab:400,700						✓
76	https://bugbountytraining.com	GET	/fastfoodhacking/book.php						200
									9347
									HTML
									php

The booking information is entered into the website



The website returns an alert that its underprocess.



And you are then redirected to a site that says that your order is pending confirmation.

So now I tried to intercept the traffic, This time I entered my own details and tried again (But not recommended practice)

The screenshot shows a web browser window and a Burp Suite interface side-by-side.

**Web Browser (Left):**

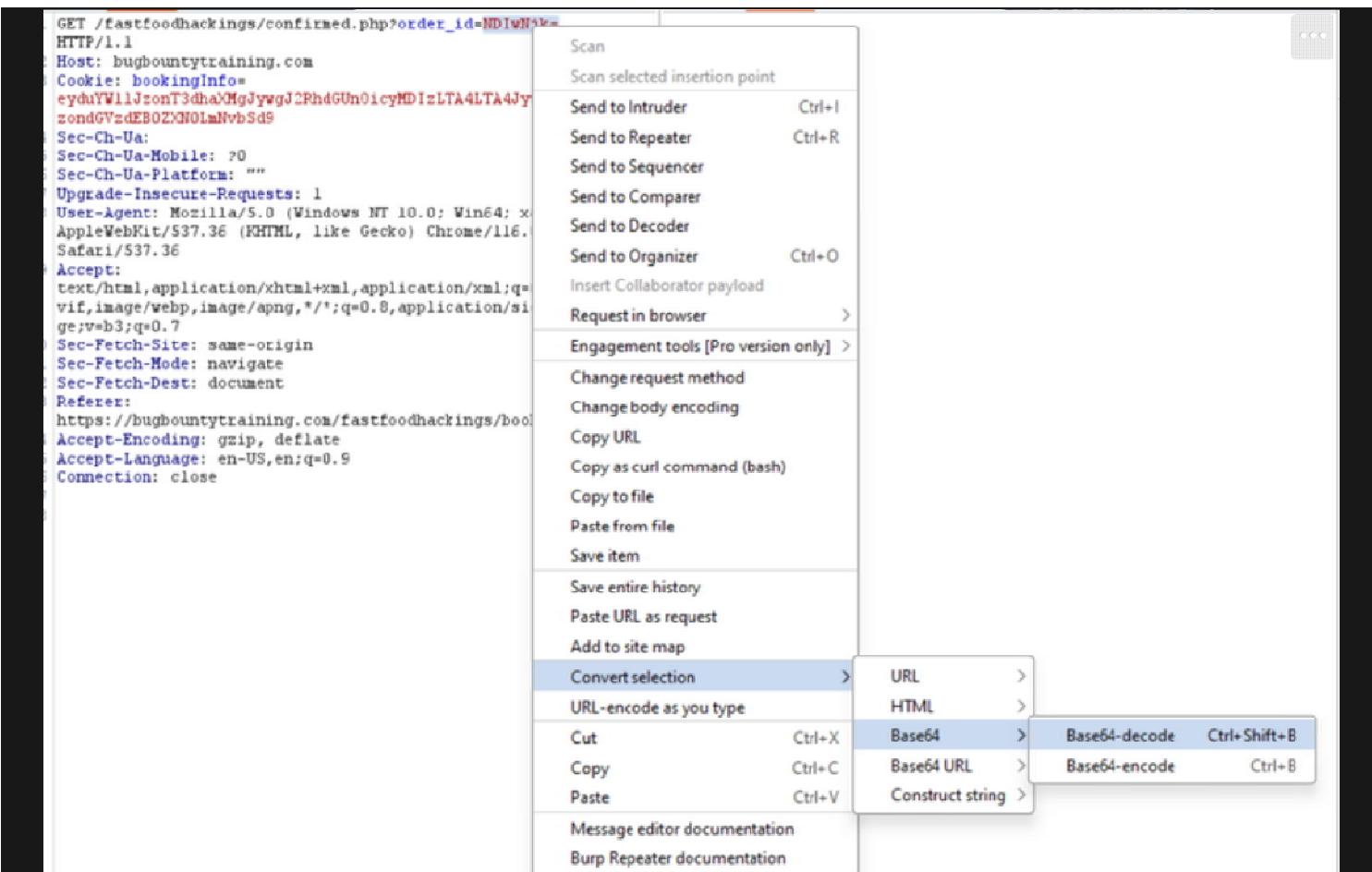
- Title: FastFoodHackings | Ethical Hack
- URL: <https://bugbountytraining.com/fastfoodhacking/confirmed.php>
- Content: Displays a "Your booking" page with fields for Full Name (Owais), Email Address (test@test.com), and Date (08-08-2023). A message says "This order is pending confirmation."
- Buttons: BROWSE OUR MENU, Famous Pizzas, Vegan Friendly.

**Burp Suite (Right):**

- Toolbar: Burp Suite Community Edition v2023.9.3 - Temporary Project, Dashboard, Target, Proxy (highlighted), Intruder, Repeater, Collaborator, Sequencer, Decoder, Composer, Settings.
- Sub-Menu: Intercept is on (highlighted).
- Message: "Intercept is on" with a blue shield icon.
- Description: "Requests sent by Burp's browser will be held here so that you can analyze and modify them before forwarding them to the target server." with "Learn more" and "Open browser" buttons.

And after the alert's redirect, the packets intercepted were as follows:

The `order_id` parameter seemed interesting, since it had a garbled value after it being sent thru a HTTP GET request. So I tested this in the repeater. Since the value ended in an equals sign, this hints that it may be a `base64` encoding. So I decoded it returned the order id of `#42069` which was the order I got before.



## Request

Pretty	<b>Raw</b>	Hex	≡	vn	☰
--------	------------	-----	---	----	---

```

1 GET /fastfoodhackings/confirmed.php?order_id=42069 HTTP/1.1
2 Host: bugbountytraining.com
3 Cookie: bookingInfo=
eyduYW11JzonT3dhaXGjywgJ2RhGU0icyMDIzLTA4LTA4JywgJ2VtYWlsJ
zondGVzdEB0ZXN0LmNvbSd9
4 Sec-Ch-Ua:

```

So I tested this to see if we can send the order IDs of people before me to be able to trick the website to show me their confirmation page. I entered 42068

The screenshot shows a web browser window with a form titled "Reserve a booking". The form includes fields for "Full Name" (Owais), "Email Address" (test@test.com), and "Date" (00-00-2023). Below the form is a section titled "BROWSE OUR MENU" featuring images of pizzas.

On the right, the Burp Suite interface is visible. It shows a "Request" pane with a raw HTTP message and a "Response" pane. A context menu is open over the request message, specifically over the "headers" section. The menu path "Convert selection > Base64" is highlighted. Other options in the menu include URL, HTML, Cut, Copy, Paste, and various encoding/decoding functions like Base64-decode and Base64-encode.

At the bottom of the browser window, a status bar says "Waiting for bugbountytraining.com".

