Task 2: Introduction to Web Application Security

Nicholas Massei

Cybersecurity Intern

Redynox

July 12, 2025

Introduction

During my second task, I focused on identifying and exploiting web application vulnerabilities – as mentioned in the objectives, focusing on SQL Injection and Cross-Site Scripting (XSS) using tools like bWAPP, WebGoat, and ZAP. My goal was to understand these vulnerabilities in practice and learn mitigation techniques.

Setup WebGoat Environment

To begin, I installed Docker to run WebGoat on my host machine – and later just installed WebGoat on my Kali Linux VM. WebGoat is an intentionally vulnerable web application including its own lessons for practicing techniques, and I accessed it via browser at http://localhost:8080/WebGoat. The platform offers guided modules that simulate real attack scenarios, which made it super easy to engage with and test techniques like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) – all are techniques I have become familiar with beforehand.

Logging in and exploring these modules on WebGoat provided a controlled space to break things, learn from mistakes, and better understand how attackers think. So far, I am just thinking about how incredible this tool is for learning and sandboxing.

Gaining Background Information

I then used an incredibly helpful YouTube tutorial titled *"Automated Hacking Tool?! | OWASP ZAP Tutorial"* to deepen my understanding of ZAP and how to properly interpret the results it was giving me. This resource helped me move beyond just passively running scans -- it taught me how to read alerts, understand their risk levels, and even explore follow-up manual testing based on what was discovered. With this video, I was able to take more **intentional** steps in using ZAP as an actual penetration testing tool, not just an automated scanner.

The tutorial specifically connected to vulnerabilities like Cross-Site Scripting **(XSS)**, **SQL Injection**, and Cross-Site Request Forgery **(CSRF)** by demonstrating how ZAP detects these issues and what manual techniques can be used to exploit and confirm them. This made it much easier to grasp the practical implications of the vulnerabilities I encountered in WebGoat and understand how attackers might leverage them. It also showed me that ZAP definitely pulls false negatives (which should be expected) so it is good to verify every vulnerability ZAP finds.

Reference: https://youtu.be/QJ5u_dHwoAk?si=1si6gh2HH_V62lAZ

Basic Vulnerability Analysis

Now to start getting interactive. I have everything set up and am ready to manually explore WebGoat's lessons and the site itself, actively investigating and exploiting vulnerabilities while using ZAP as my primary tool for identifying security weaknesses. While going through the lessons for SQL injection, XSS, and CSRF on WebGoat I had ZAP manually configured to log, spider, and further vulnerability scan using, and it obviously logged more than one example of each of these vulnerabilities – there were many more to explore but I will focus on these for this task. The alerts listed gave me solid proof of each vulnerability.

- **SQL Injection** and **SQL Injection - Hypersonic SQL –** SQL Injection
- **Vulnerable JS Library –** XSS
- **Absence of Anti-CSRF Tokens –** CSRF

Evidence screenshotted and included on the last page(s).

Exploring Vulnerabilities

After reviewing ZAP's alert descriptions and with the lessons I went through on WebGoat, I deepened my understanding of how each vulnerability functions – not just in theory, but even in the context of a real (insecure) application.

- For **SQL Injection** (Windows – WebGoat), I used inputs in the lessons such as the first one where I used "SELECT department FROM employees WHERE first_name = 'Bob'" to retrieve data without authentication. ZAP's alert for "SQL Injection – Hypersonic SQL" confirmed the potential impact of insecure query handling and helped reinforce how backend logic can be manipulated with crafted input.
- For **XSS** (Windows – WebGoat), I explored where simple JavaScript payloads like <script>alert('XSS')</script> could be submitted — particularly in comment or input fields. While WebGoat provides safe examples, ZAP identified the usage of an outdated version of Underscore.js, linked to a known CVE (CVE-2021-23358, as listed in the description), which supports the potential for script injection attacks.
- For **CSRF** (Windows – WebGoat), ZAP detected the absence of anti-CSRF tokens in many requests. Without these tokens, an attacker could forge state-changing requests from a victim's browser. This aligns closely with what I learned when going through WebGoat's CSRF lessons.

Each of these findings was confirmed through both ZAP's logging, automated scanning, and manual interaction with the WebGoat interface.

Using Kali Linux VM with ZAP in bWAPP to explore SQL Injection and XSS further

Attempting to manually exploit a site (with consent - bWAPP) using techniques I have learned.

bWAPP (buggy web application) is an intentionally vulnerable app -- like WebGoat, but different.

- **SQL Injection:** For SQL Injection (Search/GET – bWAPP on Kali), I used payloads like ' OR 1=1 # to bypass search filters and retrieve the entire dataset from the backend without proper validation. The success of this injection demonstrated how vulnerable queries can be exploited to expose all records. This further reinforced the importance of sanitizing input and using parameterized queries to defend against unauthorized data access.
- **Cross-Site Scripting XSS:** For XSS testing on bWAPP, I used payload '><script>alert('You got hacked!')</script> and injected it. This successfully triggered an alert, demonstrating that the application failed to properly sanitize user input before rendering it on the page. This vulnerability highlights the risk of attackers injecting malicious scripts that can steal user data or perform unauthorized actions. It further emphasizes the need for input validation to prevent such attacks.

Evidence screenshotted and included on the last page(s) as always.

Challenges Faced

Most tasks were clear and manageable, though I faced some formatting issues with documentation and had to familiarize myself with new security tools and set ups. I addressed these by researching best practices and searching for help when needed.

Report

SQL Injection: Bypassed input validation to access unauthorized data.

Cross-Site Scripting (XSS): Executed malicious scripts via un-sanitized user input.

