# Web Application Penetration Testing: A Hands-On Workshop

Welcome to our comprehensive workshop on web application penetration testing.

This workshop is for developers and/or security enthusiasts who have a basic understanding of web applications and want to learn how to find real security issues. You'll get hands-on experience using popular tools and practical methods that professionals use to test web apps in real-world situations.

SK **by Sheshananda Reddy Kandula**

# Whoami

@Sheshananda Reddy Kandula

15 years in Application Security/Cyber Security

# Agenda

- Introductions to Web App Pen Testing - 45 min - 1:30 - 2:15 PM
  - Introductions (5 min) - 1:35PM
  - Web Application Security and OWASP top 10 risks (10 minutes) - 1:45PM
  - CIA Triad and Encryption, Hashing, and Encoding (10 minutes) - 1:55PM
  - Setting Up the Environment and Tools (10 minutes) - 2:05PM
  - Reconnaissance and Information Gathering (10 minutes) - 2:15PM
- Common Web Application Vulnerabilities (2+ hours) - 2:15PM - 4:30PM
  - SQL Injection (20 minutes) - 2:35PM
  - Cross-Site Scripting (XSS) (20 minutes) - 2:55PM
- —BREAK — 5 min
  - How HTTPS works (10 minutes) - 3:10PM
  - Password Storage in DB (5 min) - 3:15PM
  - Server-Side Request Forgery (SSRF) (15 minutes) - 3:30PM
  - Local File Inclusion (LFI) (10 min) - 3:40 PM
  - Broken Authentication and Session Management (15 minutes) - 3:55PM
  - OAuth 2.0 (15 minutes) - 4:10PM
  - Security Misconfigurations and Logic Flaws (15 minutes) - 4:25PM
- Reporting and Mitigation Strategies (15-20 minutes) - 4:30 - 5:00PM
  - Secure Coding Best Practices
  - Key Takeaways and Next Steps
  - Q&A and Wrap-Up

# Introductions

Who are you?

What brings you here?

Do you have CyberSecurity experience?


# Why Web Security Matters

## 1 Rising Threats

Data breaches and cyberattacks are increasing at an alarming rate. No organization is immune.

## 2 Real-world Impact

Recent attacks include social media hacks, financial data leaks, and ransomware incidents.

## 3 Devastating Consequences

Organizations face financial losses, reputation damage, and erosion of user trust.

# Recent High-Profile Cyberattacks

- **MOVEit (2023):** SQL Injection vulnerability exposed 60+ million users' data across multiple organizations.
- **MGM Resorts (2023):** Ransomware forced casino operations offline, costing millions in revenue and recovery.
- **LastPass (2022-2023):** Password manager breach exposed encrypted vaults, threatening millions of stored credentials.
- **Key Lesson:** Timely patching, multi-factor authentication, and security training are essential defensive measures.

Zero-Day Vulnerability in MOVEit Transfer Exploited for Data Theft | Mandiant | Google Cloud Blog

ALPHV: Hackers Reveal Details of MGM Cyber Attack

Experts Fear Crooks are Cracking Keys Stolen in LastPass Breach

# Notable Web Application Breaches and Their Impact


## Equifax (2017)

A vulnerability in the Apache Struts framework led to the exposure of 147 million Americans' personal data. The breach cost over $1.4 billion in remediation and settlements.


## Capital One (2019)

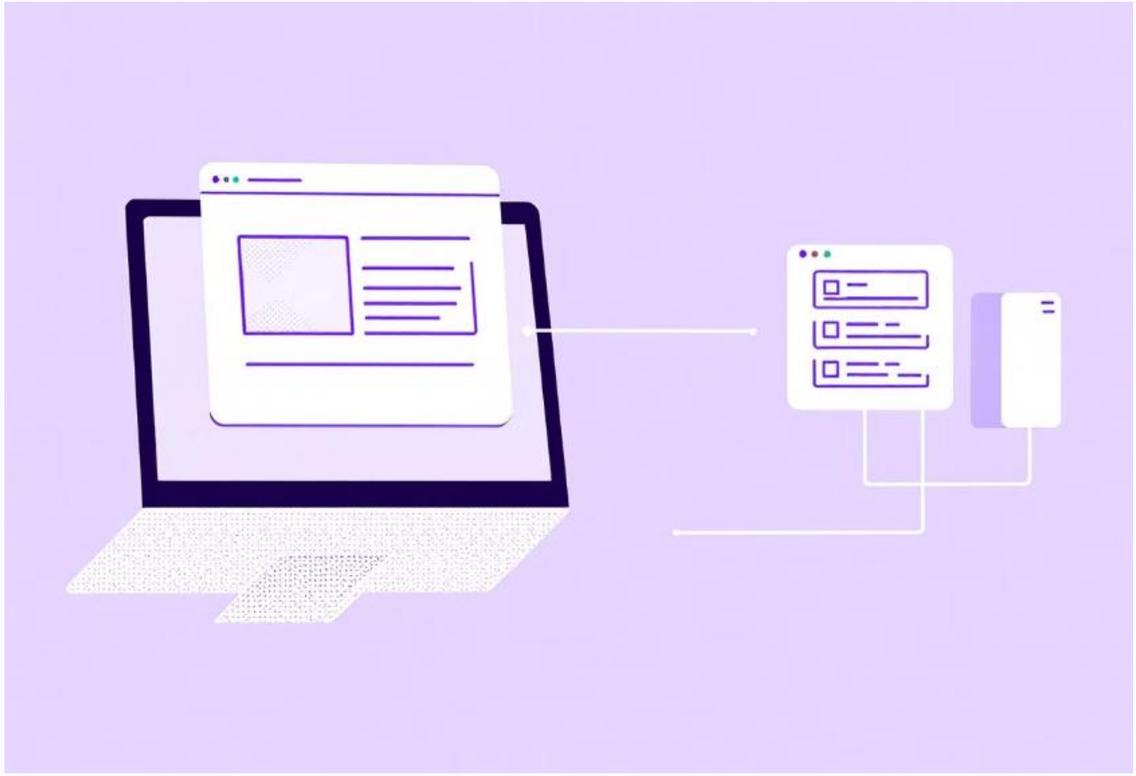
A server-side request forgery (SSRF) vulnerability allowed access to over 100 million customer records. The breach resulted from a misconfigured web application firewall.

## Marriott International (2018)

An unpatched web application vulnerability in the Starwood reservation system exposed 500 million guest records over four years, resulting in a £18.4 million GDPR fine.

These high-profile breaches demonstrate how web application vulnerabilities can lead to catastrophic data exposure, financial losses, and reputational damage. In each case, proper penetration testing might have identified the vulnerabilities before they could be exploited by malicious actors.

# How Web Apps Work: A Simplified View



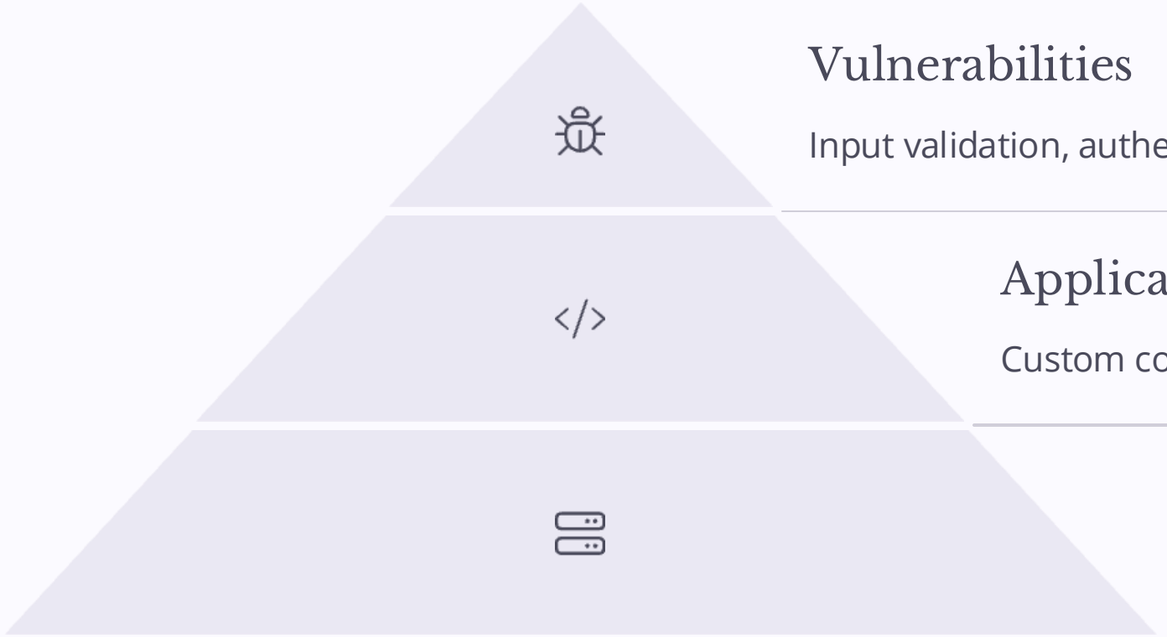
## Client-Server Interaction

Web applications operate on a client-server model. The client (usually a web browser) sends requests to the server, which processes these requests and sends back the appropriate responses.

## Request-Response Cycle

1. The client initiates a request by entering a URL or interacting with a web page.
2. The request is sent over the internet to the server.
3. The server receives the request and processes it, often involving database queries or other operations.
4. The server formulates a response, typically in HTML, JSON, or other formats.
5. The response is sent back to the client.
6. The client's browser renders the response, displaying the web page or data to the user.

# Understanding the Web Application Attack Surface



## Vulnerabilities

Input validation, authentication, and access control flaws

## Application Layer

Custom code, frameworks, APIs, and business logic

## Infrastructure

Web servers, databases, and cloud services

Web applications present a complex attack surface that spans multiple layers of technology. Unlike traditional network applications, modern web apps involve client-side processing, APIs, microservices, and cloud infrastructure—all potential entry points for attackers.

The complexity is further increased by frequent updates and the integration of third-party components, which can introduce new vulnerabilities with each deployment. This multi-layered architecture makes comprehensive testing essential but challenging.

# OWASP Top 10: The Most Critical Web Application Risks

### Broken Access Control

Restrictions on authenticated users are not properly enforced, allowing attackers to access unauthorized functionality or data.

### Cryptographic Failures

Failures related to cryptography that often lead to sensitive data exposure through inadequate encryption.

### Injection

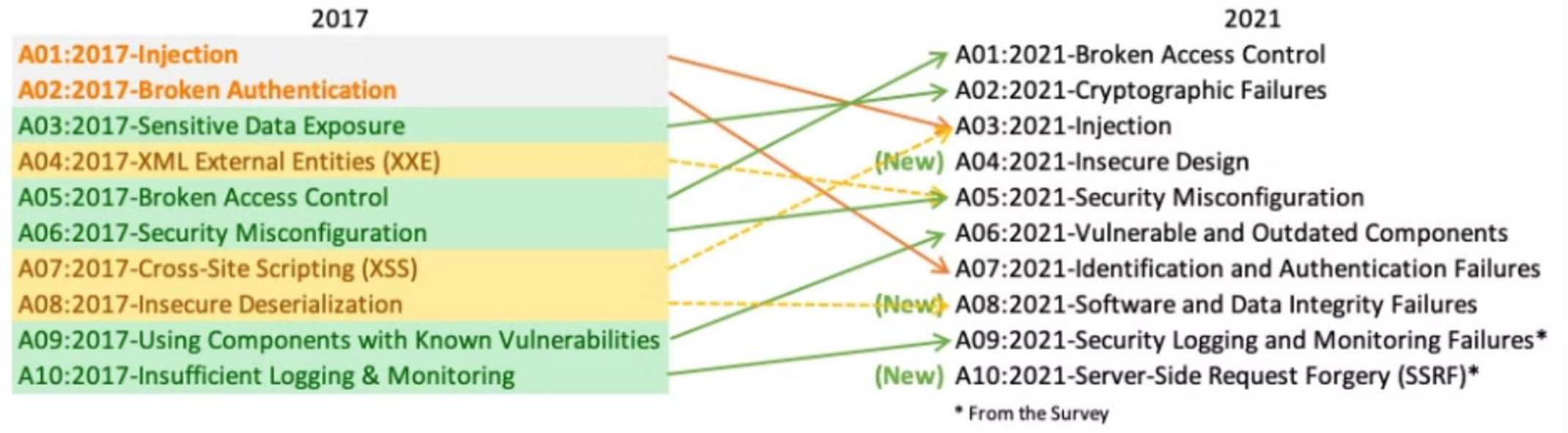
User-supplied data is not validated, filtered, or sanitized, enabling SQL, NoSQL, OS, and LDAP injection attacks.

### Insecure Design

Flaws in the design and architecture of applications that require secure design patterns and principles.

The OWASP Top 10 represents the most critical security risks to web applications based on frequency, exploitability, detectability, and potential impact. Understanding these vulnerabilities is essential for effective penetration testing, as they provide a framework for prioritizing your assessment efforts.

