Penetration Testing on OWASP Juice Shop

This presentation covers a comprehensive security assessment of the OWASP Juice Shop application, which is widely used to demonstrate common web vulnerabilities.

Our expert team conducted thorough testing to uncover potential vulnerabilities that could be exploited by attackers, focusing on both known and novel threats.

The goal was to identify security weaknesses, demonstrate exploit techniques, and provide actionable recommendations to improve the application's security posture.

Throughout the assessment, we followed industry best practices and utilized both automated tools and manual testing methods to ensure a detailed analysis.

by ramadan ragab

Project Overview

Goal

Identify and exploit vulnerabilities in OWASP Juice Shop to understand the security weaknesses and potential risks associated with web applications. Our aim is to improve awareness and demonstrate practical exploitation techniques.

Scope

Conduct a comprehensive security assessment covering the entire OWASP Juice Shop application, including frontend and backend components. The evaluation targets all critical modules to ensure thorough vulnerability identification.

Duration

The assessment was performed over a predefined timeframe allowing sufficient time for detailed reconnaissance, manual testing, and verification of findings. This balanced approach ensured accuracy and completeness in results.

Team

The security testing team comprised Ramadan Ragab, Abdelrahman Alaa, Abdelrahman Adel, and Mansour Ayman, each bringing expertise in penetration testing, web security, and application analysis. Their collaboration enhanced the depth of the assessment.

Methodology

Manual Testing

Performed in-depth analysis of application workflows, user inputs, and business logic to identify potential security weaknesses that automated tools might miss. This involved carefully crafting and executing test cases to simulate real-world attack scenarios.

Automated Scanning

Utilized industry-standard automated tools to quickly and efficiently detect common vulnerabilities such as SQL injection, cross-site scripting (XSS), and security misconfigurations. Automated scans complemented manual testing by covering a broader attack surface.

Tools Used

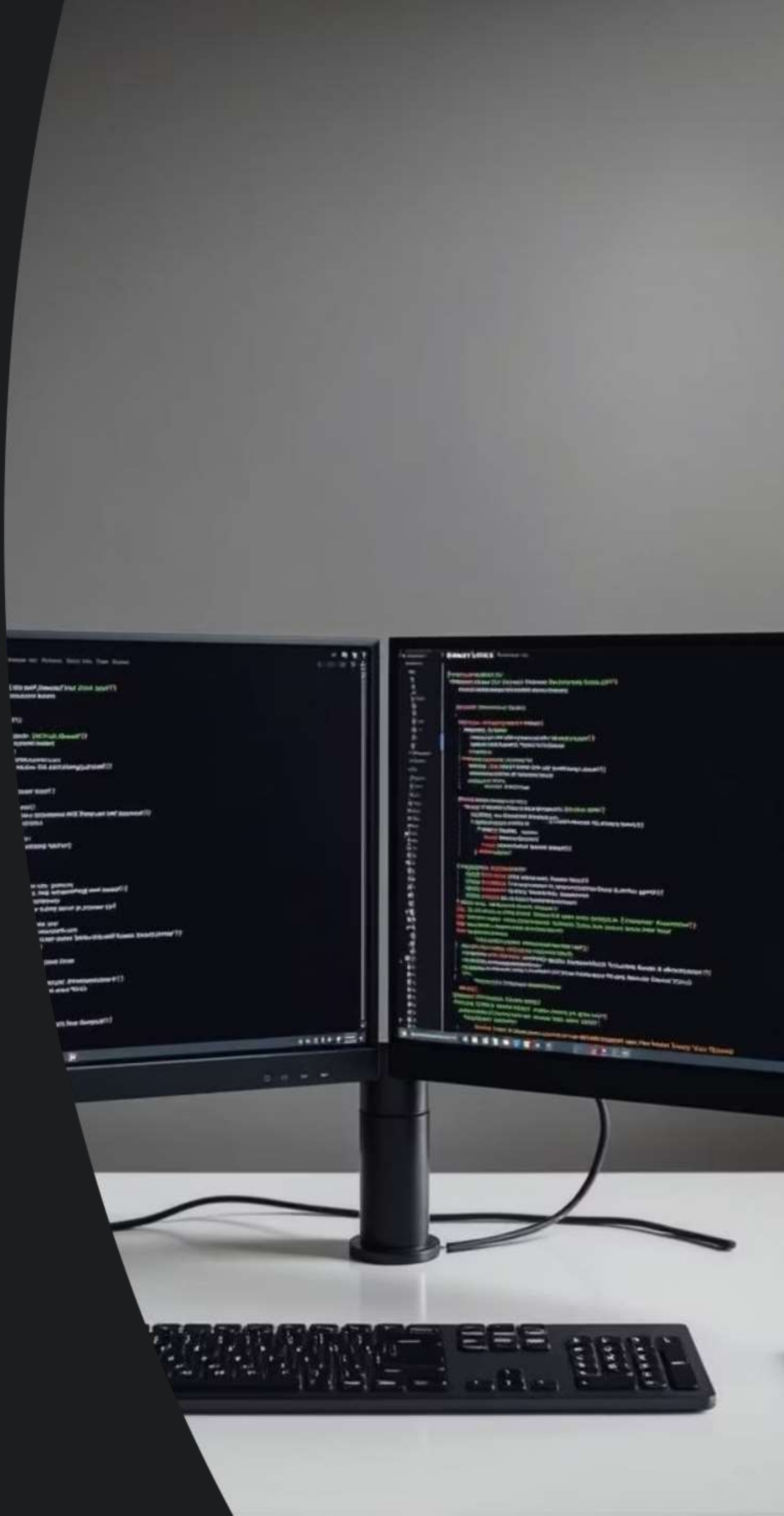
A diverse set of tools was leveraged for comprehensive coverage, including subfinder and AMASS for subdomain discovery, Burp Suite and OWASP ZAP for dynamic application security testing, Nmap for network scanning, HTTPX and FFUF for HTTP enumeration and fuzzing, and Shodan for external asset discovery. Each tool was selected for its unique strengths in the testing process.

Reconnaissance and Information Gathering

Extensive reconnaissance was conducted to understand the application's architecture, available endpoints, and underlying technologies. This phase aided in prioritizing targets and tailoring the testing approach to the specific environment.

Verification and Validation

Potential vulnerabilities identified were verified through multiple testing techniques to confirm their existence and assess their impact. This ensured accuracy in reporting and helped guide remediation efforts effectively.



Team Members and Discovered Vulnerabilitie

Team Member	Discovered Vulnerabilities
Ramadan Ragab	Broken Access Control, Input Validation Bypass
Mansour Ayman	SQL Injection, XSS,
Abdelrahman Adel	Forgotten Developer Backup, Changing Bender's Password
Abdelrahman Alaa	Remote Code Execution (RCE) via Server-Side Template Injection (SSTI), XML External Entity (XXE)

