Server-side Template Injection

I. What is SSTI ?

Server-side template injection is a vulnerability where the attacker injects malicious input into a template to execute command on the server-side. This vulnerability occurs when invalid input is directly embedded into the template engine which can lead to RCE.

Template engines are designed to combine templates with a data model to produce result documents which helps populating dynamic data into web page. Some of the most popular template engines can be listed as the following:

- PHP: Smarty, Twigs
- JAVA: Velocity, Freemaker
- Python: Jinja, Mako, Tornado
- JS: Jade,Rage
- RUBY: Liquid, ERB

II. How does it work?

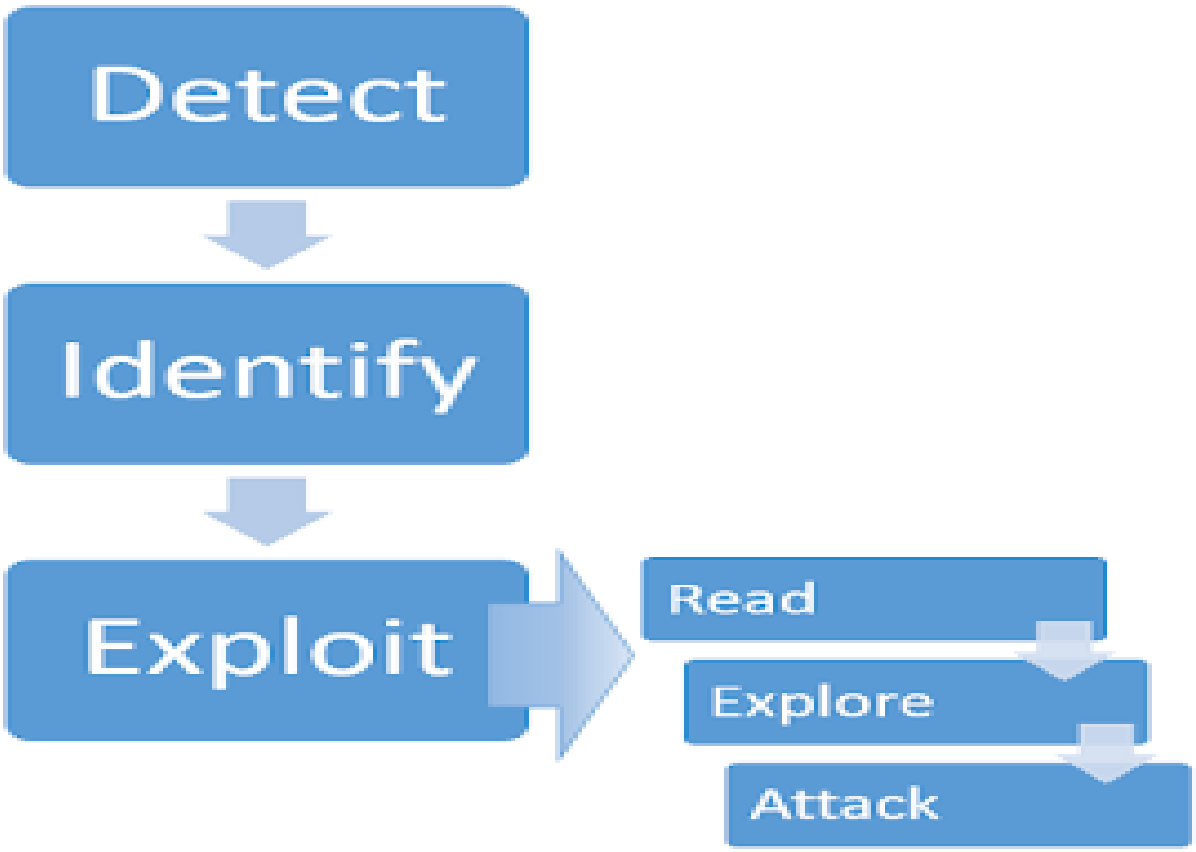
When attacker injects payload into template, then the command within it will be executed on the server-side

III. What is the impact of SSTI ?

The impact of SSTI vulnerability is generally critical, result in RCE by taking full control of the backend server. Even without RCE, the attacker may be able to read sensitive data on the server. There are also rare cases where an SSTI vulnerability isn’t critical, depending on the template engine.

IV. How to perform a SSTI attack ?

Identifying SSTI and crafting a susscessful attack typically involves the following high-level process.



1. Detect

To detect the vulnerabiity, use the payload as the value of the parameter which is a sequence of special characters such as the following:

Parameter = ${{<%[%'"}}%\.

If an exception is raised, this indicates that the injected template syntax is potentially being interpreted by the server in some way. This is signal that a vulnerability to SSTI may exist.

Plaintext context

Most template languges allow younto freely input content by using HTML tags directly or by using template's native syntax,Which will be rendered to GTML on the back-end server before reponse.

For example, consider a template that contains the following vulnerable code:

render('Hello ' + username)

During auditing, we might test for server-side template injection by requesting a URL such as:

```
http://vulnerable-website.com/?username=${7*7}
```

If the resulting output contains Hello 49, this shows that the mathematical operation is being evaluated server-side. This is a good proof of concept for a server-side template injection vulnerability.

Code context

In other cases, the vulnerability is exposed by user input being placed within a template expression. Such as:

```
greeting = getQueryParameter('greeting')
engine.render("Hello {{"+greeting+"}}", data)
```

2. Identify

To identify SSTI vul, use a payload composed of special character commonly used in template expressions to fuzz the template:

${{<%[%'"}}%\.

In case of a vulnerability, an erro message can be return or the exception can be raised by the server. This can be used to identify the template engine in use.

The following cheat sheet canbe used to identify the template engine in use:

3. Exploit

After detecting that a potential vulnerability exists and successfully identifying the template engine. We will follow the manuals for the specific template engine and exploit the vulnerability.

Payload for Exploit:
https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#jade--codepen

VI. How to prevent SSTI ?

Prevention for the SSTI vulnerability depend on the different template engines in use. There are 2 common suggestions to prevent this vulnerability:

- **Sanitization:** Sanitize user input before passing it into the templates ti minimize vulnerabilities from any malicious.
- **Sandboxing:** In case using risky characters is a business need, it is recommended to use a sandbox within a safe environment.

VII. Reference.

Blog: https://blog.cobalt.io/a-pentesters-guide-to-server-side-template-injection-ssti-c5e3998eae68

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection

Web Security Academy: https://portswigger.net/web-security/server-side-template-injection

https://portswigger.net/web-security/server-side-template-injection/exploiting