On rendering the response I got from sending 42068 in the repeater, I got the following order page:

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'Request' pane displays a captured GET request to `/FastFoodHackings/confirmed.php?order_id=42068`. The 'Response' pane shows the resulting page from `https://bugbountytraining.com`, which is a booking confirmation page for Order ID #42068. The page includes fields for 'Full Name' (Deirdre Slocum), 'Email Address' (zhagenes@yahoo.com), and 'Date' (dd-mm-yyyy). A note at the bottom states: "This order is pending confirmation." Below this, there's a banner for "Famous Pizzas".

Since we found access to another person's order, it is considered to be broken access control and we have successfully exploited this website for the given CWE.

## 2. Cryptographic Failures

Cryptographic failures refer to vulnerabilities that arise from weak or improperly implemented cryptographic mechanisms. Cryptography is used to secure data transmission, authentication, and confidentiality. When cryptographic mechanisms are flawed, attackers can exploit these weaknesses to compromise sensitive information.

**Impact:**

Cryptographic failures can result in the exposure of sensitive data, the compromise of communication channels, and the undermining of authentication and integrity. Attackers may be able to decrypt encrypted data, impersonate users, or execute man-in-the-middle attacks.

**List of Mapped CWEs:**

CWE Code	Vulnerability Name
CWE-261	Weak Encoding for Password
CWE-296	Improper Following of a Certificate's Chain of Trust
CWE-310	Cryptographic Issues
CWE-319	Cleartext Transmission of Sensitive Information
CWE-321	Use of Hard-coded Cryptographic Key
CWE-322	Key Exchange without Entity Authentication
CWE-323	Reusing a Nonce, Key Pair in Encryption
CWE-324	Use of a Key Past its Expiration Date
CWE-325	Missing Required Cryptographic Step
CWE-326	Inadequate Encryption Strength
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-328	Reversible One-Way Hash
CWE-329	Not Using a Random IV with CBC Mode
CWE-330	Use of Insufficiently Random Values
CWE-331	Insufficient Entropy
CWE-335	Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)
CWE-336	Same Seed in Pseudo-Random Number Generator (PRNG)
CWE-337	Predictable Seed in Pseudo-Random Number Generator (PRNG)
CWE-338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
CWE-340	Generation of Predictable Numbers or Identifiers
CWE-347	Improper Verification of Cryptographic Signature
CWE-523	Unprotected Transport of Credentials
CWE-720	OWASP Top Ten 2007 Category A9 - Insecure Communications
CWE-757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')

CWE-759	Use of a One-Way Hash without a Salt
CWE-760	Use of a One-Way Hash with a Predictable Salt
CWE-780	Use of RSA Algorithm without OAEP
CWE-818	Insufficient Transport Layer Protection
CWE-916	Use of Password Hash With Insufficient Computational Effort

## **Example: CWE-319 (Cleartext Transmission of Sensitive Information)**

### **Description**

The product transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors. Many communication channels can be "sniffed" (monitored) by adversaries during data transmission. Adversaries might have privileged access to network interfaces or links, such as routers, enabling them to collect underlying data. This vulnerability can lead to security-critical data being exposed.

### **Business Impact**

Transmitting sensitive information in cleartext through communication channels that can be monitored by unauthorized actors can result in severe confidentiality breaches. Adversaries can gain access to the transmitted data, leading to unauthorized access, data leaks, identity theft, and other malicious activities. Additionally, when security-critical data is exposed, it can undermine user trust, violate compliance regulations, and cause reputational damage. The vulnerability significantly reduces the difficulty of exploitation for attackers, as they can easily intercept and misuse the transmitted sensitive information.

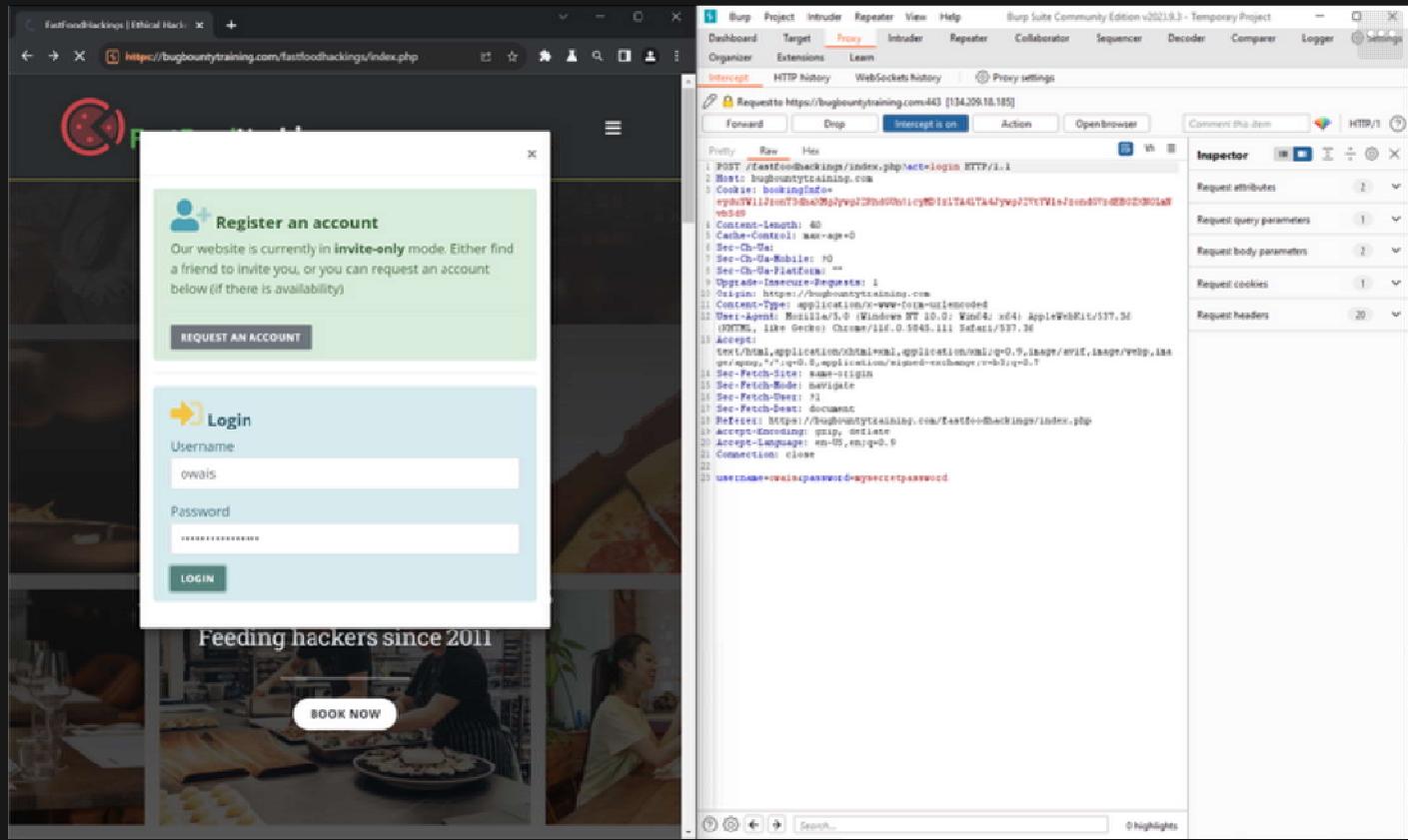
## **Overview of the Target**

I targetted the same website as before (\_FastFoodHackings) and as iterated previously, I am authorized to exploit it for the vulnerability since it is meant for bug bounty practice.