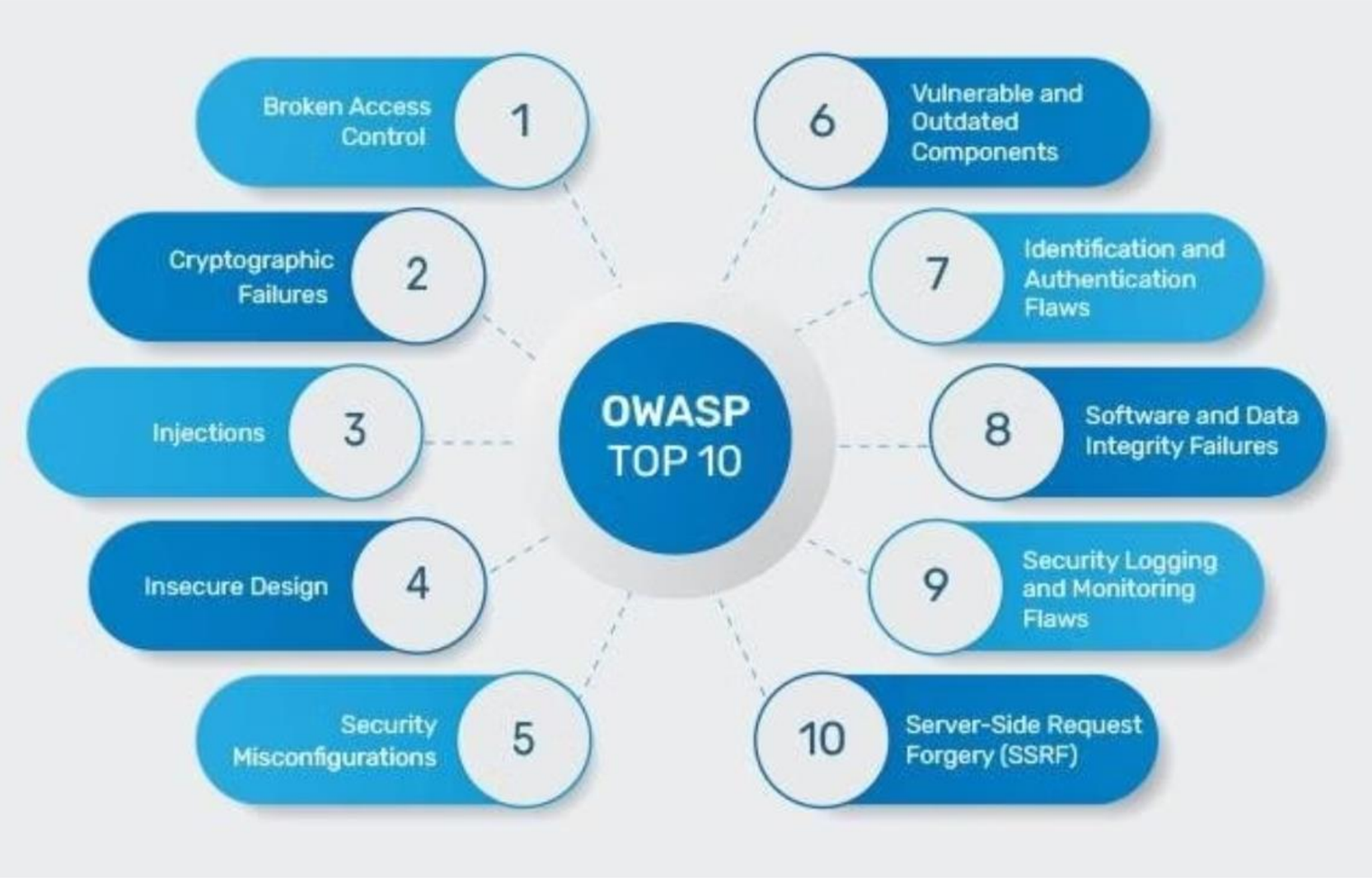
# OWASP Top 10 Vulnerabilities



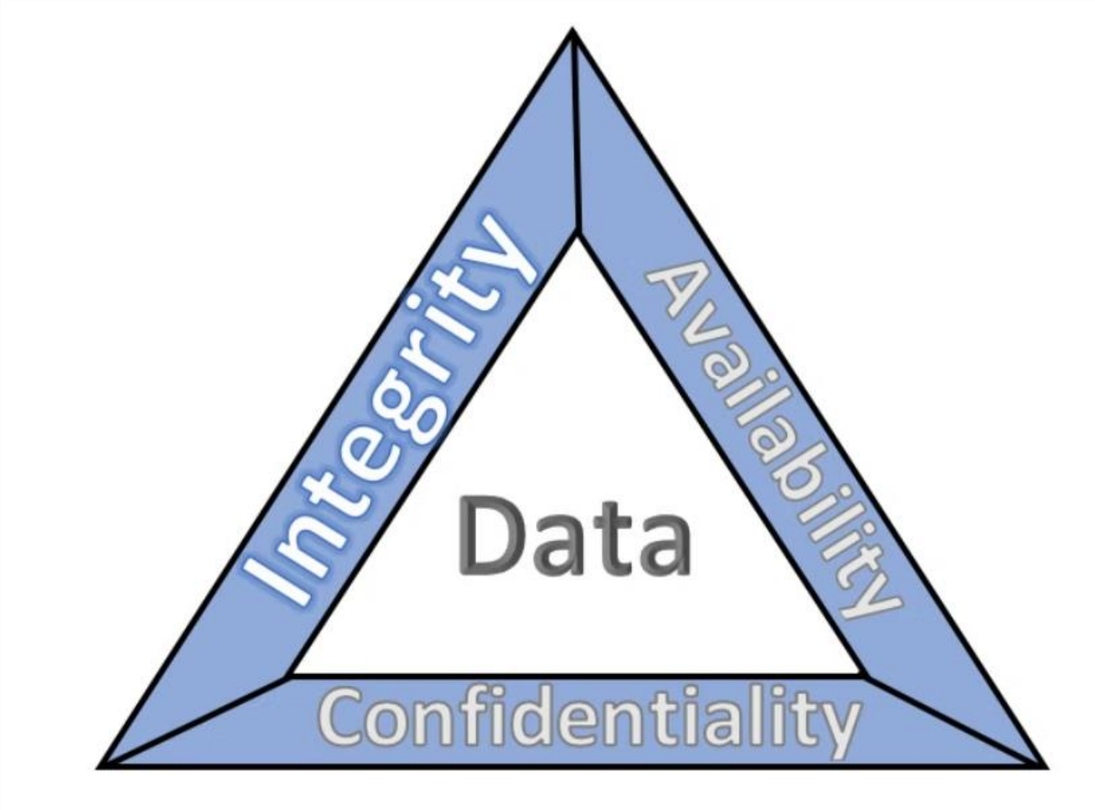

https://owasp.org/

https://owasp.org/Top10/

# OWASP Top 10 2021



# CIA Triad




Executive Summary — NIST SP 1800-26  documentation

Extended Pillars:

- Authenticity and
- Non-repudiation

The Five Pillars of Information Security: CIA Triad and More

# Encryption Fundamentals

- **Definition:** Encryption is the process of transforming information (plaintext) into an unreadable format (ciphertext) to protect its confidentiality.
- **Key Concepts:**
  - **Plaintext:** The original, readable data.
  - **Ciphertext:** The encrypted, unreadable data.
  - **Key:** A secret value used by an encryption algorithm.
  - **Encryption Algorithm:** A mathematical process for encryption and decryption.
  - **Decryption:** The process of converting ciphertext back into plaintext.
- **Importance:** Encryption is essential for protecting sensitive data in web applications, including passwords, user data, financial transactions, and more.
- **Types of Encryption:**
  - Symmetric Encryption
  - Asymmetric Encryption

# Encryption Types

- **Symmetric Encryption:**
  - **Description:** Uses the same key for both encryption and decryption.
  - **Example:** AES (Advanced Encryption Standard)
  - **Advantages:** Fast and efficient.
  - **Disadvantages:** Key distribution can be complex and requires a secure channel.
- **Asymmetric Encryption:**
  - **Description:** Uses a pair of keys: a public key for encryption and a private key for decryption.
  - **Examples:** RSA (Rivest–Shamir–Adleman), ECC (Elliptic Curve Cryptography)
  - **Advantages:** Enables secure key exchange and digital signatures.
  - **Disadvantages:** Slower than symmetric encryption.
- **Encryption in Web Applications:**
  - **In Transit:** HTTPS/TLS encrypts data transmitted between web browsers and servers.
  - **At Rest:** Database encryption protects stored data; file encryption secures uploaded files.
  - **Hashing:** A one-way function used to store passwords securely.

# Encryption, Hashing and Encoding

| Feature | Encryption | Hashing | Encoding |
|---|---|---|---|
| **Purpose** | Protect data confidentiality | Ensure data integrity | Convert data into a readable format |
| **Reversible** | Yes (with the key) | No (one-way function) | Yes (decoding restores original data) |
| **Use Case** | Secure data transfer (e.g., messages, files) | Password storage, file checksums | Data transmission (e.g., Base64 in emails) |
| **Key Required** | Yes (for encryption and decryption) | No | No |
| **Examples** | AES, RSA, DES | SHA-256, SHA-1, MD5 | Base64, ASCII, URL encoding |
| **Security Focus** | Confidentiality | Integrity and verification | Readability and transport |
| **Output Format** | Appears random | Fixed-length hash | Readable format (e.g., text, URL-safe) |

We will revisit all these after some time with more details

# Essential Tools for Web Application Penetration Testing

## Burp Suite

- Intercept and modify HTTP/S traffic
- Automated vulnerability scanning
- Session analysis and manipulation
- Brute-force password attacks

## OWASP ZAP

- Open-source alternative to Burp
- Active and passive scanning
- Spider functionality for discovery
- Automated exploitation tools

## Specialized Tools

- SQLMap for SQL injection testing
- Nikto for web server scanning
- WPScan for WordPress vulnerabilities
- Nuclei for template-based scanning

Mastering these tools is crucial for effective web application penetration testing. While automated tools can identify many common vulnerabilities, the most successful penetration testers combine tool usage with manual testing techniques for comprehensive coverage.

Understanding how each tool works "under the hood" allows you to customize and extend their capabilities for your specific testing scenarios.

Download Burp Suite Community Edition - PortSwigger

# Setting Up Your Penetration Testing Environment

## 1 Install Docker

Set up Docker

## 2 Deploy Vulnerable Applications

Install deliberately vulnerable applications like OWASP's Damn Vulnerable Web App (DVWA), Juice Shop, or WebGoat to practice exploitation safely.

A properly configured testing environment is essential for both learning and conducting professional penetration tests. The isolation protects production systems while giving you freedom to experiment with techniques that would be dangerous in a live environment.

# DVWA

https://github.com/digininja/DVWA?tab=readme-ov-file#download

GitHub - digininja/DVWA: Damn Vulnerable Web Application (DVWA)

```
shesha@Sheshananadas-MacBook-Pro DVWA % pwd
/Users/shesha/Desktop/BSidesCharm/demoapps/DVWA
shesha@Sheshananadas-MacBook-Pro DVWA % docker version
Client:
 Version:           27.3.1
......
shesha@Sheshananadas-MacBook-Pro DVWA % docker compose version
Docker Compose version v2.29.7-desktop.1
shesha@Sheshananadas-MacBook-Pro DVWA %
```

```
shesha@Sheshananadas-MacBook-Pro DVWA % docker compose version
Docker Compose version v2.29.7-desktop.1
shesha@Sheshananadas-MacBook-Pro DVWA % docker compose up -d
[+] Running 20/20
 ✔ dvwa Pulled                                                                                                                                14.2s
   ✔ b9e2f83d4e30 Download complete                                                                                                            0.3s
   ✔ 19883a5b6404 Download complete                                                                                                            3.5s
   ✔ d49a5d4d78e5 Download complete                                                                                                            0.1s
   ✔ 99e83134a031 Download complete                                                                                                            5.1s
   ✔ 9be9496ea2d8 Download complete                                                                                                            1.8s
   ✔ e78f4a695473 Download complete                                                                                                            0.3s
   ✔ 4f4fb700ef54 Already exists                                                                                                               0.0s
   ✔ 56cf2d8e9e09 Download complete                                                                                                            3.1s
   ✔ dce5011ff25f Download complete                                                                                                            0.1s
   ✔ 64dbcd12588e Download complete                                                                                                            0.3s
   ✔ c569283c96bc Download complete                                                                                                            0.1s
   ✔ 51cb2ba39d1e Download complete                                                                                                            0.3s
   ✔ a843a84fd431 Download complete                                                                                                            0.3s
   ✔ 7d6f3098fbfc Download complete                                                                                                            0.1s
   ✔ 54df264c65c0 Download complete                                                                                                            8.0s
   ✔ 4799d5c848fe Download complete                                                                                                            0.7s
   ✔ dc325eebd405 Download complete                                                                                                            3.5s
   ✔ 298e1b158820 Download complete                                                                                                            4.4s
   ✔ f01400cc6ea2 Download complete                                                                                                            0.1s
[+] Running 3/3
 ✔ Container dvwa-db-1                                                                                                        Running        0.0s
 ✔ Container dvwa-dvwa-1                                                                                                      Started        1.5s
 ! dvwa The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested  0.0s
shesha@Sheshananadas-MacBook-Pro DVWA %
```

# JuiceShop

https://hub.docker.com/r/bkimminich/juice-shop

docker pull bkimminich/juice-shop

docker run --rm -p 3000:3000 bkimminich/juice-shop

http://localhost:3000

```
shesha@Sheshananadas-MacBook-Pro DVWA % docker pull bkimminich/juice-shop
Using default tag: latest
latest: Pulling from bkimminich/juice-shop
b56c1478af91: Download complete
53c19e94a2ee: Download complete
26cf43e0702c: Download complete
Digest: sha256:bdeabb57aa4e455d4ab38eff7e212193e1ce416d65410624071c0592e5994e87
Status: Downloaded newer image for bkimminich/juice-shop:latest
docker.io/bkimminich/juice-shop:latest
shesha@Sheshananadas-MacBook-Pro DVWA % docker run --rm -p 3000:3000 bkimminich/juice-shop
info: Detected Node.js version v20.17.0 (OK)
info: Detected OS linux (OK)
info: Detected CPU arm64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file index.html is present (OK)
info: Required file main.js is present (OK)
info: Required file styles.css is present (OK)
info: Required file tutorial.js is present (OK)
info: Required file polyfills.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file vendor.js is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Server listening on port 3000
```

# PortSwigger Academy Free Online

Web Security Academy: Free Online Training from PortSwigger

All Web Security Academy topics | Web Security Academy

# Browser Extensions for Web Application Testing - May be we can skip this

## Wappalyzer

Identifies technologies used in web applications, helping testers target known vulnerabilities in specific frameworks, CMS platforms, and server technologies.

## EditThisCookie

Allows direct manipulation of cookie values to test authentication and session management vulnerabilities, including privilege escalation and session fixation attacks.

## HackBar

Provides quick access to encoding/decoding functions, SQL injection payloads, and XSS testing capabilities directly from the browser interface.

## Postman

Facilitates testing of RESTful APIs with custom headers, authentication, and payload manipulation to identify vulnerabilities in backend services.