Reconnaissance: Building the Target Profile

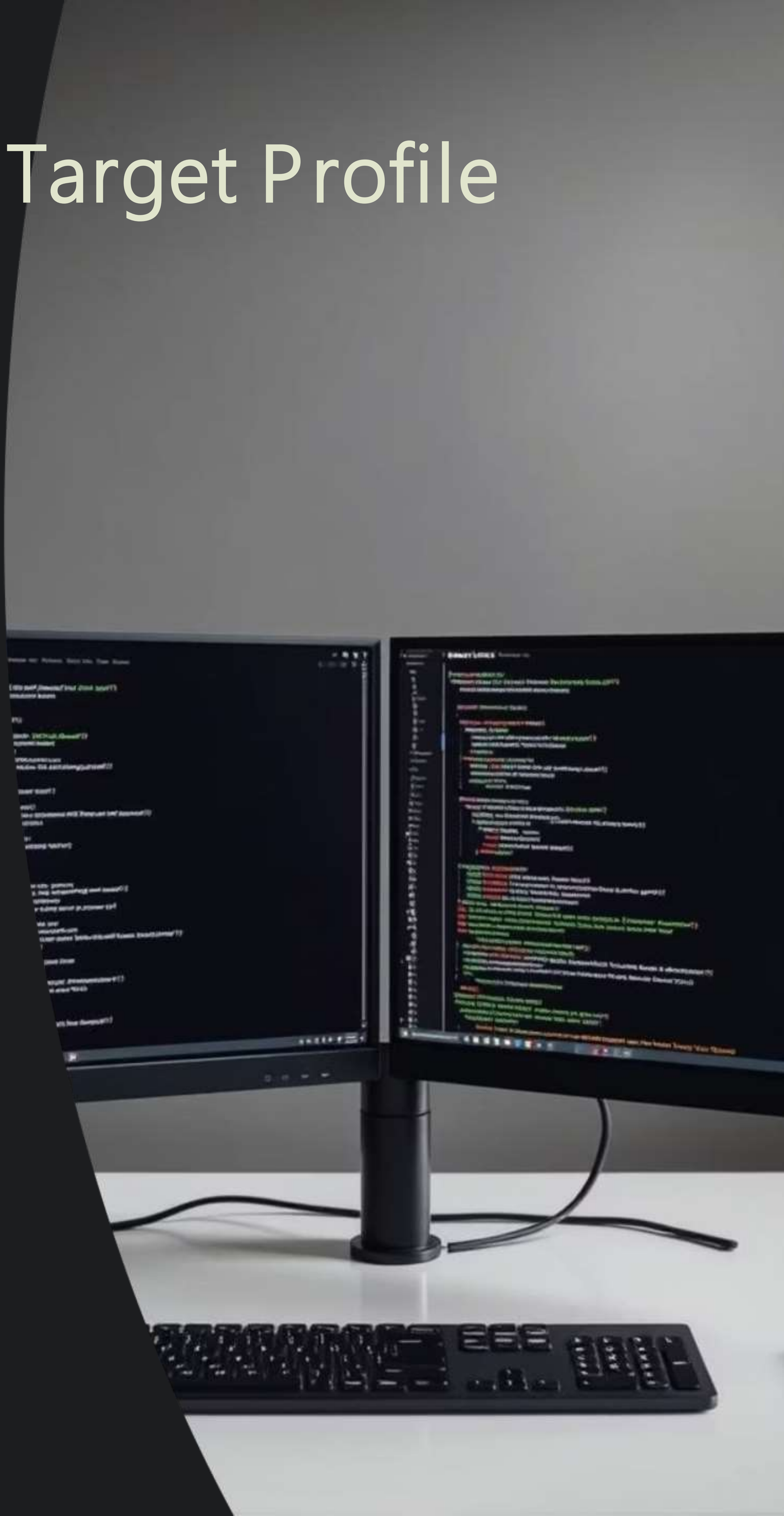
Reconnaissance begins with gathering core infrastructure elements such as IP addresses, subdomains, web directories, parameters, and associated email accounts linked to the target.

Specialized tools like theHarvester and Amass are leveraged to discover Autonomous System Numbers (ASNs), which are then translated into CIDR blocks to create comprehensive network maps that highlight potential attack surfaces.

Network sweeping is conducted through Nmap scans to identify active hosts and open ports, providing an overview of the live systems within the network perimeter.

Subdomain enumeration utilizes advanced reconnaissance tools including Subfinder and FFUF, enabling deeper surface mapping to uncover hidden assets, web endpoints, and vulnerable points that might otherwise be overlooked.

Additional information such as metadata, technology stacks, and third-party integrations are also collected during this phase to enrich the target profile and guide subsequent testing activities.



Vulnerability: Broken Access Control

Observation

This vulnerability occurs when users can perform actions beyond their permissions.

Exploit

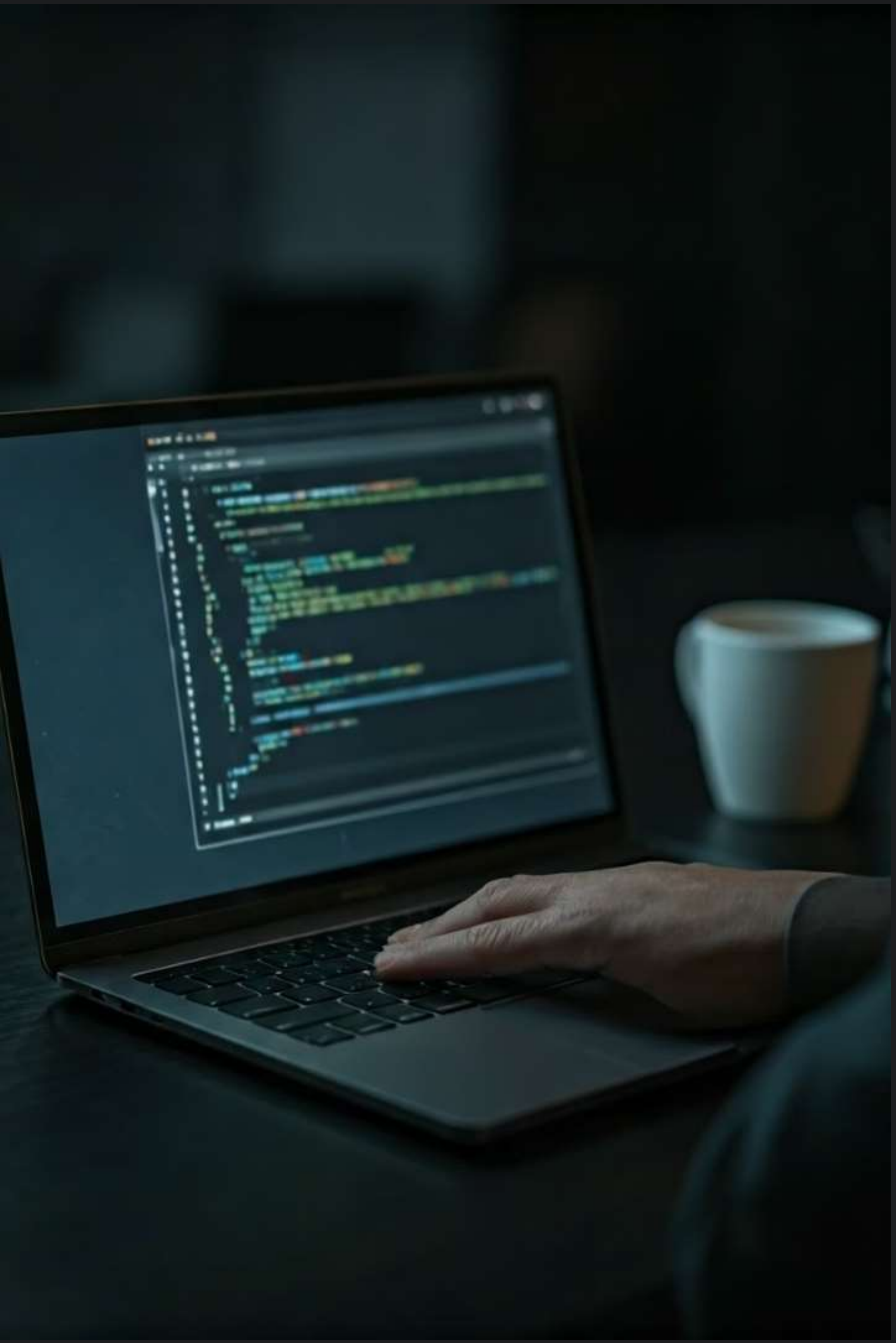
Attackers exploit flaws to access unauthorized data or functions. Common causes include lack of server-side checks and reliance on client-side validation.

Impact

This vulnerability allows attackers to manipulate another user's shopping basket by injecting multiple BasketId parameters. It bypasses authorization controls, leading to unauthorized access, data integrity issues, and potential financial or reputational damage.

Manipulate Basket: Exploiting Client-Side Validation

We begin by logging into an account and adding a product to the basket, observing the request and response details for any anomalies. Initially, the basket is empty, establishing a clear baseline and setting the stage for thorough testing of product additions and potential vulnerabilities within the client-side validation process. By carefully monitoring how the system handles these interactions, we aim to uncover weaknesses that could be exploited via manipulated client requests or basket ID tampering, which may lead to unauthorized actions or data exposure.



Adding Products and Analyzing Requests

Adding an item to the basket generates a request containing the product ID, basket ID, and quantity. For example, adding product 7 to basket 6 with quantity 1 is reflected in the request payload.

This step helps us understand the structure of requests and responses, which is crucial for testing how the system handles basket manipulation.

