

Application Security

Simulation

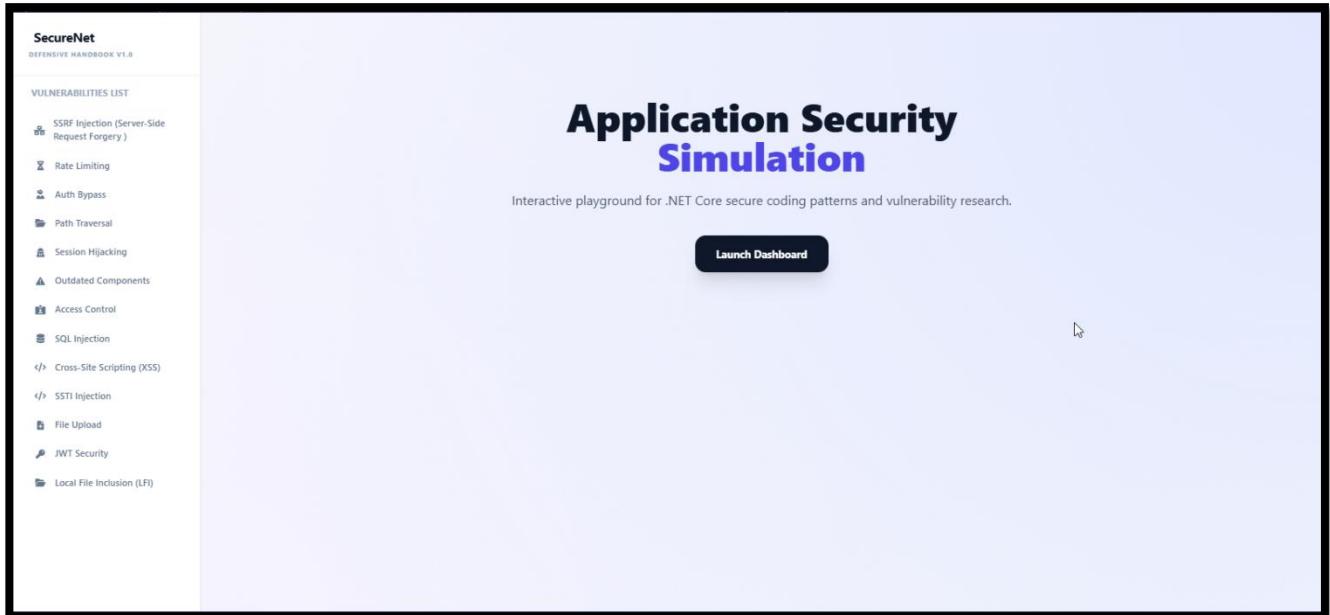
Developed by BRPL Cyber security Team

1. Application Security Simulator

Live Link :- <https://harvinder300.github.io/ApplicationSecuritySimulatorV1.0/>

Github Link :- <https://github.com/harvinder300/ApplicationSecuritySimulatorV1.0.git>

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



3. Click on the Launch Dashboard

The screenshot shows the "Vulnerabilities Dashboard". The title is "Vulnerabilities Dashboard" and it says "Master the OWASP Top 10 with 13 live examples.". There is a circular icon with a pentagon inside. Below are nine cards, each representing one of the OWASP Top 10 vulnerabilities:

- 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 flags.
- Outdated Components**: Definition: Using libraries with known CVEs (e.g., old Newtonsoft.Json versions).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.

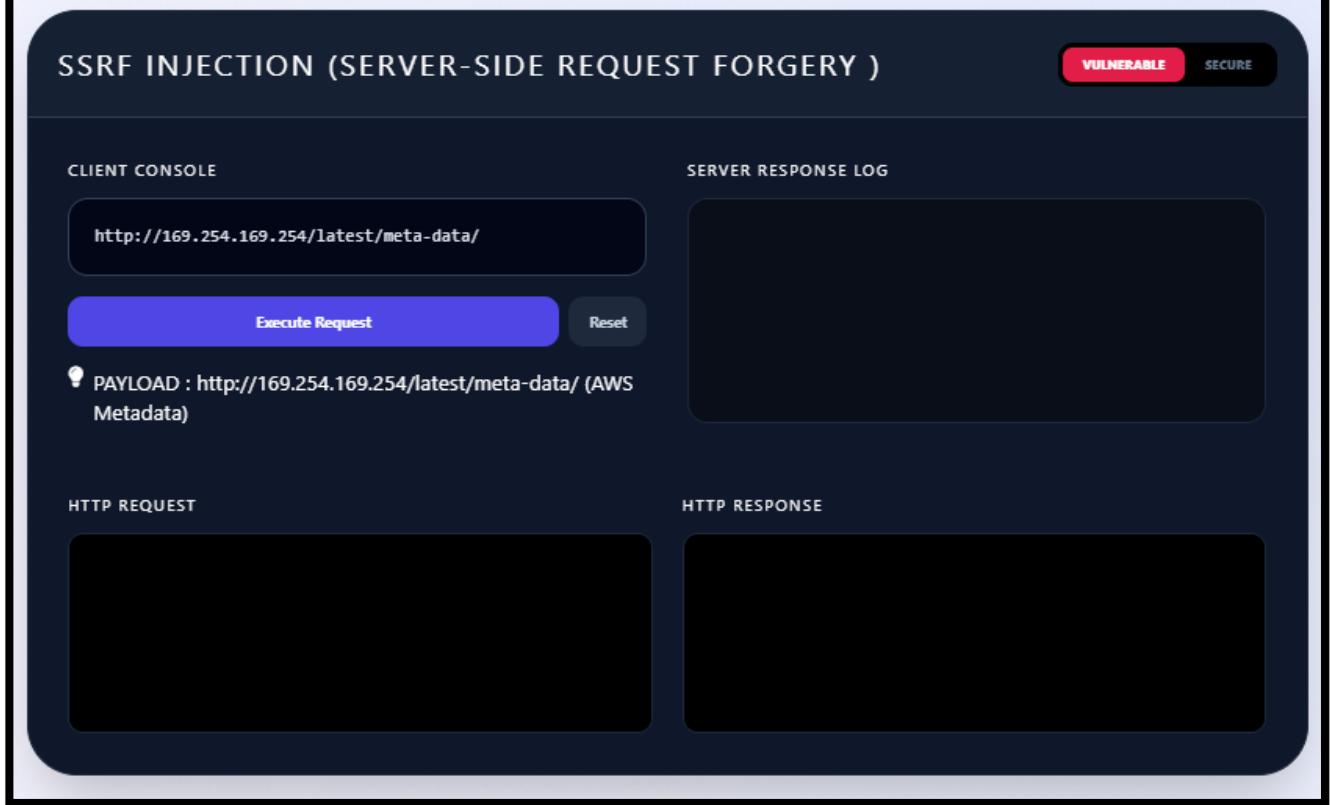
4. Topic 1: SSRF (Server Side Request Forgery)