Browser extensions serve as lightweight alternatives to full-featured tools in specific testing scenarios. They excel at quick checks and can be particularly useful during the reconnaissance phase of a penetration test, where identifying technologies and basic configurations provides direction for more in-depth testing.

# Ethical and Legal Considerations in Penetration Testing

## Obtain Written Permission

Always secure documented authorization before testing any system

## Define Scope Boundaries

Clearly establish which systems and techniques are permitted

## Schedule Testing Windows

Coordinate timing to minimize business disruption

## Document Everything

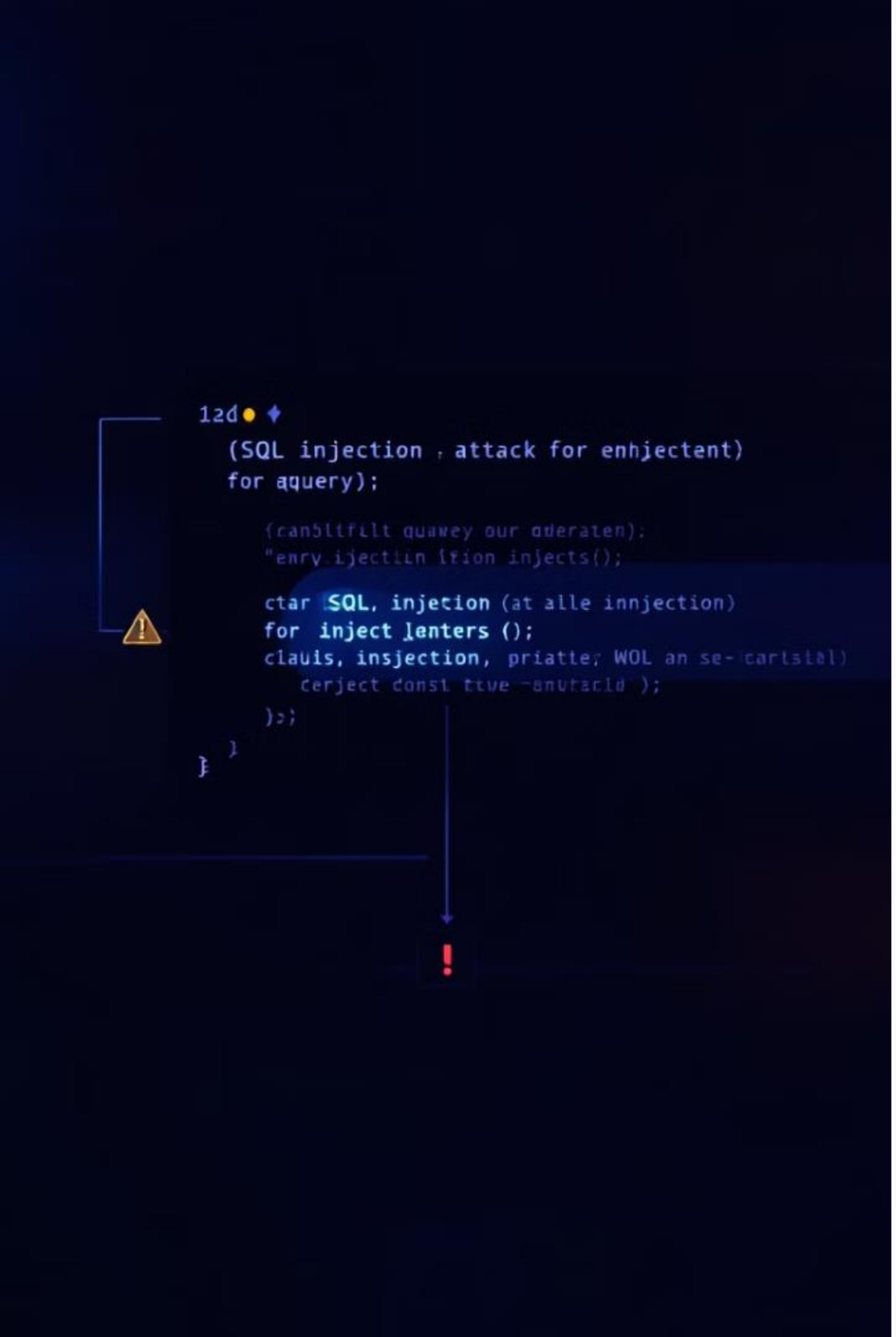
Maintain detailed records of all testing activities

## Handle Data Responsibly

Protect any sensitive information discovered during testing

The line between security testing and illegal hacking is defined by authorization, not technical methods. Without proper permission, even basic scanning can violate computer crime laws in most jurisdictions, potentially resulting in severe legal penalties.

Remember that penetration testing often reveals sensitive information about an organization's security posture. Maintaining confidentiality and following responsible disclosure protocols is essential for protecting both the client and yourself as a security professional.



# SQL Injection Fundamentals

## Definition

Code injection technique where attackers insert malicious SQL statements into entry fields. They manipulate backend databases to access unauthorized data.

## Common Targets

Login forms, search boxes, and URL parameters often accept unfiltered user input. Legacy applications are particularly vulnerable.

## Impact

Data theft, authentication bypass, and complete database compromise. Can lead to system takeover and regulatory violations.

# Identifying Injectable Parameters

## Error-Based Detection

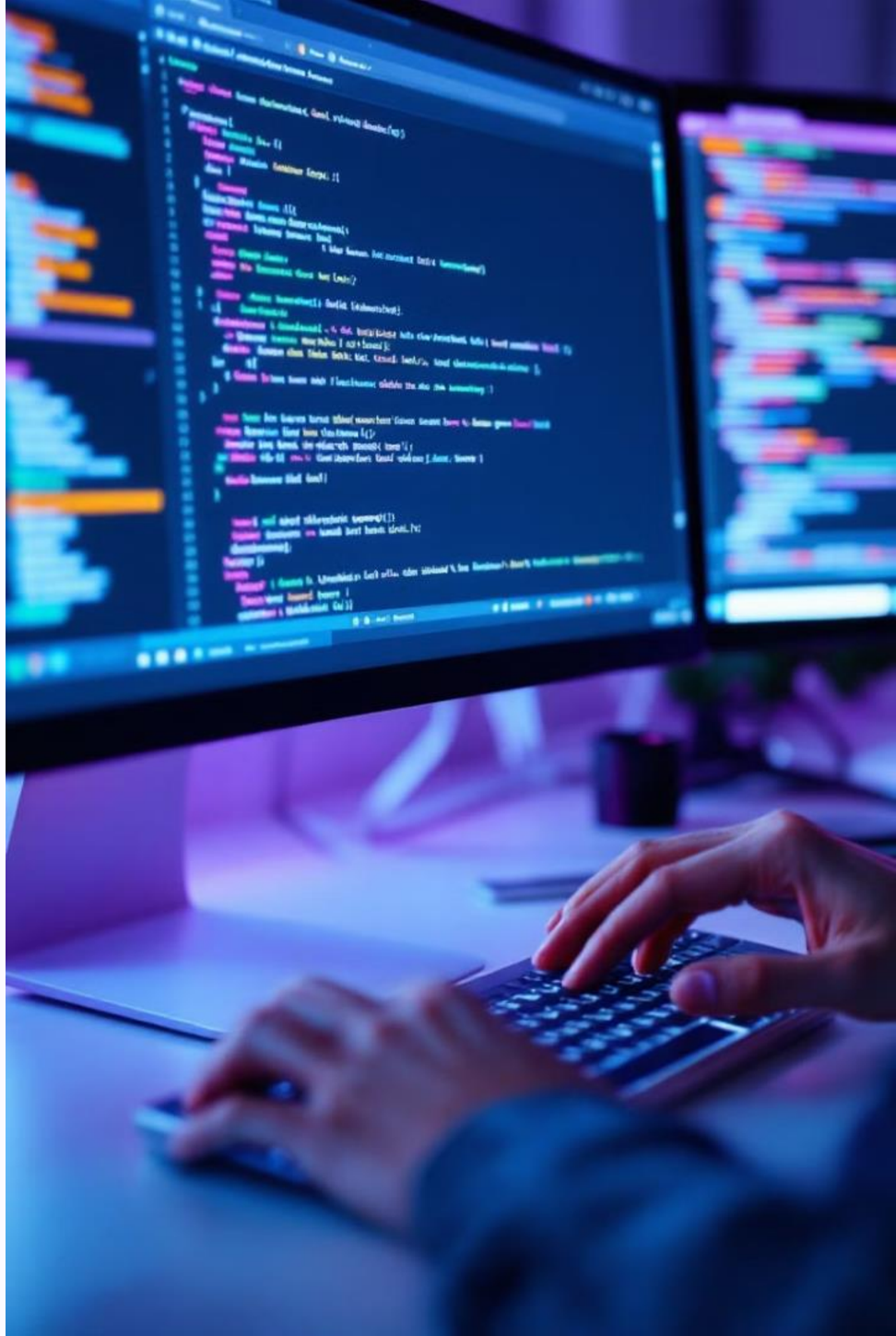
Insert special characters like quotes or semicolons. Database error messages often reveal injection points.

## Blind Testing

Use logical conditions that alter application behavior. Look for differences in responses.

## Automated Scanning

Tools like SQLMap and Burp Suite can identify vulnerable parameters. They test inputs systematically.



# Exploiting SQL Injection

## Reconnaissance

Determine database type and structure. Extract schema information to map the environment.

## Authentication Bypass

Use OR 1=1 conditions to trick applications into granting access. Target login forms.

## Data Extraction

Use UNION queries to combine results with legitimate queries. Extract data systematically.

## Advanced Exploitation

Execute system commands or access files through database functionality. Elevate the attack.

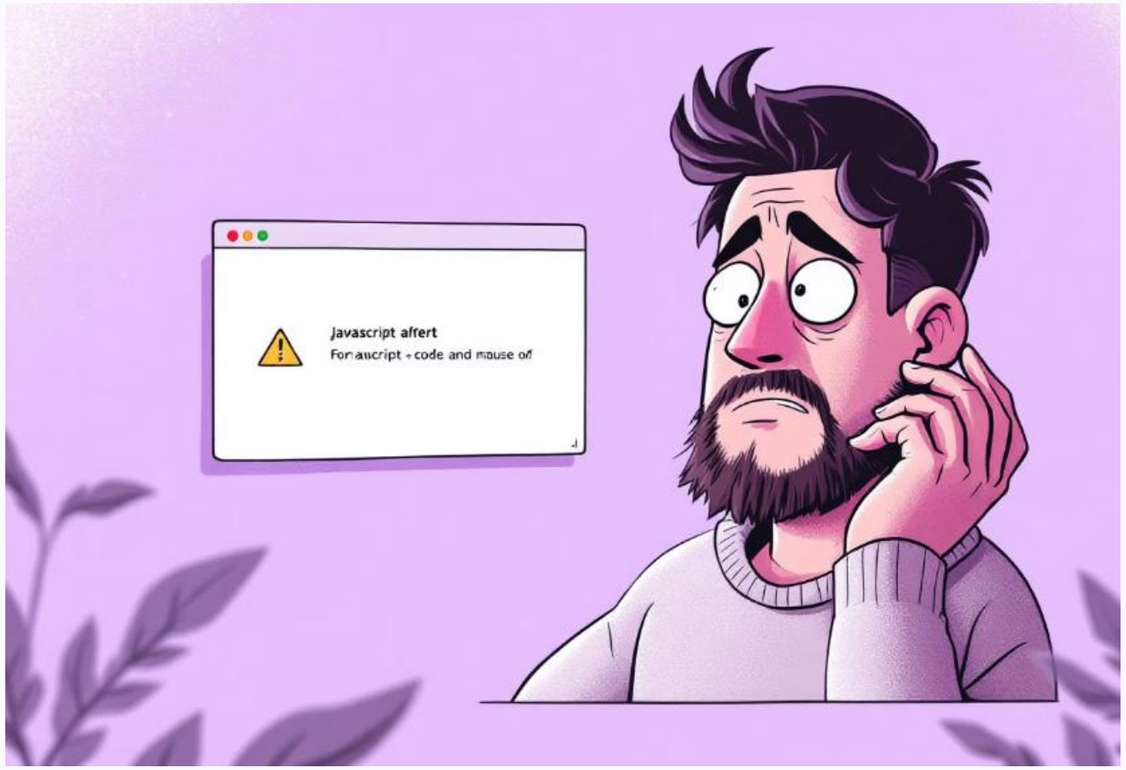
# Injection Attacks Deep Dive

## SQL Injection

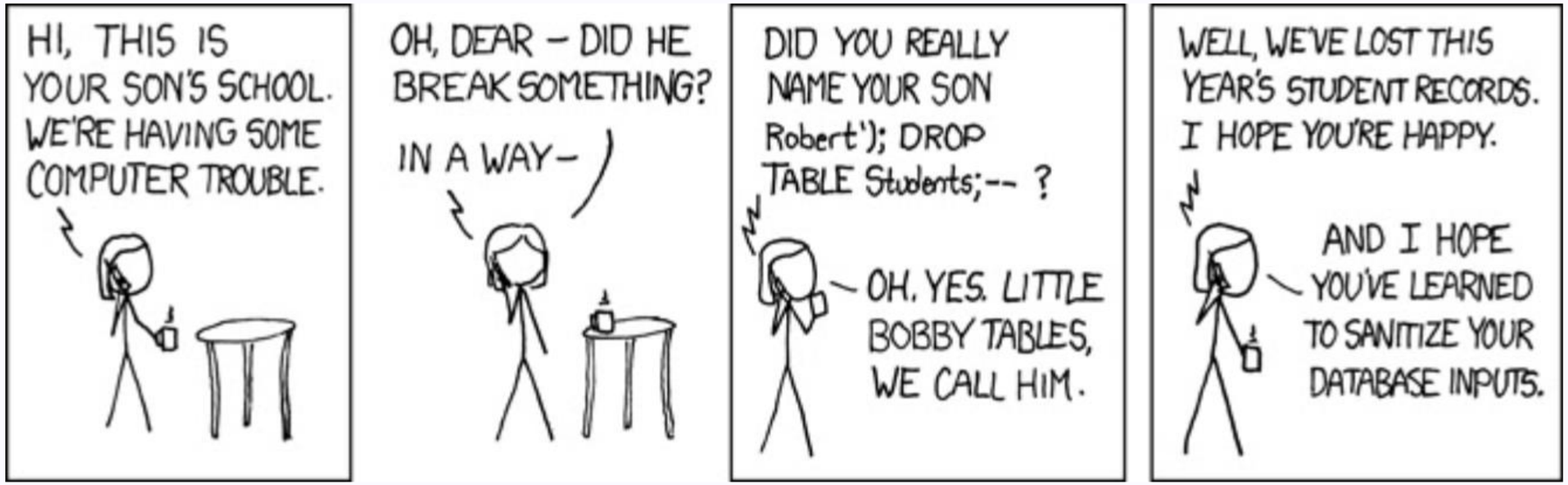
Attackers manipulate SQL queries to access or delete sensitive data. They can bypass login forms with simple code tricks.