## **Methodology/Procedure to Exploit**

This was a relatively simpler vulnerability to exploit. I just had to intercept the `HTTP` packets carrying the information while logging in. Since the website does not show `HTTPS` in the URL bar, I know that the data won't be encrypted in transit.

And sure enough, once I used my name as username `omkar` and the password as something like `123PASS`, I was able to pick it up in the proxy in plain text.



You can even pick this packet up over something like wireshark which only picks up packets and has no encryption decryption features whatsoever.

### 3. Injection

Injection vulnerabilities occur when untrusted input is improperly sanitized and then executed as part of a command or query. This allows attackers to manipulate the behavior of an application by injecting malicious code or commands into it. Common types of injection attacks include SQL injection and cross-site scripting (XSS).

### **Impact:**

Injection attacks can lead to unauthorized data access, data manipulation, and remote code execution. Attackers can steal sensitive information, modify or delete data, and compromise the security of an application and its users.

### **List of Mapped CWEs:**

CWE Code	Vulnerability Name
CWE-20	Improper Input Validation
CWE-74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
CWE-75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-80	Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
CWE-83	Improper Neutralization of Script in Attributes in a Web Page
CWE-87	Improper Neutralization of Alternate XSS Syntax
CWE-88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
CWE-91	XML Injection (aka Blind XPath Injection)
CWE-93	Improper Neutralization of CRLF Sequences ('CRLF Injection')
CWE-94	Improper Control of Generation of Code ('Code Injection')
CWE-95	Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
CWE-96	Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
CWE-97	Improper Neutralization of Server-Side Includes (SSI) Within a Web Page

CWE-98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')
CWE-99	Improper Control of Resource Identifiers ('Resource Injection')
CWE-100	Deprecated: Was catch-all for input validation issues
CWE-113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
CWE-116	Improper Encoding or Escaping of Output
CWE-138	Improper Neutralization of Special Elements
CWE-184	Incomplete List of Disallowed Inputs
CWE-470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
CWE-471	Modification of Assumed-Immutable Data (MAID)
CWE-564	SQL Injection: Hibernate
CWE-610	Externally Controlled Reference to a Resource in Another Sphere
CWE-643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')
CWE-644	Improper Neutralization of HTTP Headers for Scripting Syntax
CWE-652	Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
CWE-917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

## CWE-80 (Improper Neutralization of Script-Related HTML Tags in a Web Page / Basic XSS)

### Description

The product receives input from an upstream component, but it does not neutralize or incorrectly neutralizes special characters such as "<", ">", and "&" that could be interpreted as web-scripting elements when they are sent to a downstream component that processes web pages. This vulnerability allows such characters to be treated as control characters, which are executed client-side in the context of the user's session. This vulnerability is commonly referred to as Cross-Site Scripting (XSS). Attackers exploit XSS vulnerabilities by injecting malicious scripts into web pages viewed by other users. When these scripts run in the user's browser, they can perform various malicious actions, including stealing sensitive data, manipulating user sessions, and executing unauthorized code. Properly neutralizing script-related HTML tags is crucial to prevent XSS attacks and protect user data and system integrity.

## **Business Impact**

The incorrect neutralization of script-related HTML tags in a web page can result in a significant security vulnerability known as Cross-Site Scripting (XSS). This vulnerability allows attackers to inject malicious scripts into web pages viewed by other users. When these malicious scripts execute in the user's browser, they can steal sensitive data, manipulate user sessions, deface websites, and perform other malicious actions on behalf of the attacker. The impact ranges from information disclosure and user impersonation to complete takeover of user accounts and unauthorized execution of arbitrary code. Proper neutralization of script-related tags is critical to prevent XSS attacks and safeguard user data and system integrity.

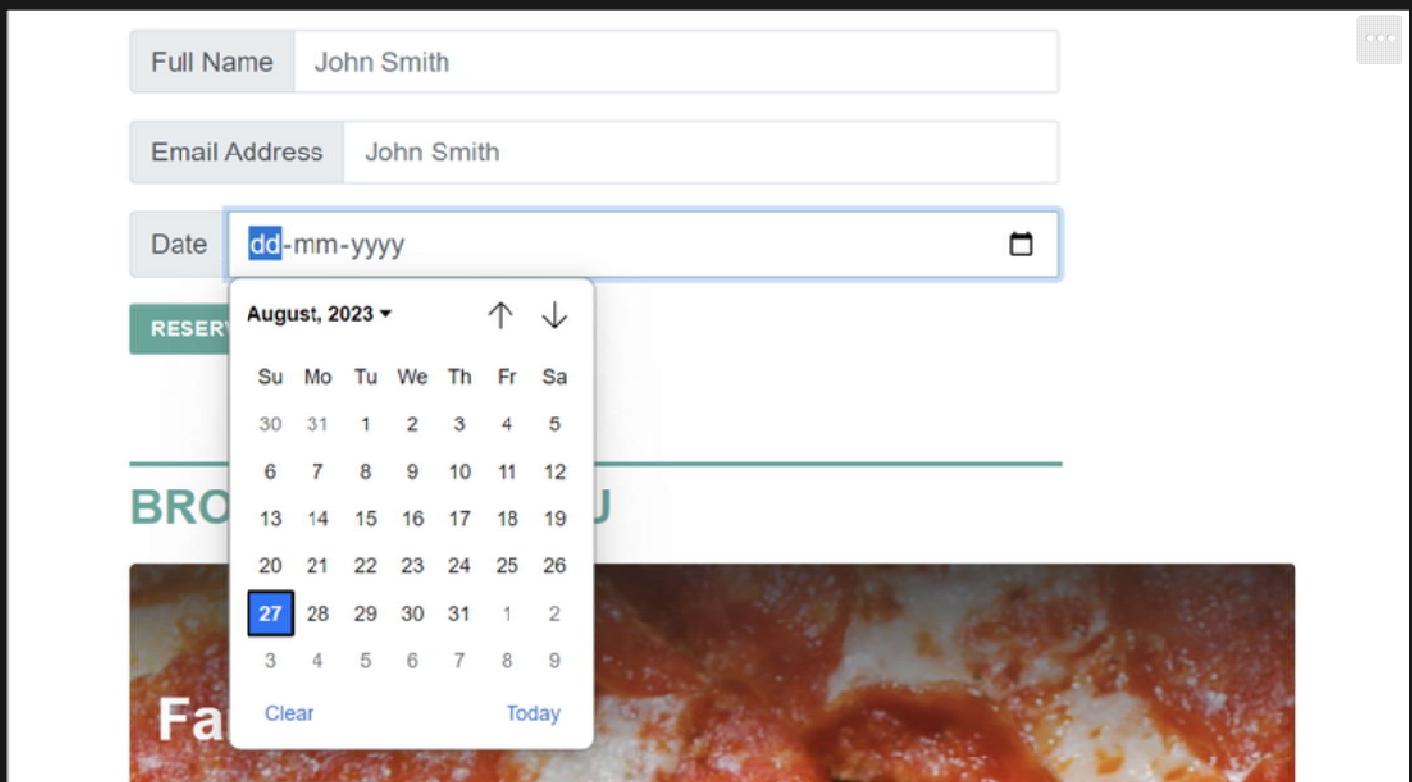
## **Overview of the Target**

Again, I targetted the same website as before (\_FastFoodHackings) and as iterated previously, I am authorized to exploit it for the vulnerability since it is meant for bug bounty practice.

