**Khai thác lỗ hổng File Upload với Race Conditions (Challenge)**

Chắc hẳn những lỗ hổng liên quan tới file upload đã không còn xa lạ gì. Việc cho phép người dùng tải file lên hệ thống mà không kiểm tra đầy đủ những thứ như phần mở rộng, loại file, nội dung, tên file và kích thước sẽ dẫn tới những hệ quả vô cùng nghiêm trọng. Theo đó, với kịch bản tệ nhất, kẻ tấn công có thể chiếm được quyền điều khiển của máy chủ.

Trong quá trình làm việc mình đã gặp một case khá thú vị khi mà lập trình viên đã xử lý hết mọi vấn đề đã nhắc tới ở trên nhưng vẫn có gì đó lạ lắm… Mình đã dựng lại một challenge dựa trên case này, các bạn có thể chạy bằng docker như sau:

docker pull thaihoang2789/vnpt\_challenge:race\_condition

docker run -p 80:80 [IMAGE\_ID]

Nội dung chính của bài viết sẽ hướng tới việc phân tích root-cause và khai thác lỗ hổng. Theo mình, các bạn nên thử challenge trước rồi hẵng quay lại đọc bài viết. Còn nếu các bạn đã sẵn sàng với spoiler, thì hãy cùng mình tìm hiểu đôi chút về Race Condition nhé:

**1- Race Condition là gì?**

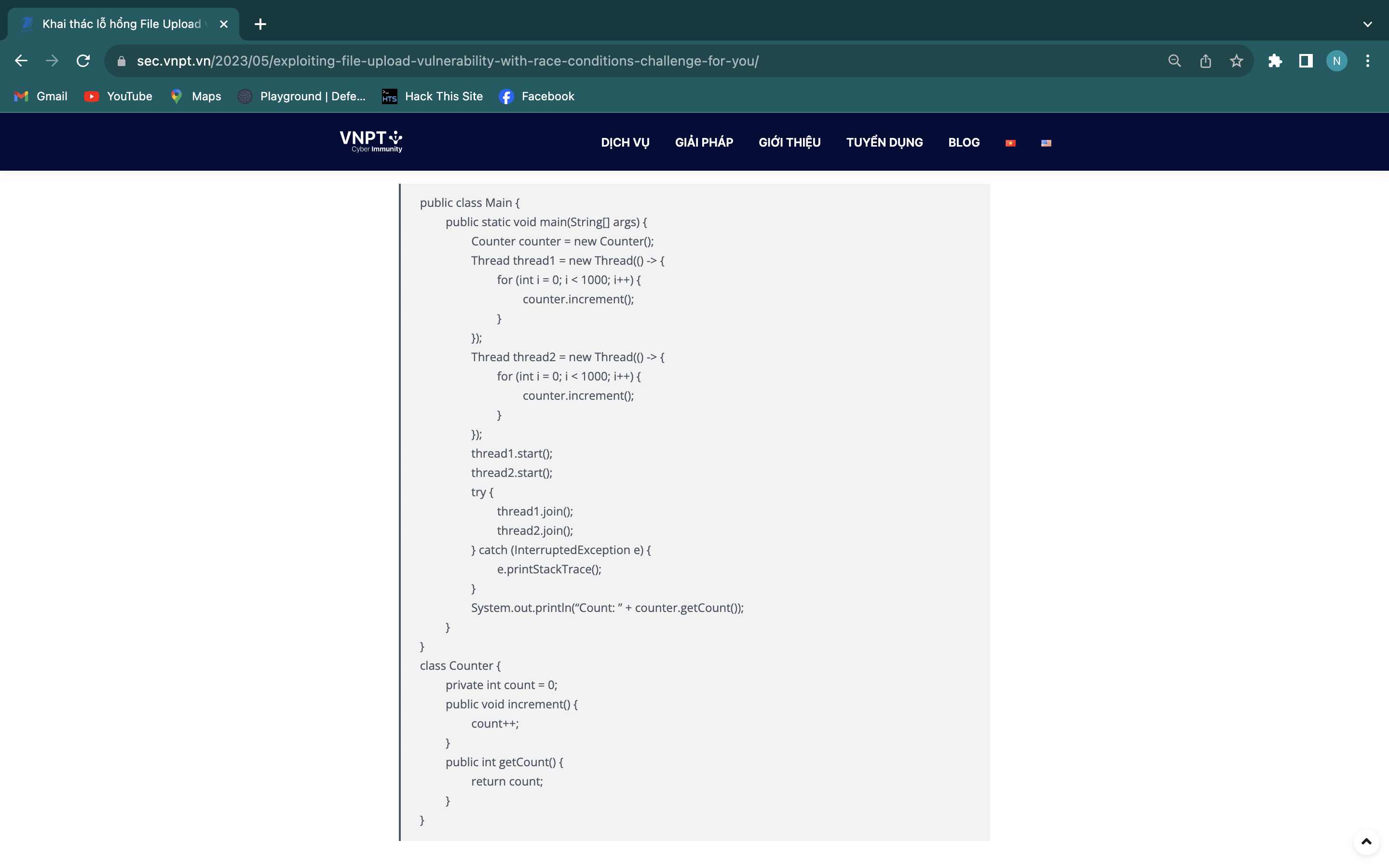
Đúc kết từ nhiều nguồn tài liệu thì:

“Race condition xảy ra khi hai hoặc nhiều tiến trình cùng truy cập vào một tài nguyên và thực hiện các thao tác trên tài nguyên đó mà không được đồng bộ hóa đúng cách. Khi đó, kết quả của các thao tác này có thể không đúng hoặc không như mong đợi.”

Ví dụ, giả sử có một ứng dụng web cho phép người dùng mua hàng trực tuyến. Khi người dùng thực hiện thanh toán, ứng dụng sẽ thực hiện các thao tác để cập nhật cơ sở dữ liệu về đơn hàng và số lượng sản phẩm còn lại trong kho. Nếu có hai người dùng cùng thực hiện thanh toán cho cùng một sản phẩm còn lại duy nhất trong kho, và các thao tác của họ không được đồng bộ hóa đúng cách, thì có thể xảy ra Race condition. Kết quả có thể là cả hai người dùng đều được xác nhận mua sản phẩm đó, trong khi thực tế chỉ còn một sản phẩm duy nhất trong kho.

Rất là loằng ngoằng phải không? Hiểu đơn giản thì nó giống như bạn và một gã nào đó đang cùng thích một cô nàng, hai người là hai tiến trình còn cô nàng là đích đến, kết quả thì…Chậc, bạn mong đợi gì ở một mối tình tay ba…bruh.

Để dễ hình dung hơn thì ở đây mình có một đoạn code minh hoạ về Race condition trong Java:



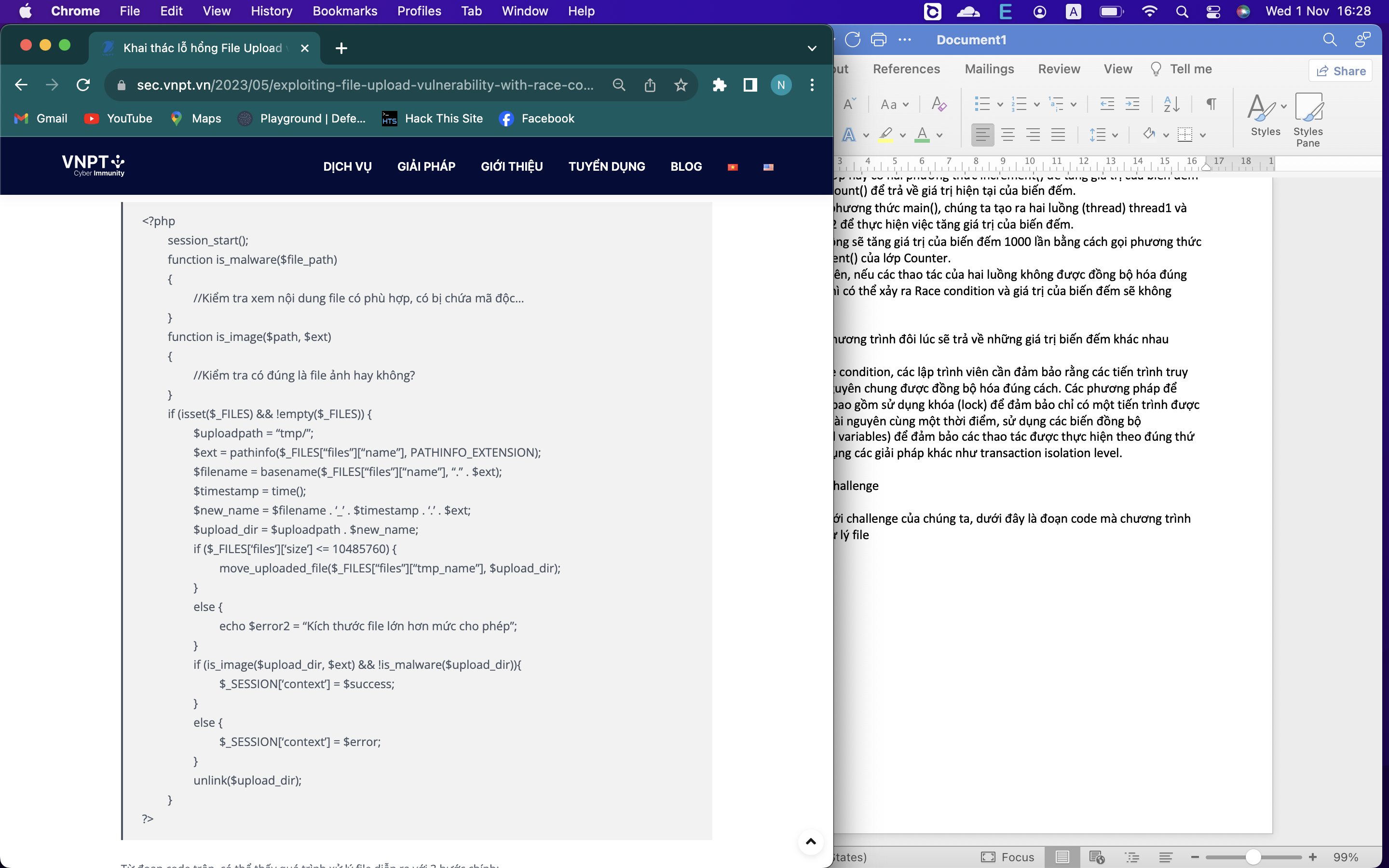
* Trong đoạn code trên, chúng ta có một lớp Counter đại diện cho một biến đếm. Lớp này có hai phương thức increment() để tăng giá trị của biến đếm và getCount() để trả về giá trị hiện tại của biến đếm.
* Trong phương thức main(), chúng ta tạo ra hai luồng (thread) thread1 và thread2 để thực hiện việc tăng giá trị của biến đếm.
* Mỗi luồng sẽ tăng giá trị của biến đếm 1000 lần bằng cách gọi phương thức increment() của lớp Counter.
* Tuy nhiên, nếu các thao tác của hai luồng không được đồng bộ hóa đúng cách, thì có thể xảy ra Race condition và giá trị của biến đếm sẽ không đúng.

Có thể thấy chương trình đôi lúc sẽ trả về những giá trị biến đếm khác nhau

Để tránh Race condition, các lập trình viên cần đảm bảo rằng các tiến trình truy cập vào tài nguyên chung được đồng bộ hóa đúng cách. Các phương pháp để đồng bộ hóa bao gồm sử dụng khóa (lock) để đảm bảo chỉ có một tiến trình được truy cập vào tài nguyên cùng một thời điểm, sử dụng các biến đồng bộ (synchronized variables) để đảm bảo các thao tác được thực hiện theo đúng thứ tự, hoặc sử dụng các giải pháp khác như transaction isolation level.

**2- Phân tích challenge**

Quay trở lại với challenge của chúng ta, dưới đây là đoạn code mà chương trình sử dụng để xử lý file