## Cross-Site Scripting (XSS)

Malicious scripts injected into web pages execute in users' browsers. Attackers can steal session cookies and hijack accounts.



# Bobby Table



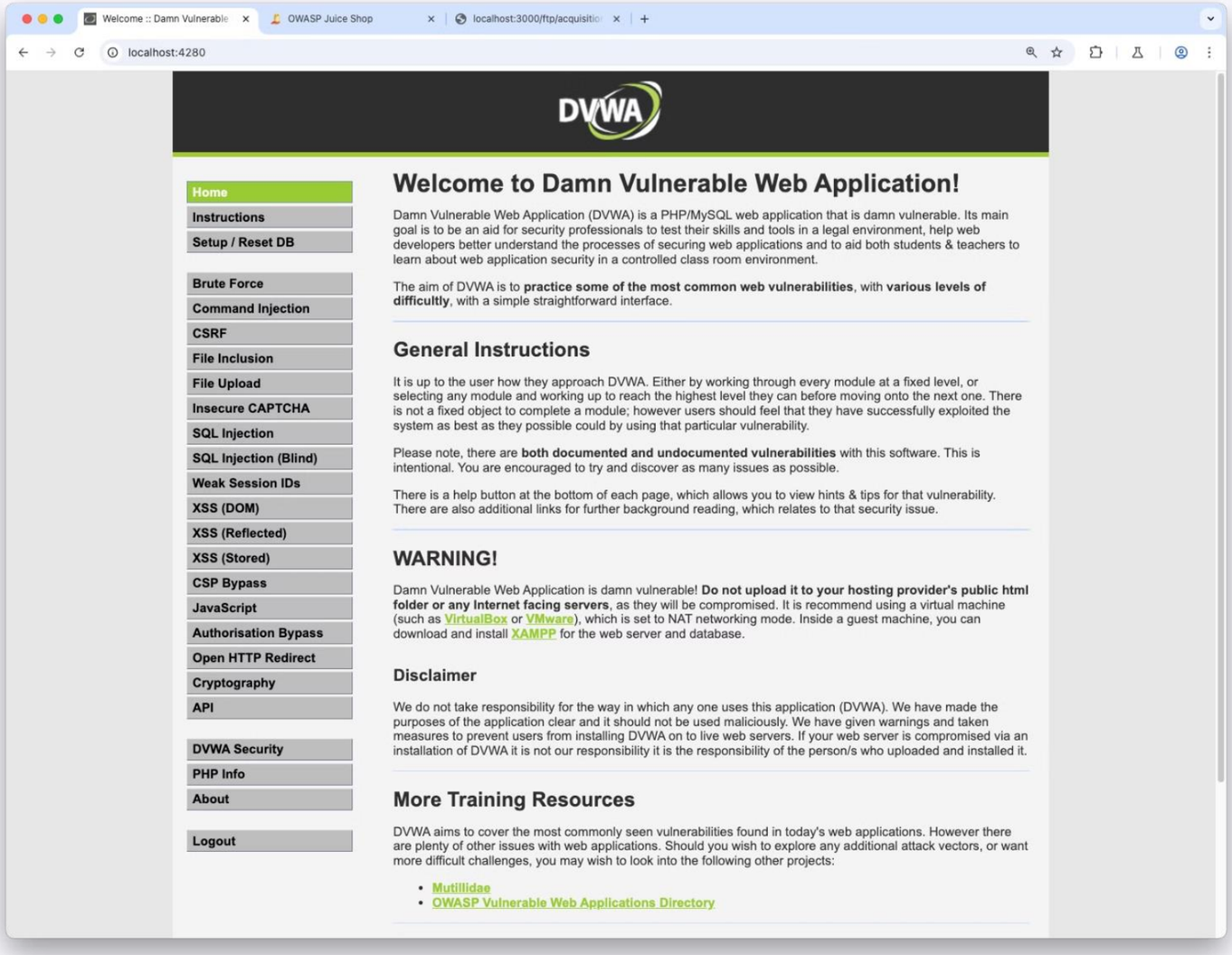

xkcd

# Demo

DVWA : Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. https://github.com/digininja/DVWA