## **Methodology/Procedure to Exploit:**

This vulnerability was also found in the bookings page that we have explored before. The XSS part however was found specifically in the date picker. This was because the design of the site was such that the developer had assumed that the HTML date picker would be enough validation to sanitize the input given from the user.

What this means is, since you cannot manually enter characters and the only way to enter information is thru the graphic interface of the date picker, the developer assumes that is enough data sanitization.



However we can intercept the request in between thru the proxy and then modify and sent this request to the server and play around with it.

On entering information normally, the rendered output looks like this in HTML:

The screenshot shows a browser window with the URL <https://bugbountytraining.com/fastfoodhacking/confirmed.php>. The page displays a booking confirmation for Order ID #42069, with fields for Full Name (John), Email Address (John), and Date (10-08-2023). A message indicates the order is pending confirmation. Below the browser is the Burp Suite interface, specifically the Proxy tab, showing the raw request sent to <https://maxcdn.bootstrapcdn.com> on port 443. The request includes headers such as User-Agent, Accept-Language, and Accept-Encoding.

Specifically, we notice that the date entered is being rendered in an HTML element like so:

The screenshot shows the browser's developer tools Elements tab. It highlights the HTML code for a date input field. The rendered HTML is as follows:

```

<input type="date" class="form-control" id="date" aria-label="Username" aria-describedby="basic-addon1" value="2023-08-10" locked="">

```

Whatever we enter shows up in this section:

HTML ▾

```
<input type="date" class="form-control" id="date" aria-lab  
el="Username" aria-describedby="basic-addon1" value="{our  
entered information / text}" locked="">
```

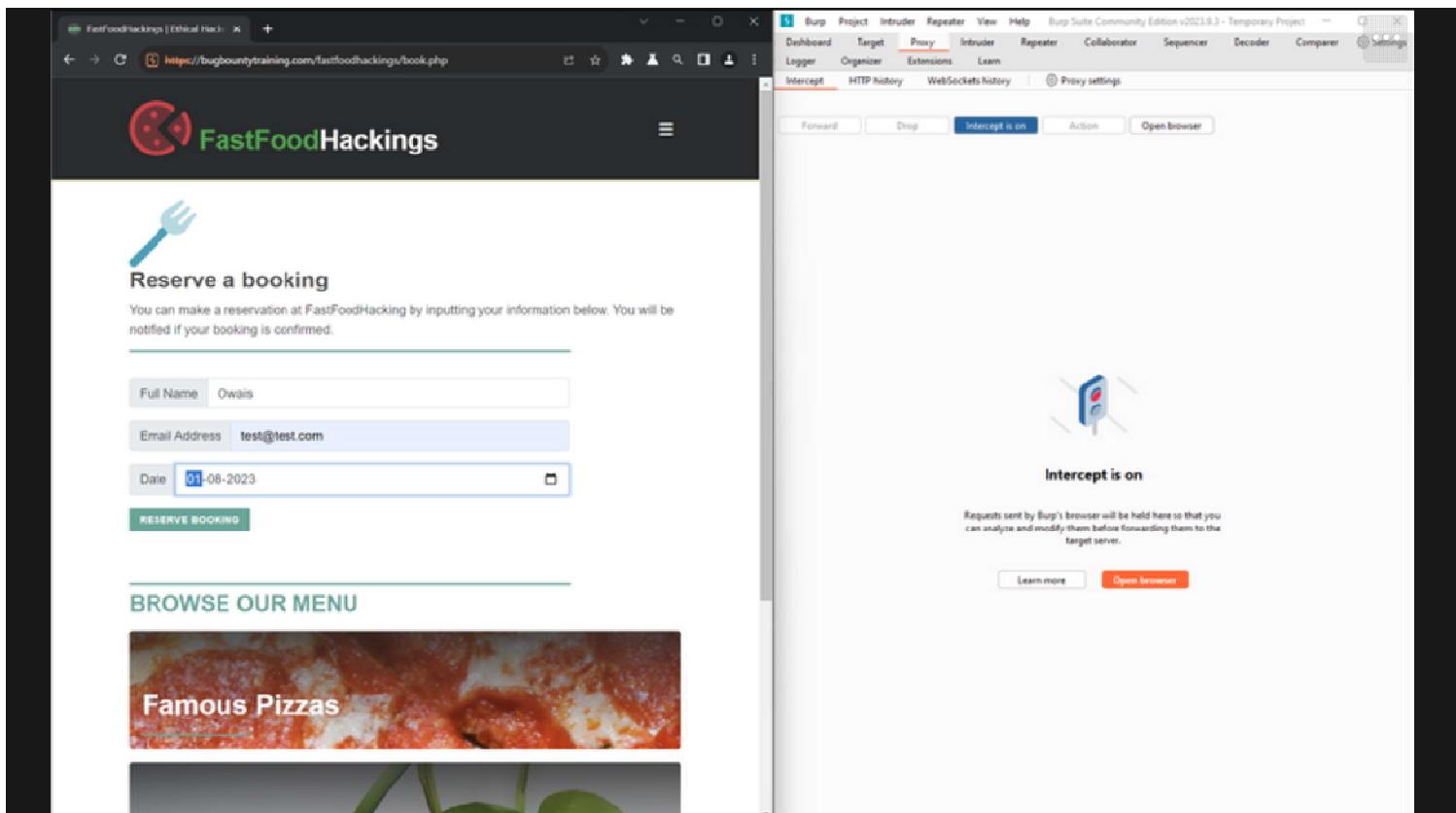
So to design an XSS attack, we can see if we can end the HTML tag here itself and then start a new script tag there with our own javascript code. I will be trying to trigger the `print()` function which usually sends the page to the printer. This should look something like:

HTML ▾

```
<input type="date" class="form-control" id="date" aria-lab  
el="Username" aria-describedby="basic-addon1" value=" "><s  
cript>print()</script>" locked="">
```

So I will try to intercept the packet in between and try to send the terms `"><script>print()</script>"`

To do this, I opened the same page again for another order booking, entered some random information and switched on the intercept.



The screenshot shows a browser window on the left and the Burp Suite interface on the right. In the browser, there's a form titled "Reserve a booking" with fields for Full Name (Owais), Email Address (test@test.com), and Date (01-08-2023). Below the form is a "RESERVE BOOKING" button. To the right, the Burp Suite interface has "Intercept" selected. A message says "Intercept is on" and a tooltip explains: "Requests sent by Burp's browser will be held here so that you can analyze and modify them before forwarding them to the target server." Buttons for "Learn more" and "Open browser" are also visible.