Request Payload

{"ProductId":7,"BasketId":"6","quantity":1}

Basket Status

Basket 6 belongs to the logged-in user and is initially empty.

Testing Basket ID Manipulation

The basket ID is numeric, suggesting a sequence of basket IDs like 1, 2, 3, and so on. This opens the possibility of targeting other users' baskets by changing the basket ID in the request.

By modifying the ProductId and BasketId in the request to {"ProductId":5,"BasketId":"4","quantity":1}, an attempt is made to add a product to another user's basket.

Original Request

```
{"ProductId":7,"BasketId":"6","quantity":1}
```

Modified Request

```
{"ProductId":5,"BasketId":"4","quantity":1}
```

Authorization Validation and JWT Analysis

The server responds with 401 Unauthorized when trying to add a product to another basket, indicating some validation is in place. This validation likely involves the JWT token stored in the cookie.

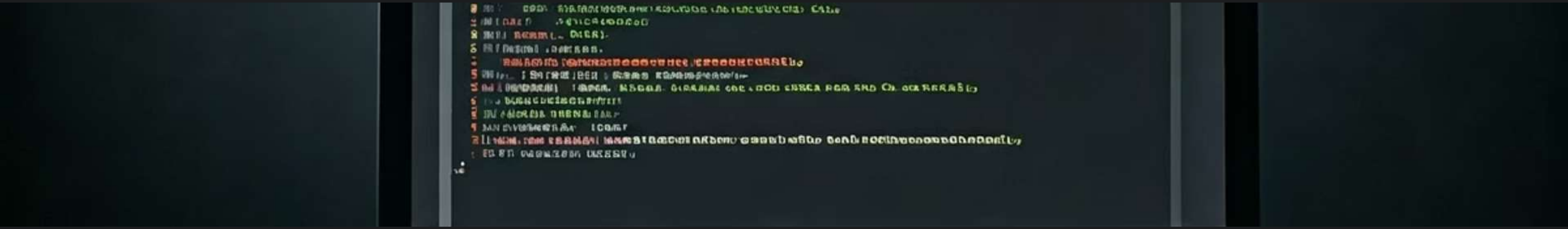
Decoding the JWT reveals it does not contain any basket ID information, suggesting the basket ID is not tied directly to the token's claims.

401 Unauthorized

Server rejects unauthorized basket manipulation attempts.

JWT Token

Decoded token lacks basket ID data, implying separate validation mechanisms.



Parameter Pollution Exploit

By sending a request with duplicate BasketId parameters, such as {"ProductId":5,"BasketId":"6","BasketId":"4","quantity":1}, the system processes the request successfully. This is an example of parameter pollution.

This vulnerability allows bypassing authorization checks and manipulating other users' baskets by exploiting how the server handles multiple parameters with the same name.

1

Original Request

Single BasketId parameter with user's own basket ID.

2

Polluted Request

Multiple BasketId parameters with different values, bypassing validation.

3

Result

Unauthorized product addition to another user's basket succeeds.

Vulnerability: Input Validation Weakness

Login Form Weakness

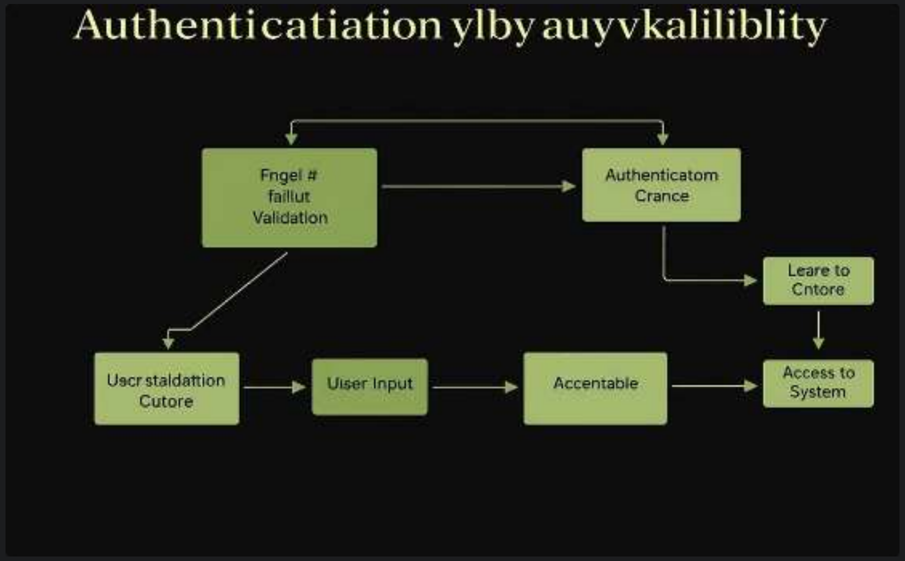
Improper input validation mechanisms in the login form permit the submission of malformed or malicious data, which can bypass client-side checks.

This weakness often stems from insufficient sanitization and failure to enforce strict input rules, exposing the system to unauthorized access.

Exploitation Potential

Attackers can exploit this vulnerability by crafting specially designed inputs that manipulate the authentication logic.

Such manipulation can effectively bypass standard authentication checks, allowing attackers to log in as privileged users without valid credentials.



Impact

The lack of robust input validation can lead to unauthorized access, granting attackers full control over user accounts and potentially exposing sensitive data.

This can result in data breaches, system manipulation, and significant damage to the integrity and confidentiality of the application.

Demonstration: Imborer input validation

Register User with Burp Suite

Submit user registration while intercepting the request to inspect data sent to server.

Modify Role Attribute

Change "role" from "customer" to "admin" in the intercepted request payload.

Send Crafted Request

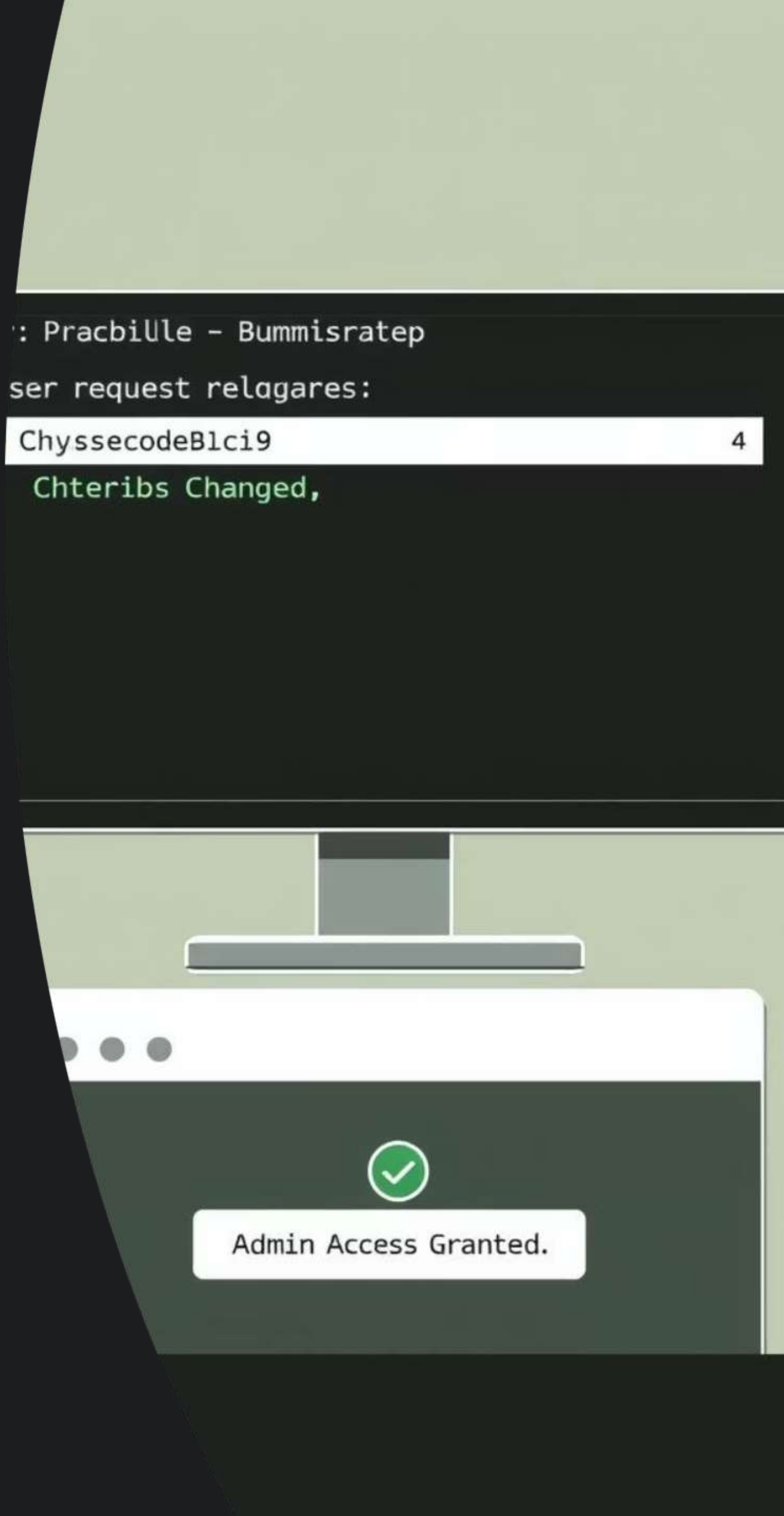
Submit modified registration data and receive a 201 Created response from server.