http://localhost:4280/

1. SQL Injection
2. XSS (Cross Site Scripting)

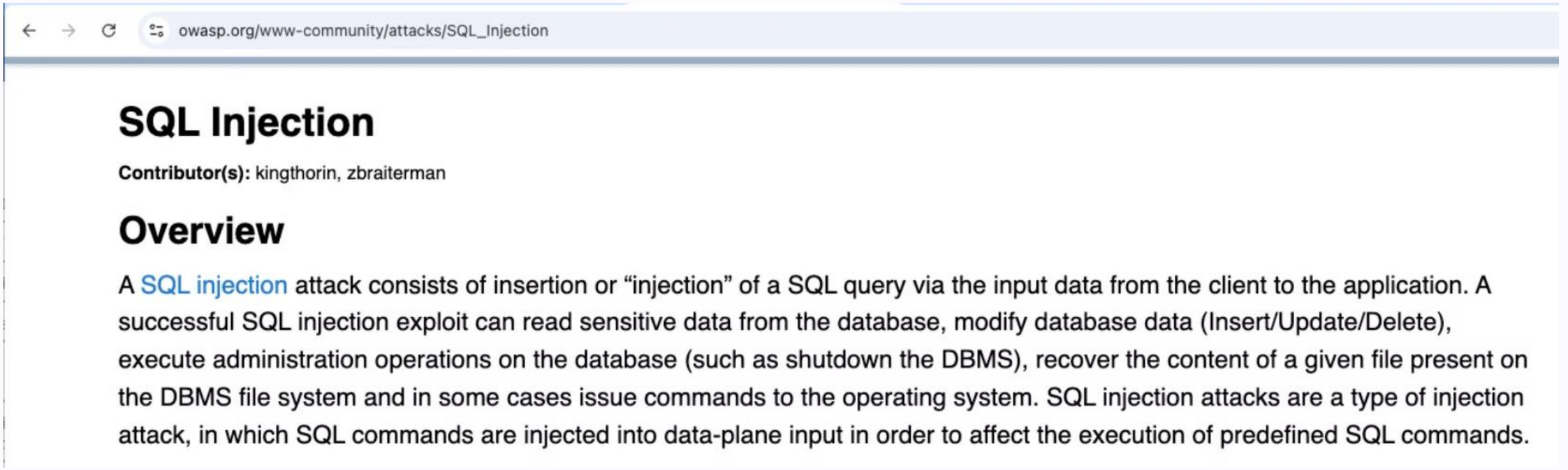


# SQL Injection Source

## SQL Injection Source

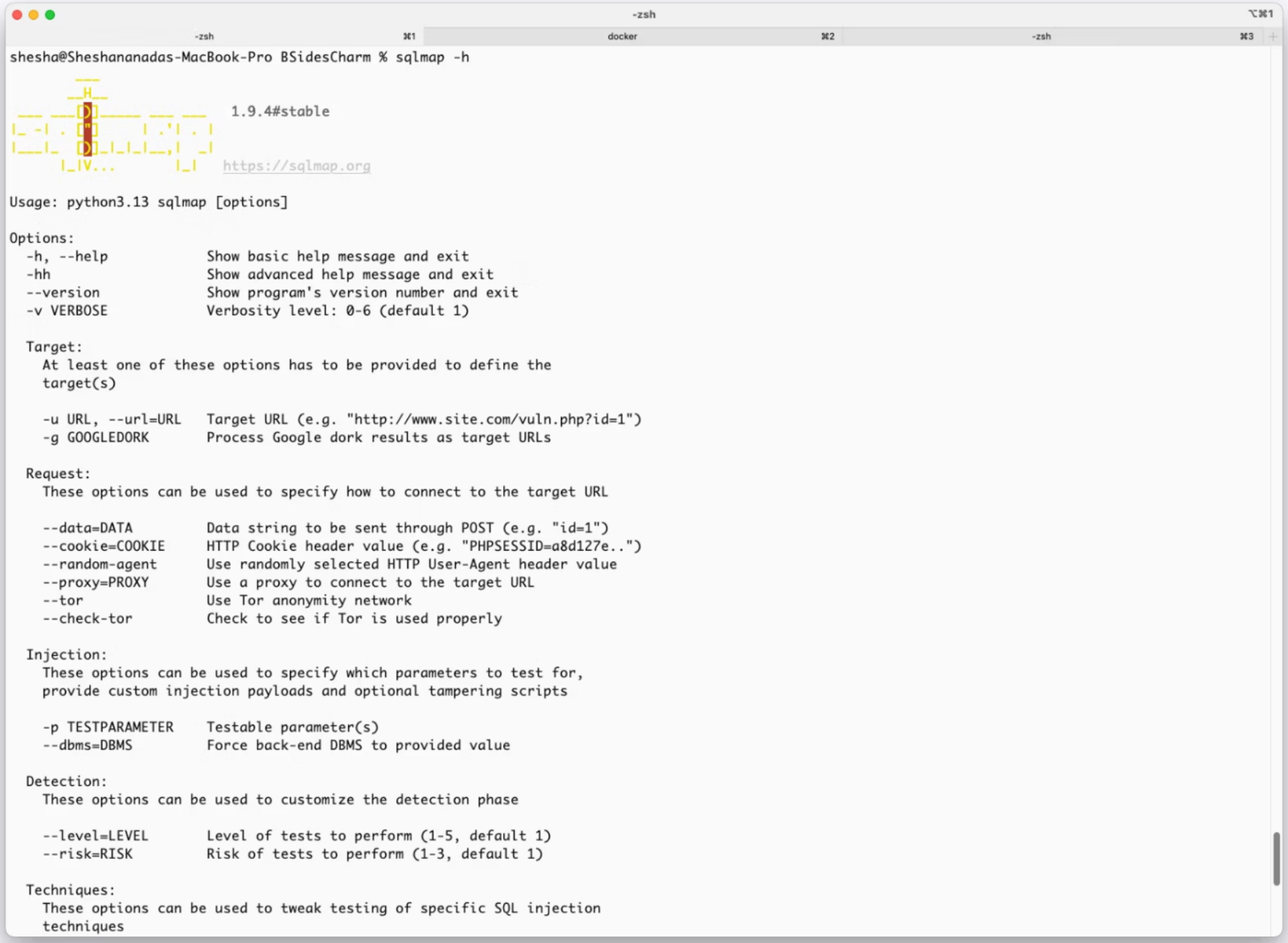
### vulnerabilities/sqli/source/low.php

```php
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check database
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```



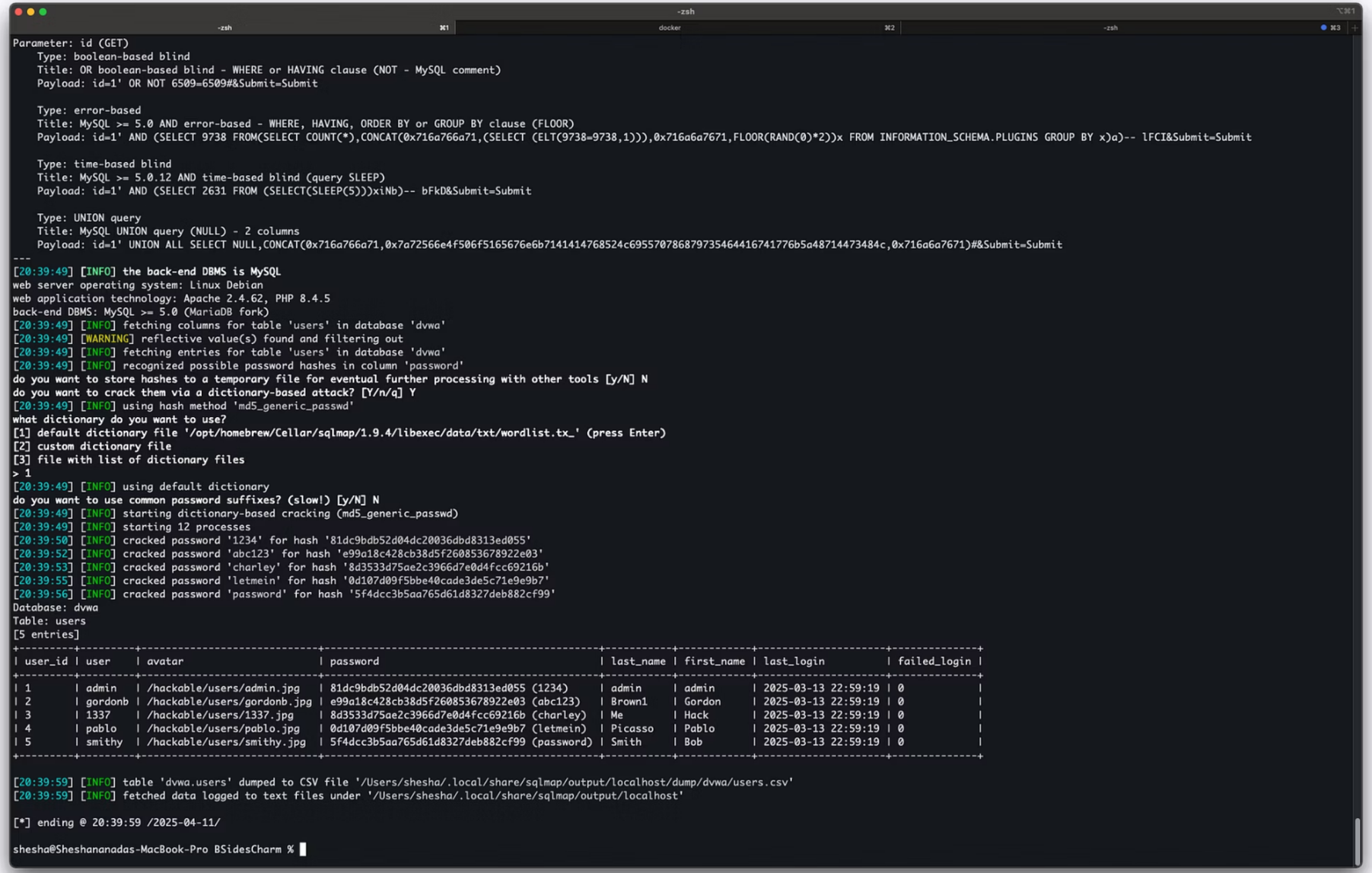
# SQLMap

- Automates the process of detecting and exploiting **SQL injection vulnerabilities**.
- Supports **various databases**: MySQL, PostgreSQL, Oracle, Microsoft SQL Server, SQLite, etc.



# SQLMap

sqlmap -r SQLi.txt --batch -D dvwa -T users --dump



# Portswigger Academy

SQL Injection:

Lab: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data | Web Security Academy

# Fixing Injection Attacks

## Identify Vulnerable Code

Locate string concatenation in database queries. Find unvalidated user inputs in forms.

## Implement Input Validation

Check input type, length, and format. Reject unexpected characters and patterns.

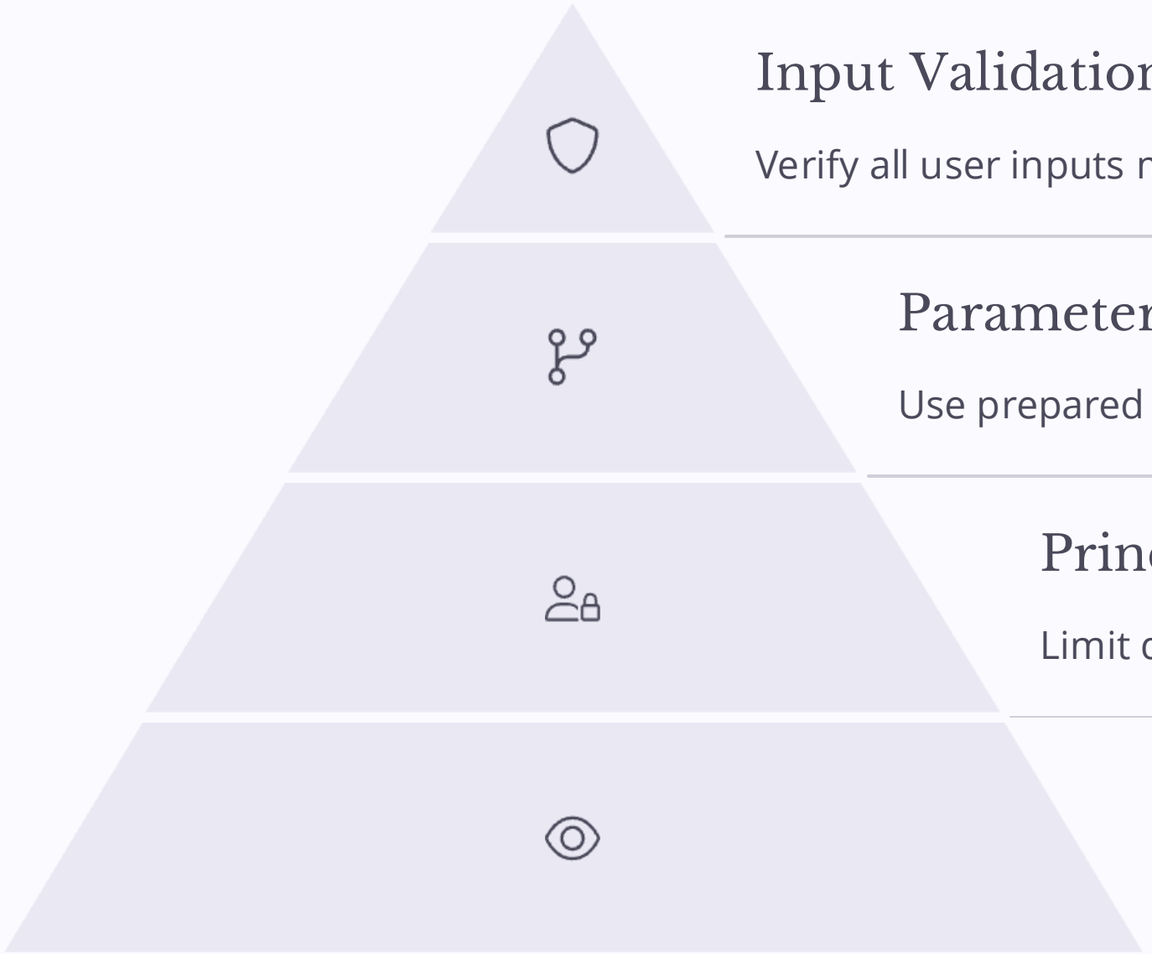
## Use Prepared Statements

Separate SQL logic from data. Let the database distinguish between code and input.

## Apply ORM Frameworks

Leverage frameworks that handle parameter escaping automatically. Avoid raw SQL queries.

# SQL Injection: Practical Defense



## Input Validation

Verify all user inputs match expected patterns

## Parameterized Queries

Use prepared statements to separate SQL from data

## Principle of Least Privilege

Limit database account permissions strictly

## Regular Code Reviews

Audit database interactions systematically

# OS Command Injection

## What is It?

- A **security vulnerability** that allows an attacker to execute **arbitrary system-level commands** on the host operating system.
- Happens when **untrusted input** is passed to a system shell or command interpreter.

## How It Works

- Web app takes user input (e.g., a username or file name).
- Passes it directly to a system command like ping, ls, or cat.
- Attacker injects extra commands using ;, &&, |, etc.

http://localhost:4280/vulnerabilities/exec/#

# XSS (Cross Site Scripting)

## What is XSS?

- A **web security vulnerability** that allows attackers to inject **malicious scripts** into trusted websites.
- These scripts run in the **browser of unsuspecting users**.

## How It Works

- Attacker injects script (e.g., ) into a form or URL.
- Website reflects this input without proper sanitization.
- Script executes in the victim’s browser as if it were trusted content.

## Attack Goals

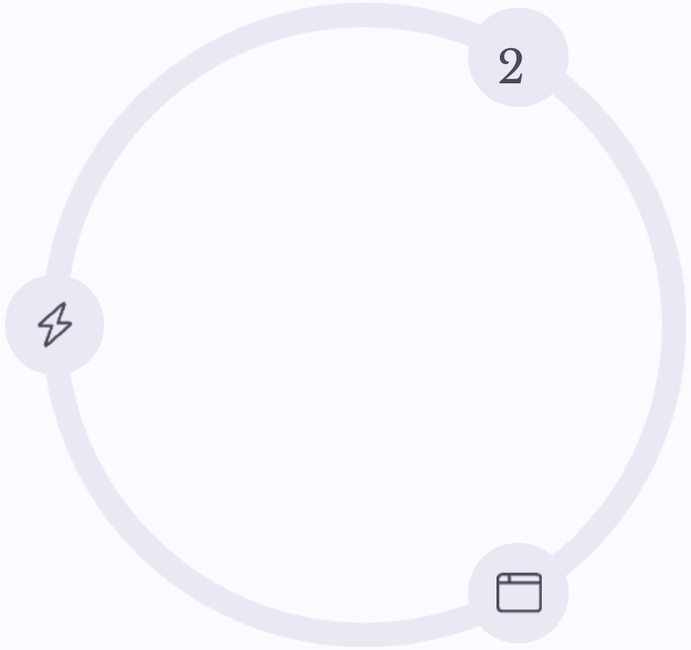
- Steal cookies, sessions, or user data.
- Perform actions on behalf of the user.
- Deface or redirect webpages.

# Cross-Site Scripting (XSS) Types

## Reflected XSS

Non-persistent attacks where malicious script reflects off web server. Typically delivered via URLs.

- Search results