Then I tapped the `Reserve Booking` to catch the request in between.

The screenshot shows a browser window for 'FastFoodHackings' and a Burp Suite interface. The browser displays a booking form with fields for Full Name, Email Address, and Date. The Burp Suite proxy shows the raw HTTP request being sent to the target server. The request includes a POST method, a URL with a parameter 'battleofthebackends', and a body containing the user's input.

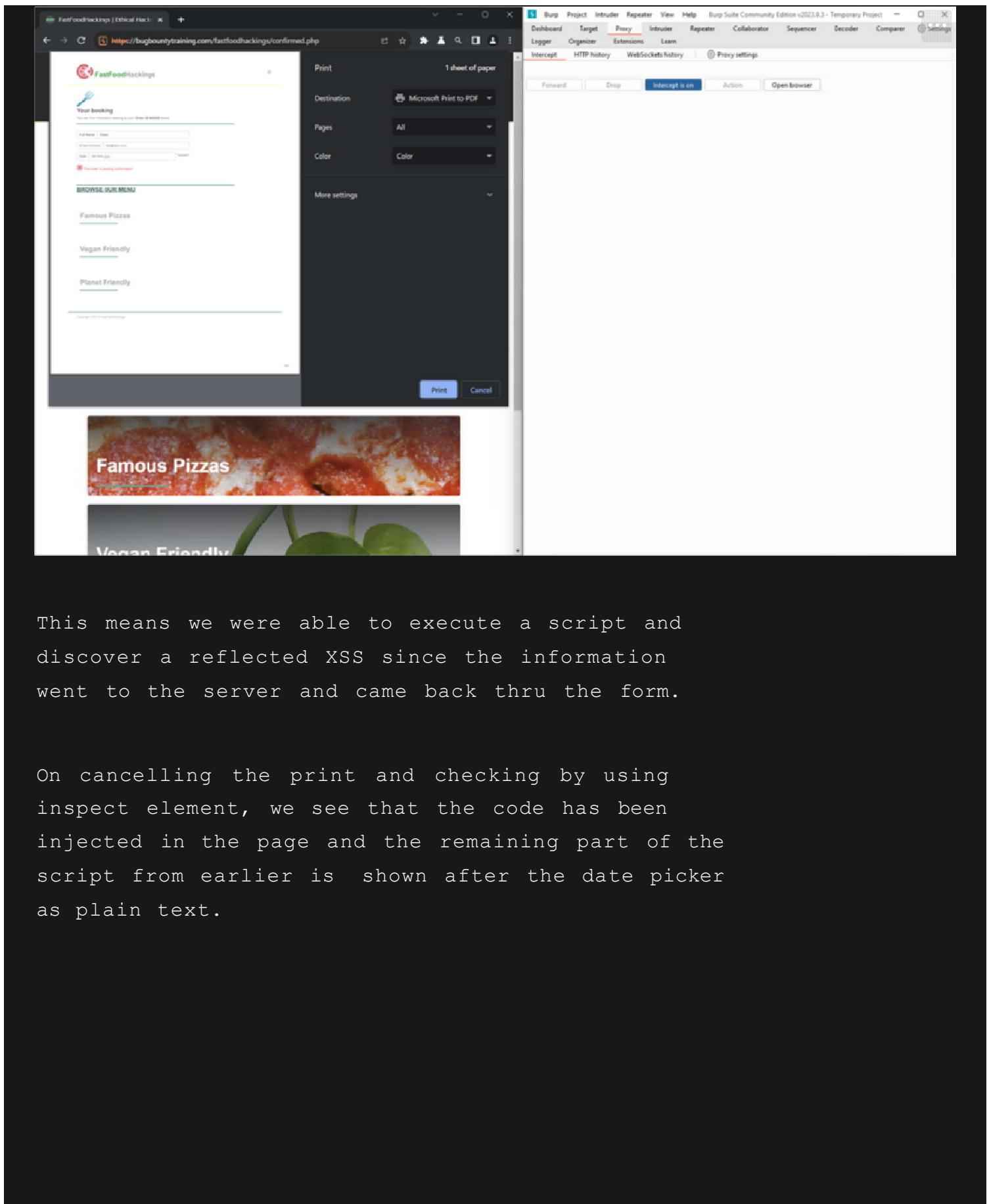
Here the request being sent (line 21) is:

```
HTML ▾
email=test@test.com&date=2023-08-01&userFN=omkar
```

I modified this to:

```
HTML ▾
email=test@test.com&date="><script>print()</script>&userFN
=omkar
```

And when I forwarded this packet, the print function triggered.



This means we were able to execute a script and discover a reflected XSS since the information went to the server and came back thru the form.

On cancelling the print and checking by using inspect element, we see that the code has been injected in the page and the remaining part of the script from earlier is shown after the date picker as plain text.

The screenshot shows a web browser window with a form titled "Your booking". The form includes fields for "Full Name" (Owais), "Email Address" (test@test.com), and "Date" (dd-mm-yyyy). A message below the date field states "This order is pending confirmation." To the right of the browser is the Burp Suite interface, specifically the "Proxy" tab. It displays a captured HTTP request to https://www.googleapis.com/443. The request details show various headers such as User-Agent, Host, and Accept-Language, along with the URL and method.

## 4. Insecure Design

Insecure design vulnerabilities system from the inadequate consideration of security during the software design phase. These vulnerabilities can manifest as poor architecture, lack of threat modeling, and failure to address potential attack vectors during the design and planning stages.

### Impact:

Insecure design can result in fundamental flaws that are difficult to mitigate after the software is developed. Such vulnerabilities may enable various attacks, compromise user privacy, and require substantial rework to address properly.

**List of Mapped CWEs:**

CWE Code	Vulnerability Name
CWE-73	External Control of File Name or Path
CWE-183	Permissive List of Allowed Inputs
CWE-209	Generation of Error Message Containing Sensitive Information
CWE-213	Exposure of Sensitive Information Due to Incompatible Policies
CWE-235	Improper Handling of Extra Parameters
CWE-256	Unprotected Storage of Credentials
CWE-257	Storing Passwords in a Recoverable Format
CWE-266	Incorrect Privilege Assignment
CWE-269	Improper Privilege Management
CWE-280	Improper Handling of Insufficient Permissions or Privileges
CWE-311	Missing Encryption of Sensitive Data
CWE-312	Cleartext Storage of Sensitive Information
CWE-313	Cleartext Storage in a File or on Disk
CWE-316	Cleartext Storage of Sensitive Information in Memory
CWE-419	Unprotected Primary Channel
CWE-430	Deployment of Wrong Handler
CWE-434	Unrestricted Upload of File with Dangerous Type
CWE-444	Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
CWE-451	User Interface (UI) Misrepresentation of Critical Information
CWE-472	External Control of Assumed-Immutable Web Parameter
CWE-501	Trust Boundary Violation
CWE-522	Insufficiently Protected Credentials
CWE-525	Use of Web Browser Cache Containing Sensitive Information
CWE-539	Use of Persistent Cookies Containing Sensitive Information