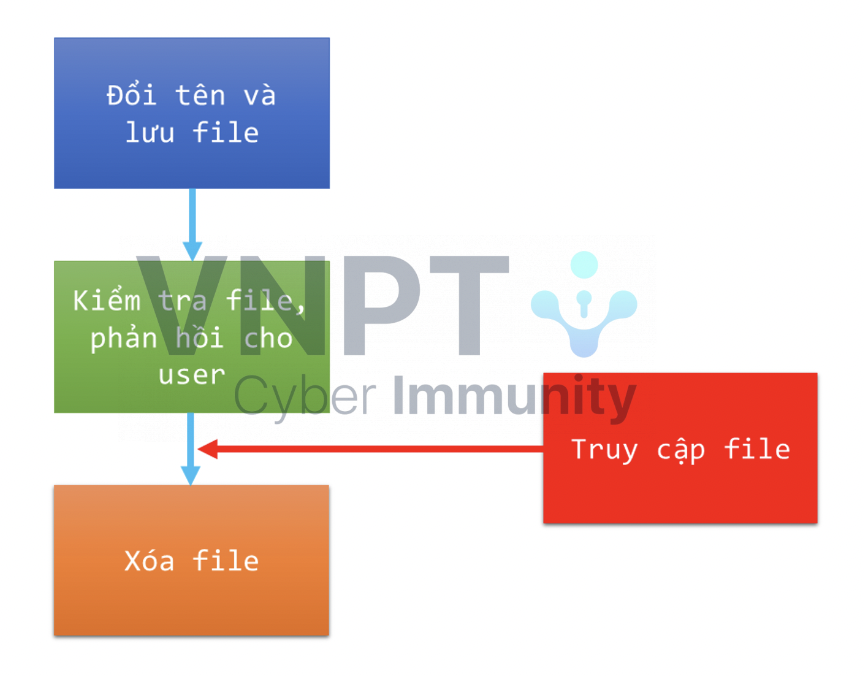
Từ đoạn code trên, có thể thấy quá trình xử lý file diễn ra với 3 bước chính:

* Đổi tên file kết hợp timestamp rồi lưu và thư mục /tmp
* Kiểm tra xem liệu file tải lên có phải file ảnh và chứ mã độc hay không rồi phản hồi cho người dùng
* Xóa file

⇒ Đúng kiểu Xem Xong Xóa 😏, nhưng ở đây mình chưa kịp xem mà đã xóa rồi

Ơ nhưng mà từ từ, nhìn qua thì nó hợp lý mà nhỉ? Có vấn đề gì đâu?

Nhưng mà các bạn ạ, một nhà tiên tri đã từng nói “Không có bông tuyết nào là trong sạch cả”, đoạn code trên cũng vậy hãy nhìn vào sơ đồ dưới đây:



Có thể thấy chương trình sẽ tiến hành xóa file khi đã xử lý xong nhưng mọi loại file tải lên đều được lưu ngay sau khi đổi tên, để ý một chút các bạn sẽ thấy tên file là một giá trị có thể tính toán được (sử dụng timestamp) và theo lý thuyết, nếu chúng ta đủ nhanh để truy cập vào file đó trước khi nó bị xóa, tức là chúng ta sẽ tạo một tiến trình truy cập file. Khi đó sẽ thì xóa và truy cập là hai gã đàn ông còn file là cô nàng. Hai tiến trình tranh nhau để đến được với file. Và đó chính là Race condition.

**3- RCE như thế nào?**

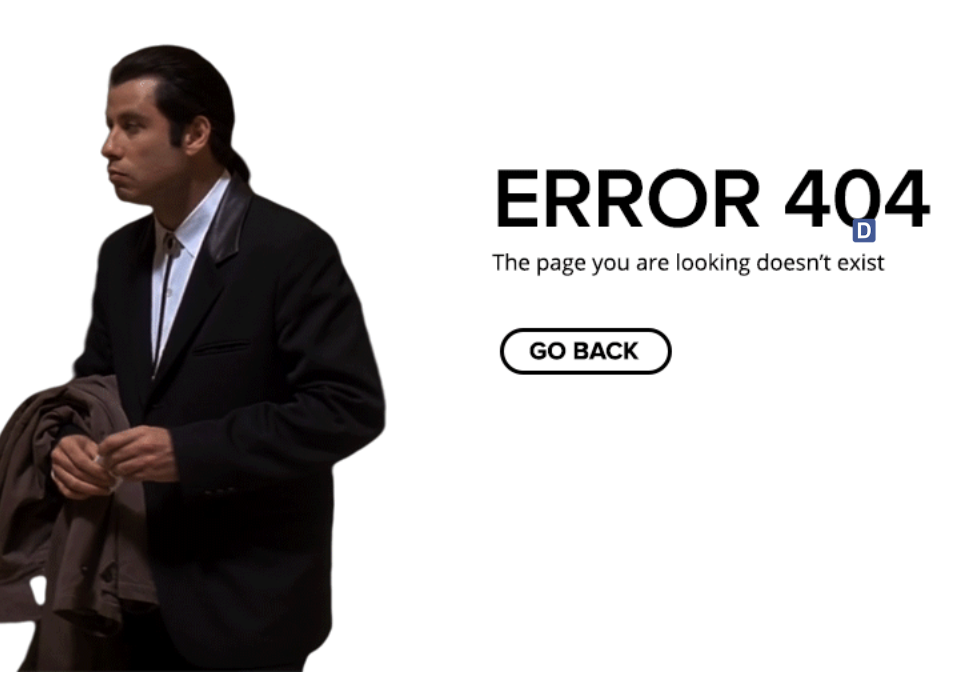
Lý thuyết là như vậy, giờ hãy cùng bắt tay vào thực tiễn

