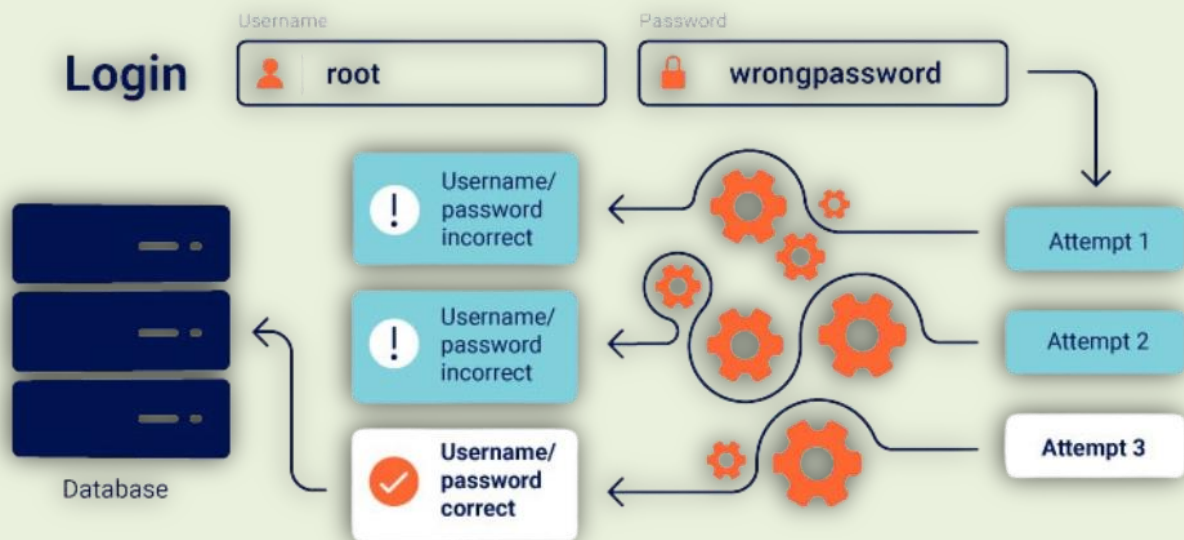


Business Logic Vulnerabilities

RESEARCH
REPORT



GUIDE : Mr KULDEEP L M



JAYASHANKAR P - SPYDER 9

RED TEAM INTERN @ CYBERSAPIENS

28th November, 2024

INTRODUCTION TO BUSINESS LOGIC VULNERABILITIES

Business logic vulnerabilities arise when an attacker manipulates the business rules or processes to their advantage, leading to potential losses or damages. The most significant characteristic of business logic Vulnerabilities is that they cannot be detected by traditional firewalls or security scanners. This is because unlike common vulnerabilities, such as SQL injection or Cross-Site Scripting (XSS), business logic vulnerabilities do not exploit technical flaws. Instead, they exploit the legitimate functionality of the application or system.

In simpler terms: Business logic vulnerabilities are weaknesses in how a system or application works that can be exploited to get unauthorized benefits or cause harm.

Let's understand this in simple example;

Imagine you have a vending machine. It has a specific way to work: you insert money, choose a product, and it dispenses the product.

Business logic vulnerabilities are like finding a secret way to get free candy from that machine. It's not about breaking the machine itself, but exploiting the rules it follows to get unintended results.

Here are some examples:

- **Price manipulation:** Finding a way to change the price of a product to a lower value.
- **Inventory exploitation:** Exploiting a loophole in the inventory system to get more products than you paid for.
- **Timing attacks:** Taking advantage of how the machine processes requests to get extra products or bypass limits.
- **Session hijacking:** Stealing someone else's "ticket" to use the machine and get their products.

These vulnerabilities happen when the rules of the machine (the business logic) aren't designed carefully enough to prevent these kinds of tricks.



EXAMPLES OF BUSINESS LOGIC VULNERABILITIES

Business logic vulnerabilities are flaws in the design and implementation of an application's business rules and processes. These vulnerabilities can be exploited by attackers to bypass security controls, manipulate data, or gain unauthorized access to resources.

Here are some common examples of business logic vulnerabilities:

1. Excessive Trust in Client-Side Controls: (Demo : vulnerability check : page-10)

Issue: Relying solely on client-side validation (e.g., JavaScript) to enforce security rules.