CWE-579	J2EE Bad Practices: Non-Serializable Object Stored in Session
CWE-598	Use of GET Request Method With Sensitive Query Strings
CWE-602	Client-Side Enforcement of Server-Side Security
CWE-642	External Control of Critical State Data
CWE-646	Reliance on File Name or Extension of Externally-Supplied File
CWE-650	Trusting HTTP Permission Methods on the Server Side
CWE-653	Insufficient Compartmentalization
CWE-656	Reliance on Security Through Obscurity
CWE-657	Violation of Secure Design Principles
CWE-799	Improper Control of Interaction Frequency
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-840	Business Logic Errors
CWE-841	Improper Enforcement of Behavioral Workflow
CWE-927	Use of Implicit Intent for Sensitive Communication
CWE-1021	Improper Restriction of Rendered UI Layers or Frames
CWE-1173	Improper Use of Validation Framework

### Example: CWE-840 (Business Logic Errors)

#### Description

Weaknesses in this category identify problems that often allow attackers to manipulate the business logic of an application. These errors can have a severe impact on the application's functionality. They are challenging to detect automatically as they often involve legitimate use of the application's features. Business logic errors may exhibit patterns similar to well-known implementation and design weaknesses.

## **Business Impact**

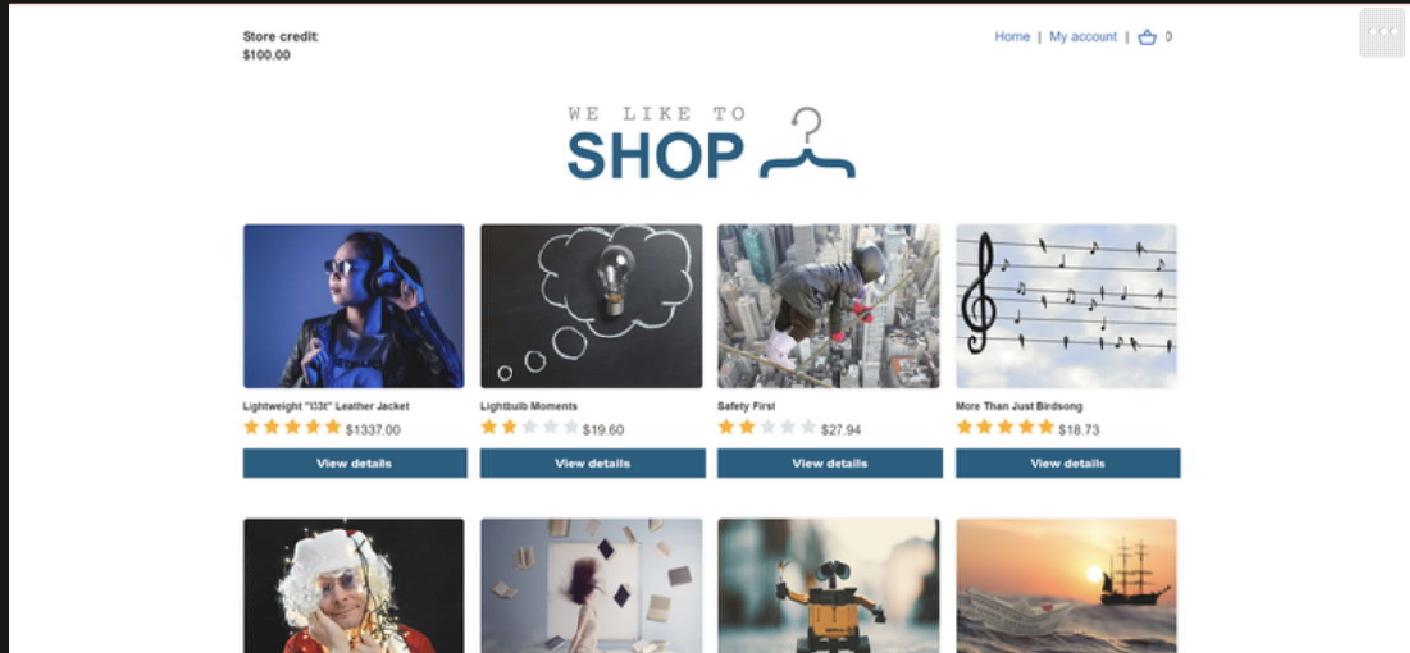
Business logic errors can lead to significant consequences. Attackers exploiting these errors could manipulate the application's intended behavior, allowing unauthorized access to sensitive data, unauthorized transactions, or unauthorized control over system functions. This can result in financial losses, data breaches, regulatory non-compliance, reputational damage, and legal liabilities. Such errors can be difficult to identify and remediate, making them a serious concern in application security.

## **Overview of the Target**

For this example, I picked up a lab from PortSwigger. This is the company that created BurpSuite so the target sites are very well crafted. And since this is also for Bug Bounty practice, I am authorized to pick this up for testing.

## Methodology/Procedure to Exploit:

The target site is an ecommerce application and on logging in, we have a store credit of \$100 .



Here there is a vulnerability in the add to cart feature that lets you buy items at a price you have set by yourself.

When you normally add an item to the cart:

Lighweight "I33t" Leather Jacket  
★★★★★  
\$1337.00

**Description:**  
Do you often feel as though people aren't aware of just how "I33t" you are? Do you find yourself struggling to make others feel inferior with public displays of your advanced "I33t-ness"? If either of these things are at the top of your priority list, it's time to the welcome Lightweight "I33t" Leather Jacket into your life.

Handcrafted from leather and single strands of recycled biomass, so you can enjoy environmental smugness on top of your high-ranking leather clad "I33t" levels, this jacket is far superior to anything currently available on the high street. Once you've explained to your friends and colleagues what "I33t" means, we guarantee you'll be at least 18% cooler when donning your "I33t" leather. Inspired by the term-cookers, the jacket comes with hand-stitched GISSP insignia so you can channel the original elite every time you rock your Lightweight "I33t" Leather Jacket.

Make your apparel as formidable as your intellect, and dazzle noobs the world over, with the Lightweight "I33t" Leather Jacket.  
"Every purchase comes with a free leaflet, detailing how best to explain the superiority of being "I33t" to noobs.

1

Add to cart

Forward Drop Intercept is off Action Open browser

Intercept is off

When enabled, requests sent by Burp's browser are held here so that you can analyse and modify them before forwarding them to the target server.

Learn more Open browser

And try to buy it, The transaction gets blocked and there's not enough credit for the purchase.