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



5. 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/**

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/`

6. 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".

The "CLIENT CONSOLE" section contains a text input field with the URL "http://169.254.169.254/latest/meta-data/" and a blue "Execute Request" button.
The "HTTP REQUEST" section shows the raw POST request:

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

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


The "HTTP RESPONSE" section displays the server's 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/
```


The "SERVER RESPONSE LOG" section shows the log entry:

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


A status indicator at the bottom right shows a red dot next to "SSRF EXPLOITED".

7. 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 SSRF INJECTION (SERVER-SIDE REQUEST FORGERY) interface with the "SECURE" button highlighted in green at the top right.

The "CLIENT CONSOLE" section contains a text input field with the URL "http://169.254.169.254/latest/meta-data/" and a blue "Execute Request" button.
The "HTTP REQUEST" section shows the raw POST request:

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

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


The "HTTP RESPONSE" section displays the server's 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.
```


The "SERVER RESPONSE LOG" section shows the log entry:

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


A status indicator at the bottom right shows a green dot next to "SSRF BLOCKED".

8. 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>

9. 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

10. 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"}

11. 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."}

12. 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 Task<IAActionResult> to return a result, which is then checked against a 'LoginModel' object. The password is hashed using SHA-256 and compared with the provided password. If they match, an 'Ok(result)' response is returned. A callout box notes that attackers can automate thousands of requests per second to guess passwords.

```
"code-keyword">public "code-keyword">async "code-keyword">Task<"code-keyword">IAActionResult>
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.

On the right, under 'SECURE IMPLEMENTATION', is a snippet of Java code using the 'RateLimiter' interface. It adds a rate limiter to the application's configuration. The 'login_policy' is used to define a fixed window of 60 seconds with a permit limit of 5. A callout box suggests implementing rate limiting globally or on high-value endpoints like '/login'.

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

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

13. Topic 3 : Auth Bypass

The image shows a web-based tool for testing authentication bypass. The title is 'Auth Bypass'. At the top, there are tabs for 'OWASP:2025 – Authentication Failures' and 'Common Weakness Enumeration - CWE-287-306'. Below the tabs, there are two main sections: 'CLIENT CONSOLE' and 'SERVER RESPONSE LOG'. The 'CLIENT CONSOLE' section contains a text input field with placeholder 'Enter payload...', a blue 'Execute Request' button, and a 'Reset' button. A note below the input field says 'Try header: "X-Admin-Bypass: true"'. The 'SERVER RESPONSE LOG' section is currently empty. At the bottom, there are two large sections: 'HTTP REQUEST' and 'HTTP RESPONSE', both of which are also currently empty. A red 'VULNERABLE' button is located at the top right of the interface.

14. 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'.

15. 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'.

16. 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 a POST request to "/api/auth_bypass" with headers "Host: securenet.local" and "X-Admin-Bypass: true".
- HTTP RESPONSE:** Shows a 401 Unauthorized response with the date "Mon, 09 Feb 2026 12:40:17 GMT" and the message "Authentication required. Header ignored."
- PROTECTED:** A green circular icon with a white dot.

At the bottom left, a note says "Try header: 'X-Admin-Bypass: true'" with a lightbulb icon. Below the 'Execute Request' button are "Execute Request" and "Reset" buttons.