Access Admin Panel

Use directory brute force to locate admin panel, confirming unauthorized admin access.

Prevent Unauthorized Admin Creation

Remove public admin registration and enforce backend controlled role assignments.



```
: PracbiUle - Bummisratep
ser request relagares:
ChyssecodeBlci9                                    4
 Chteribs Changed,
```

Vulnerability: Cross-Site Request Forgery (CSRF)

Attackers exploit the change name field to perform unauthorized actions on behalf of authenticated users without their knowledge.

This vulnerability allows malicious requests to be sent from a trusted user’s browser without their consent or interaction, enabling attackers to manipulate application state or perform actions with the user's privileges.

By tricking victims into submitting forged requests, attackers can change user information, transfer funds, or escalate privileges, potentially compromising sensitive data and application integrity.

Demonstration: Cross-Site Request Forgery (CSRF) Attack

This demo shows how to change a username via CSRF from a different origin.

Without a CSRF token, the server cannot verify the request source, leaving the application vulnerable.

A crafted HTML form submits a malicious username change request automatically when the victim visits the attacker's page.

The attack successfully modifies the user's profile without their knowledge or consent, demonstrating the risk of unauthorized actions.

This vulnerability allows attackers to perform actions on behalf of authenticated users, potentially leading to account takeovers or data corruption.

Mitigating CSRF requires implementing anti-CSRF tokens and validating request origins to protect user sessions.

The demonstration highlights the importance of securing state-changing requests against forgery attacks in web applications.

Vulnerability: SQL Injection

Location

The login form is the primary entry point where this vulnerability is found. It handles user input data for authentication purposes.

Details

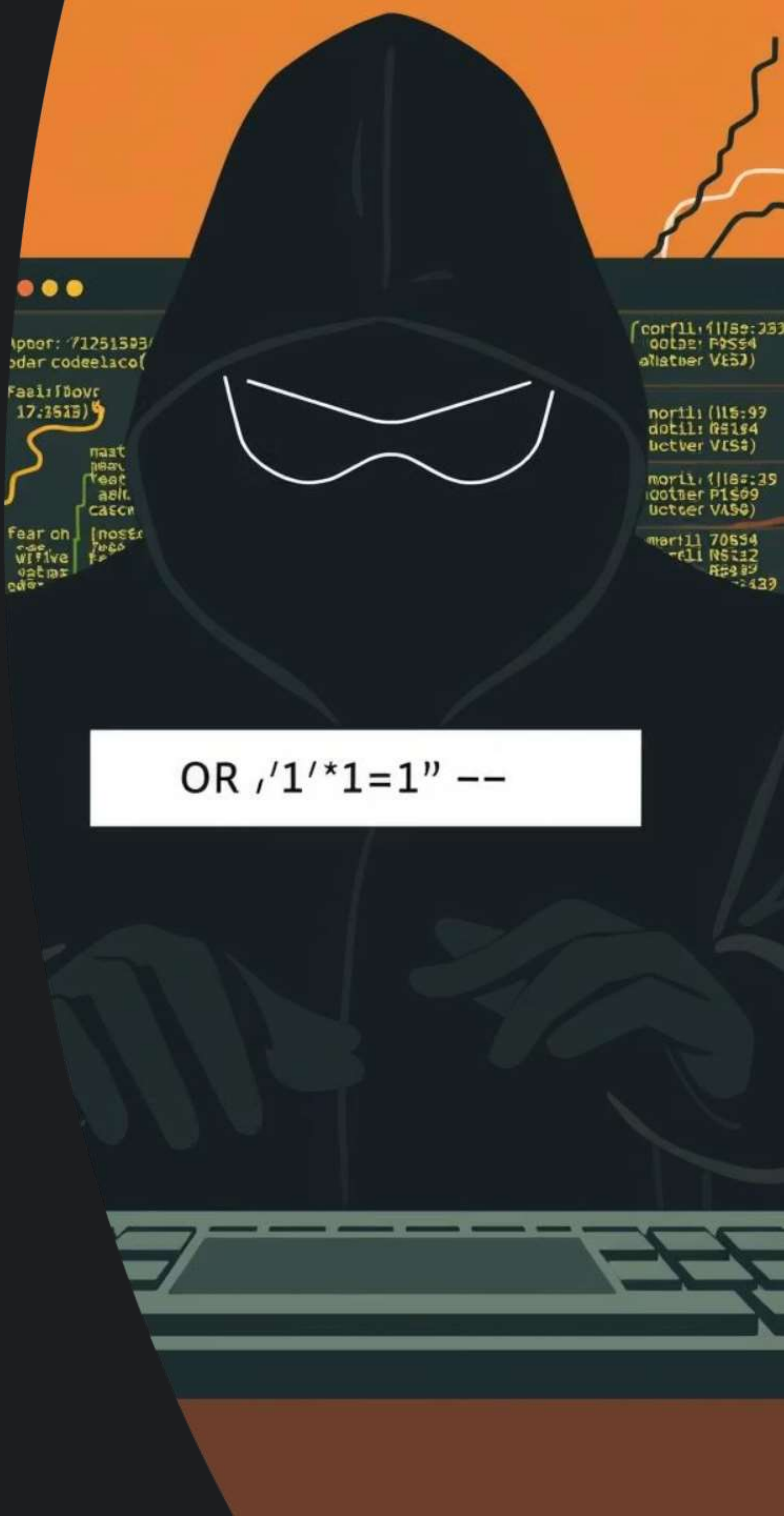
This vulnerability allows attackers to bypass authentication by injecting malicious SQL commands into input fields such as username or password. The application fails to properly sanitize or parameterize these inputs, allowing unauthorized database queries.

Impact

If exploited successfully, attackers can gain unauthorized access to user accounts and potentially escalate privileges. This can lead to complete compromise of the system, exposure of sensitive data, and manipulation of the database content.

Steps

Attackers inject specially crafted SQL statements into the username or password input fields to manipulate or bypass the intended authentication logic. This often involves techniques like tautology-based injections or union-based queries to extract or alter data.



Demonstration: SQL Injection Exploitation

This demonstration illustrates how an attacker can leverage SQL injection vulnerabilities to bypass the authentication mechanism of an application.

By injecting carefully crafted SQL commands into the login input fields, the attacker manipulates the backend database queries to gain unauthorized access to the system.

This technique exploits improper input validation and lack of parameterized queries on the server side, allowing malicious input to alter the intended SQL logic.

1 **Step 1: Craft Injection**

Develop a malicious SQL payload designed to manipulate the authentication query, typically by using tautologies or union-based injection to bypass standard login verification.

2 **Step 2: Submit Credentials**

Input the crafted SQL injection string into the username or password fields of the login form, triggering the backend to execute the altered query.

3 **Step 3: Gain Access**

By successfully bypassing authentication checks, the attacker gains unauthorized entry into the application without providing valid user credentials.