- Error messages
- URL parameters




## Stored XSS

Persistent attacks where script is saved on the server. Affects all visitors to the page.

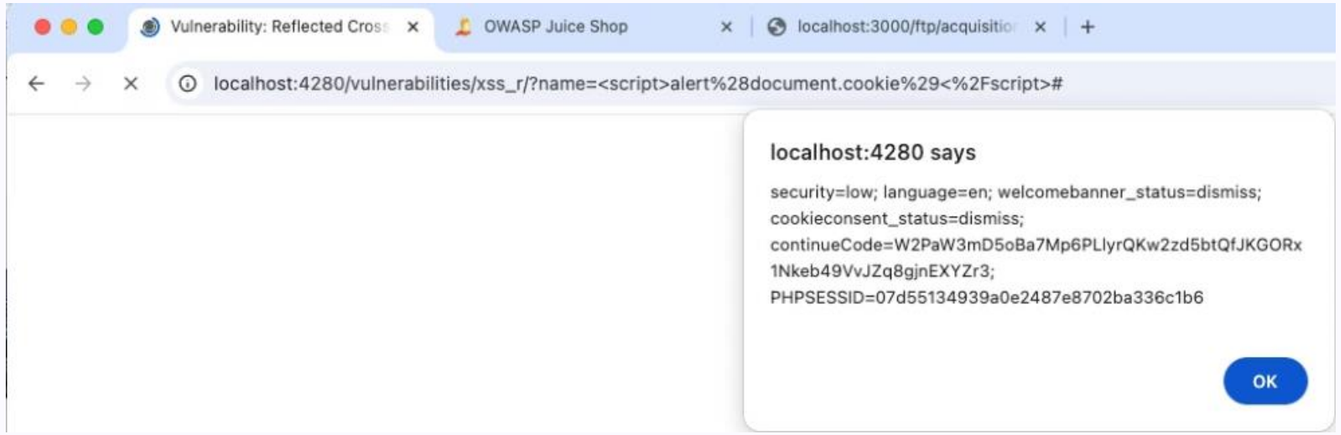
- Comments
- User profiles
- Forum posts

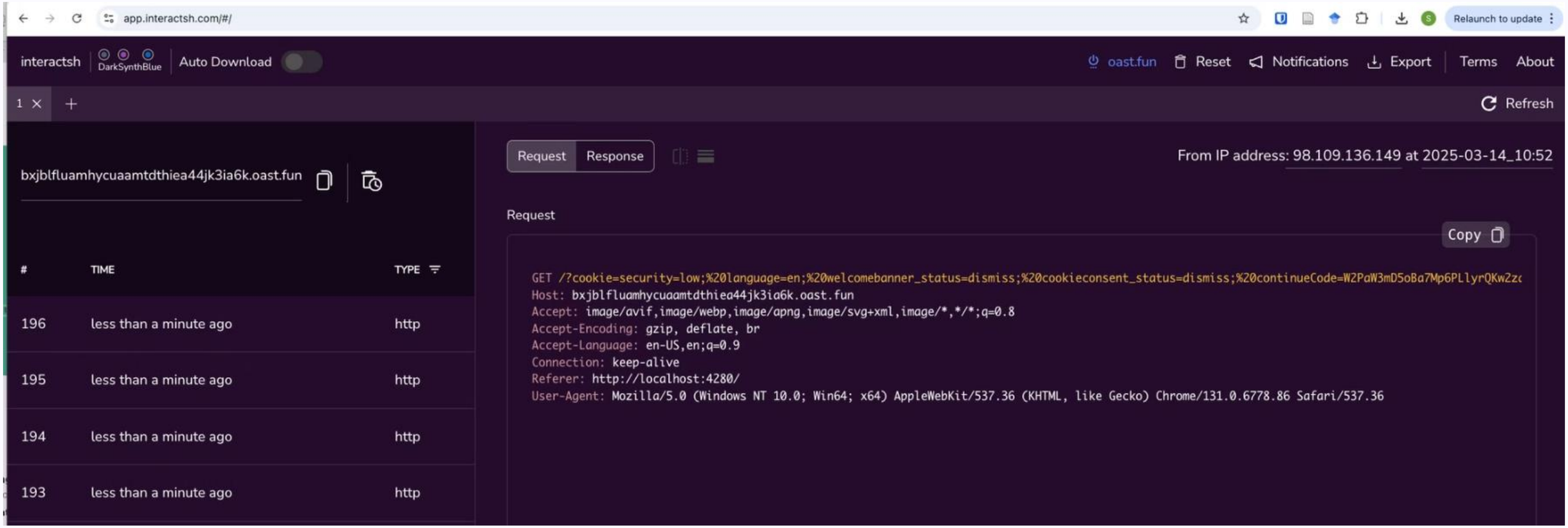
## DOM-based XSS

Client-side vulnerability where malicious code executes in browser DOM. No server interaction required.

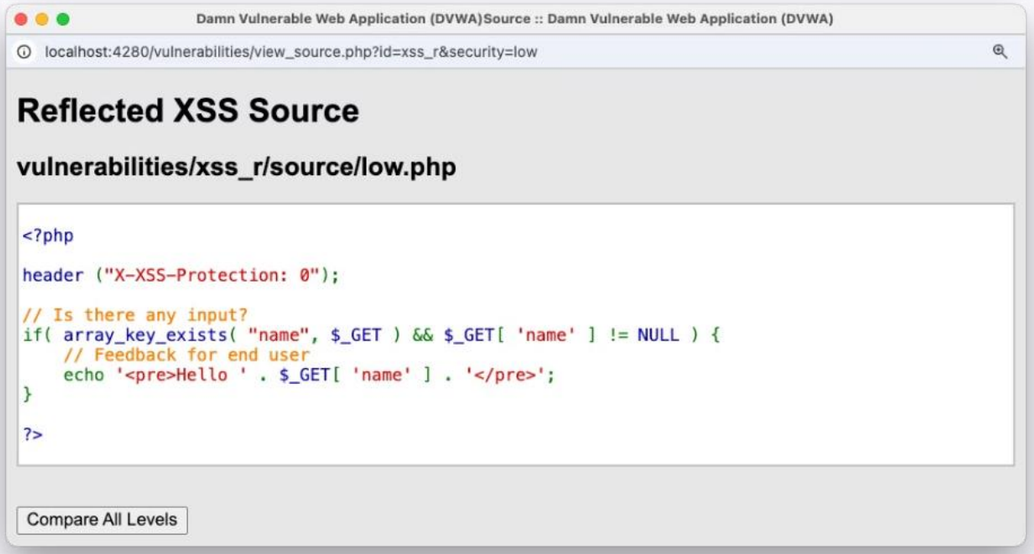
- Client-side routing
- JavaScript frameworks

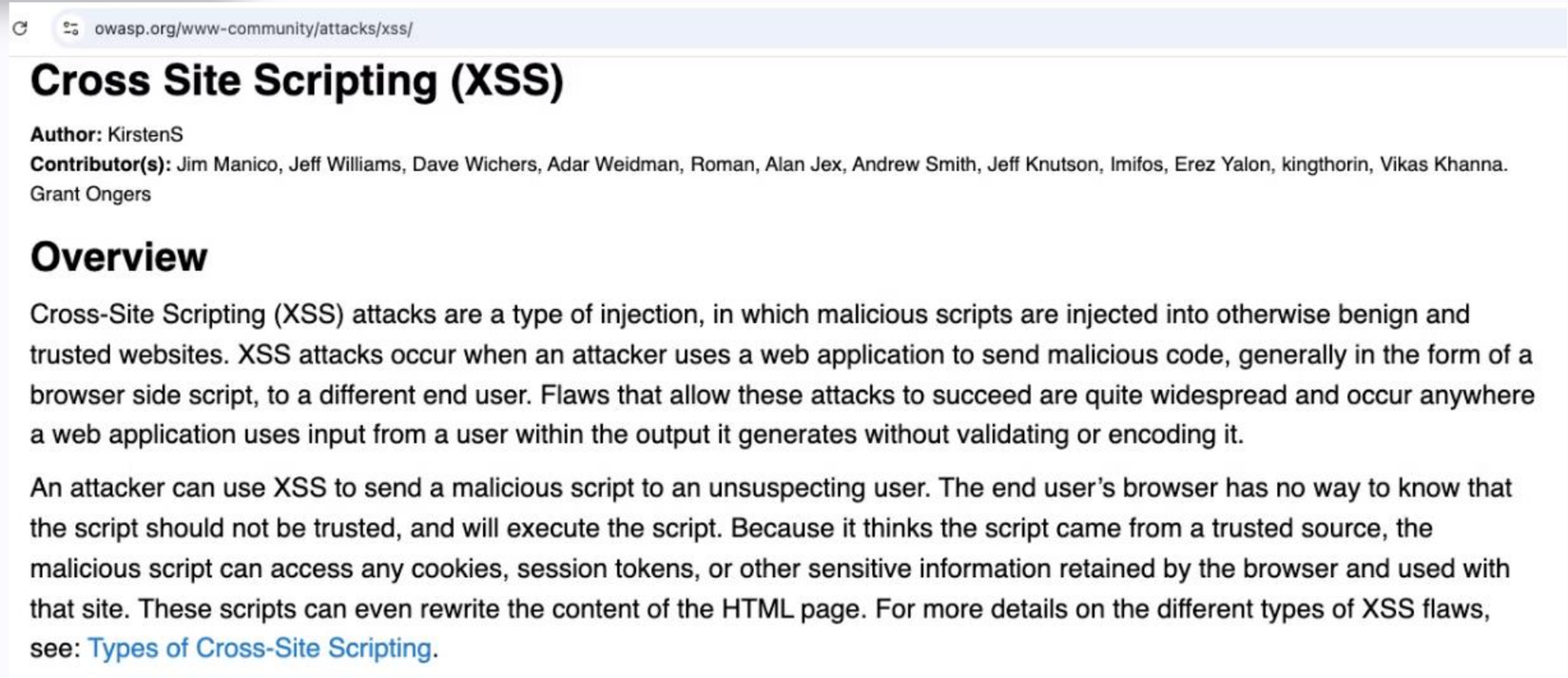
# XSS (Cross Site Scripting) Attack

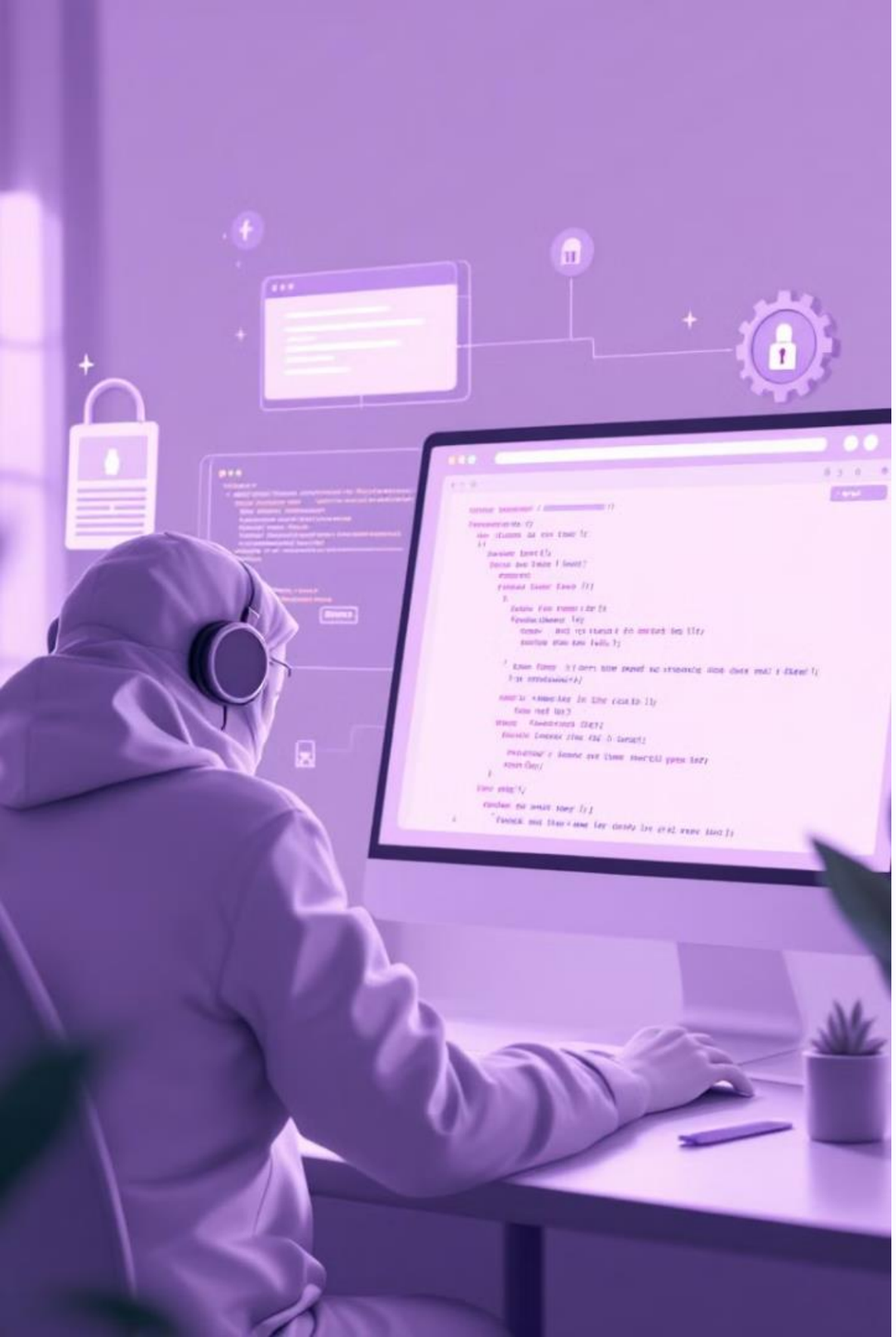


# XSS Source


Damn Vulnerable Web Application (DVWA)Source :: Damn Vulnerable Web Application (DVWA)
localhost:4280/vulnerabilities/view_source.php?id=xss_r&security=low
Reflected XSS Source
vulnerabilities/xss_r/source/low.php
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
// Feedback for end user
echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
?>
Compare All Levels


owasp.org/www-community/attacks/xss/
Cross Site Scripting (XSS)
Author: KirstenS
Contributor(s): Jim Manico, Jeff Williams, Dave Wichers, Adar Weidman, Roman, Alan Jex, Andrew Smith, Jeff Knutson, lmifos, Erez Yalon, kingthorin, Vikas Khanna. Grant Ongers
Overview
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.
An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page. For more details on the different types of XSS flaws, see: Types of Cross-Site Scripting.



# XSS Identification & Exploitation

## Input Testing

Insert simple JavaScript payloads like alert(1) in all user inputs. Check if they execute in the browser.

## Context Analysis

Determine how user input is processed. Different encoding bypasses are needed for HTML, JavaScript, and attribute contexts.

## Payload Creation

Craft context-specific payloads that evade filters. Use HTML entities, encoding tricks, or event handlers.

## Impact Demonstration

Show cookie theft, session hijacking, or phishing capabilities. Document the full attack chain.

# Defending Against XSS Attacks

## 3

### Key Defense Layers

Input validation, output encoding, and content security policy

## 60%

### Vulnerability Reduction

Modern frameworks provide automatic XSS protection

## 100%

### Testing Coverage

All user inputs must undergo security validation

Implement Content Security Policy headers. Use modern frameworks with built-in protections. Validate all inputs and sanitize outputs.

# First Few Milli Seconds of HTTPS

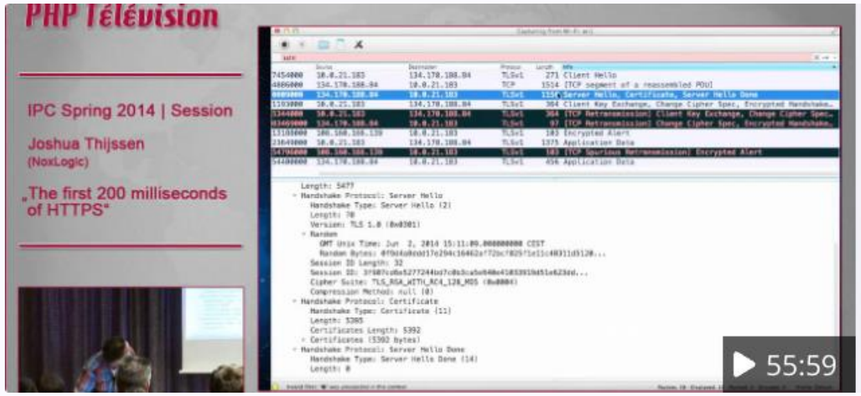
Information Security Stack Exchange

**The First Few Milliseconds of an HTTPS Connection [TLS 1.2 / TLS_ECH...**

In his blog post, 'The First Few Milliseconds of an HTTPS Connection', Jeff Moser does a wonderful job of walking through the TLS/SSL handshake process, and explaining...

[MIT CSAIL on Twitter / X]

[The First Few Milliseconds of an HTTPS Connection]



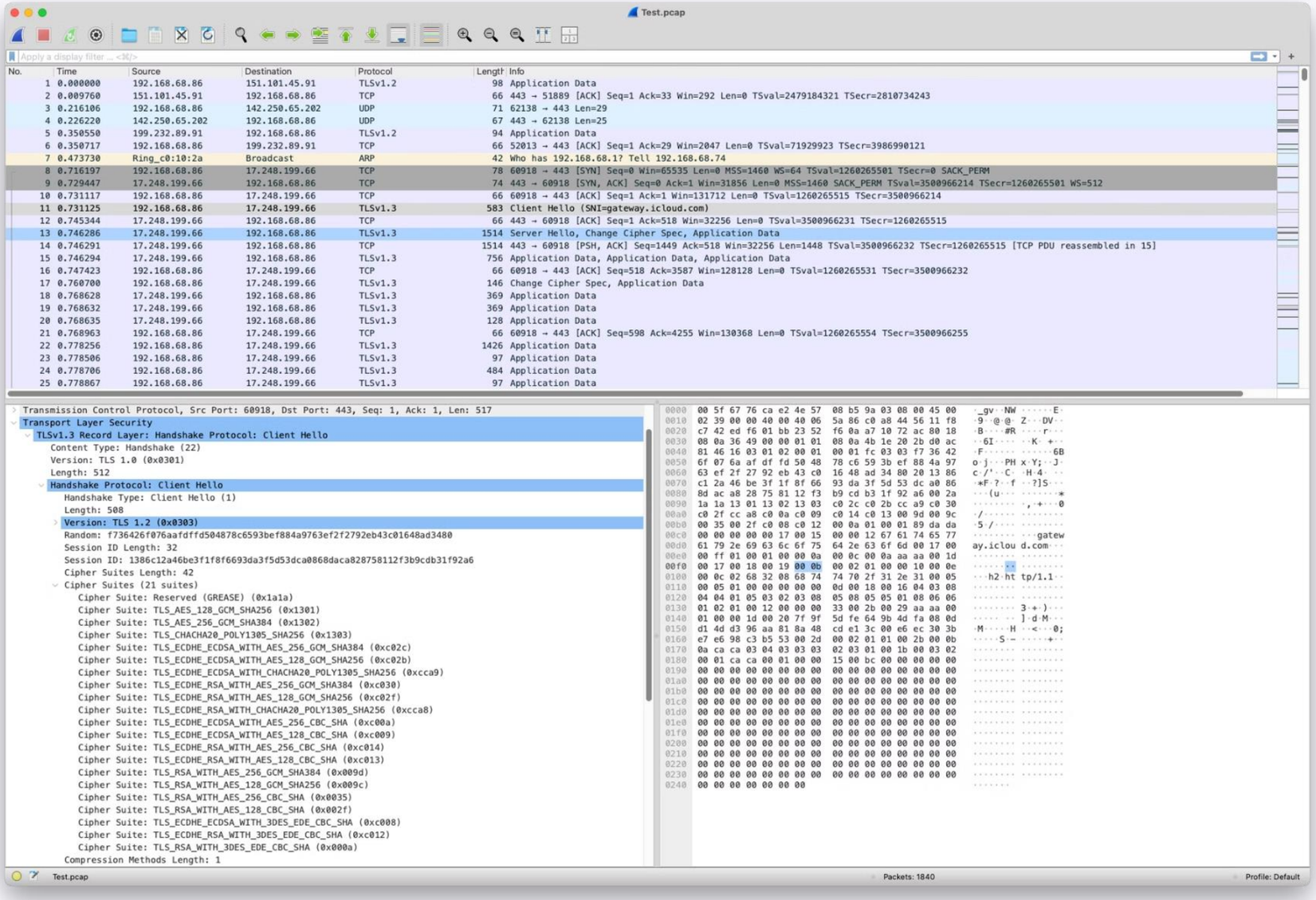