Cross-Site Request Forgery (CSRF): Missing anti-CSRF tokens would allow unauthorized actions.

Possible Simple Mitigations

Input Sanitization & Parameterized Queries: Prevent SQL injection by using prepared statements and validating user inputs rigorously.

Output Encoding & Content Security Policy (CSP): Mitigate XSS by encoding output, sanitizing input, and implementing CSP headers to restrict script execution.

Implement Anti-CSRF Tokens: Include unique tokens in state-changing requests to verify legitimate user actions and block forged requests.

Reference:

OWASP Foundation. *OWASP Cheat Sheet Series*. https://cheatsheetseries.owasp.org/index.html

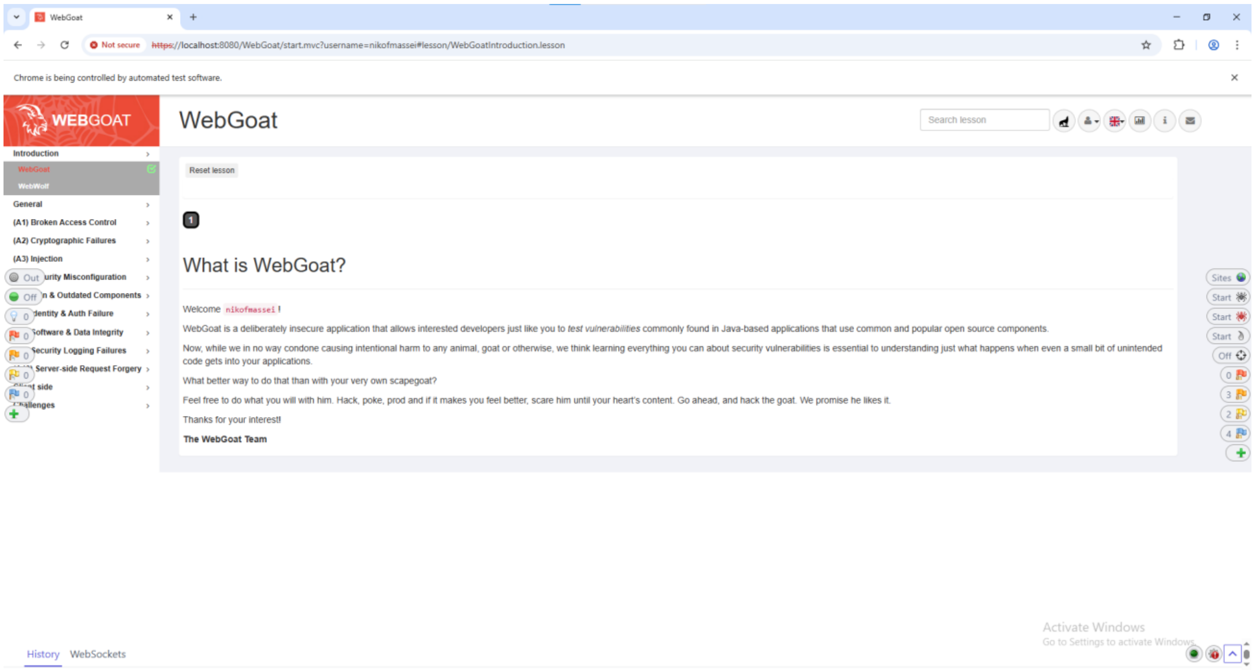
Conclusion

This internship has been a great learning experience that took my cybersecurity skills from theory to hands-on application. Working through real vulnerabilities like SQL injection, cross-site scripting, and CSRF while using tools such as WebGoat, ZAP, and Wireshark allowed me to better understand both offensive **and** defensive techniques. From network traffic monitoring to vulnerability scanning and safe configuration practices, every task showed the importance of layered security and its implementation.

Closing Remarks

I’m grateful to the Redynox team for creating supportive, challenge-driven tasks. The structure and resources provided helped me grow technically and professionally. For what it is, I couldn’t think of anything to make this opportunity better. Overall, this internship has strengthened my confidence and prepared me for future roles in cybersecurity. Thank you for the opportunity!

Windows - WebGoat


WebGoat
https://localhost:8080/WebGoat/start.mvc?username=nikofmassei#lesson/WebGoatIntroduction.lesson
Not secure
Chrome is being controlled by automated test software.
WEBGOAT
Search lesson
Introduction
WebGoat
WebWolf
General
(A1) Broken Access Control
(A2) Cryptographic Failures
(A3) Injection
urity Misconfiguration
n & Outdated Components
dentity & Auth Failure
Software & Data Integrity
Security Logging Failures
Server-side Request Forgery
side
allenges
Out
Off
Sites
Start
Reset lesson
History
WebSockets
Activate Windows
Go to Settings to activate Windows

WebGoat

1

What is WebGoat?

Welcome `nikofmassei` !

WebGoat is a deliberately insecure application that allows interested developers just like you to *test vulnerabilities* commonly found in Java-based applications that use common and popular open source components.

Now, while we in no way condone causing intentional harm to any animal, goat or otherwise, we think learning everything you can about security vulnerabilities is essential to understanding just what happens when even a small bit of unintended code gets into your applications.