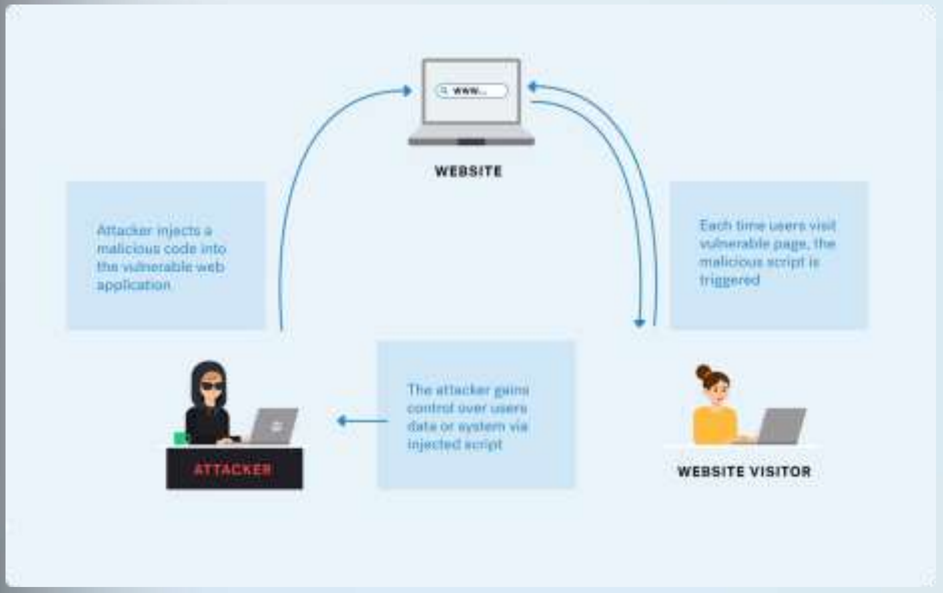
Cross-Site Scripting (XSS): Understanding, Exploitation, and Prevention

What is XSS?

What is XSS? Cross-Site Scripting vulnerabilities occur when attackers attackers inject malicious scripts into trusted websites, which then execute in users' browsers. This undermines trust and security by enabling an attacker to steal sensitive data, hijack sessions, deface websites, or redirect users to to malicious pages.

XSS Types

There are three main types of XSS: **Reflected**, triggered via a single request; **Stored**, where malicious scripts persist on servers such as databases or logs; and **DOM-based**, which exploits client-side JavaScript to modify the page's environment dynamically. Understanding these types is key to both exploitation and prevention.


www...
WEBSITE
Attacker injects a malicious code into the vulnerable web application
Each time users visit vulnerable page, the malicious script is triggered
The attacker gains control over users data or system via injected script
ATTACKER
WEBSITE VISITOR

Real-World Testing, Payloads, and Bypassing

1

Input Field Analysis

Initial step involves identifying fields vulnerable to script injections, such as search bars, comment boxes, or form inputs.

2

Payload Crafting

Malicious scripts are crafted to trigger alerts or execute unauthorized actions. Example payloads include **<script>alert("XSS")</script>** and **<img src=x onerror=alert('XSS')>**.

3

Exploitation Outcomes

Successful exploitation can lead to session ID theft, cookie extraction, or user redirection to attacker-controlled sites.

4

Bypassing Techniques

Attackers often bypass defenses like Content Security Policy (CSP) using advanced methods, such as embedding malicious code in data URIs with inline event handlers, evading CSP restrictions.

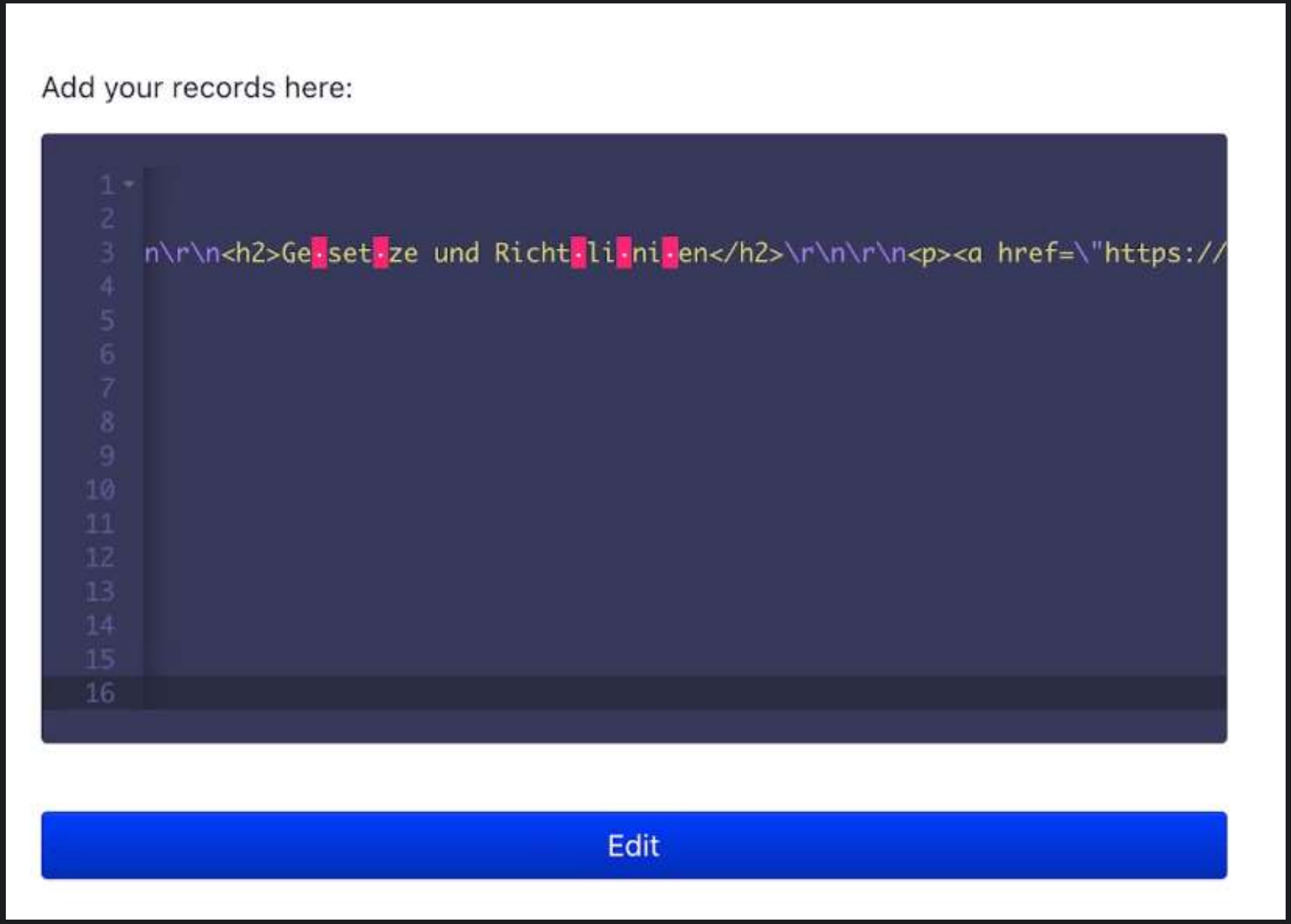
XSS Prevention Techniques

Content Security Policy (CSP)

CSP acts as an HTTP response header controlling what sources of content the browser can load, significantly reducing the risk of unauthorized script execution. A typical policy might look like: **Content-Security-Policy: default-src 'self'**.

Input Validation and Output Sanitization

Proper input validation, such as whitelisting characters or using regular expressions, prevents malicious data from entering servers. Output sanitization tools like **DOMPurify** clean potentially harmful HTML before rendering, preventing script execution.