17. 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 note below states: "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 note below states: "Enforce a single point of authentication. Remove all debug flags."

18. 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...

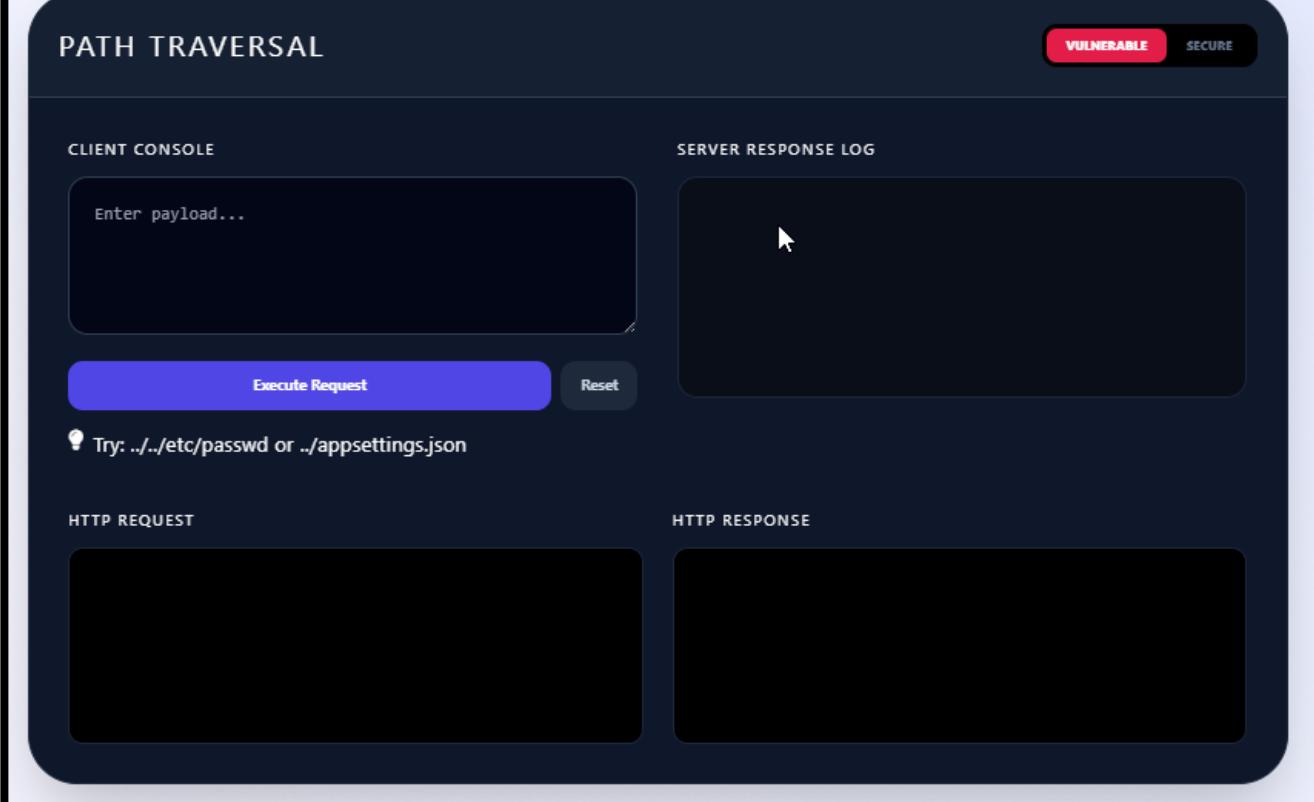
Execute Request Reset

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

SERVER RESPONSE LOG

HTTP REQUEST

HTTP RESPONSE



19. 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

SERVER RESPONSE LOG

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

● PATH TRAVERSAL EXPLOITED

HTTP REQUEST

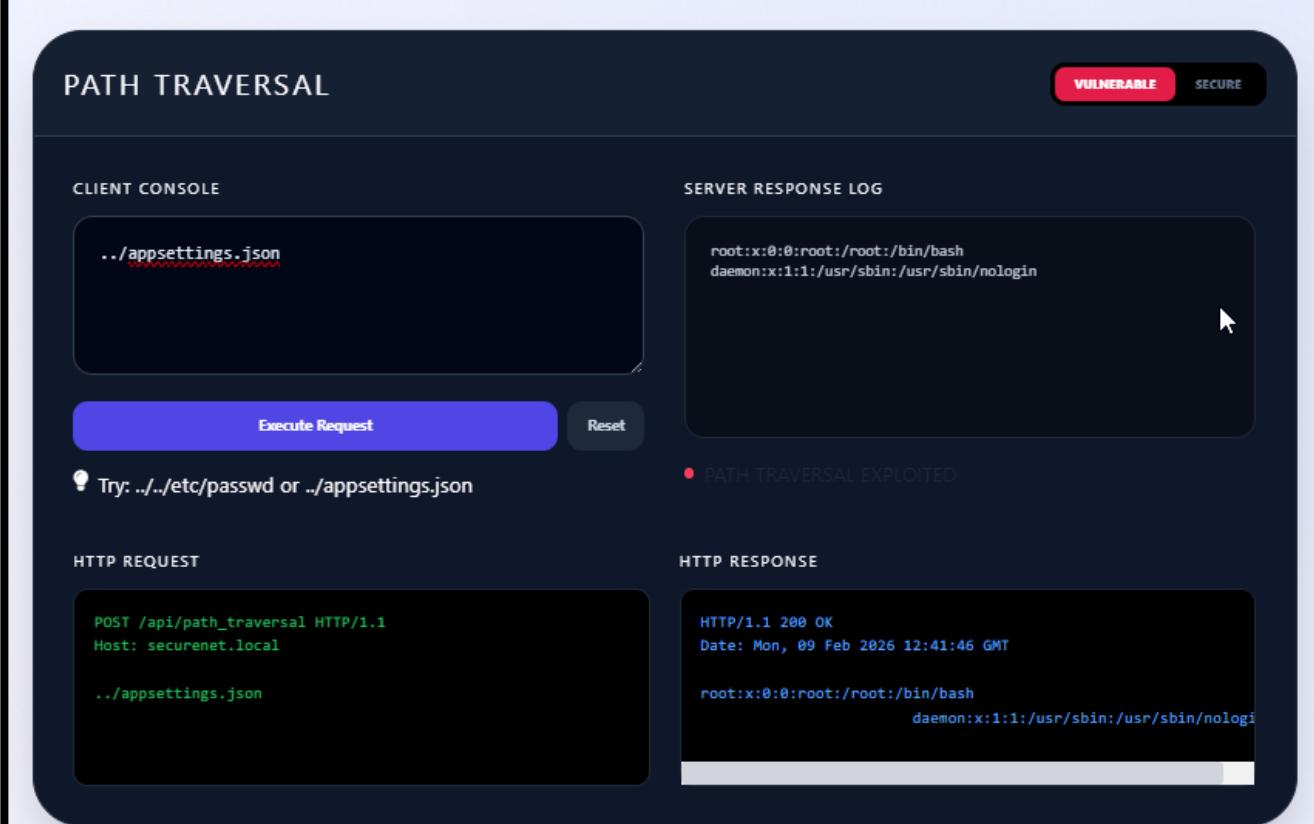
POST /api/path_traversal HTTP/1.1
Host: securenet.local

../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



20. 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 payload `..//appsettings.json`. Below it are two buttons: "Execute Request" (highlighted with a mouse cursor) and "Reset". A note below says "Try: ../../etc/passwd or ..//appsettings.json".

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

HTTP REQUEST: Shows the raw POST request sent to `/api/path_traversal`:
POST /api/path_traversal HTTP/1.1
Host: securenet.local

..//appsettings.json

HTTP RESPONSE: Shows the raw response:
HTTP/1.1 403 Forbidden
Date: Mon, 09 Feb 2026 12:42:13 GMT

Invalid file path.

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

The comparison highlights the difference in handling path traversal between a "VULNERABLE LOGIC" and a "SECURE IMPLEMENTATION".

VULNERABLE LOGIC: The code uses `Path.Combine(_rootPath, filename)` without proper validation, allowing directory traversal. A note states: "An attacker can read /etc/passwd by jumping directories."

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

SECURE IMPLEMENTATION: The code uses `Path.GetFileName(filename)` to get the file name only and then `Path.Combine(_rootPath, "safe_dir", fileNameOnly)` to ensure the final destination is within a safe directory. A note states: "Always strip path information and validate the final destination path."

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

An orange callout box on the left side of the page contains the text: "This is how you can prevent Path Traversal attacks".

22. 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 web-based application for demonstrating 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 blue, and a 'Reset' button. A tooltip '💡 Try JS snippet: document.cookie' is positioned near the input field. On the right, a 'SERVER RESPONSE LOG' section is shown as a large, empty black box. At the bottom, there are four sections: 'HTTP REQUEST' (empty), 'HTTP RESPONSE' (empty), and 'HTTP REQUEST' (empty again). The overall interface has a dark theme with blue highlights for interactive elements.

23. 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 the payload 'document.cookie' was entered in the client console. The 'VULNERABLE' button remains highlighted. In the 'CLIENT CONSOLE' section, the input field now contains 'document.cookie'. The 'HTTP REQUEST' section shows a POST request to '/api/session_hijacking' with the 'Host' header set to 'securenet.local'. The 'HTTP RESPONSE' section displays an 'HTTP/1.1 200 OK' status with a timestamp and an 'ALERT' message stating 'Cookie Exfiltrated! SessionId=12345; Path=/; Domain=securenet.local'. The 'SERVER RESPONSE LOG' section is still empty.

24. 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. Below the buttons, the title "SESSION HIJACKING" is displayed. On the left, under "CLIENT CONSOLE", a red box highlights the 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". On the right, under "SERVER RESPONSE LOG", the text "Cookie Access: [Empty/Filtered] (HttpOnly prevents JS access)" is shown. At the bottom, there are two sections: "HTTP REQUEST" and "HTTP RESPONSE". The "HTTP REQUEST" section contains the POST data: "POST /api/session_hijacking HTTP/1.1" and "Host: securenet.local", followed by "document.cookie". The "HTTP RESPONSE" section shows the server's response: "HTTP/1.1 200 OK" and "Date: Mon, 09 Feb 2026 12:43:41 GMT", along with the same "Cookie Access" message.

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

The screenshot compares two snippets of code. On the left, under "VULNERABLE LOGIC", the code is:

```
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 below states: "Lack of HttpOnly allows XSS to steal cookies."

On the right, under "SECURE IMPLEMENTATION", the code is:

```
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 below states: "Always use HttpOnly to prevent JS access and Secure to enforce HTTPS."

26. 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" panels, both of which are currently empty.

27. 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" panel now contains the payload "Newtonsoft.Json 9.0.1". The "SERVER RESPONSE LOG" panel displays the message "CVE-2017-15708 DETECTED: Package Newtonsoft.Json 9.0.1 is vulnerable to RCE via Object Injection." The "HTTP REQUEST" panel shows a POST request to "/api/outdated_components" with the host "securenet.local" and the payload "Newtonsoft.Json 9.0.1". The "HTTP RESPONSE" panel shows a successful HTTP/1.1 200 OK response with the date "Mon, 09 Feb 2026 12:48:58 GMT" and the message "CVE-2017-15708 DETECTED: Package Newtonsoft.Json 9.0.1 is vulnerable to RCE via Object I".

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

The screenshot shows a web-based tool for managing outdated components. At the top, there's a navigation bar with tabs for 'A03:2025 – Software Supply Chain Failures' and 'Common Weakness Enumeration - CWE-1104'. Below the navigation is a main panel titled 'OUTDATED COMPONENTS'. The panel has two main sections: 'CLIENT CONSOLE' on the left and 'SERVER RESPONSE LOG' on the right. In the 'CLIENT CONSOLE' section, there's a text input field containing 'Newtonsoft.Json 9.0.1' with a red underline, indicating it's a vulnerable version. Below the input are two buttons: 'Execute Request' (in blue) and 'Reset'. A note below the input says 'Try: Newtonsoft.Json 9.0.1'. In the 'SERVER RESPONSE LOG' section, the text reads 'Package List Scanned: All versions within support lifecycle.' At the very top of the main panel, there are two status indicators: 'VULNERABLE' (in grey) and 'SECURE' (in green, which is highlighted). The overall interface is dark-themed.

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

This screenshot compares two snippets of XML code related to package references. On the left, under 'VULNERABLE LOGIC', the code includes a package reference for 'Newtonsoft.Json' with a specific version ('9.0.1'). A callout bubble from this section states: 'Old package versions contain public exploits like RCE.' On the right, under 'SECURE IMPLEMENTATION', the same code is shown, but the version is explicitly set to '13.0.1'. A callout bubble from this section states: 'Regularly run vulnerability scanners like 'dotnet list package -vulnerable''. A cursor icon is visible above the 'SECURE IMPLEMENTATION' code.

```
<"code-keyword">PackageReference "code-keyword">Include="Newtonsoft">.Json" "code-keyword">Version="9.0.1" />
```

```
<"code-keyword">PackageReference "code-keyword">Include="Newtonsoft">.Json" "code-keyword">Version="13.0.1" />
```

30. 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

VULNERABLE

SECURE

ACCESS CONTROL

CLIENT CONSOLE

Enter payload...

Execute Request

SERVER RESPONSE LOG

HTTP REQUEST

HTTP RESPONSE

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

Reset

31. 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

ACCESS CONTROL

VULNERABLE

SECURE

CLIENT CONSOLE

101

Execute Request

SERVER RESPONSE LOG

{"id": 101, "owner": "Yourself", "balance": "\$50"}

Reset

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

HTTP REQUEST

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

101

HTTP RESPONSE

```
HTTP/1.1 200 OK  
Date: Mon, 09 Feb 2026 12:50:56 GMT
```

{"id": 101, "owner": "Yourself", "balance": "\$50"}

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" (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: Shows a POST request to /api/bac with Host: securenet.local. The body of the request contains the value "102".

HTTP RESPONSE: Shows an HTTP/1.1 200 OK response with Date: Mon, 09 Feb 2026 12:51:40 GMT. The response body is {"id": 102, "owner": "Admin", "balance": "\$9,999,999", "ssn": "XXX-XX-0000"}.

A tooltip message "Change ID: 101 (Your Account) to 102 (Admin Account)" is displayed near the "Execute Request" button.

32. 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" (blue) and "Reset".

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

HTTP REQUEST: Shows a POST request to /api/bac with Host: securenet.local. The body of the request contains the value "101".

HTTP RESPONSE: Shows an HTTP/1.1 200 OK response with Date: Mon, 09 Feb 2026 12:52:56 GMT. The response body is {"id": 101, "owner": "Yourself", "balance": "\$50"}.

A tooltip message "Change ID: 101 (Your Account) to 102 (Admin Account)" is displayed near the "Execute Request" button.

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."}
```

33. 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)">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.

34. Topic 8 : SQL Injection vulnerability

SQL Injection

Definition : Manipulate database queries through untrusted input parameters.

OWASP:2025 – Injection CWE-89

SQL INJECTION

VULNERABLE SECURE

CLIENT CONSOLE

Enter payload...

Execute Request Reset

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

HTTP REQUEST

HTTP RESPONSE

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

SQL INJECTION

VULNERABLE SECURE

CLIENT CONSOLE

1 OR 1=1

Execute Request Reset

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

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...

