Web Attacks: Concepts, Examples Visualizations

Generated for Abdo Wael

Last updated:

Abstract

This document provides a comprehensive overview of common web attacks: classification, concrete examples (payloads and vulnerable snippets), step-by-step attack flows, and visualizations drawn with TikZ suitable for Overleaf. It also covers detection, mitigation, and recommended secure coding practices.

Document history

Version	Date	Notes
1.0	2025-10-14	Initial comprehensive draft with examples and basic visuals.
1.1	2025-10-14	Added timeline, clearer TikZ sequence diagrams, payload table and compilation notes

Contents

1 Introduction **2**

2 Classification of Web Attacks **2**

3 Detailed Examples and Code **2**

3.1 Cross-Site Scripting (XSS) 2
3.2 SQL Injection 3
3.3 Cross-Site Request Forgery (CSRF) 3
3.4 Server-Side Request Forgery (SSRF) 4
3.5 Insecure Direct Object Reference (IDOR) 4

4 Visualizations (TikZ) **4**

4.1 XSS sequence diagram (clearer) 5
4.2 SQL Injection: annotated flow 5
4.3 Unified attack timeline 5

5 Detection and Logging **6**

6 Mitigation Checklist (Developer Guide) **6**

7 Appendix: Useful Payloads and Resources **6**

8 Compilation notes **6**

1 Introduction

Web applications face a broad range of attacks that target user input processing, authentication, session management, business logic, and the environment (servers, databases, third-party services). Understanding both how attacks work and how to visualize them helps developers and security teams mitigate risks.

2 Classification of Web Attacks

Category	**Description and examples**
Injection	SQL Injection, NoSQL Injection, Command Injection (RCE), LDAP injection.
Cross-Site Scripting (XSS)	Stored, Reflected, DOM-based XSS.
Cross-Site Request Forgery (CSRF)	Unauthorized state-changing requests using victim's credentials.
Broken Authentication	Credential stuffing, session fixation, weak session management.
Insecure Direct Object Reference (IDOR)	Accessing objects by predictable IDs.
Server-Side Request Forgery (SSRF)	Forcing server to make internal/remote requests.
Local/Remote File Inclusion (LFI/RFI)	Including system files or remote resources.
Security Misconfiguration	Unpatched software, excessive permissions, exposed admin panels.
Sensitive Data Exposure	Leaking PII, weak encryption or no TLS.
Denial of Service (DoS)	Resource exhaustion via floods or expensive operations.
Supply Chain Attacks	Compromised libraries or CI/CD pipelines.

3 Detailed Examples and Code

Each sub-section includes a minimal vulnerable code snippet and a demonstration payload. For each example we include a small "attack steps" list and a short mitigation checklist.

3.1 Cross-Site Scripting (XSS)

Vulnerable snippet (PHP):

```

<?php
// search.php
$q = $_GET['q'] ?? '';
echo "<h1>Search␣results␣for:␣" . $q . "</h1>";
?>
```

Listing 1: Vulnerable PHP snippet that echoes user input directly

Attack payload (reflected XSS):

```
?q=<script>alert('XSS')</script>
```

Attack steps (reflected XSS):

1. Attacker crafts a URL containing the payload.
2. Victim clicks or is redirected to the URL.
3. Server reflects the payload into the response; browser executes it.
4. Attacker may steal session cookies or perform actions on behalf of the victim.

Mitigation:

- Properly escape output (e.g., `htmlspecialchars` in PHP).
- Use Content Security Policy (CSP).
- Validate and sanitize inputs server-side and client-side.

3.2 SQL Injection

Vulnerable snippet (Node.js + MySQL):

```
// app.js
app.get('/user', (req, res) => {
  const id = req.query.id;
  db.query("SELECT * FROM users WHERE id = " + id, (err, rows) => {
    res.json(rows);
  });
});
```

Listing 2: Vulnerable query concatenation

Attack payloads:

```
?id=1 OR 1=1
?id=1; DROP TABLE users; --
```

Attack steps:

1. Attacker identifies an input that is used directly in SQL.
2. They craft payloads that alter the query logic.
3. The database returns unexpected results or performs destructive actions.