Example: A web application that validates input using JavaScript can be easily bypassed by manipulating the client-side code.

2. Failing to Handle Unconventional Input: (Demo : vulnerability check : page-11)

Issue: Not considering unexpected or malicious input formats.

Example: A web application that expects a specific input format might be vulnerable to attacks that exploit unusual input, such as long strings, special characters, or code injection.

3. Making Flawed Assumptions about User Behavior: (Demo : vulnerability check : page-12)

Issue: Assuming that users will always follow the intended workflow.

Example: A web application that assumes users will only access certain pages in a specific order might be vulnerable to attacks that bypass the intended sequence.

4. Domain-Specific Flaws: (Demo : vulnerability check : page-13)

Issue: Vulnerabilities specific to a particular industry or application domain.

Example: In e-commerce, a flaw in the pricing or discount calculation logic could allow attackers to purchase items at significantly reduced prices or even for free.

5. Providing an Encryption Oracle:

Issue: Leaking information about the encryption process, which can aid in decryption attacks.

Example: A web application that reveals partial information about encrypted data, such as its length or hash value, can be exploited to crack the encryption.

6. Email Address Parser Discrepancies:

Issue: Inconsistent handling of email addresses, leading to potential vulnerabilities.

Example: A web application that allows users to register with email addresses might be vulnerable to attacks that exploit email address validation flaws.

TEST CASES

Business logic vulnerabilities are often overlooked but can have significant security implications. To effectively test for these vulnerabilities, consider the following test cases:

1. ATO via Google SSO Sign-Up function
2. Parameter Tampering
3. Response Manipulation
4. Critical Parameter Manipulation and Access to Unauthorized Information/Content (IDOR)

1. ACCOUNT TAKE OVER (ATO) via GOOGLE SSO SIGN-UP

Account Takeover (ATO) is a threat where malicious actors gain unauthorized access to a user's account. In the context of Google's Single Sign-On (SSO) sign-up process, ATO can occur through various methods, often exploiting vulnerabilities in the system or user behavior.



Here's a breakdown of the process and potential vulnerabilities:

User Initiates Sign-Up	Google Authentication	Redirection and Authorization
User clicks on the "Sign Up with Google" button on a third-party website or application. This redirects the user to Google's authentication page.	User enters their Google credentials (email and password). Google verifies the credentials and grants access.	Google redirects the user back to the original website or application, along with a token or authorization code. This token allows the website or application to access certain user information from Google.

Potential ATO Vectors:

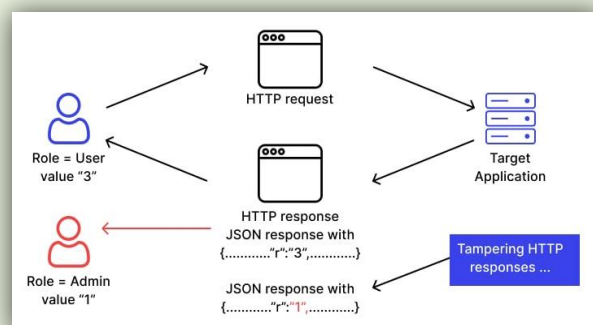
Weak Password Security Users choosing weak or easily guessable passwords can make their accounts vulnerable to brute-force attacks. Phishing attacks can trick users into revealing their credentials.	Phishing Attacks Malicious actors can send phishing emails or messages that mimic legitimate Google communications. These attacks can trick users into entering their credentials on fake login pages.
Session Hijacking Attackers can steal a user's valid session token, allowing them to access the user's account without their knowledge. This can be achieved through various techniques, such as exploiting vulnerabilities in web applications or network infrastructure	Man-in-the-Middle Attacks Attackers can intercept the communication between the user and Google's servers, capturing sensitive information like passwords and tokens. This can be done by setting up malicious Wi-Fi hotspots or exploiting vulnerabilities in network devices.
Social Engineering Attacks Attackers can manipulate users into revealing sensitive information through social engineering tactics. This can involve building trust with the user and then asking for personal information or access to their accounts	

Mitigation Strategies for ATO attacks :

1. Strong Password Practices: Encourage users to create strong, unique passwords and enable two-factor authentication (2FA).
2. Security Awareness Training: Educate users about phishing attacks, social engineering, and other common threats.
3. Regular Security Audits: Conduct regular security audits to identify and address vulnerabilities in the application and infrastructure.
4. Robust Security Measures: Implement strong security measures, such as encryption, secure coding practices, and regular security updates.
5. User Monitoring and Alerting: Monitor user activity for unusual patterns and suspicious behavior.
6. Incident Response Plan: Have a well-defined incident response plan to quickly detect and respond to security breaches.