Store credit:  
\$100.00

Cart

Not enough store credit for this purchase

Name	Price	Quantity
Lightweight "I33t" Leather Jacket	\$1337.00	- 1 + Remove

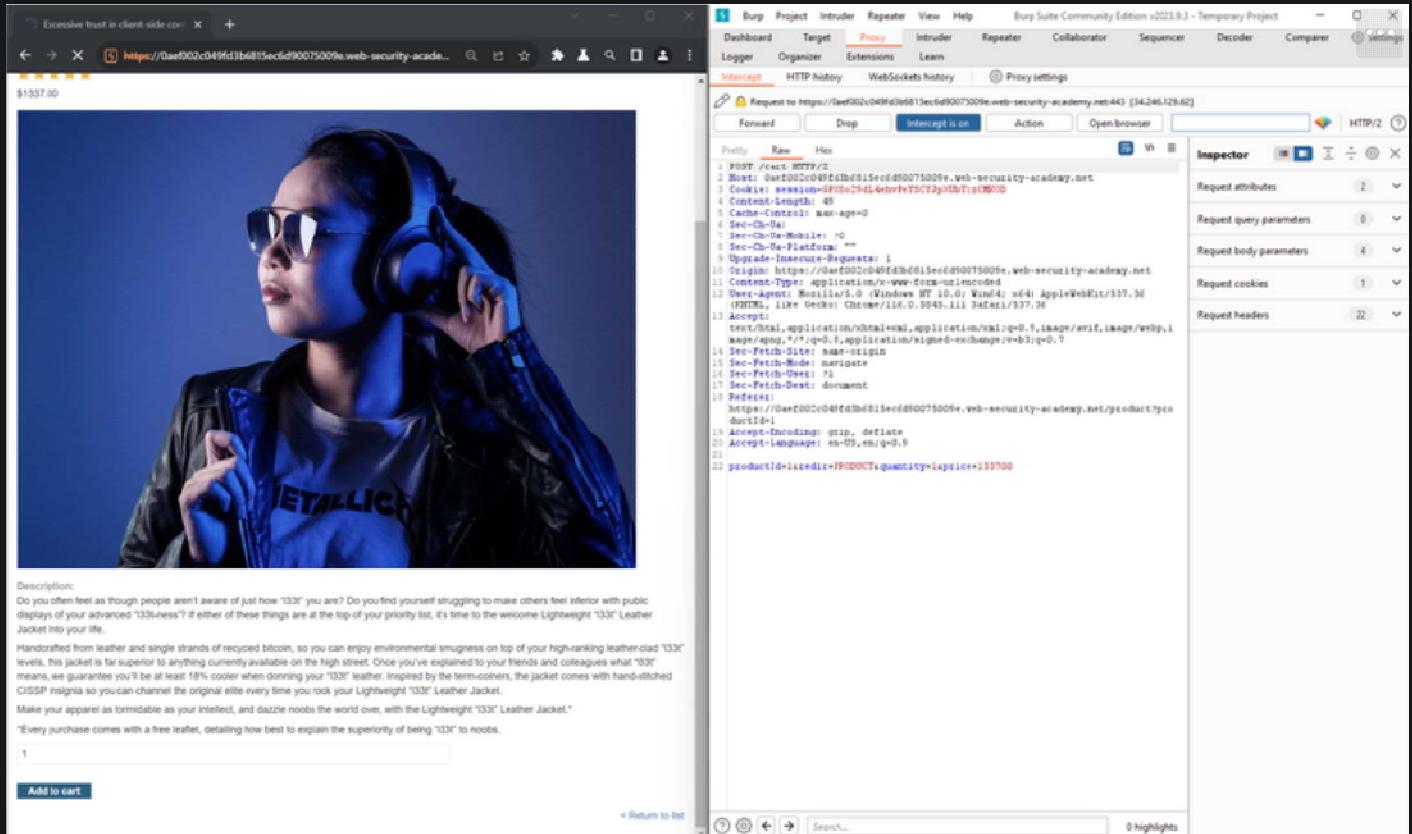
Coupon:  
Add coupon

Apply

Total: \$1337.00

Place order

So we try adding it again but this time we intercept the data while sending it over. And we see that the price is being sent over as a parameter.



The screenshot shows a browser window displaying a product page for a "Lightweight '03R" Leather Jacket. The page features a woman wearing headphones and a jacket with "METALLICA" printed on it. Below the image is a detailed product description. To the right of the browser is the Burp Suite interface, specifically the "Proxy" tab. A POST request is selected in the list. The "Raw" tab shows the modified HTTP request. The "productID" parameter has been changed from "1" to "123456789+PRODUCT+quantity=1+price=133700". Other parameters like "categoryID" and "size" remain at their original values. The "Inspector" tab on the right shows the request attributes, query parameters, body parameters, cookies, and headers.

```

1 POST /excessive-trust-in-client-side-code HTTP/2
2 Host: 12ae002c049fd3b4d15ec0075009e.web-security-academy.net
3 Cookie: session=4739c2941464fbf9fc739cb71cb090
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Not_A�;v=87"
6 Sec-Ch-Ua-Mobile: <none>
7 Sec-Ch-Ua-Platform: <none>
8 Upgrade-Insecure-Requests: 1
9 Origin: https://12ae002c049fd3b4d15ec0075009e.web-security-academy.net
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5443.111 Safari/537.36
12 Accept: */*
13 Accept-Encoding: gzip, deflate
14 Accept-Language: en-US,en;q=0.9
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-Dest: document
18 Referer: https://12ae002c049fd3b4d15ec0075009e.web-security-academy.net/product?categoryID=1
19 Accept-Charset: <none>
20 Accept-Content-Type: <none>
21
22 productID=123456789+PRODUCT+quantity=1+price=133700

```

So we modify it to an arbitrary integer just to see if it works and forward the request.

Description:  
Do you often feel as though people aren't aware of just how "33" you are? Do you find yourself struggling to make others feel inferior with public displays of your advanced "33"ness? If either of these things are at the top of your priority list, it's time to welcome Lightweight "33" Leather Jacket into your life.

Handcrafted from leather and single strands of recycled fibres, so you can enjoy environmental smugness on top of your high-ranking leather clad "33" levels, this jacket is far superior to anything currently available on the high street. Once you've explained to your friends and colleagues what "33" means, we guarantee you'll be at least 18% cooler when donning your "33" leather. Inspired by the hem-cookers, the jacket comes with hand-stitched GS&SP insignia so you can channel the original elite every time you rock your Lightweight "33" Leather Jacket.

Make your apparel as formidable as your intellect, and dazzle noobs the world over, with the Lightweight "33" Leather Jacket.

\*Every purchase comes with a free leaflet, detailing how best to explain the superiority of being "33" to noobs.

1

Add to cart

Request attributes: 1  
Request query parameters: 0  
Request body parameters: 4  
Request cookies: 1  
Request headers: 22

```

1 POST /443 HTTP/2
2 Host: 0aef002c049fd3fb485ec6d90075009e.web-security-academy.net
3 Cookie: session=d9f43bd815ec6d90075009e; Web-security-academy.NET
4 Content-Length: 49
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: 
7 Sec-Ch-Ua-Mobile: no
8 Sec-Ch-Ua-Platform: 
9 Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5845.113 Safari/537.36
11 Accept: 
12 Accept-Language: en-US,en;q=0.9
13 Accept-Encoding: gzip, deflate
14 Sec-Fetch-Dest: navigator
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Prefetch: 
19 https://0aef002c049fd3fb485ec6d90075009e.web-security-academy.net/product/prod1001
20 Accept-Charset: 
21 
22 product_id=1&size=L&PRODUCTquantity=laptop+1000
  
```

And now when we access the cart, we see it being added for our price instead. (\$10)

The screenshot shows a web-based shopping cart interface. At the top right, there are links for "Home" and "My account" and a shopping cart icon with a count of 1. On the left, it says "Store credit: \$100.00" and "Cart". Below that is a table with columns "Name", "Price", and "Quantity". A single item is listed: "Lightweight 'I33t' Leather Jacket" at "\$10.00" with a quantity of "1". There are buttons for "-", "+", and "Remove". Below the table is a "Coupon" input field with a placeholder "Add coupon" and an "Apply" button. At the bottom, it shows "Total: \$10.00" and a prominent "Place order" button.