Add your records here:

```
n\r\n<h2>Ge·set·ze und Richt·li·ni·en</h2>\r\n\r\n<p><a href=\"https://
```

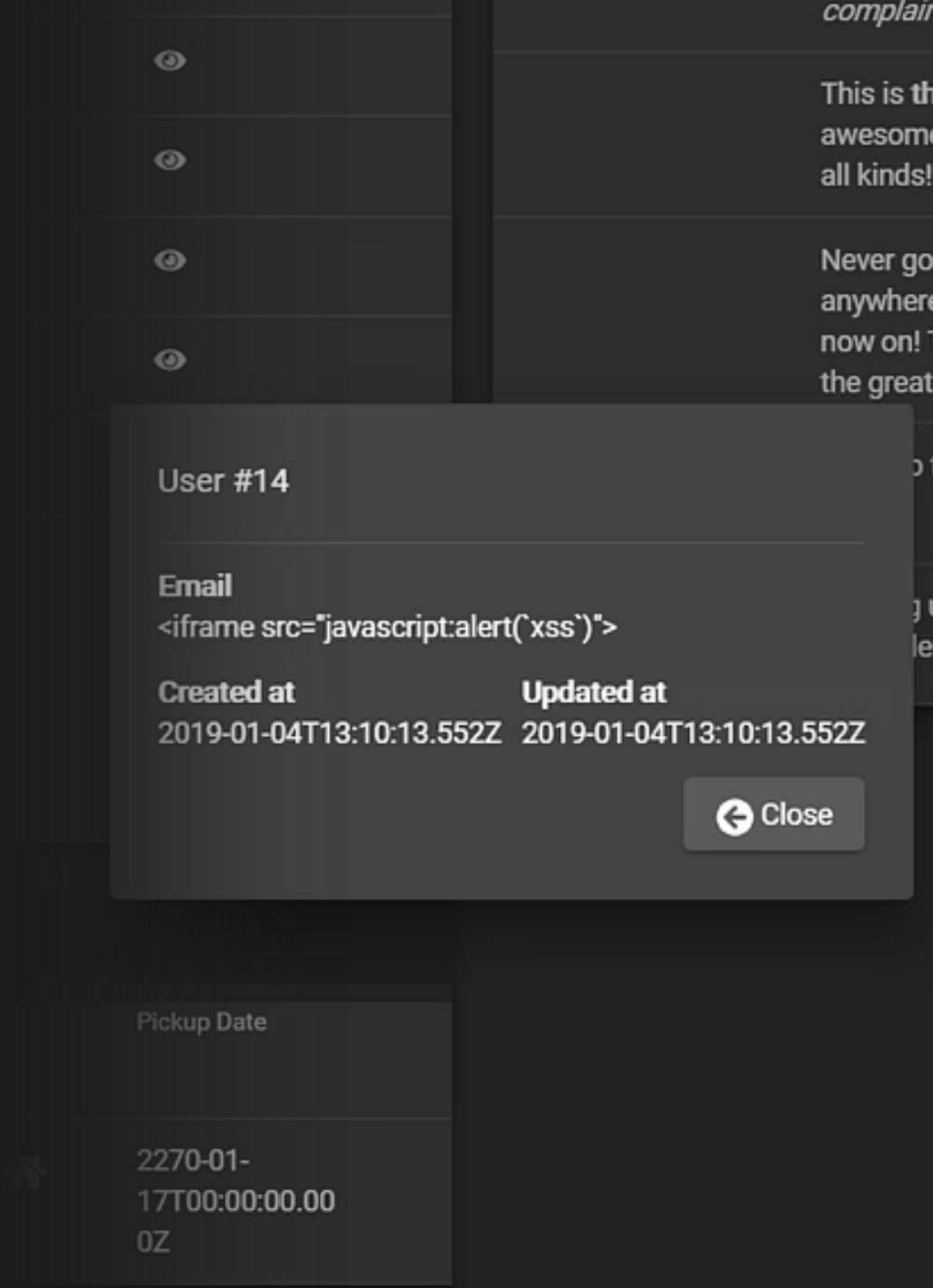
Demonstration: XSS Attack on OWASP Juice Shop

Visualize a real cross-site scripting attack targeting the OWASP Juice Shop, a purposely vulnerable web application used for security training and awareness.

Watch closely as input fields, such as search bars and comment sections, are exploited by attackers to inject and execute malicious scripts in a user's browser.

This demonstration highlights how unsanitized inputs can lead to critical security breaches, including data theft, session hijacking, or unauthorized actions performed without the user's consent.

Understanding this attack in a real-world context emphasizes the importance of secure coding practices and robust security defenses.



User #14

Email
<iframe src="javascript:alert(`xss`)">

Created at	Updated at
2019-01-04T13:10:13.552Z	2019-01-04T13:10:13.552Z

Forgotten Developer Backup: Null Byte Injection

Access FTP Server

Locate the backup file stored on the FTP server, noting that it lacks a readable or expected file extension, which prevents normal access.

Encounter Access Restriction

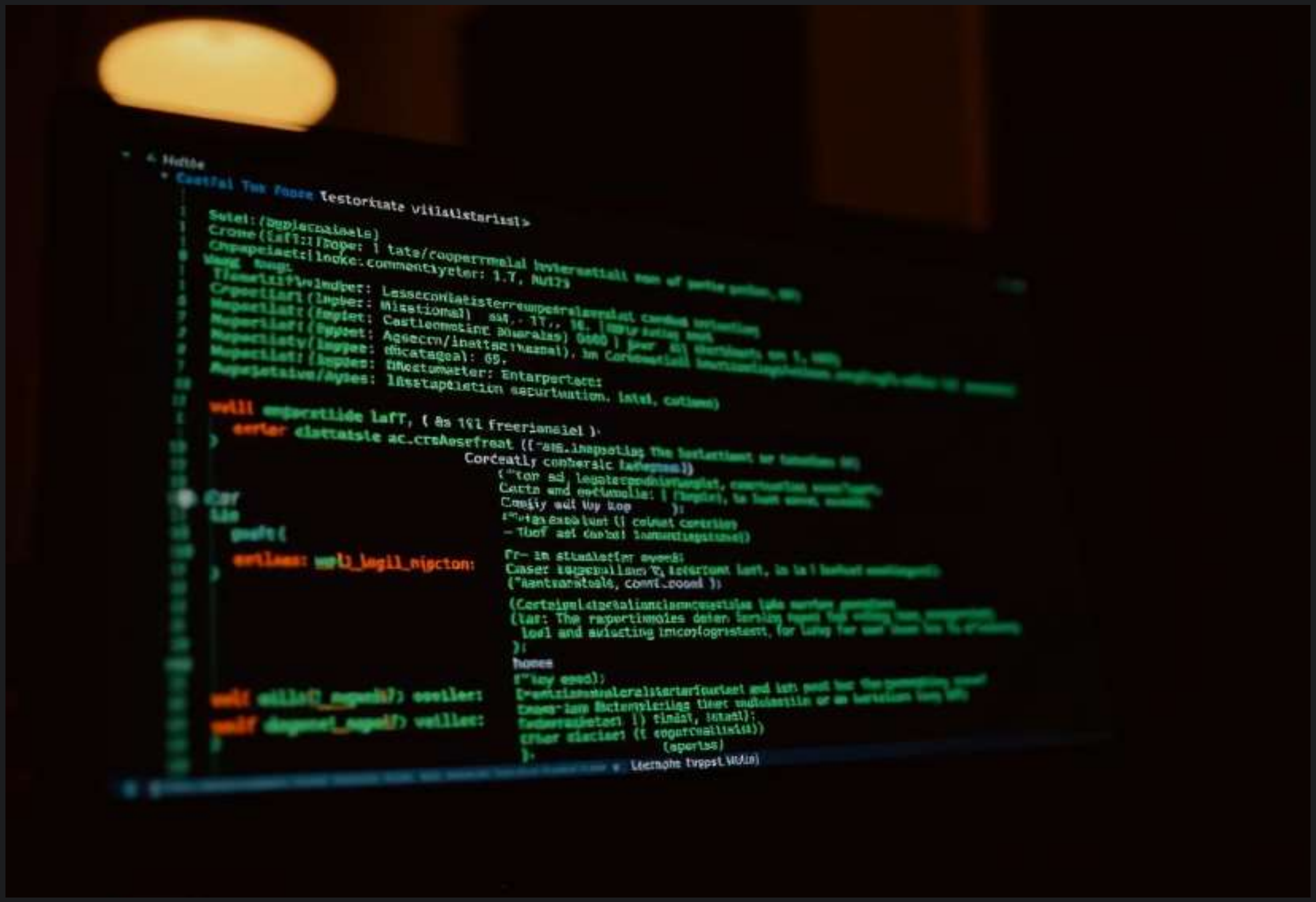
Observe that the FTP server has a strict file extension validation, blocking any access to files unless the extension is explicitly **.md** for Markdown files.