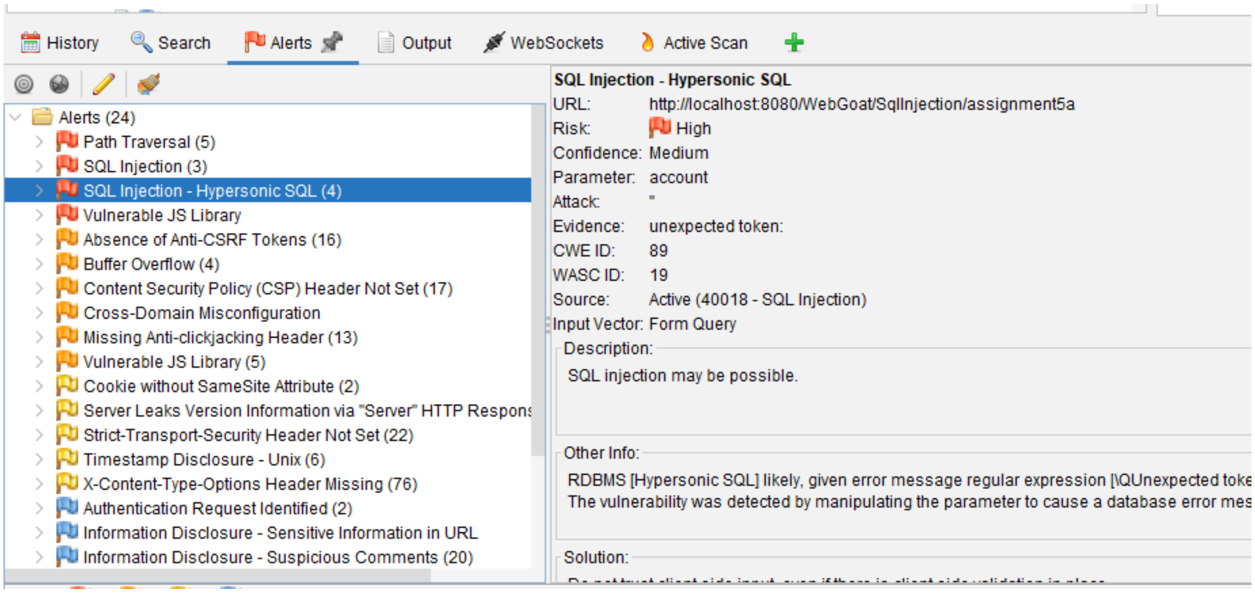
What better way to do that than with your very own scapegoat?

Feel free to do what you will with him. Hack, poke, prod and if it makes you feel better, scare him until your heart's content. Go ahead, and hack the goat. We promise he likes it.

Thanks for your interest!

The WebGoat Team

Logged in & running WebGoat successfully. Manual explore active via ZAP (with HUD enabled)



- Alerts (24)
  - Path Traversal (5)
  - SQL Injection (3)
  - SQL Injection - Hypersonic SQL (4)
  - Vulnerable JS Library
  - Absence of Anti-CSRF Tokens (16)
  - Buffer Overflow (4)
  - Content Security Policy (CSP) Header Not Set (17)
  - Cross-Domain Misconfiguration
  - Missing Anti-clickjacking Header (13)
  - Vulnerable JS Library (5)
  - Cookie without SameSite Attribute (2)
  - Server Leaks Version Information via "Server" HTTP Respons
  - Strict-Transport-Security Header Not Set (22)
  - Timestamp Disclosure - Unix (6)
  - X-Content-Type-Options Header Missing (76)
  - Authentication Request Identified (2)
  - Information Disclosure - Sensitive Information in URL
  - Information Disclosure - Suspicious Comments (20)

SQL Injection - Hypersonic SQL

URL: http://localhost:8080/WebGoat/SqlInjection/assignment5a
Risk: High
Confidence: Medium
Parameter: account
Attack: '
Evidence: unexpected token:
CWE ID: 89
WASC ID: 19
Source: Active (40018 - SQL Injection)
Input Vector: Form Query

Description:

SQL injection may be possible.

Other Info:

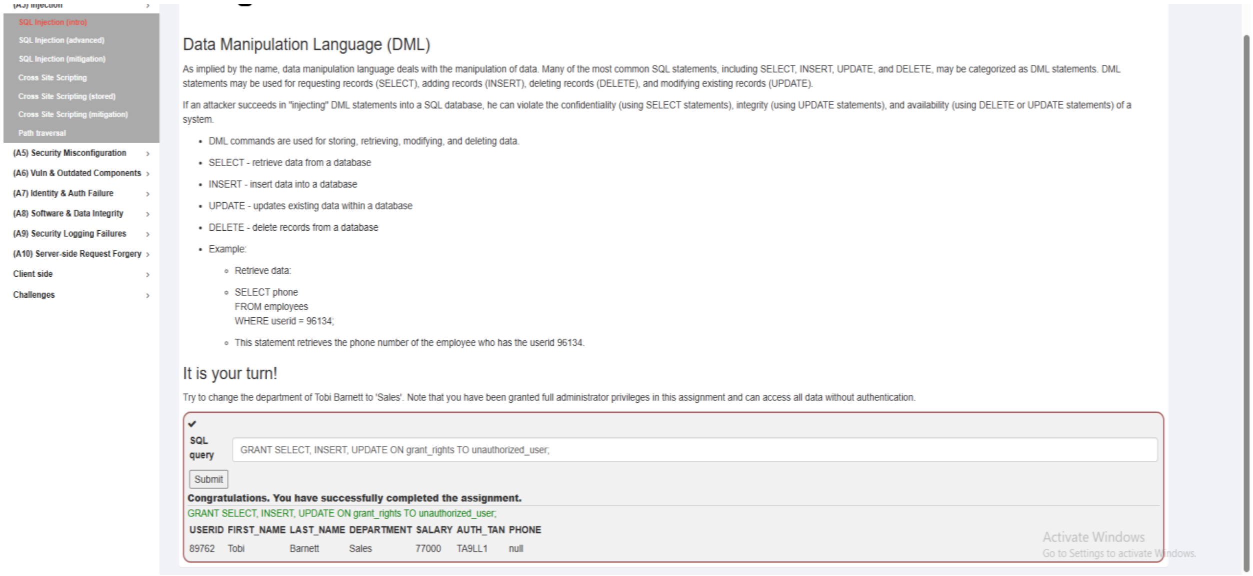
RDBMS [Hypersonic SQL] likely, given error message regular expression [\QUnexpected toke
The vulnerability was detected by manipulating the parameter to cause a database error mes

Solution:

Do not trust client side input, even if there is client side validation in place.