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

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

- CLIENT CONSOLE:** A text input field containing the payload "1 OR 1=1". Below it are two buttons: "Execute Request" (highlighted with a cursor) and "Reset".
- SERVER RESPONSE LOG:** A text area displaying the message "Query executed safely. No records found."
- STATUS INDICATOR:** A green dot followed by the text "QUERY SAFE".
- 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.
```

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

The screenshot compares two pieces of code for handling user input in SQL queries:

- VULNERABLE LOGIC:** Shows a string concatenation vulnerability where the query is constructed as "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 where the query is constructed as "SELECT * FROM Users WHERE ID = @id;". The command part of the code is annotated with ".Parameters" and ".AddWithValue("@id", id);". A callout box notes: "Parameterized queries treat all input as literal data."

38. 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 for testing Cross-Site Scripting (XSS). At the top right, there are two buttons: "VULNERABLE" (highlighted in red) and "SECURE".
The main interface is divided into several sections:

- CLIENT CONSOLE:** A text input field containing "Enter payload...". Below it are two buttons: "Execute Request" (in blue) and "Reset". A tooltip below the input field says: "Try XSS payloads like <script>alert('XSS')</script>".
- SERVER RESPONSE LOG:** A large empty box where the results of the request will be displayed.
- HTTP REQUEST:** A large empty box where the raw HTTP request is shown.
- HTTP RESPONSE:** A large empty box where the raw HTTP response is shown.

A cursor is visible at the bottom right of the interface.

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

This screenshot shows a browser window with a modal dialog and a background XSS testing interface.
The modal dialog from "127.0.0.1:5500 says" displays the message "XSS" and has an "OK" button.
Below the modal, the text "Definition : User input is rendered back to the browser without proper output encoding." is visible.
The OWASP:2025 – Injection status is shown as "INJECTION".
The background shows the XSS testing interface with the following details:

- CLIENT CONSOLE:** Contains the payload: <script>alert('XSS')</script>.
- SERVER RESPONSE LOG:** Displays the response "Welcome".
- HTTP REQUEST:** Shows the POST request: "POST /api/xss HTTP/1.1" and "Host: securinet.local" followed by the payload.
- HTTP RESPONSE:** Shows the response: "HTTP/1.1 200 OK", "Date: Mon, 09 Feb 2026 12:55:36 GMT", and the HTML content: <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

The screenshot shows a web-based testing tool for XSS attacks. At the top, there's a header "CROSS-SITE SCRIPTING (XSS)" with two status indicators: "VULNERABLE" and "SECURE".

CLIENT CONSOLE: A text input field containing the payload "<script>alert('XSS')</script>". Below it are two buttons: "Execute Request" (highlighted with a cursor icon) and "Reset". A note below says "Try XSS payloads like <script>alert('XSS')</script>".

SERVER RESPONSE LOG: Displays the response "Welcome" followed by the injected payload "<script>alert('XSS')</script>". To the right, a green circular icon indicates "XSS BLOCKED".

HTTP REQUEST: Shows a POST request to "/api/xss" with the following headers: "POST /api/xss HTTP/1.1" and "Host: securenet.local". The body of the request contains the XSS payload "<script>alert('XSS')</script>".

HTTP RESPONSE: Shows the response "HTTP/1.1 200 OK" with the date "Date: Mon, 09 Feb 2026 12:56:11 GMT". The body of the response includes the injected script: "<div class='xss-output-container'><h1>Welcome</h1><script>alert('XSS');</script></div>".

39. XSS : Below is the secure and vulnerable logic

VULNERABLE LOGIC: Shows the following C# code:

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

A callout box notes: "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: Shows the following C# code:

```
"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");
}
```

A callout box notes: "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

The screenshot shows a web-based tool for testing SSTI. At the top, there's a header "SERVER-SIDE TEMPLATE INJECTION (SSTI)" with two status indicators: "VULNERABLE" (in a grey box) and "SECURE" (in a green box).

The "CLIENT CONSOLE" section contains a text input field with the placeholder `{{6*6}}` and two buttons: "Execute Request" (blue) and "Reset". Below the input field is a note: "Try: {{7*7}} or {{6*6}}".

The "SERVER RESPONSE LOG" section displays the message "Template rendering blocked or safely escaped."

The "HTTP REQUEST" section shows a POST request to "/api/ssti" with the host "securenet.local". The body of the request contains the template `{{6*6}}`.

The "HTTP RESPONSE" section shows a successful response (HTTP/1.1 200 OK) from "Mon, 09 Feb 2026 13:25:46 GMT". The response body also says "Template rendering blocked or safely escaped."

A status indicator "● TEMPLATE SAFE" is located between the Response Log and the Response sections.

43 . SSTI : Below is the secure and vulnerable logic

The comparison highlights two approaches to prevent SSTI:

- VULNERABLE LOGIC:** Shows a code snippet where a template string is constructed by concatenating user input into a larger string. A callout notes: "Inputting template tags like {{7*7}} allows remote execution."
- SECURE IMPLEMENTATION:** Shows a code snippet where the template is defined as a static string and data is passed through a dedicated object model. A callout notes: "Use static templates and pass data through a dedicated object model."

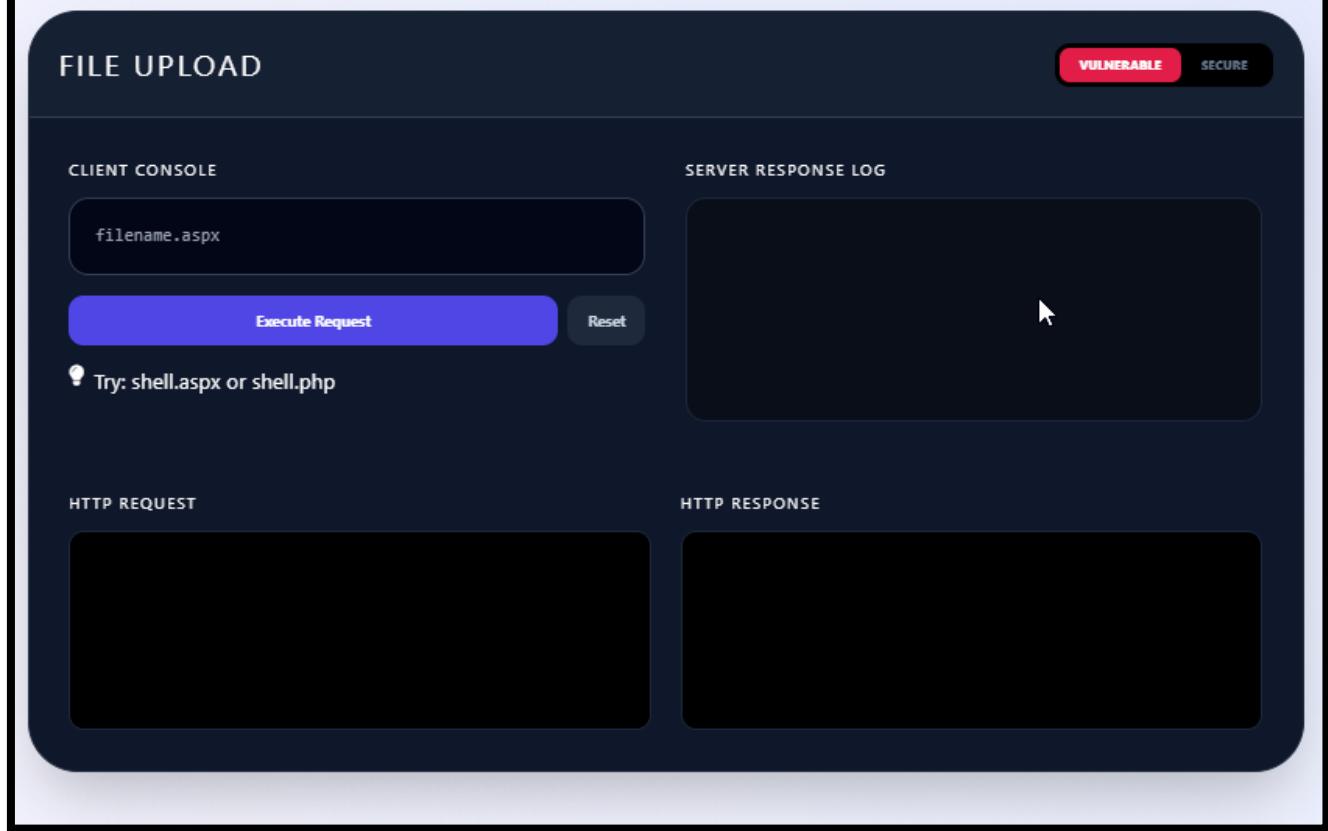
44. Topic 11 :File Upload

File Upload

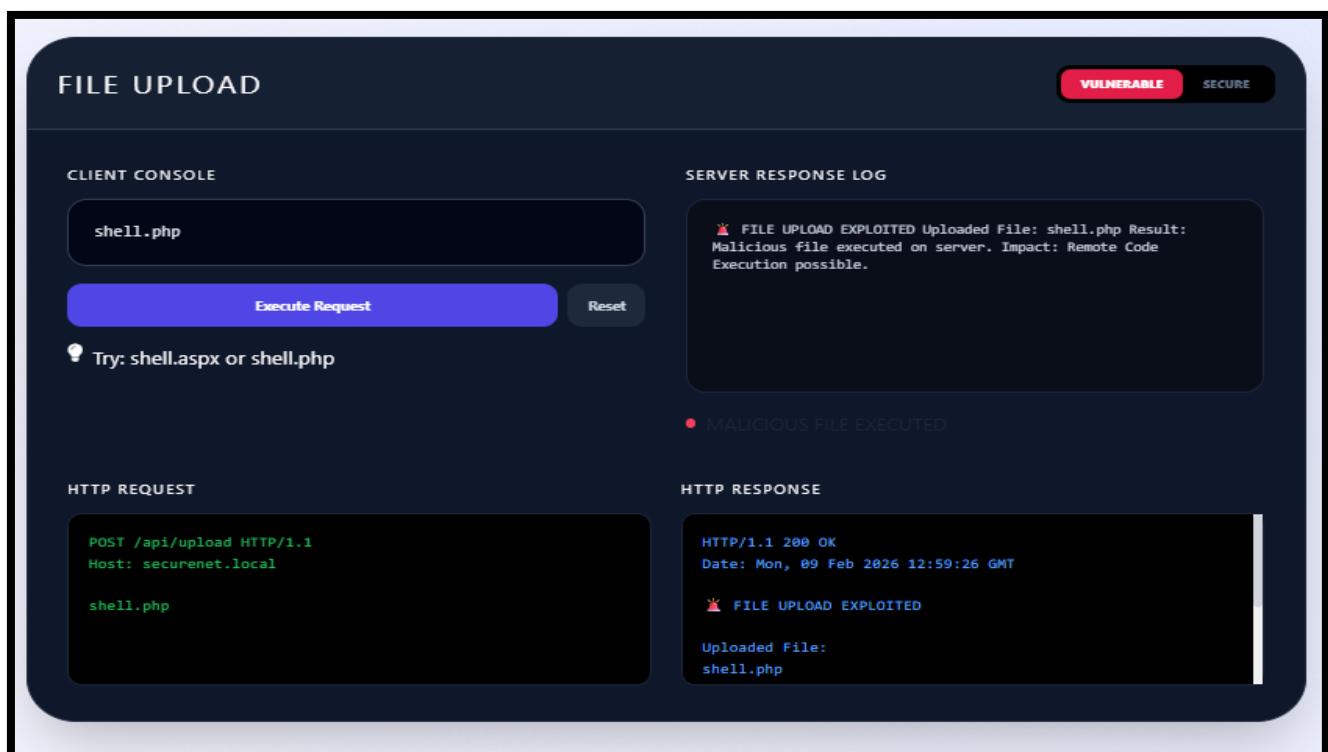
Definition : Execution of arbitrary code through malicious file uploads.

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 accept the .php and .aspx

The screenshot shows a web-based file upload interface. At the top right, there are two buttons: 'VULNERABLE' (in grey) and 'SECURE' (in 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' (grey).

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 Host: 'securenet.local' and a file named 'shell.php'.

HTTP RESPONSE: Returns an HTTP/1.1 400 Bad Request with Date: Mon, 09 Feb 2026 12:59:44 GMT. It also displays the 'UPLOAD BLOCKED' message and 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

The screenshot shows the 'JWT SECURITY' tool interface. At the top right, there are two buttons: 'VULNERABLE' (highlighted in red) and 'SECURE'. Below this, the interface is divided into several sections:

- CLIENT CONSOLE:** A text input field containing "Enter payload...". Below it are two buttons: "Execute Request" (in blue) and "Reset".
- SERVER RESPONSE LOG:** An empty text area.
- HTTP REQUEST:** An empty text area.
- HTTP RESPONSE:** An empty text area.
- Alert:** A warning message: "💡 Try weak-secret signed token or change role to admin using HS256".

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

The screenshot shows the 'JWT SECURITY' tool interface after entering the payload. The 'VULNERABLE' button is still highlighted in red. The interface sections are as follows:

- CLIENT CONSOLE:** The input field now contains the payload: {"user": "attacker", "role": "admin"}.
- SERVER RESPONSE LOG:** Displays a success message: "⚠️ JWT FORGERY SUCCESS Algorithm: HS256 Weak secret used. Decoded Claims: { "user": "attacker", "role": "admin" } Impact: Privilege escalation successful." Below this, a note says "● TOKEN FORGED".
- HTTP REQUEST:** Shows the POST request details: "POST /api/jwt HTTP/1.1" and "Host: securenet.local". The payload is also shown: {"user": "attacker", "role": "admin"}.
- HTTP RESPONSE:** Shows the server response: "HTTP/1.1 200 OK" and "Date: Mon, 09 Feb 2026 13:00:51 GMT". It also displays the forged token details: "⚠️ JWT FORGERY SUCCESS" and "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:** A log entry: "TOKEN REJECTED Reason: * Signature verification failed * Token forged without valid key Security: Strong secret enforced".
- HTTP REQUEST:** Shows the POST /api/jwt HTTP/1.1 request with Host: securonet.local and the same JSON payload.
- HTTP RESPONSE:** Shows the HTTP/1.1 401 Unauthorized response with Date: Mon, 09 Feb 2026 13:01:19 GMT. It includes a "TOKEN REJECTED" entry with the reason: "Signature verification failed".

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

The comparison highlights the differences in how tokens are generated and handled between a "VULNERABLE LOGIC" approach and a "SECURE IMPLEMENTATION".

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 recommends: "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, there's a navigation bar with tabs for 'VULNERABLE' and 'SECURE'. Below the tabs, there are two main sections: 'CLIENT CONSOLE' and 'SERVER RESPONSE LOG'. The 'CLIENT CONSOLE' section contains a text input field with placeholder text 'Enter payload...', a blue button labeled 'Execute Request', and a 'Reset' button. A note below the input field says 'Try: ../../etc/passwd or ./web.config'. The 'SERVER RESPONSE LOG' section is currently empty. Below these are 'HTTP REQUEST' and 'HTTP RESPONSE' sections, both of which are also empty. The overall interface is clean and modern, designed for penetration testing.