Mitigation:

- Use parameterized queries / prepared statements.
- Use least-privilege database accounts.
- Validate input types and lengths.

3.3 Cross-Site Request Forgery (CSRF)

Vulnerable snippet (HTML form):

```

<form action="/transfer" method="POST">
  <input name="amount" value="1000">
  <input name="to" value="attacker-account">
  <input type="submit">
</form>
```

Listing 3: Bank transfer form that lacks CSRF protection

Attack scenario: If a logged-in user visits an attacker's page that auto-submits this form via JavaScript, the transfer runs using the user's session.

Mitigation:

- Use anti-CSRF tokens validated server-side.
- Require same-site cookies or set SameSite=strict/lax.
- Use double-submit cookie pattern.

3.4 Server-Side Request Forgery (SSRF)

Vulnerable snippet (Python/Flask):

```
@app.route('/fetch')
def fetch():
  url = request.args.get('url')
  r = requests.get(url)
  return Response(r.content, mimetype=r.headers.get('Content-Type'))
```

Listing 4: Image fetch proxy that accepts an image URL and returns it

Attack payload: Request internal endpoints: `/fetch?url=http://127.0.0.1:2375/containers/json`

Mitigation:

- Whitelist allowed domains.
- Block localhost and private IP ranges.
- Use timeouts and limit redirects.

3.5 Insecure Direct Object Reference (IDOR)

Vulnerable snippet (Express.js):

```
app.get('/download/:id', (req, res) => {
  const id = req.params.id;
  res.sendFile('/data/' + id + '.pdf');
});
```

Listing 5: Return file by id without authorization check

Attack: Change the URL `/download/123` to `/download/124` to access another user's document.

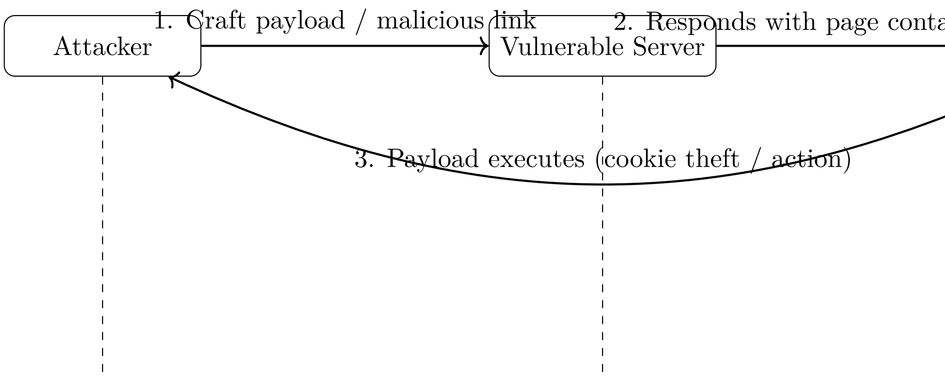
Mitigation:

- Enforce authorization checks for object access.
- Use unpredictable identifiers (GUIDs) and map them to owners.

4 Visualizations (TikZ)

Below are clearer, more detailed diagrams that visualize an XSS flow, an SQL Injection attack chain, and a unified attack timeline. Paste into Overleaf and compile with PDFLaTeX. These diagrams include legends and numbered steps.