2. PARAMETER TAMPERING

Parameter tampering is a type of web application vulnerability where an attacker manipulates parameters passed between a client and a server to gain unauthorized access or modify data. This can lead to a variety of security risks, including data breaches, financial loss, and reputational damage.



How Does Parameter Tampering Work?

Parameter Identification	Manipulation
Attackers identify parameters within a web application's requests, such as those found in URLs, form fields, or cookies.	They alter these parameters to bypass security controls, exploit vulnerabilities, or gain unauthorized access.
Exploitation	
The manipulated parameters are sent to the server, which may process them incorrectly, leading to unintended consequences.	

Common Techniques Used in Parameter Tampering:

Modifying Input Fields	Manipulating URL Parameters
Altering values in form fields to trigger unexpected behavior or bypass validation rules.	Changing values in the query string of a URL to access restricted resources or execute malicious code.
Modifying Cookies	Injecting Malicious Code
Tampering with cookies to change user session information or bypass authentication checks	Injecting malicious code, such as JavaScript, into input fields to execute arbitrary code on the server-side.

Real-World Examples of Parameter Tampering:

<p>Price Manipulation</p> <p>Attackers can modify product prices in shopping cart applications to purchase items at significantly reduced prices.</p>	<p>Unauthorized Access</p> <p>By altering user roles or permissions, attackers can gain access to sensitive data or functionalities.</p>
<p>Data Corruption</p> <p>Manipulating critical data can lead to system failures, data loss, or incorrect calculations.</p>	<p>Cross-Site Scripting (XSS)</p> <p>Injecting malicious scripts into web pages to steal user information or compromise the website.</p>

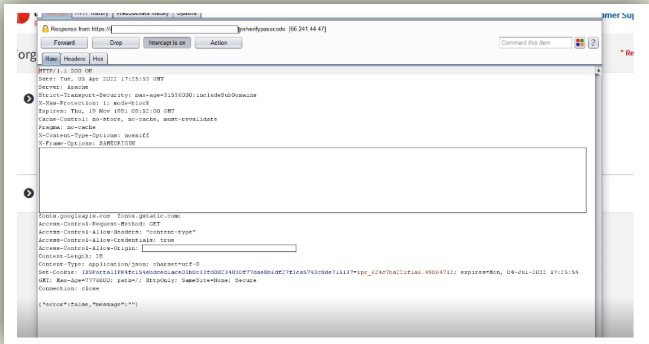
Mitigation Strategies for Parameter Tampering Attacks:

- 1. **Input Validation and Sanitization:** Validate and sanitize all user input to prevent malicious data from being processed.
- 2. **Secure Coding Practices:** Follow secure coding guidelines to avoid common vulnerabilities like SQL injection and cross-site scripting.
- 3. **Robust Authentication and Authorization:** Implement strong authentication mechanisms and authorization controls to protect access to sensitive resources.
- 4. **Web Application Firewalls (WAFs):** Deploy WAFs to filter and block malicious requests.
- 5. **Regular Security Testing:** Conduct regular penetration testing and vulnerability assessments to identify and address weaknesses.
- 6. **Security Awareness Training:** Educate users about the risks of clicking on malicious links and downloading suspicious attachments.
- 7. **Monitoring and Logging:** Monitor system logs for unusual activity and suspicious requests.

3. RESPONSE MANIPULATION

Response manipulation is a type of application security vulnerability that occurs when an attacker modifies the server’s response to a client request before reaching a browser or an Application. This is mostly carried out by modifying data using a proxy interceptor.

Response manipulation exploits the trust relationship between the client and the server. By manipulating server responses, an attacker can deceive the application client into believing that the server has authorized certain actions or provided specific data. This deception can lead to significant security breaches.



How Does Response Manipulation Work?