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

This screenshot shows the same LFI tool after a payload has been executed. The 'CLIENT CONSOLE' now displays the payload '.../web.config'. The 'SERVER RESPONSE LOG' section shows the output of the command, listing system users: 'root:x:0:0:root:/root:/bin/bash', 'daemon:x:1:1:daemon:/usr/sbin/nologin', and 'www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin'. Below the log, a red circular icon with a white dot and the text 'LFI EXPLOITED' are visible. The 'HTTP REQUEST' section shows a POST request to '/api/lfi' with 'Host: securenet.local' and the payload '.../web.config'. The 'HTTP RESPONSE' section shows a successful response with status 'HTTP/1.1 200 OK', date 'Mon, 09 Feb 2026 13:02:42 GMT', and the same user list as the log. The interface remains consistent with the first screenshot, maintaining a professional and user-friendly design.

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's a header with the title "LOCAL FILE INCLUSION (LFI)" and two status indicators: "VULNERABLE" (in red) and "SECURE" (in green). The main area is divided into several sections:

- CLIENT CONSOLE:** A text input field containing the path ".../web.config".
- Execute Request:** A blue button to trigger the request.
- Reset:** A small button to reset the form.
- SERVER RESPONSE LOG:** A text area displaying the response: "Access denied. Requested file is not on the allowlist."
- HTTP REQUEST:** A text area showing the raw HTTP POST request: "POST /api/lfi HTTP/1.1" and "Host: securenet.local" followed by the path ".../web.config".
- HTTP RESPONSE:** A text area showing the raw HTTP response: "HTTP/1.1 200 OK" and "Date: Mon, 09 Feb 2026 13:03:00 GMT" followed by the message "Access denied. Requested file is not on the allowlist."
- LFI BLOCKED:** A green indicator message at the bottom left.

55.LFI : Below is the secure and vulnerable logic

The screenshot compares two pieces of code: "VULNERABLE LOGIC" and "SECURE IMPLEMENTATION".

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 below this code 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 below this code states: "Only predefined filenames are allowed (allowlist). Directory traversal characters are never processed, preventing access to unauthorized files."

Thank you