SERVER-SIDE TEMPLATE INJECTION (SSTI)

1. Concept & Root Cause

- **Definition:** An attack where the attacker injects native template syntax into a web page, which the server then executes.
- **The Flaw:** Occurs when user input is **concatenated directly** into a template string rather than being passed as data.
    -
        - **Safe Way:** `$twig->render("Dear {name}", array("name" => $user_input));`
    - **Vulnerable Way:** `$twig->render("Dear ".$user_input);`
- **Mechanism:** Template engines mix fixed templates with volatile data. If an attacker controls the "template" part, they control the engine.

2. Impact

- **Catastrophic (Critical Severity):**
    - **Remote Code Execution (RCE):** Complete takeover of the backend server.
    - **Internal Access:** Pivot point to attack internal infrastructure.
- **Data Exfiltration:** Read sensitive files even if full RCE isn't achieved.

---

3. Attack Methodology

Phase 1: Detect

Finding the vulnerability often involves "fuzzing" or math injections.

- **Fuzzing:** Inject special template characters: `${{<%[%'"}}%\`.
    - *Indicator:* Server errors (stack traces) or blank pages.
- **Context 1: Plaintext Context** (Input is rendered as text)
    - **Test:** Inject a math operation.
    - **Payload:** `${7*7}` or `{{7*7}}`
    - **Result:** If the page renders `49`, the server executed your code.
- **Context 2: Code Context** (Input is inside an existing statement)
    - **Scenario:** `engine.render("Hello {{"+input+"}}")`
    - **Test:** Break out of the syntax.

- **Payload:** `}}<tag>` or `user}}<script>`
- **Result:** If the tag renders or the error changes, you have control.

Phase 2: Identify

Once confirmed, determine *which* engine is running (Jinja2, Twig, FreeMarker, etc.).

- **Error Messages:** Sending invalid syntax (e.g., `<%=foobar%>`) often forces the server to return the engine name in the error trace.
- **Polyglot Payloads (The Decision Tree):** Different engines handle math differently.
- Example: `{{7*'7'}}`
  - **Twig** returns: `49` (It does the math).
  - **Jinja2** returns: `7777777` (It repeats the string).

---

4. Prevention

- **Logic-Less Engines:** Use engines like **Mustache** that separate logic from presentation (preventing code execution).
- **Sandboxing:** Remove dangerous modules/functions from the template environment (though difficult to secure completely).
- **Containerization:** Deploy the template environment in a locked-down Docker container to limit impact if compromised.
- **Best Practice:** Never allow users to submit or edit template strings.