<p>Intercepting the Response</p> <p>The attacker intercepts the response from the server, usually by using a proxy tool like Burp Suite or OWASP ZAP.</p>	<p>Modifying the Response</p> <p>The attacker modifies the response, such as changing the content, removing security headers, or altering authentication tokens.</p>
<p>Forwarding the Manipulated Response</p> <p>The attacker sends the modified response to the client's browser or application.</p>	

Common Techniques Used in Response Manipulation:

Modifying HTTP Headers Altering headers like Set-Cookie or Content-Security-Policy to bypass security controls.	Tampering with HTML Content Changing the HTML structure or content to display malicious content or redirect users to malicious websites.
Manipulating JSON or XML Responses Altering data within JSON or XML responses to gain unauthorized access or modify application behavior.	Session Hijacking Stealing or modifying session cookies to impersonate legitimate users.

Real-World Examples of Response Manipulation:

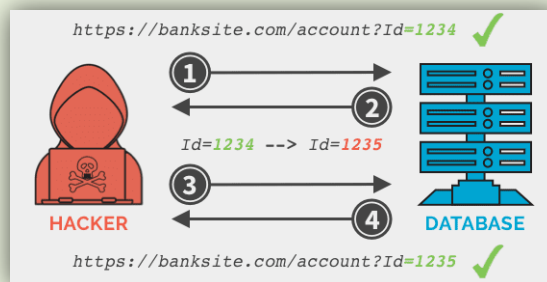
Price Manipulation Attackers can modify product prices on e-commerce websites to purchase items at discounted rates.	Unauthorized Access Attackers can modify authentication tokens or session cookies to gain access to restricted areas of a website.
Data Theft Attackers can steal sensitive information, such as credit card numbers or personal data, by modifying the response to a request.	Cross-Site Scripting (XSS) Injecting malicious scripts into web pages to steal user information or compromise the website.

Prevention and Mitigation Techniques:

1. **Secure Coding Practices:** Follow secure coding guidelines to prevent vulnerabilities that can be exploited for response manipulation.
2. **Input Validation and Sanitization:** Validate and sanitize all user input to prevent malicious data from being processed.
3. **Robust Authentication and Authorization:** Implement strong authentication mechanisms and authorization controls to protect access to sensitive resources.
4. **Web Application Firewalls (WAFs):** Deploy WAFs to filter and block malicious requests.
5. **Secure HTTP Headers:** Configure security headers like Content-Security-Policy, X-Frame-Options, and Strict-Transport-Security to mitigate risks.
6. **Regular Security Testing:** Conduct regular penetration testing and vulnerability assessments to identify and address weaknesses.
7. **Security Awareness Training:** Educate users about the risks of clicking on malicious links and downloading suspicious attachments.
8. **Monitoring and Logging:** Monitor system logs for unusual activity and suspicious requests.

4. IDOR – INSECURE DIRECT OBJECT REFERENCE OR CRITICAL PARAMETER MANIPULATION AND ACCESS TO UNAUTHORIZED INFORMATION / CONTENT

An Insecure Direct Object Reference (IDOR) is a web application vulnerability that arises when a web application exposes internal object identifiers, such as database keys or file paths, to users without proper access controls. 1 This can allow attackers to manipulate these identifiers and gain unauthorized access to sensitive data or perform unauthorized actions on the system



How IDOR works?

Identifier Exposure	Manipulation
The web application exposes object identifiers in URLs, form fields, or other parameters.	An attacker can modify these identifiers to access resources they are not authorized to view.
Unauthorized Access	
If the application's access controls are insufficient, the attacker can gain access to sensitive data or perform privileged actions.	

Example for IDOR Vulnerability

Consider a web application that displays user profiles. The URL to view a user's profile might look like this:

`https://spyder9.com/profile?user_id=123`

If the application doesn't properly validate the `user_id` parameter, an attacker could modify it to view other users' profiles, potentially accessing sensitive information.

Mitigating IDOR Vulnerabilities

- Input Validation and Sanitization:**
 - ✓ Validate and sanitize all user-supplied input, including parameters in URLs and form fields.
 - ✓ Ensure that only valid and authorized values are accepted.
- Robust Access Controls:**
 - ✓ Implement strong access control mechanisms to restrict access to resources based on user roles and permissions.
 - ✓ Use appropriate authorization techniques, such as role-based access control (RBAC) or attribute-based access control (ABAC).
- Avoid Exposing Sensitive Information in URLs:**
 - ✓ Avoid exposing sensitive information, such as user IDs or database keys, directly in URLs.
 - ✓ Use opaque identifiers or session tokens to reference objects.
- Secure Session Management:**
 - ✓ Implement secure session management practices to prevent session hijacking & unauthorized access.
 - ✓ Use strong session IDs and secure cookie settings.
- Regular Security Testing:**
 - ✓ Conduct regular penetration testing & vulnerability assessments to identify IDOR vulnerabilities.
 - ✓ Use automated tools and manual techniques to find and exploit potential weaknesses.