Alerts were found after manual interaction with all WebGoat lessons and active scan in the ZAP GUI. Used the HUD in WebGoat to spider and scan.

SQL Injection & XSS through WebGoat


(A3) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Cross Site Scripting
Cross Site Scripting (stored)
Cross Site Scripting (mitigation)
Path traversal
(A5) Security Misconfiguration
(A6) Vuln & Outdated Components
(A7) Identity & Auth Failure
(A8) Software & Data Integrity
(A9) Security Logging Failures
(A10) Server-side Request Forgery
Client side
Challenges
Submit
Activate Windows
Go to Settings to activate Windows.

Data Manipulation Language (DML)

As implied by the name, data manipulation language deals with the manipulation of data. Many of the most common SQL statements, including SELECT, INSERT, UPDATE, and DELETE, may be categorized as DML statements. DML statements may be used for requesting records (SELECT), adding records (INSERT), deleting records (DELETE), and modifying existing records (UPDATE).

If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and availability (using DELETE or UPDATE statements) of a system.

- DML commands are used for storing, retrieving, modifying, and deleting data.
- SELECT - retrieve data from a database
- INSERT - insert data into a database
- UPDATE - updates existing data within a database
- DELETE - delete records from a database
- Example:
  - Retrieve data:
  - SELECT phone
    FROM employees
    WHERE userid = 96134;
  - This statement retrieves the phone number of the employee who has the userid 96134.

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

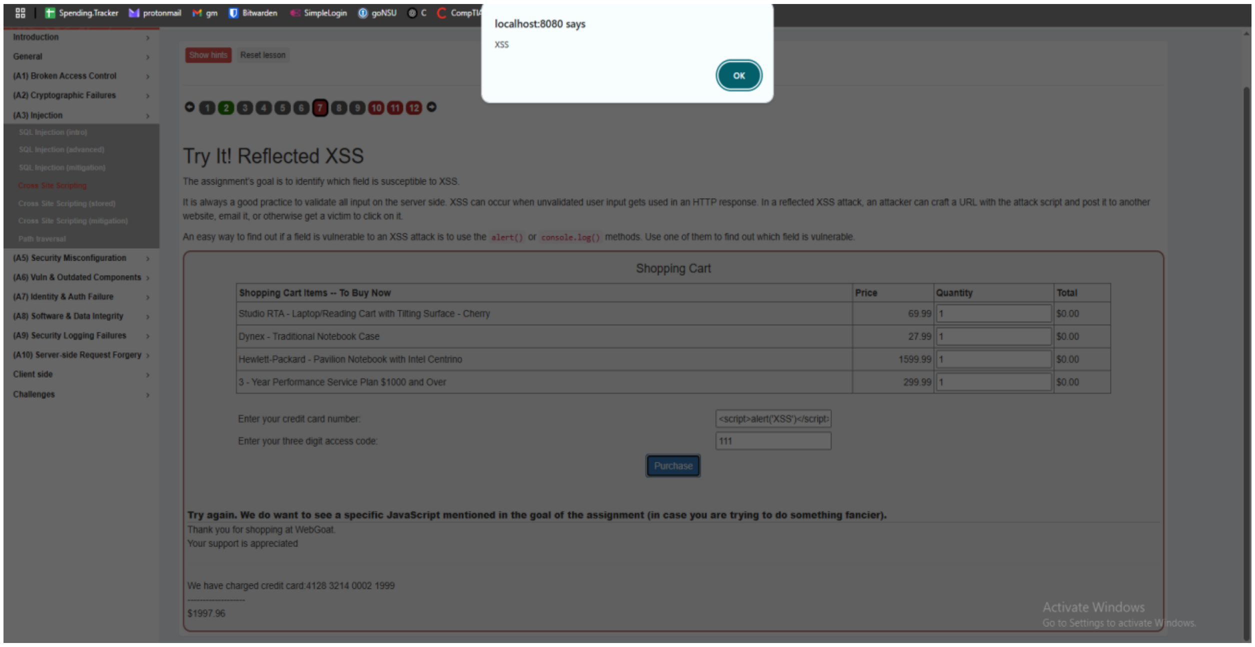
✓

SQL query: GRANT SELECT, INSERT, UPDATE ON grant_rights TO unauthorized_user;

Congratulations. You have successfully completed the assignment.

```
GRANT SELECT, INSERT, UPDATE ON grant_rights TO unauthorized_user;
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN PHONE
89762  Tobi       Barnett   Sales      77000  TA9LL1   null
```

A later example of using SQL Injection to manipulate a database in WebGoat via host machine


Spending.Tracker
protonmail
gm
Bitwarden
SimpleLogin
goNSU
C
CompTIA
OK
Introduction
General
(A1) Broken Access Control
(A2) Cryptographic Failures
(A3) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Cross Site Scripting
Cross Site Scripting (stored)
Cross Site Scripting (mitigation)
Path traversal
(A5) Security Misconfiguration
(A6) Vuln & Outdated Components
(A7) Identity & Auth Failure
(A8) Software & Data Integrity
(A9) Security Logging Failures
(A10) Server-side Request Forgery
Client side
Challenges
Show hints
Reset lesson
1 2 3 4 5 6 7 8 9 10 11 12
Purchase
Activate Windows
Go to Settings to activate Windows.

localhost:8080 says

XSS

Try It! Reflected XSS

The assignment's goal is to identify which field is susceptible to XSS.

