Что такое SSTI (Server-Side Template Injection) и как она работает?

SSTI (Server-Side Template Injection) — это уязвимость, возникающая, когда злоумышленник может внедрить произвольный код в шаблон, который затем выполняется на стороне сервера. Шаблонизаторы (Jinja2, Twig, Smarty и др.) предназначены для динамического генерации HTML, подставляя данные в заранее подготовленные "заготовки". Если пользовательский ввод передается в шаблон без проверки, атакующий может "сломать" контекст шаблона и выполнить свой собственный код.

Как работает атака?

Представим уязвимое веб-приложение на Python с использованием фреймворка Flask

```
from flask import Flask, request, render_template_string  # <-- Часто используют render_template_string для динамических шаблонов
app = Flask(__name__)
@app.route('/hello')
def hello():
    name = request.args.get('name', 'World')  # Получаем значение 'name' из запроса
    # И это опасно: пользовательский ввод напрямую подставляется в шаблон
    template = f"<h1>Hello, {name}!</h1>"  # Или используют render_template_string(name)
    return render_template_string(template)
if __name__ == '__main__':
    app.run()
```

1. Нормальное использование: пользователь переходит на /hello?name=Alice. Сервер рендерит шаблон <h1>Hello, Alice!</h1>.
2. Обнаружение уязвимости: злоумышленник пытается "сломать" синтаксис шаблона. Он вводит простое выражение: /hello?name={{ 1+1 }}.
   - Сервер интерпретирует {{ 1+1 }} не как текст, а как код шаблонизатора Jinja2.
   - Результат: пользователь видит <h1>Hello, 2!</h1>. Это подтверждает, что ввод выполняется как код.
3. Эксплуатация: после подтверждения уязвимости атакующий определяет шаблонизатор (Jinja2 в данном случае) и переходит к выполнению опасных команд. В Jinja2 есть доступ к некоторым Python-объектам.

- Пример получения доступа к файловой системе:

/hello?name={{ config.__class__.__init__.__globals__['os'].popen('ls /').read() }}

  - Этот сложный payload "пробивается" от объекта config к встроенному модулю os и выполняет команду ls /, выводя список файлов в корневой директории сервера.
- Угроза: таким образом можно читать любые файлы (включая конфиги с паролями к БД), выполнять команды, устанавливать бэкдоры.

Реальные угрозы SSTI:

- Выполнение произвольных команд (RCE): полный контроль над сервером.
- Чтение конфиденциальных данных: доступ к файлам с паролями, ключами, базе данных.
- Компрометация всей системы: с сервера-жертвы атака может распространиться на другие внутренние системы сети.
- Дефейс сайта: изменение внешнего вида ресурса.

Сравнение с XSS: важно не путать. При XSS вредоносный код (обычно JavaScript) выполняется в браузере *жертвы*. При SSTI код выполняется непосредственно *на сервере*, что делает эту атаку гораздо более опасной.

Как защититься от SSTI-атак?

1. Использование "чистых" шаблонов: самый надежный метод. Всегда использовать статические шаблоны с заранее определенными переменными. Данные должны только *подставляться* в шаблон, а не быть его частью.

- Правильно (на Flask):

```
# hello.html - отдельный файл шаблона
# <h1>Hello, {{ name }}!</h1>
@app.route('/hello')
def hello():
    name = request.args.get('name', 'World')
    return render_template('hello.html', name=name)  # Данные передаются отдельно
```

2. Санктизация (очистка) пользовательского ввода: если возможность динамических шаблонов необходима, нужно реализовать строгую фильтрацию, запретить использование специальных символов шаблонизатора ({ { } }, <% %> и т.д.). Однако это ненадежно, так как обходится сложными способами.

3. Запуск в песочнице (Sandbox): некоторые шаблонизаторы поддерживают режим "песочницы", где ограничен доступ к опасным функциям и классам. Но этот метод тоже часто оказывается уязвимым.

4. Принцип наименьших привилегий: запускать серверное приложение от пользователя с минимальными возможностями, чтобы даже в случае успешной атаки у злоумышленника был ограничен доступ к системе.