COMMON END-POINTS

- **Auth Mechanisms** : Login mechanisms, role-based access controls.
- **Transaction Processing** : Order placements, payment processing.
- **User Privileges** : Profile settings, account management.
- **Workflow Processes** : Order fulfillment, document approvals.
- **Inadequate Access Controls** : Unauthorized access to functionalities or data.
- **Inconsistent Validation** : Validation errors leading to unintended outcomes.
- **Session Management** : Incorrect handling of user sessions
- **Race Conditions** : Arising from concurrent execution of processes.

IMPACT OF BUSINESS LOGIC VULNERABILITIES

The business impact of business logic vulnerability can be significant, as it can potentially lead to financial losses, data breaches, and damage to an organization's reputation and customer trust.

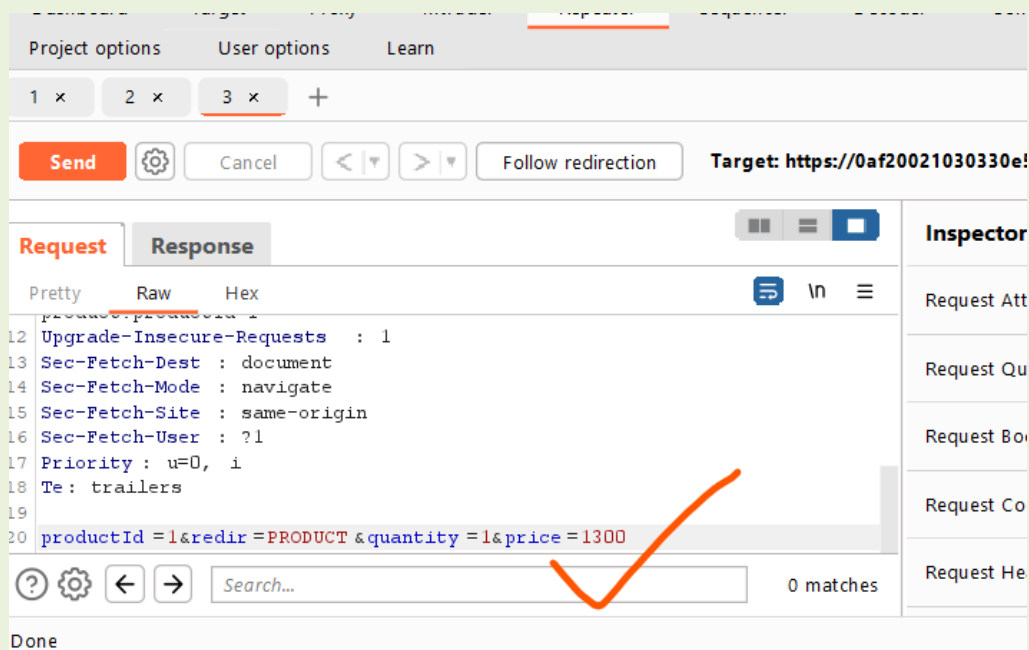
1. **Financial Loss:** Attackers may be able to conduct fraudulent transactions, unauthorized purchases, or gain access to financial accounts/data.
2. **Data Breaches:** Business logic flaws can allow unauthorized access to sensitive data, such as personal information, financial records, or intellectual property.
3. **Reputational Damage:** A publicized data breach or exploitation of a logic flaw can severely damage an organization's reputation and credibility.
4. **Competitive Disadvantage:** Intellectual property or trade secrets could be compromised, providing competitors with an unfair advantage.
5. **Operational Disruptions:** Attackers may be able to disrupt business operations, e.g., by depleting resources or overwhelming systems through exploitation of logic flaws.
6. **Non-Compliance Penalties:** Depending on the nature of the vulnerability, it may result in non-compliance with industry regulations or standards, especially in the financial services industry, leading to penalties or loss of certifications.