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	$0.00
Dynex - Traditional Notebook Case	27.99	1	$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	$0.00
3 - Year Performance Service Plan $1000 and Over	299.99	1	$0.00

Enter your credit card number: <script>alert('XSS')</script

Enter your three digit access code: 111

Try again. We do want to see a specific JavaScript mentioned in the goal of the assignment (in case you are trying to do something fancier).

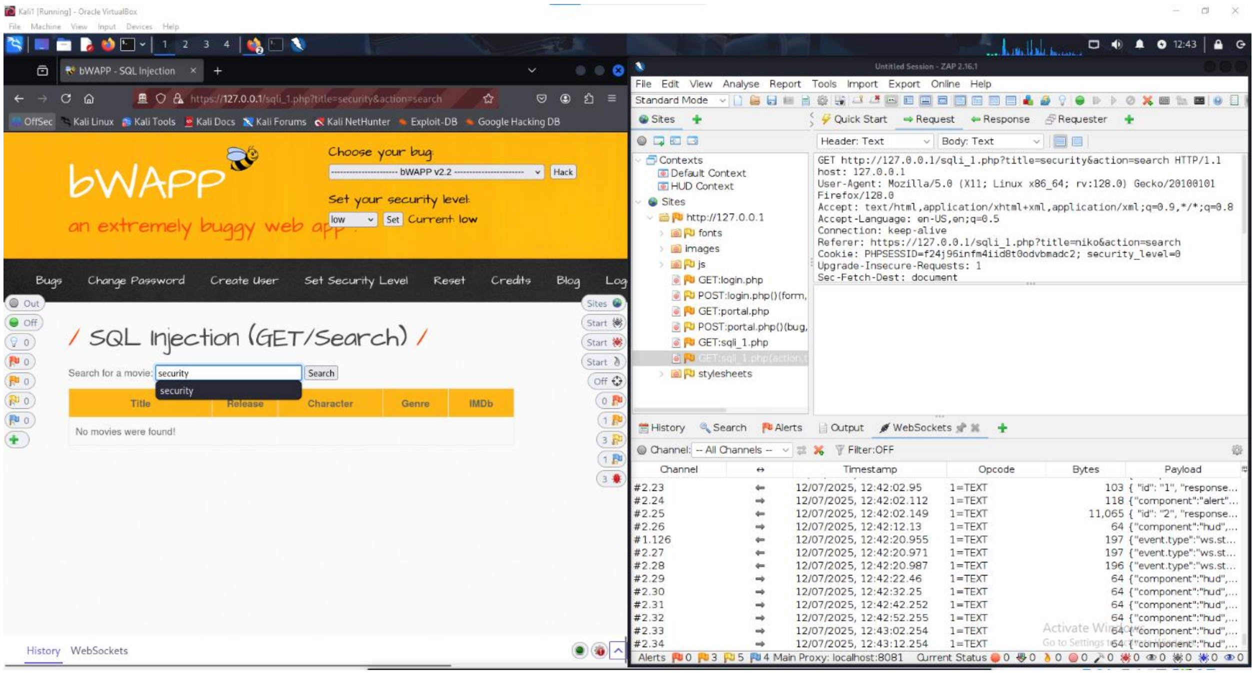
Thank you for shopping at WebGoat.
Your support is appreciated

We have charged credit card:4128 3214 0002 1999

$1997.96

XSS attack successful in a later lesson in WebGoat via host machine

Kali – bWAPP

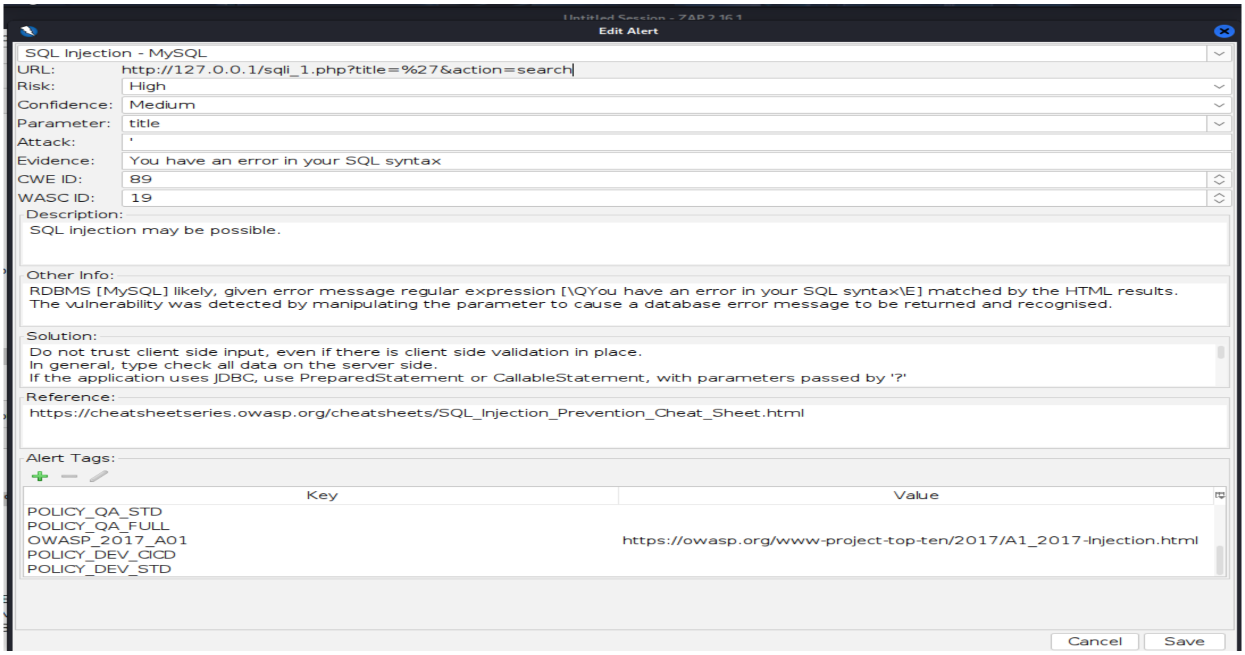


```
GET http://127.0.0.1/sqli_1.php?title=security&action=search HTTP/1.1
host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101
Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: https://127.0.0.1/sqli_1.php?title=niko&action=search
Cookie: PHPSESSID=f24j96infm4iid8t0odvbmadc2; security_level=0
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
```

