Server-side Template Injection

Overview

Template-engine: Template engine is a module that makes it possible to inject dynamic data into modern web pages. They allow developers to separate the logic of data processing and presentation code. They also provide rich functionality through Wikis, CMS, blogs. Template engines are often used to display dynamic data about web pages or about sending bulk emails.

- The engine templates come in many different types with many different types of programming languages:
    - PHP – Smarty, Twigs
    - JAVA – Velocity, FreeMaker
    - Python – JINJA, Mako, Tornado
    - JavaScript – Jade, Rage
    - Ruby - Liquid

Server-side template injections: is a type of vulnerability where an attacker can take advantage of the template's initial syntax to inject malicious payloads into the template, which are then executed on the server side. These attacks can occur when user input is connected directly to the template instead of being passed as data. This often allows an attacker to inject arbitrary template directives to manipulate the template engines to take complete control of the server. SSTI attacks are often mistaken for XSS. As a result, many SSTI vulnerabilities can be overlooked. Server-side template injection can be much more dangerous than Client-side injection.

Impact

Server-side template injection can expose websites to a variety of attacks that depend mainly on the template engine used and its correct usage. In certain cases, SSIT will not pose a security problem, but most of these attacks will cause extremely serious problems.

The most serious is that the attacker can Remote Code Execution (RCE), take control of the entire back end and use it to build other attacks on the internal infrastructure.

Even the failure to perform RCE would be the basis for other attacks, or it would be able to access memory on the server to read sensitive data.

Detect

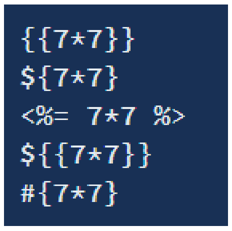
The simplest detection of SSTI is probably fuzzing the template by injecting special characters used in the template engine, such as the polyglot ${{<%[%'"}}%\.

To detect if the server is vulnerable, you need to detect the difference between the response and the normal data or the injected payloads and parameters.

If the server returns an error message, we can easily determine if the server is under attack or even what engine is running.

You can also find vulnerable servers if you expect to return the given payload from the server, but it is not returned or returned but lost some characters.

Usually, we will rely on these two types to detect:



```
{{7*7}}
${7*7}
<%= 7*7 %>
${{7*7}}
#{7*7}
```

- Plaintext context: This type is often easily confused as a simple XSS type because the given input is reflected directly in the responses. To distinguish SSTI from XSS, we can put operations inside of a template expression.
- Code contex: User input must be placed inside a template expression. The URL parameter must display the user input expression. Now if we change the user parameter to a different value (and have to close the tempalte expression) if and an error occurs during such testing then we can determine that the server is vulnerable.

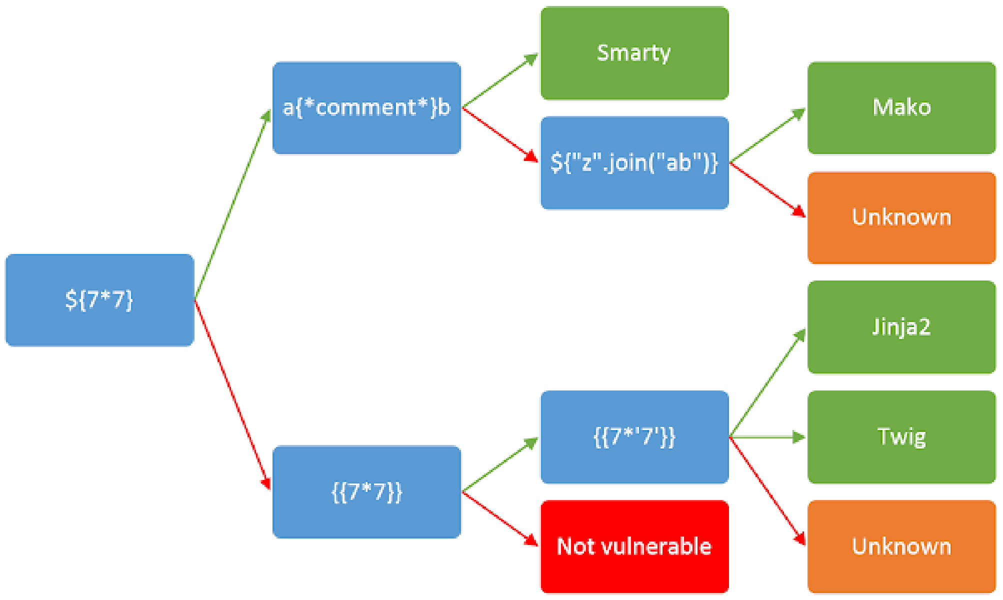
How to recognize template engine

After identifying the server-side that can be attacked, the next step is to check to see what type of template engine is used.

- If the server returns an error, then we can determine the type of template engine used inside. There are a few payloads that can cause errors:

`${}`	`{{}}`	`<%= %>`
`${7/0}`	`{{7/0}}`	`<%= 7/0 %>`
`${foobar}`	`{{foobar}}`	`<%= foobar %>`
`${7*7}`	`{{7*7}}`	``

- If the server does not return an error, then we must manually check that inject the different language-specific payloads and study how they are implemented by the template engine. We can use a decision tree like the following:



Exploit

After knowing that the server-side can be attacked and the template engine, the next step is how to exploit this vulnerability.

- Find and read all relevant documents. The types of documents required for reading are:
    - "For Template Authors" section covers basic syntax.
    - 'Security considerations' - chances are anyone who developed the app you are testing has not read this, and it may contain some helpful hints.
    - List of built-in methods, functions, filters, and variables.
    - List of extensions/plugins - some may be enabled by default
- After reading and digging through the theory, the next step we will practice understanding the environment to see what services you have access to. We can find the default objects provided by the template engine, and application specific. There are many pattern systems that specify certain objects or enumerate their properties and methods.
  If the object does not exist, then we must brute-force the boundary name. There are several commonly used variable names that are collected and released to the public through the Seclists collection.
  Objects can contain sensitive information and can vary between templates in an app, so this procedure should be applied to individual templates.
- Now that we have an idea and how to attack, we can look at all the functions of the vulnerability to be able to exploit. Some functions can be used to exploit application-specific features. After successful debriefing, we can read/write arbitrary files, included remote files, information disclosure and privilege escalation vulnerabilities.

Prevent

The best way to prevent server-side template injection is to not allow any user to modify or submit a new template.

One of the simplest is to always use a "logic-less" template engine. Separating logic from presentation can greatly reduce exposure to template-based attacks.

Another way is to execute user code only in a sandboxed environment where dangerous modules and functions are completely removed. However, the sandbox's unreliable code is easy to bypass and very difficult.

Finally, we can implement the acceptance of arbitrary code execution, but we can apply our own sandbox by deploying the template environment in a secure container.

References

- https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection#generic
- https://portswigger.net/research/server-side-template-injection
- https://portswigger.net/web-security/server-side-template-injection#identify
- https://owasp.org/www-pdf-archive/Owasp_SSTI_final.pdf