TESTING PRACTICALS - DEMO

Demo 1: EXCESSIVE TRUST IN CLIENT-SIDE CONTROLS

Lab Goal : To exploit a logic flaw in its purchasing workflow to buy items for an unintended price. Have to purchase product of more value, even though the wallet credit limit is too less.

Solution: By changing the amount through burp repeater we can change the product price and purchase it.



Excessive trust in client-side controls

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#) [Continue learning >>](#)

Store credit:
\$87.00

[Home](#) | [My account](#) | [0](#)

Your order is on its way!

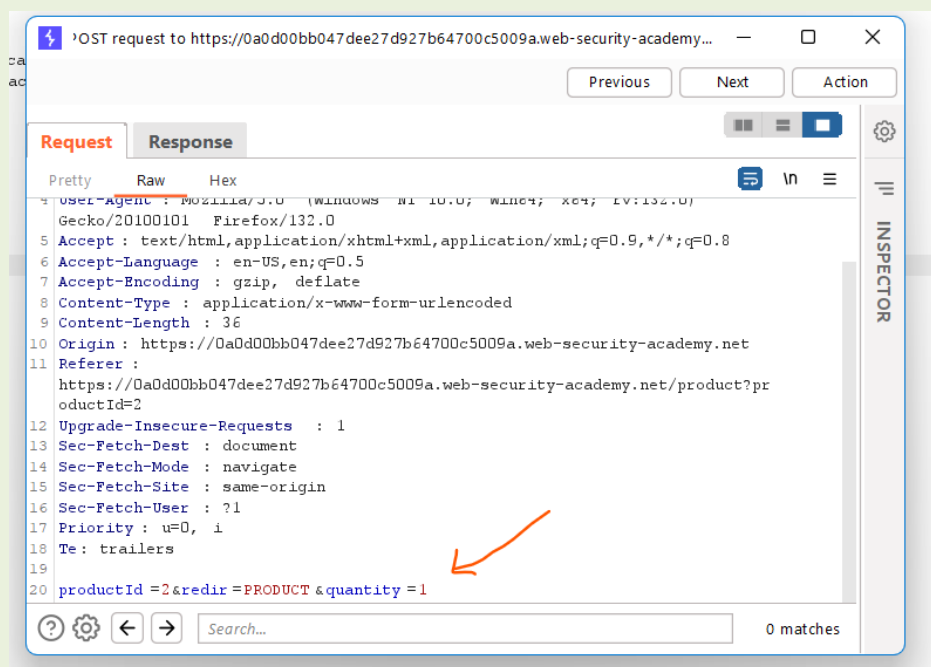
Name	Price	Quantity
Lightweight "I33t" Leather Jacket	\$1337.00	1

Total: \$13.00

Demo 2: HIGH – LEVEL LOGIC VULNERABILITIES

Lab Goal: To exploit a logic flaw in its purchasing workflow to buy items for an unintended price. Have to purchase product of more value, even though the wallet credit limit is too less.

Solution: (POST Request) - By changing the product quantity in negative value, we can reduce the purchase value



Web Security Academy

High-level logic vulnerability

LAB Solved

Back to lab description >>

Congratulations, you solved the lab!

Share your skills! Continue learning >>

Store credit:
\$10.46

Home | My account | 0

Your order is on its way!