Channel	↔	Timestamp	Opcode	Bytes	Payload
#2.23	←	12/07/2025, 12:42:02.95	1=TEXT	103	{ "id": "1", "response...
#2.24	→	12/07/2025, 12:42:02.112	1=TEXT	118	{"component":"alert"...
#2.25	←	12/07/2025, 12:42:02.149	1=TEXT	11,065	{ "id": "2", "response...
#2.26	→	12/07/2025, 12:42:12.13	1=TEXT	64	{"component":"hud",...
#1.126	←	12/07/2025, 12:42:20.955	1=TEXT	197	{"event.type":"ws.st...
#2.27	←	12/07/2025, 12:42:20.971	1=TEXT	197	{"event.type":"ws.st...
#2.28	←	12/07/2025, 12:42:20.987	1=TEXT	196	{"event.type":"ws.st...
#2.29	→	12/07/2025, 12:42:22.46	1=TEXT	64	{"component":"hud",...
#2.30	→	12/07/2025, 12:42:32.25	1=TEXT	64	{"component":"hud",...
#2.31	→	12/07/2025, 12:42:42.252	1=TEXT	64	{"component":"hud",...
#2.32	→	12/07/2025, 12:42:52.255	1=TEXT	64	{"component":"hud",...
#2.33	→	12/07/2025, 12:43:02.254	1=TEXT	64	{"component":"hud",...
#2.34	→	12/07/2025, 12:43:12.254	1=TEXT	64	{"component":"hud",...

Alerts 0 3 5 4 Main Proxy: localhost:8081 Current Status 0 0 0 0 0 0 0 0 0 0

Normal input through bWAPP to log in ZAP for automated attacks and alerts – as seen in the HTTP request



SQL Injection - MySQL

URL:	http://127.0.0.1/sqli_1.php?title=%27&action=search
Risk:	High
Confidence:	Medium
Parameter:	title
Attack:	'
Evidence:	You have an error in your SQL syntax
CWE ID:	89
WASC ID:	19

Description:
SQL injection may be possible.

Other Info:
RDBMS [MySQL] likely, given error message regular expression [\QYou have an error in your SQL syntax\E] matched by the HTML results.
The vulnerability was detected by manipulating the parameter to cause a database error message to be returned and recognised.

Solution:
Do not trust client side input, even if there is client side validation in place.
In general, type check all data on the server side.
If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'

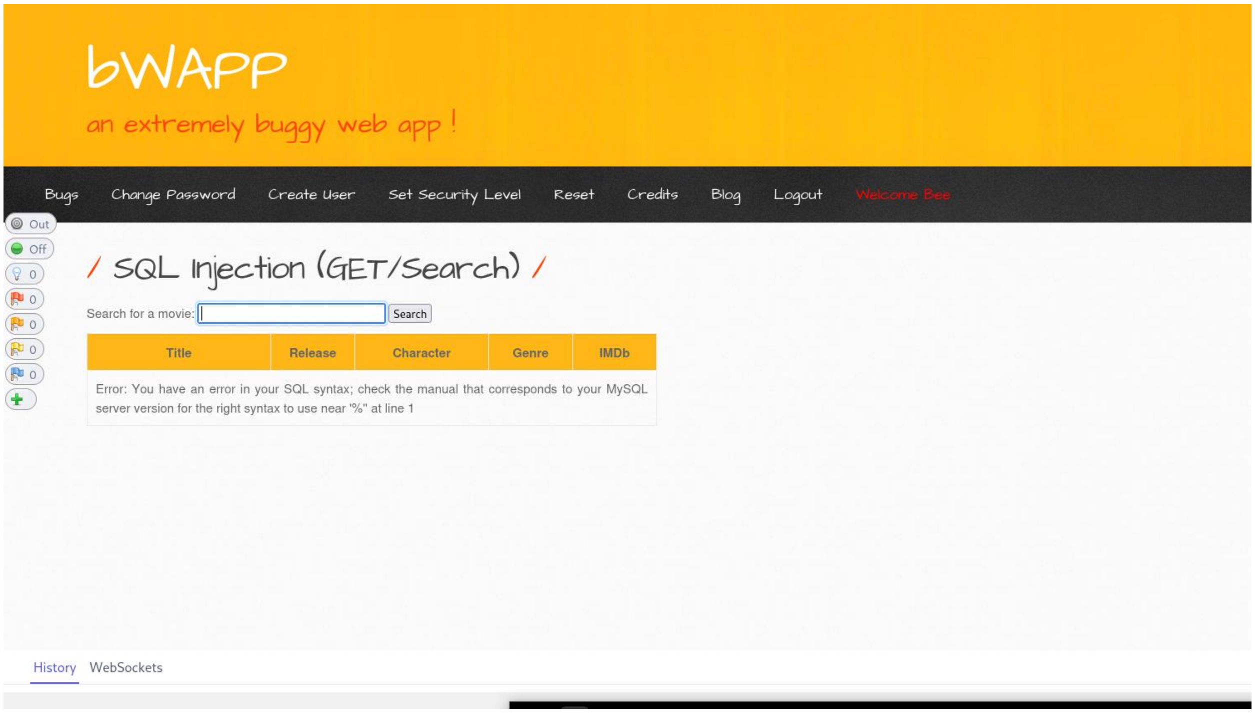
Reference:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Alert Tags:

Key	Value
POLICY_QA_STD	
POLICY_QA_FULL	
OWASP_2017_A01	https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html
POLICY_DEV_CICD	
POLICY_DEV_STD	

Possible SQL Injection details listed from active scan

SQL Injection



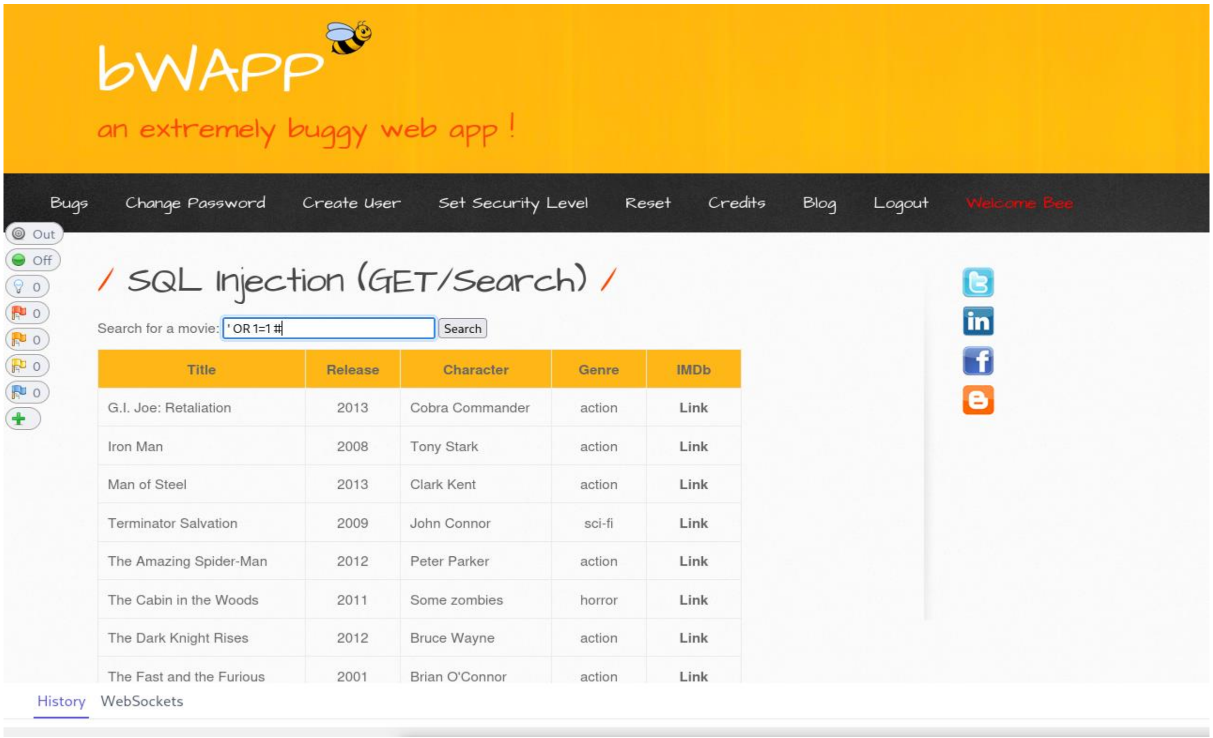
/ SQL Injection (GET/Search) /

Search for a movie: Search

Title	Release	Character	Genre	IMDb

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '%'' at line 1

I followed the HTTP address for the possible injection, and this confirms that SQL Injection can be exploited