YouTube

**The first 200 milliseconds of HTTPS - Joshua Thijssen | IPC14**

What happens when your browser connects to a HTTPS secure site? We all know it has to do something with certificates, blue and green address bars and sometimes...

# Wireshark

/Users/shesha/Desktop/BSidesCharm/Test.pcap



# SSH - Secure Shell

- **Definition:**
  - SSH (Secure Shell) is a cryptographic network protocol for operating network services securely over an unsecured computer network.
  - It provides a secure channel over an insecure network by encrypting all the traffic.
- **Primary Uses:**
  - Remote command-line login
  - Remote command execution
  - Secure file transfer
  - Port Forwarding (tunneling)
- **Key Security Features:**
  - **Encryption:** Protects data confidentiality.
  - **Authentication:** Verifies the identity of the server and (optionally) the client.
  - **Integrity:** Ensures that data is not modified in transit.

# How SSH Works

- **Key Exchange:**
  - SSH uses algorithms like Diffie-Hellman to securely establish a shared secret key between the client and the server.
  - This key is then used to encrypt the session.
- **Server Authentication:**
  - The server proves its identity to the client, typically using public-key cryptography.
  - The client may have a "known_hosts" file to store trusted server keys.
- **Client Authentication (Two Main Methods):**
  - **Password-based:** The client provides a username and password (less secure).
  - **Public-key authentication:**
    - More secure method.
    - Involves generating a key pair (public and private key).
    - The public key is placed on the server, and the client uses its private key to authenticate.
- **Simplified Handshake:**
  1. Key exchange to establish a shared secret.
  2. Server authentication.
  3. Client authentication (password or public key).
  4. Encrypted communication.

EP124: How does SSH work?

How To Use SSH to Connect to a Remote Server | DigitalOcean

# Password Hashing

**What's the Problem?**

- Storing plain-text passwords is **dangerous**.
- If a database is breached, every password is **instantly exposed**.

**What's the Solution?**

- **Hash** passwords before storing them.
- A **hash** is a one-way cryptographic function that turns a password into a fixed-length string.

**Hashing Basics**

| Property | Description |
| --- | --- |
| One-Way | Cannot reverse a hash back to password |
| Salted | Add a random value (salt) to each password before hashing |
| Deterministic | Same input always gives same output (without salt) |
| Fixed Length | Output is always the same size |

# Best Practices for Password Storage

## Use Secure Hashing Algorithms:

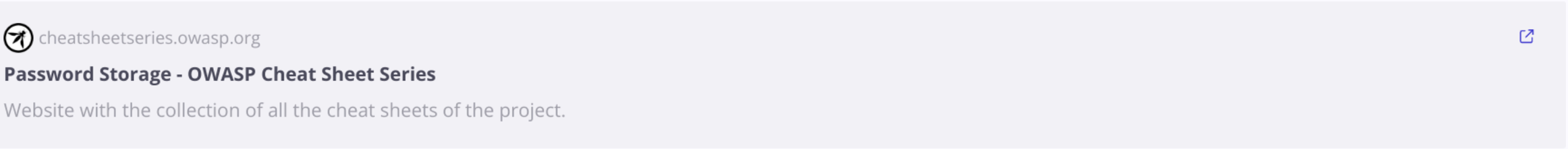
- **bcrypt** – slow, secure, adds salt automatically
- **scrypt**, **Argon2** – modern, memory-hard (resist GPU cracking)

## Avoid:

- MD5, SHA-1 – too fast and broken
- Storing passwords without a salt