Apply Null Byte Injection

Exploit the vulnerability by appending a null byte character (**%00**) before the **.md** extension in the filename, using the payload **backup.%00md** to bypass the extension filter.

Gain File Access

Successfully bypass the file extension validation and open the backup file, enabling retrieval of potentially sensitive backup contents that were otherwise inaccessible.



Vulnerability: Changing Bender's Password

Bypass Login

Use the payload **user'--** to bypass authentication and log in.

Access Change Password Page

Navigate to the password change interface requiring the current password.

Intercept and Modify Request

Remove **currentPassword** parameter from the intercepted request via Burp Suite.

Send Modified Request

Submit the altered request successfully changing Bender’s password without the old one.

Vulnerability: Server-Side Template Injection (SSTI)

Location

The vulnerability is identified specifically on the Profile Page, where user inputs are handled dynamically within template rendering logic, creating an attack surface for SSTI.

Details

The application fails to properly sanitize user-supplied data fields, which are directly embedded into server-side templates. This flaw enables attackers to inject malicious code executed by the server's template engine.

Impact

Successful exploitation of SSTI can lead to remote code execution on the server, allowing an attacker to gain unauthorized shell access. This compromises the entire system's confidentiality, integrity, and availability.

Steps

To exploit this vulnerability, an attacker injects a carefully crafted payload into the profile fields, which gets processed by the server template engine. This triggers code execution that can establish a reverse shell session, granting command execution capabilities.

Demonstration: SSTI Reverse Shell

1

Step 1: Inject Payload

Input carefully crafted malicious code into profile page input fields to exploit the server-side template vulnerability.

This payload is designed to be executed by the server's template engine, allowing arbitrary code execution.

2

Step 2: Gain Shell

Once the payload is processed, establish a reverse shell connection back to the attacker-controlled system.

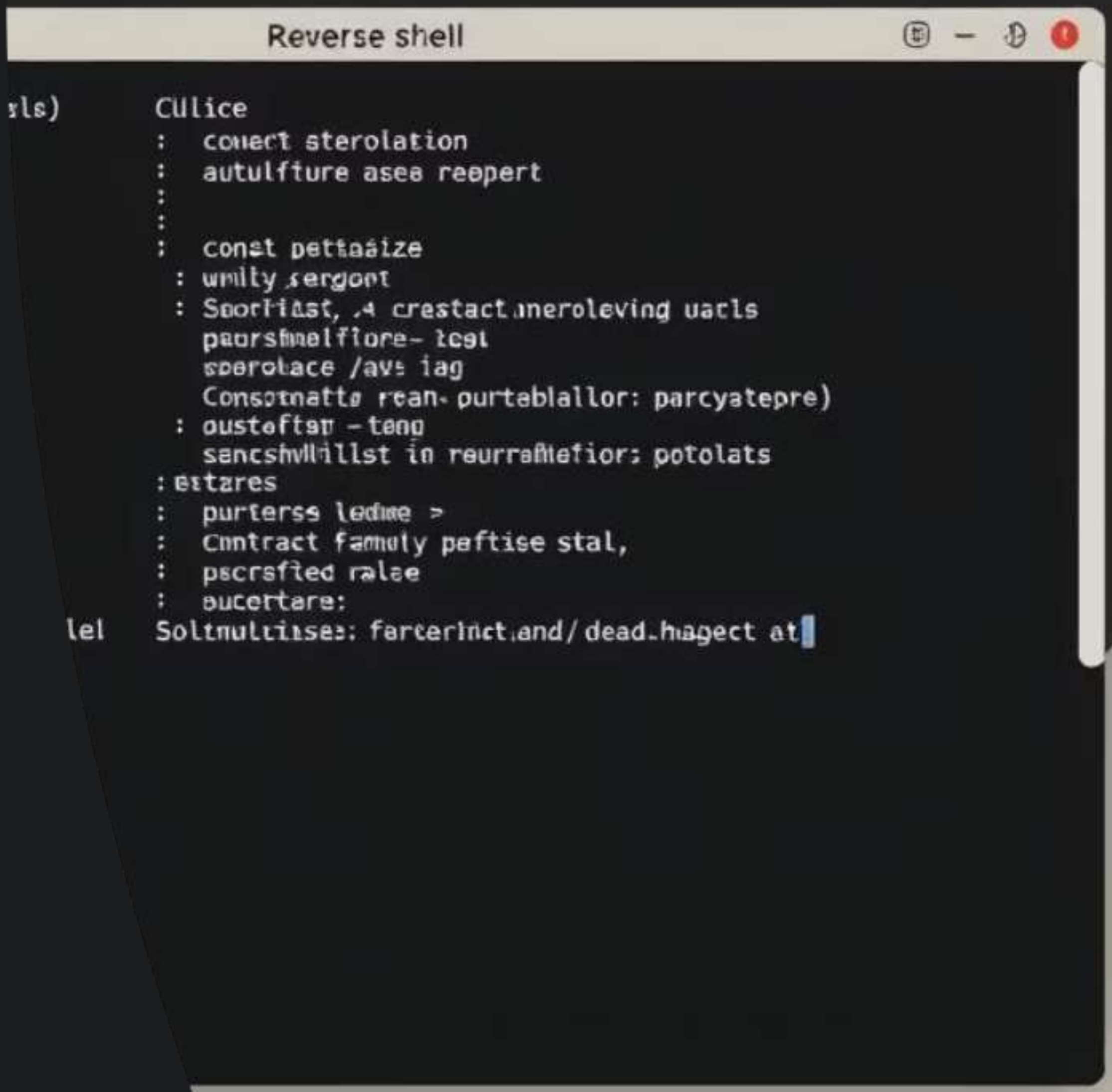
This connection provides direct access to the server environment, bypassing normal authentication mechanisms.

3

Step 3: Execute Commands

With shell access obtained, run system commands to explore the server and extract sensitive information.

Commands can include file system navigation, data exfiltration, and modifying system configurations.



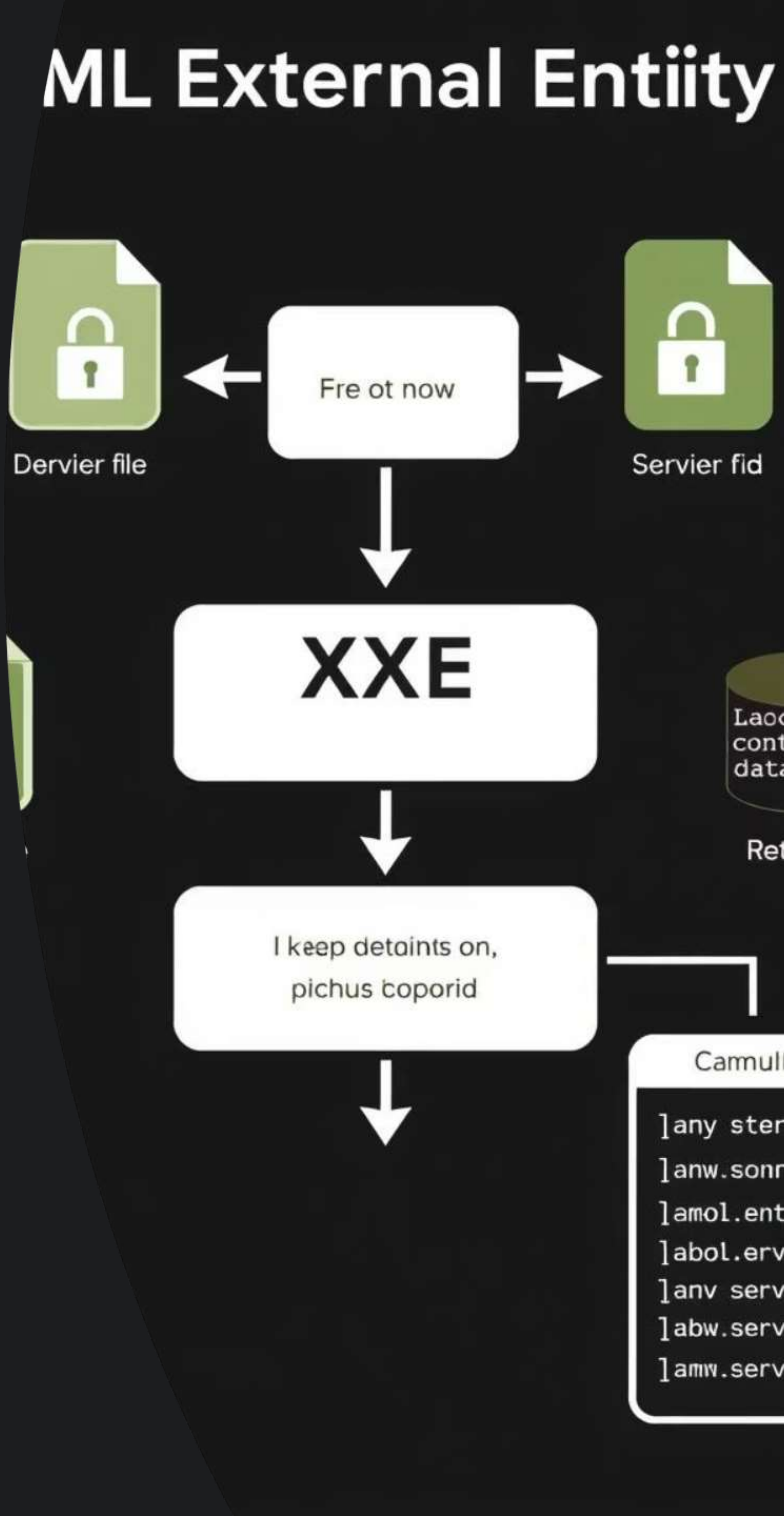
```
sls)      CUlice
          :  conect sterolation
          :  autulfture asea reepert
          :
          :
          :  const pettaaize
           : unity sergont
           : Soorliast, .4 crestact neroleving uacls
             paorsfmelflore- tcal
             sperolace /avs iag
             Consotnatta rean- purtablallor: parcyatepre)
           : oustoftar - teng
             sancshvllillst in reurrafiefior: potolats
          : estares
          :  purterss ledme >
          :  Cmntract famuly peftise stal,
          :  pscrsfied ralse
          :  oucertare:
  lel     Soltmultiises: farcerinct,and/ dead.hagect at
```