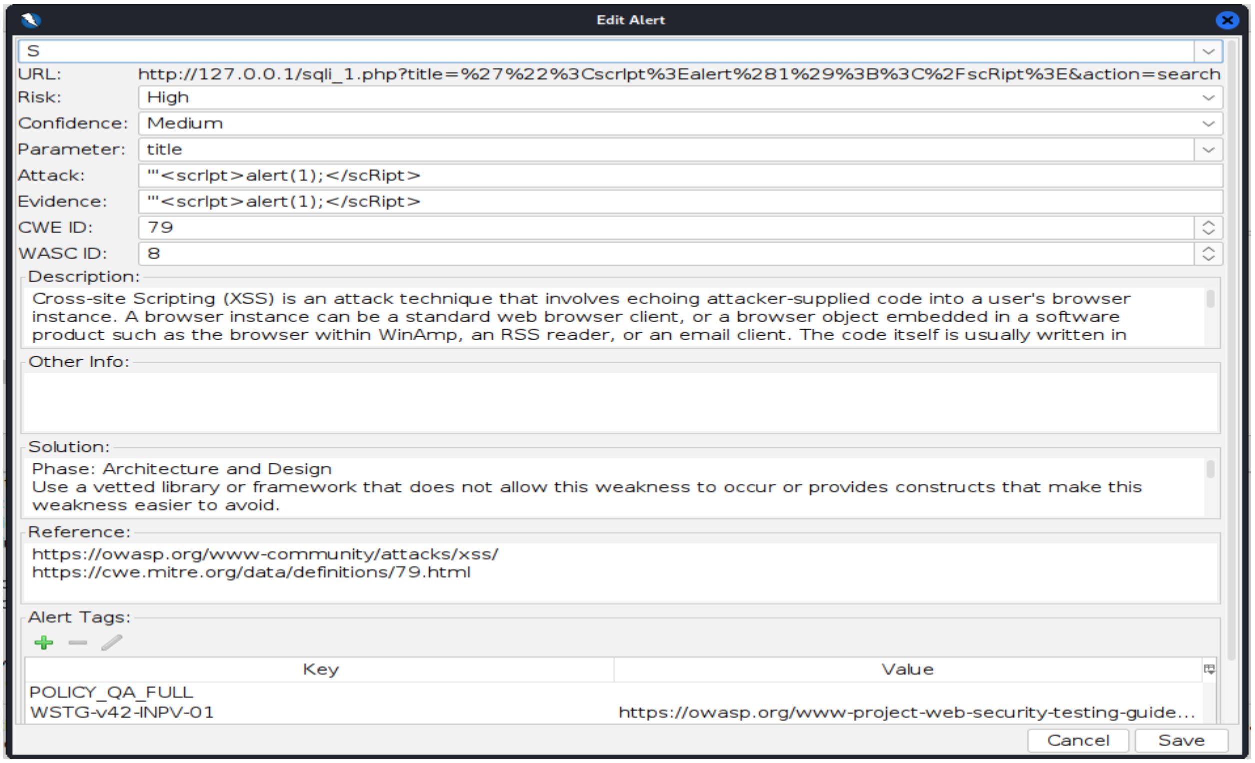
/ SQL Injection (GET/Search) /

Search for a movie: ' OR 1=1 # Search

Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	Link
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link

After attempting multiple common SQL injections, ”' OR 1=1 #” worked!

XSS



S

URL: http://127.0.0.1/sqli_1.php?title=%27%22%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E&action=search

Risk: High

Confidence: Medium

Parameter: title

Attack: '"<scrIpt>alert(1);</scRipt>

Evidence: '"<scrIpt>alert(1);</scRipt>

CWE ID: 79

WASC ID: 8

Description:

Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in

Other Info:

Solution:

Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

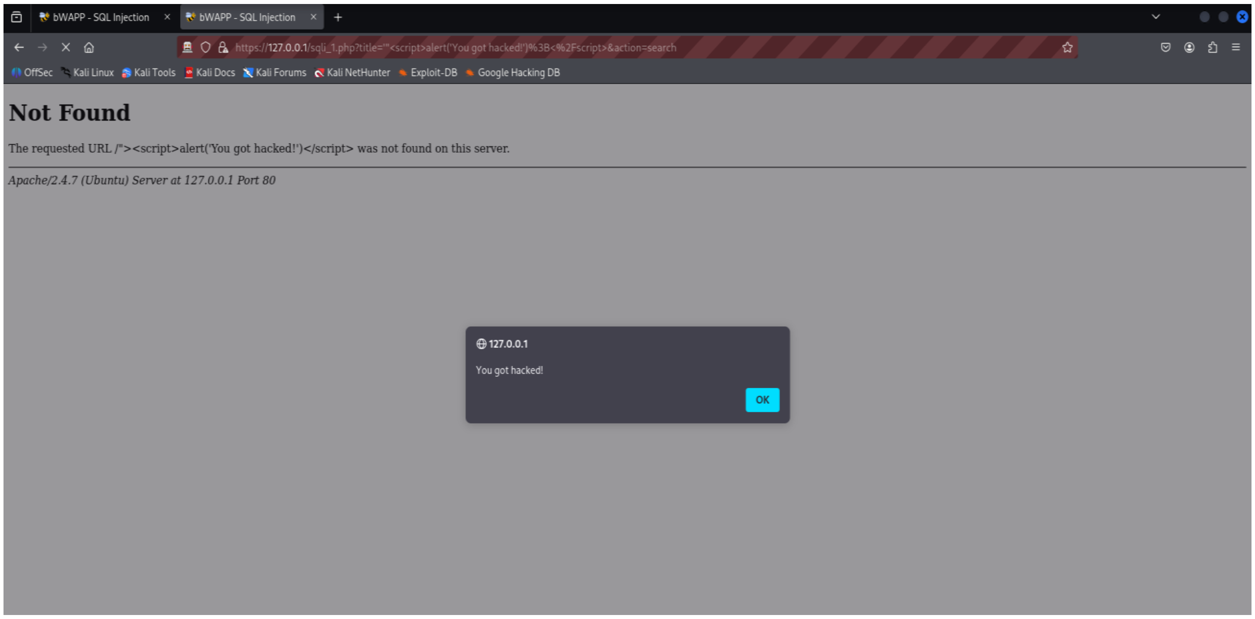
Reference:

https://owasp.org/www-community/attacks/xss/
https://cwe.mitre.org/data/definitions/79.html

Alert Tags:

Key	Value
POLICY_QA_FULL	
WSTG-v42-INPV-01	https://owasp.org/www-project-web-security-testing-guide...

XSS (Reflected) details form auto scan. Following HTTP and confirming vulnerability.



https://127.0.0.1/sqli_1.php?title="'<script>alert('You got hacked!')%3B<%2Fscript>&action=search

Not Found

The requested URL /"><script>alert('You got hacked!')</script> was not found on this server.

Apache/2.4.7 (Ubuntu) Server at 127.0.0.1 Port 80

127.0.0.1

You got hacked!

Vulnerability is confirmed. Changed payload from displaying – as seen in the ZAP details – “1” to “You got hacked!”