And when we place the order, it sends back a confirmation, indicating that we have successfully exploited this application for that vulnerability.

The screenshot shows the same web-based shopping cart interface after an order has been placed. The store credit remains at \$100.00. The confirmation message "Your order is on its way!" is displayed above the cart table. The cart table now lists the item "Lightweight 'I33t' Leather Jacket" at a total price of "\$1337.00" with a quantity of "1". The total amount is still shown as "Total: \$10.00".

## 5. Security Misconfiguration

Security misconfiguration vulnerabilities arise when system components, frameworks, or applications are not properly configured to follow security best practices. This can include leaving default credentials, unnecessary services enabled, and exposed sensitive information.

### **Impact:**

Security misconfiguration can lead to unauthorized access, data exposure, and other security breaches. Attackers can exploit misconfigured settings to gain control over systems, extract sensitive data, and disrupt services.

### **List of Mapped CWEs:**

CWE Code	Vulnerability Name
CWE-2	7PK - Environment
CWE-11	ASP.NET Misconfiguration: Creating Debug Binary
CWE-13	ASP.NET Misconfiguration: Password in Configuration File
CWE-15	External Control of System or Configuration Setting
CWE-16	Configuration
CWE-260	Password in Configuration File
CWE-315	Cleartext Storage of Sensitive Information in a Cookie
CWE-520	.NET Misconfiguration: Use of Impersonation
CWE-526	Exposure of Sensitive Information Through Environmental Variables
CWE-537	Java Runtime Error Message Containing Sensitive Information
CWE-541	Inclusion of Sensitive Information in an Include File
CWE-547	Use of Hard-coded, Security-relevant Constants
CWE-611	Improper Restriction of XML External Entity Reference
CWE-614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
CWE-756	Missing Custom Error Page
CWE-776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')
CWE-942	Permissive Cross-domain Policy with Untrusted Domains
CWE-1004	Sensitive Cookie Without 'HttpOnly' Flag
CWE-1032	OWASP Top Ten 2017 Category A6 - Security Misconfiguration
CWE-1174	ASP.NET Misconfiguration: Improper Model Validation

## CWE-537: Java Runtime Error Message Containing Sensitive Information

### Description

In many cases, attackers can exploit unhandled exception errors in java to gain unauthorized access to the system by leveraging the conditions that cause these errors.

### **Business Impact**

The exposure of sensitive error information in runtime error messages can have a confidentiality impact. Attackers may gain insights into the internal workings of the application, its file system structure, or other sensitive information contained within the error messages. This information could be used to formulate targeted attacks, further exploiting vulnerabilities in the application.

### **Overview of the Target**

The target website is an ecommerce website with different product listings. It is a website meant for bug bounty practice so I am authorized to hack into the website and/or to try to exploit common vulnerabilities.

### **Methodology/Procedure to Exploit:**

This vulnerability was relatively simple to exploit. All of the different products listed on the site had URLs with different product IDs being passed as parameter in `HTTP POST` method.

For example the first product 1 had the URL as:

HTML ▾

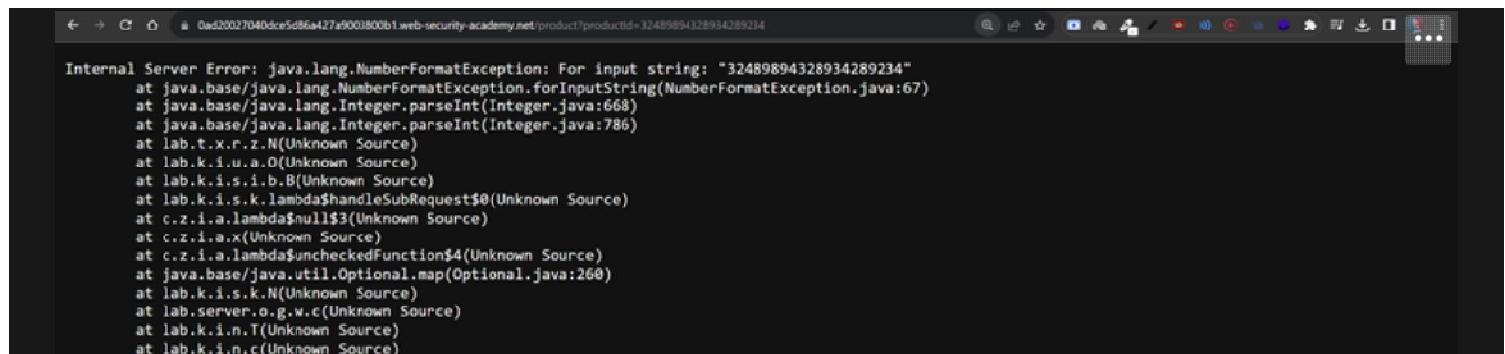
```
https://0ad20027040dce5d86a427a90038001.web-security-acade
my.net/product?productId=1
```

The 5th had a URL as:

HTML ▾

```
https://0ad20027040dce5d86a427a90038001.web-security-acade
my.net/product?productId=5
```

So I tried putting it as an arbitrarily huge number like 9328357248 since the possibility of those many entries existing on the database was slim. And instead of throwing a custom error message, I directly got an error response from the java runtime.



The screenshot shows a browser window with the URL <http://0ad2022704dce5d86a427a9003800b1.web-security-academy.net/product?productid=32489894328934289234>. The page displays an Internal Server Error message with the following Java stack trace:

```
Internal Server Error: java.lang.NumberFormatException: For input string: "32489894328934289234"
at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
at java.base/java.lang.Integer.parseInt(Integer.java:668)
at java.base/java.lang.Integer.parseInt(Integer.java:786)
at lab.t.x.r.z.N(Unknown Source)
at lab.k.i.u.a.O(Unknown Source)
at lab.k.i.s.i.b.B(Unknown Source)
at lab.k.i.s.k.lambda$handleSubRequest$0(Unknown Source)
at c.z.i.a.lambda$null$3(Unknown Source)
at c.z.i.a.X(Unknown Source)
at c.z.i.a.lambda$uncheckedFunction$4(Unknown Source)
at java.base/java.util.Optional.map(Optional.java:268)
at lab.k.i.s.k.N(Unknown Source)
at lab.server.o.g.w.c(Unknown Source)
at lab.k.i.n.T(Unknown Source)
at lab.k.i.n.c(Unknown Source)
```