```
1. User enters password: "MySecurePass123"
2. System adds random salt: "XyZ123"
3. Hashes with bcrypt: $2b$12$XyZ123...
4. Stores result in DB
```

Use pepper as well



# SSRF - Server-Side Request Forgery

- **Definition:**
  - SSRF is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary destination chosen by the attacker.
  - Even if the server itself is protected by a firewall, it can often access resources that are unavailable to external users.
- **How it Works:**
  - The attacker manipulates input parameters (URLs, file paths, etc.) that the server uses to make requests.
  - Instead of the server requesting an intended resource, it's tricked into requesting an attacker-controlled resource.
- **Attack Vectors:**
  - URLs
  - File paths
  - Headers
- **Impact:**
  - Access to sensitive data
  - Internal service access
  - Remote code execution

# SSRF

- **Exploitation Examples:**
  - **Accessing internal services:** Attacker uses the vulnerable server to access internal services (e.g., databases, admin panels) that are normally protected.
  - **Reading local files:** Attacker forces the server to read local files on the server itself (e.g., configuration files, source code).
  - **Port scanning:** Attacker uses the server to scan internal networks and identify open ports and services.
  - **Cloud metadata attacks:** In cloud environments, attackers can use SSRF to access sensitive metadata (e.g., credentials, API keys).
- **Prevention:**
  - **Input validation and sanitization:** Sanitize and validate all user-provided input.
  - **Whitelist allowed hosts/ports:** Only allow the server to make requests to a predefined list of allowed destinations.
  - **Disable unnecessary URL schemes:** Disable unused URL schemes (e.g., file://, ftp://) to prevent attackers from using them.
  - **Network segmentation:** Isolate internal networks and services.
  - **Disable HTTP redirects:** Prevent the server from automatically following HTTP redirects.
  - **Use a web application firewall (WAF):** A WAF can help detect and block SSRF attacks.

# SSRF - Portswigger Academy

What is SSRF (Server-side request forgery)? Tutorial & Examples | Web Security Academy

Lab: Basic SSRF against the local server | Web Security Academy

# LFI - Local File Inclusion

- **Definition:**
  - LFI is a vulnerability that allows an attacker to read arbitrary files on a server.
  - This occurs when an application improperly uses user-supplied input to construct file paths.
- **How it Works:**
  - Attackers manipulate input parameters that specify file locations.
  - Instead of accessing intended files, the application is tricked into exposing other files on the server.
- **Common Targets:**
  - File inclusion functions (e.g., include, require in PHP)
  - Templates
- **Impact:**
  - Reading sensitive files (e.g., source code, configuration files, passwords)
  - Remote Code Execution (in some cases)

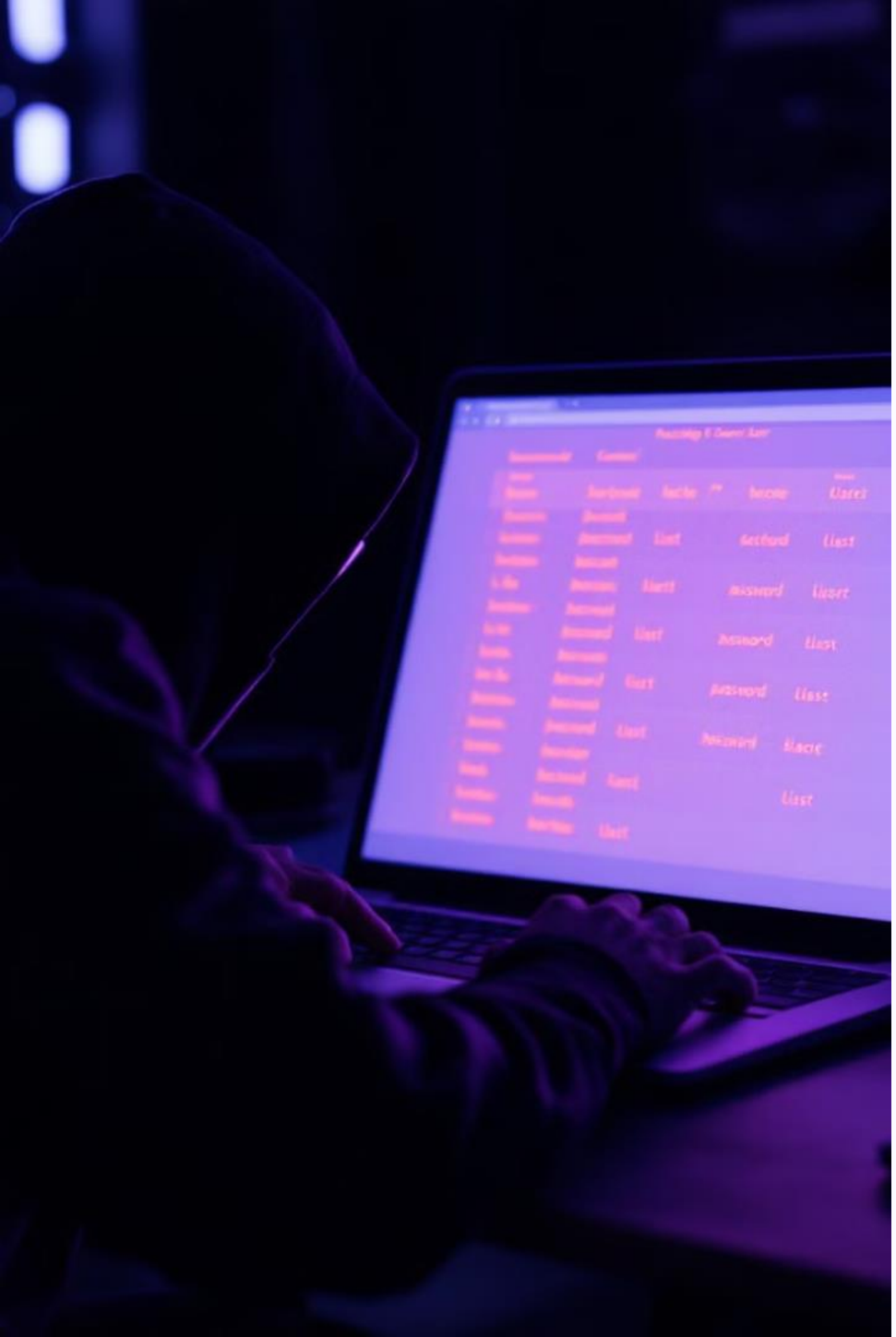
http://localhost:4280/vulnerabilities/fi/?page=include.php

What is path traversal, and how to prevent it? | Web Security Academy

Lab: File path traversal, simple case | Web Security Academy

# LFI - Prevention

- **Exploitation Techniques:**
  - **Path Traversal:** Using ".." sequences to navigate directories and access files outside the intended directory.
  - **Null Byte Injection:** In some older systems, using null bytes (%00) to terminate the file path and bypass extensions.
  - **Wrapper Exploitation:** Using wrappers (e.g., php://filter) to read file contents.
- **Prevention:**
  - **Input Validation:**
    - Strictly validate and sanitize user-provided input.
    - Avoid relying on user input to construct file paths.
    - Whitelist allowed files or directories.
  - **Path Normalization:**
    - Normalize file paths to remove any malicious sequences (e.g., "..", ".").
  - **File Access Restrictions:**
    - Enforce the principle of least privilege for file access.
    - Ensure the application only has access to necessary files.
  - **Disable Dangerous Functions:**
    - Disable or restrict the use of dangerous file inclusion functions if possible.



# Weak Authentication: The Front Door Problem

## Predictable Credentials

Default or easily-guessed usernames and passwords remain common.

## Insufficient Validation

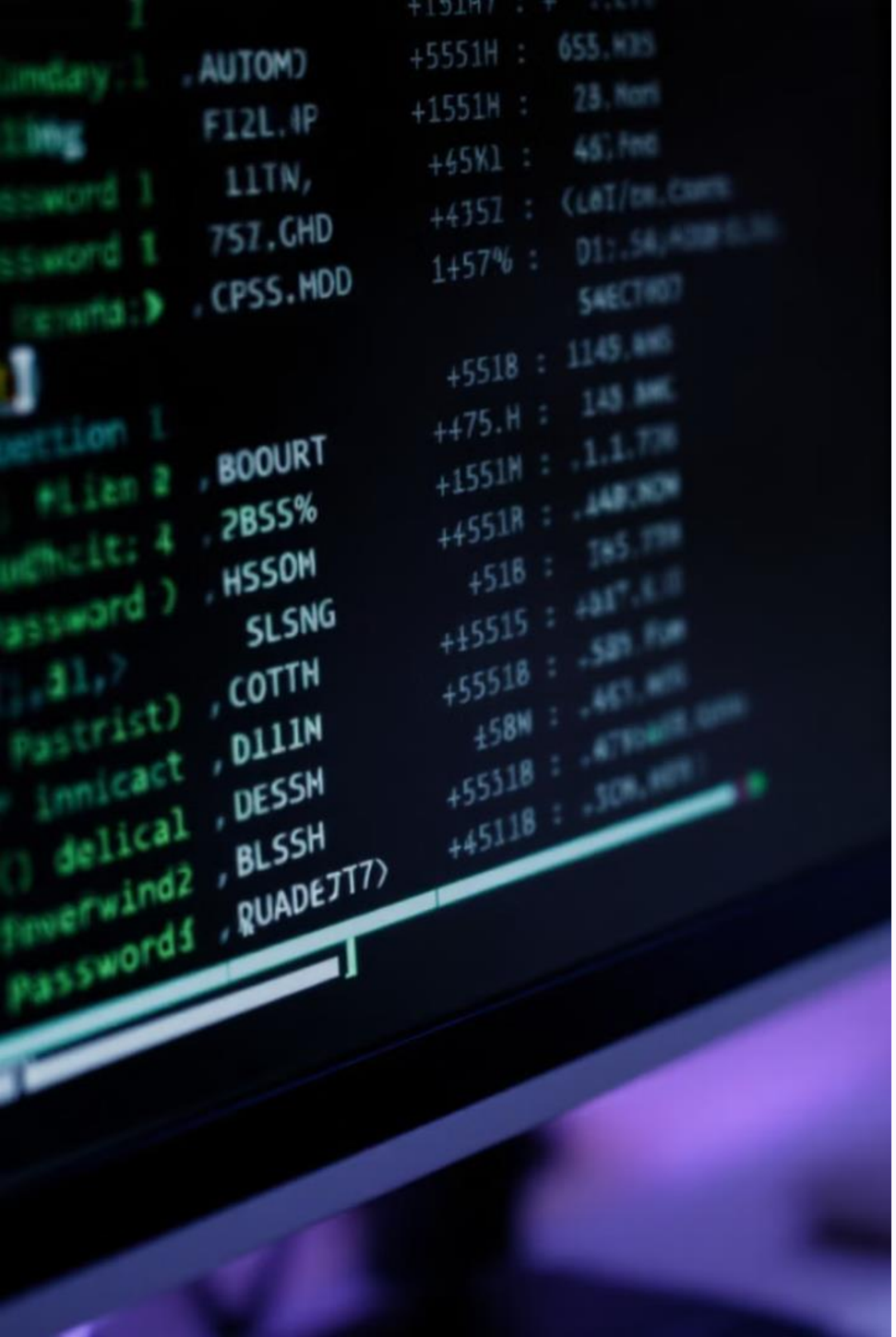
Missing input sanitization lets attackers bypass authentication flows.

## Missing Rate Limiting

Servers without login attempt limits enable brute force attacks.

## Weak Reset Mechanisms

Password recovery flows often bypass stronger primary authentication.



# Brute Force Attack Techniques

1. **Dictionary Attacks**

   Using wordlists containing common passwords.

2. **Credential Stuffing**

   Trying leaked username/password pairs from other breaches.

3. **Password Spraying**

   Testing a few common passwords against many usernames.

4. **Parameter Manipulation**

   Modifying HTTP requests to bypass client-side validation.

# OAuth 2.0

- **Definition:**
  - OAuth 2.0 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service.
  - It allows users to grant third-party applications access to their resources without exposing their credentials.
- **Key Roles:**
  - **Resource Owner:** The user who authorizes access.
  - **Client:** The application requesting access.
  - **Authorization Server:** Issues access tokens.
  - **Resource Server:** Hosts the protected resources.
- **Benefits:**
  - Improved security (no shared credentials).
  - Delegated authorization.
  - Standardized protocol.

# OAuth 2.0

- **Simplified Workflow:**
    1. The client requests authorization from the user.
    2. The user grants authorization to the client.
    3. The client obtains an access token from the authorization server.
    4. The client uses the access token to access protected resources on the resource server.
- **Common Grant Types:**
    - **Authorization Code:** For web applications (most common).
    - **Implicit:** For single-page applications (less secure).
    - **Client Credentials:** For application access.
    - **Resource Owner Password Credentials:** (Discouraged).
- **Focus on Tokens:** OAuth 2.0 revolves around the issuance and use of access tokens, which have a limited lifetime.

# OAuth Play Ground

OAuth 2.0 Playground

# Session Attack Vectors

## Session ID Theft

Capturing tokens through packet sniffing or XSS.

## Session Fixation

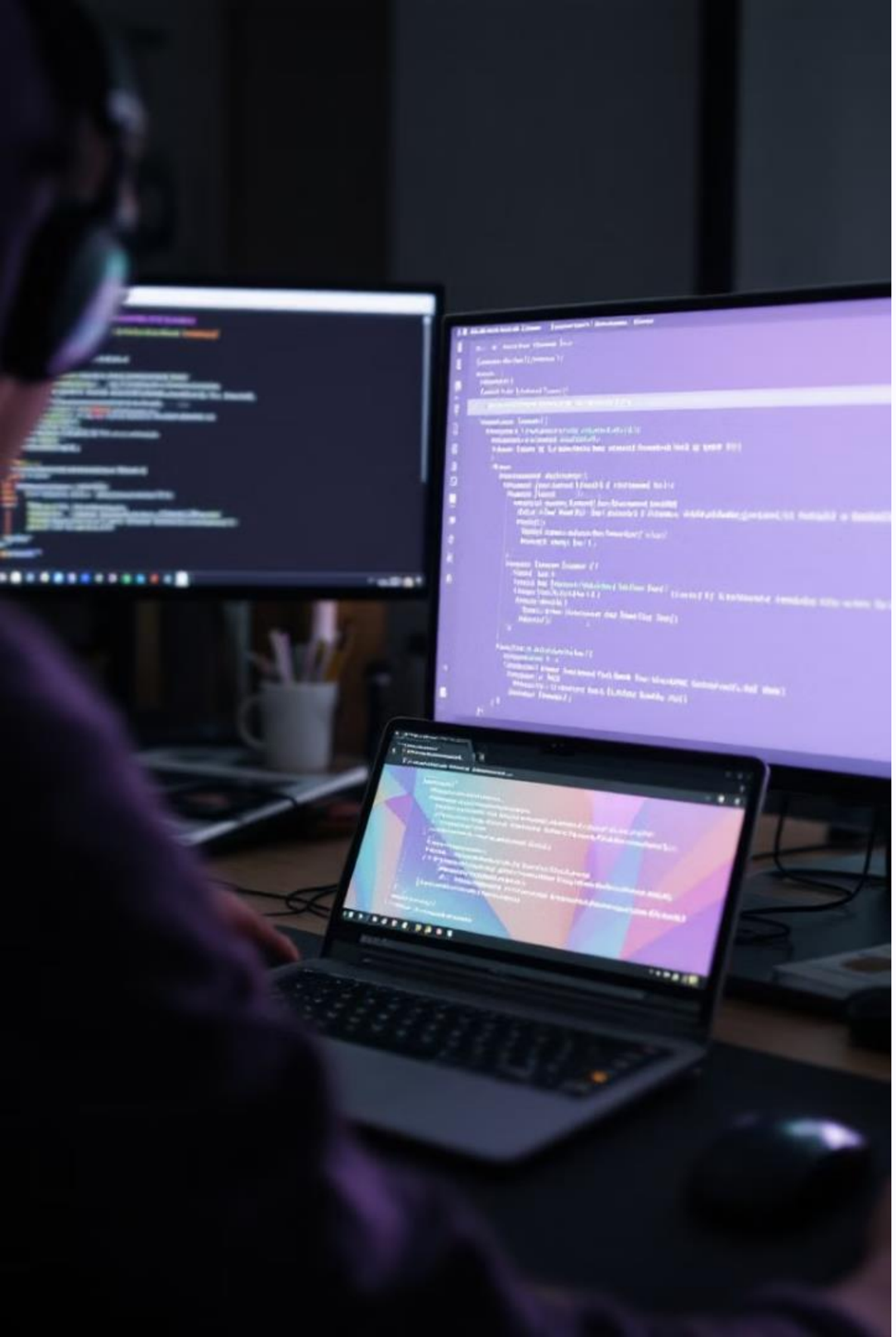
Forcing users to use attacker-controlled session identifiers.

## Predictable IDs

Guessing session tokens with sequential or weak randomization.

## Timeout Exploitation

Leveraging overly long session lifetimes.



# Hands-on Exercise: Breaking Authentication

## Identify the Login Form

Examine the application's authentication mechanism and request flow.

### Intercept Traffic

Configure your proxy to capture authentication requests.

#### Manipulate Parameters

Try SQL injection, remove validation parameters, or bypass JavaScript checks.

##### Analyze Responses

Look for information leakage in error messages.


# Security Misconfigurations

33%

## Default Credentials

Of breaches involve unchanged default passwords

70%

## Admin Panels

Of admin interfaces lack proper access controls

62%

## Debug Modes

Of production apps expose debugging information

45%

## Unnecessary Services

Of servers run unneeded, vulnerable components

# Exploiting Business Logic Flaws

## Identify Process Workflows

Map out multi-step processes like checkout or account registration.

## Test Parameter Manipulation

Try modifying prices, quantities, or discount codes.

## Change Process Order

Skip steps or perform them out of sequence.

## Exploit Race Conditions

Submit multiple concurrent requests to confuse application state.


# Effective Penetration Testing: Reporting & Mitigation

# Writing Effective Penetration Test Reports

## Executive Summary

High-level overview for decision makers

## Methodology & Scope

Clear testing boundaries and approaches used

## Findings & Evidence

Detailed vulnerability documentation with proof

## Remediation Steps

Actionable fixes with implementation guidance

A structured report builds credibility and ensures findings get addressed. Always include reproducible steps and evidence for each vulnerability.

# Vulnerability Severity Classification

## Critical Severity

- Remote code execution
- Authentication bypass
- Database compromise
- Default/hardcoded credentials

## Medium Severity

- Cross-site scripting (XSS)
- Sensitive data exposure
- Security misconfiguration
- Weak password policies

## Low Severity

- Missing security headers
- Information disclosure
- Error messages revealing details
- Outdated libraries (non-critical)

Use CVSS scoring to ensure consistent severity ratings. Include business impact analysis to help stakeholders understand risk context.

# Communicating Findings Effectively

## Tailor to Your Audience

Use technical details for developers. Focus on business impact for executives.

## Visualize Data

Include charts and graphs to show vulnerability distribution. Present trends across systems.

## Provide Context

Explain potential attack scenarios. Demonstrate real exploitation paths when possible.

## Highlight Positives

Acknowledge security measures that worked well. Balance criticism with recognition.

Avoid jargon when communicating with non-technical stakeholders. Use analogies to explain complex vulnerabilities.

# Mitigation Best Practices

## Input Validation

- Whitelist acceptable inputs
- Apply type checking
- Implement boundary validation

## Authentication Controls

- Enforce MFA
- Implement account lockouts
- Use secure password storage

## Database Security

- Use parameterized queries
- Implement least privilege
- Encrypt sensitive data

## Configuration Management

- Remove default credentials
- Apply security headers
- Disable unnecessary services

Provide specific code examples in your remediation advice. Link to official documentation when available.

# Resources and Next Steps

## Learning Resources

OWASP Testing Guide, HackerOne reports, Bug Bounty writeups

## Practice Environments

DVWA, Juice Shop, VulnHub VMs, HackTheBox

## Certifications

OSCP, SANS GPEN, GWAPT, OSWE

## Community Engagement

CTF competitions, security meetups, online forums

Continue developing your skills through regular practice. Join the security community to learn from others' experiences. Stay current with emerging attack techniques.

# Thank You


Sheshananda Reddy Kandula
Sr Security Engineer at Adobe | AppSec |
Product Securtiy | OSWE | OSCP | CISSP

# Cross-Site Request Forgery (CSRF) Explained

## How CSRF Works

CSRF tricks users into performing actions they didn't intend to do.

Attacks exploit how browsers automatically send cookies with each request.

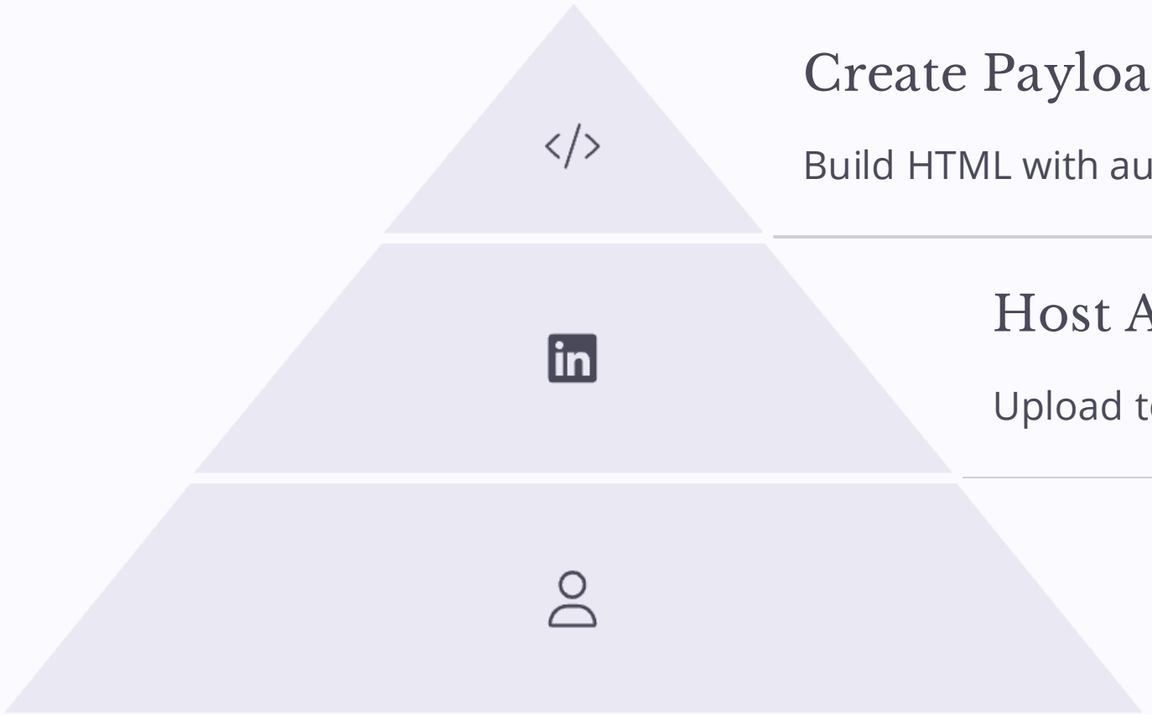
## Impact

- Account takeover
- Fund transfers
- Profile changes
- Data theft

## Signs of Vulnerability

- No anti-CSRF tokens
- Cookie-only authentication
- No referrer checks

# Crafting Malicious CSRF Requests



## Create Payload

Build HTML with auto-submitting forms

## Host Attack Page

Upload to attacker-controlled domain

## Deliver to Victim

Send via email, messages or comments

The key to CSRF is making requests indistinguishable from legitimate user actions. Cross-origin requests automatically include authentication cookies.

# Hands-on Exercise: CSRF Exploitation

## Identify Vulnerable Endpoint

Look for state-changing operations that accept GET requests or don't validate origin.

## Analyze Request Structure

Capture normal requests and identify required parameters and authentication mechanisms.

## Create Attack Template

Build an HTML file with a form that automatically submits to the target endpoint.

## Test Your Exploit

Verify the exploit works by loading your attack page while authenticated to the target.