4.1 XSS sequence diagram (clearer)

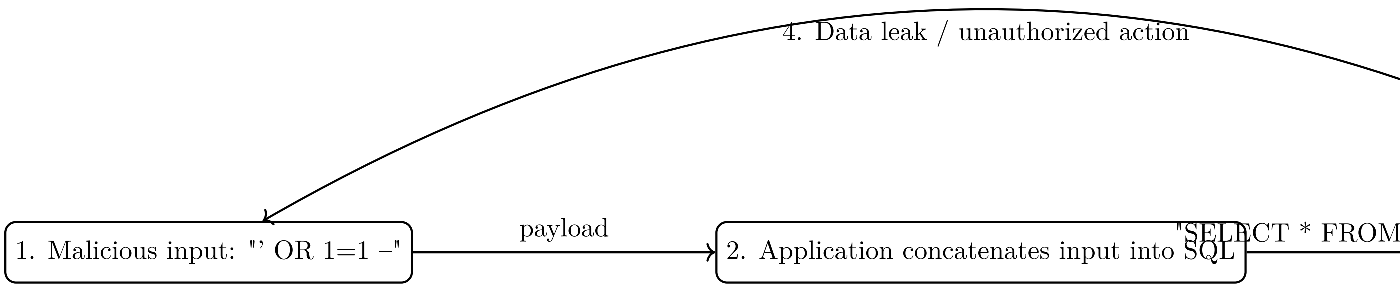


Attacker
1. Craft payload / malicious link
Vulnerable Server
2. Responds with page conta
3. Payload executes (cookie theft / action)

Legend: arrows show message flow; dashed lines = lifelines; numbered steps describe order.

Figure 1: Sequence showing how a reflected/stored XSS propagates from attacker to victim.

4.2 SQL Injection: annotated flow



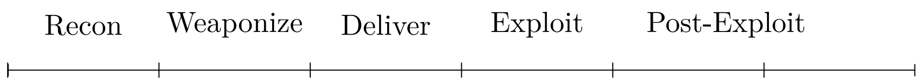
4. Data leak / unauthorized action
1. Malicious input: "' OR 1=1 –"
payload
2. Application concatenates input into SQL
"SELECT * FROM

Mitigation highlights: Use prepared statements, validate numeric IDs,
limit error messages, and use DB roles with least privilege.

Figure 2: Annotated SQL Injection flow with clear step numbers and mitigation note.

4.3 Unified attack timeline

This timeline visualizes the typical phases of a successful attack: Reconnaissance, Weaponization, Delivery, Exploitation, and Post-Exploitation/Impact.



Example: Recon = fingerprinting app; Weaponize = craft SQLi payload; Deliver = malicious form input or link; Exploit = obtain data or RCE; Post-Exploit = persistence, data exfiltration.

Figure 3: High-level attack timeline (typical stages).

5 Detection and Logging

- Log all input validation failures and unusual parameter patterns.
- Alert on repeated SQL error messages, unusual query patterns, or high error rates.
- Use web application firewalls (WAF) to detect common payload signatures.
- Monitor for new external hosts that your application requests (helps detect SSRF).

6 Mitigation Checklist (Developer Guide)

1. Validate input: types, lengths, formats.
2. Escape output according to context (HTML, JS, URL, SQL).
3. Use parameterized queries / prepared statements.
4. Implement proper authentication and session management (rotate, secure, httpOnly, SameSite).
5. Apply least privilege to DB accounts and file permissions.
6. Use CSP and other response headers (HSTS, X-Content-Type-Options).
7. Scan dependencies and pin versions; run SCA tools.
8. Harden server configuration and disable unused services.
9. Maintain an incident response plan and regular backups.

7 Appendix: Useful Payloads and Resources

Table 2: Common payloads (abbreviated) and usage notes

Attack	Example payload / notes
XSS (reflected)	`<script>alert(1)</script>` – test for reflected output.
XSS (stored)	Use POST to store payload in comment fields.
SQLi (boolean)	`' OR '1'='1` – checks if input changes boolean response.
SQLi (union)	`' UNION SELECT NULL, version(), user() - ` – enumerate columns.
CSRF	Hidden form or auto-posting fetch from victim origin.
RCE / Command inj.	`; ls -la` or `‘whoami‘` where shell interpolation exists.
SSRF	`http://127.0.0.1:2375/` – target internal services.
IDOR	Sequential numeric IDs like 1001 -> 1002 exploited.

8 Compilation notes

- This document uses TikZ and compiles with PDFLaTeX on Overleaf.
- If compilation is slow, disable TikZ previews temporarily or compile on a stronger machine.
- Fonts: lmodern is included; increase memory if TikZ uses complex libraries.

Note: This document is intended for educational purposes; always have explicit permission before testing systems you do not own.