xxe Data Access

Through this vulnerability, I was able to identify which entities were retrieving data or interacting with the server. This discovery confirms that these entities are affected by the XXE vulnerability. For example, there might be a file stored on the server that returns responses from the backend, revealing sensitive information or system behavior.

By exploiting this flaw, an attacker can manipulate the XML input to trigger the server to disclose internal files or data, which normally should be inaccessible. This can include configuration files, credentials, or other sensitive resources that provide insight into the system's structure or security controls. The data leakage enabled by XXE can also allow attackers to gain further foothold within the network, leading to more severe attacks such as remote code execution or data exfiltration.

Understanding which components are vulnerable and how they communicate is crucial for effective remediation. It helps in pinpointing the exact attack vectors and in implementing stricter input validation and parsing rules to prevent unauthorized access. Overall, the XXE vulnerability poses a significant risk to the confidentiality and integrity of the application and its underlying infrastructure.



Demonstration: XXE Data Access Steps

Step 1: Craft Malicious XML Payload

Embed external entity references within the XML structure designed specifically to access sensitive files on the target server. This may involve referencing system files like */etc/passwd* or configuration files that should not be exposed to the user. The payload exploits insecure XML parsers that do not properly restrict external entity processing.

Step 2: Submit Payload to Application

Send the crafted malicious XML payload to the vulnerable application endpoints, such as file upload modules, data input forms, or API endpoints that process XML input. The application, failing to validate or sanitize the input properly, processes the external entity and triggers the retrieval of restricted content.

Step 3: Retrieve Sensitive Data

Extract internal files or server responses that are returned by the application's XML parser as a result of the malicious external entity resolution. This sensitive data may include server configuration, user credentials, or other confidential information that should remain protected from unauthorized access.

Conclusion & Recommendations

Summary

Our security assessment found several critical vulnerabilities threatening the application's integrity and data. These include SSTI allowing remote code execution, IDOR exposing sensitive user data, SQL Injection enabling authentication bypass and data manipulation, and broken access control facilitating privilege escalation. These high-risk issues stem from weak input validation and authorization controls requiring prompt remediation.

Remediation

We recommend strict input validation and output encoding to block injection attacks. Adopt zero-trust access with robust role-based permissions and regular audits to prevent IDOR and access control flaws. Patch all vulnerabilities quickly and embed secure coding standards into development practices. Use parameterized queries, prepared statements, and secure templates to mitigate injection risks.

Continuous Monitoring

Security requires ongoing vigilance through real-time monitoring, automated scans, and scheduled penetration tests. Maintain detailed logging and regular analysis for rapid incident response, supported by a clear incident plan with defined roles and procedures.

Further Steps

Strengthen security policies and promote awareness throughout development and operations teams via regular training. Conduct audits and risk assessments to align with standards and regulations. Invest in advanced security technologies like behavioral analytics, machine learning detection, and multifactor authentication to defend against evolving threats.

Conclusion & Recommendations

Our penetration test uncovered multiple critical vulnerabilities including Server-Side Template Injection (SSTI), Insecure Direct Object References (IDOR), SQL Injection, and Broken Access Control. These flaws enable arbitrary code execution, unauthorized data access, and privilege escalation risks.

We recommend enforcing strict input validation, implementing robust access controls, and promptly patching all identified issues. Establishing continuous security monitoring and automated testing will help maintain resilience against evolving threats.

Additionally, strengthening security policies, increasing developer training on secure coding, and conducting regular audits are vital steps to safeguard the application environment long-term.

Preventing Critical Vulnerabilities

- **Input Validation:** Strictly sanitize all user inputs to block malicious data. This helps prevent attacks such as code injection, cross-site scripting, and other input-based exploits by ensuring only safe and expected data enters the system.
- **Access Controls:** Enforce role-based permissions to restrict resource access. Implement the principle of least privilege so users and components can only access what is necessary, reducing the risk of unauthorized data exposure or modification.
- **Timely Patching:** Apply security patches promptly to fix known flaws. Staying current with updates helps protect against exploits targeting vulnerabilities that could otherwise be leveraged by attackers to compromise the system.
- **Continuous Monitoring:** Use automated tools to detect emerging threats quickly. Regular scanning, real-time alerting, and anomaly detection enable early identification of suspicious activity and faster incident response.
- **Developer Training:** Educate developers on secure coding and attack vectors. Providing ongoing training ensures the team stays aware of current security best practices and common vulnerabilities to build safer applications from the ground up.

Final Thoughts on Penetration Testing

Comprehensive Coverage

Our test explored critical vulnerabilities threatening modern web apps. We thoroughly examined multiple attack vectors to understand where defenses might fail, highlighting areas requiring urgent attention.

Real-World Impact

Flaws like SSTI, IDOR, and SQL Injection can lead to severe data breaches, allowing attackers to execute arbitrary code, access sensitive information, or escalate privileges. Recognizing these risks is essential to protect user data and maintain trust.

Immediate Actions

Applying patches and enforcing strict input validation are vital first steps to mitigate existing vulnerabilities. Early remediation prevents attackers from exploiting weaknesses and reduces the attack surface significantly.

Ongoing Commitment

Continuous monitoring and developer education ensure lasting security. Regular training on secure coding practices combined with automated tools help identify new risks promptly and maintain a strong security posture over time.

Thank You for Listening

We appreciate your time and attention throughout this presentation. Your engagement and thoughtful questions have helped highlight the critical importance of web application security.

Your commitment to improving security is vital for protecting sensitive data and maintaining the trust of your users and customers. Stay vigilant and proactive by continuously assessing risks and applying best practices in your development and operational processes.

Remember, security is an ongoing journey, not just a one-time project. Regularly updating your knowledge and tools will help you stay ahead of emerging threats in an ever-evolving cybersecurity landscape.

1 **Questions Welcome**

Please ask any questions or share your thoughts. We value your input and are here to support your security initiatives with practical advice and insights.

2 **Next Steps**

Implement recommended security measures promptly to reduce exposure to vulnerabilities. Prioritize patching, enforce access controls, and validate inputs thoroughly to strengthen your defenses.

3 **Ongoing Support**

Contact us for assistance with future penetration testing, continuous monitoring, and tailored security training. Together, we can build a resilient and secure environment.