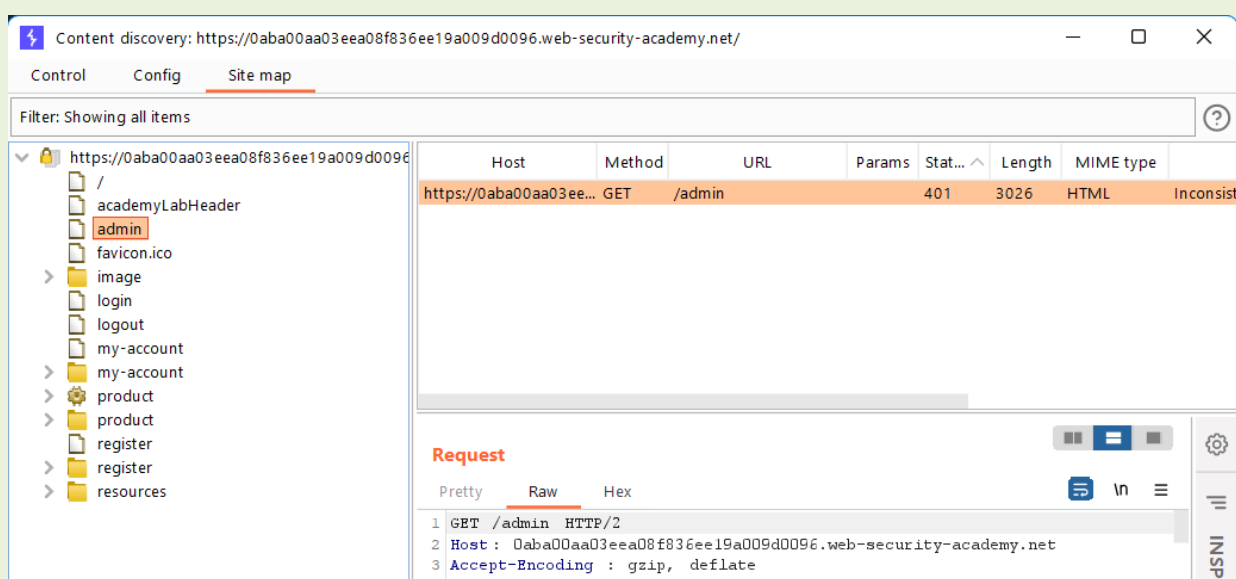
Name	Price	Quantity
Lightweight "I33t" Leather Jacket	\$1337.00	1
Conversation Controlling Lemon	\$36.69	-34
Total:	\$89.54	

Demo 3: INCONSISTENT SECURITY CONTROLS

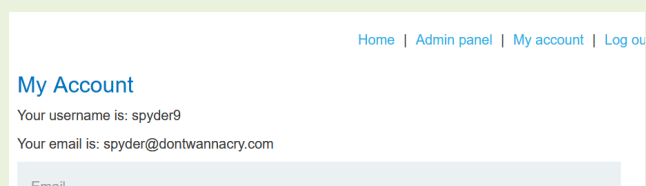
Lab Goal: To solve the lab, access the admin panel and delete the user **carlos**. Hence to show the site's vulnerability; this site's flawed logic allows arbitrary users to access administrative functionality that should only be available to company employees.

Solution:

1. Open the lab then go to the "Target" > "Site map" tab in Burp. Right-click on the lab domain and select "Engagement tools" > "Discover content" to open the content discovery tool.
2. Try and browse to `/admin`. Although you don't have access, the error message indicates that DontWannaCry users do.



3. Register with an arbitrary email address.
4. After clicking the link in the confirmation email to complete the registration, Log in using your new account and go to the "My account" page.
5. Here you can change your email address to an arbitrary spyder@dontwannacry.com address.



Notice that you now have access to the admin panel, where you can delete **carlos** user.



Demo 4: FLAWED ENFORCEMENT OF BUSINESS RULES

Lab Goal: This lab has a logic flaw in its purchasing workflow. To solve the lab, exploit this flaw to buy a "Lightweight l33t leather jacket". You can log in to your own account using the following credentials:

User Name : `wiener` and Password : `peter`

Solution: By applying 2 coupons - `NEWCUST5` and `SIGNUP30` we can make the purchase amount to zero



WebSecurity Academy

Flawed enforcement of business rules

LAB Solved


Back to lab description >>

Congratulations, you solved the lab!

Share your skills!   Continue learning >>

New customers use code at checkout: NEWCUST5

Store credit:
\$100.00

Home | My account |  0

Your order is on its way!

Name	Price	Quantity
Lightweight "l33t" Leather Jacket	\$1337.00	1
NEWCUST5	-\$5.00	
SIGNUP30	-\$401.10	
NEWCUST5	-\$5.00	
SIGNUP30	-\$401.10	
NEWCUST5	-\$5.00	
SIGNUP30	-\$401.10	
NEWCUST5	-\$5.00	
SIGNUP30	-\$401.10	

Total: \$0.00

REFERENCES

1. <https://www.imperva.com/learn/application-security/business-logic/#:~:text=Business%20logic%20vulnerabilities%20arise%20when,to%20potential%20losses%20or%20damages.>
2. <https://portswigger.net/web-security/logic-flaws/examples>
3. <https://portswigger.net/web-security/all-labs#business-logic-vulnerabilities>
4. <https://www.imperva.com/learn/application-security/insecure-direct-object-reference-idor/>

THANK YOU