Nếu upload một file ảnh hợp lệ: Chương trình sẽ phản hồi thông tin của ảnh sau khi đã xử lý bao gồm cả tên file mới

Còn nếu upload một file không hợp lệ: Chương trình sẽ báo lỗi



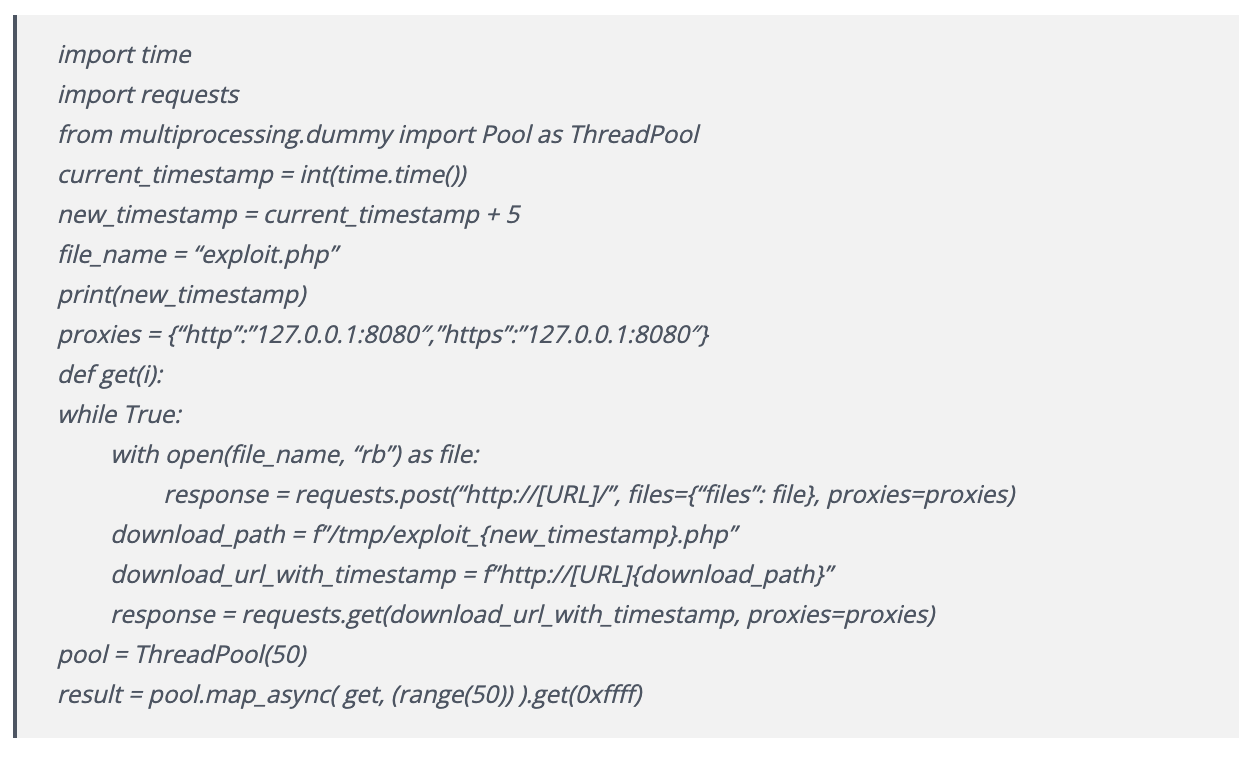
Khi thử truy cập vào file ảnh trong thư mục tmp thì cái chúng ta nhận được là… hmm, xóa rồi 😒



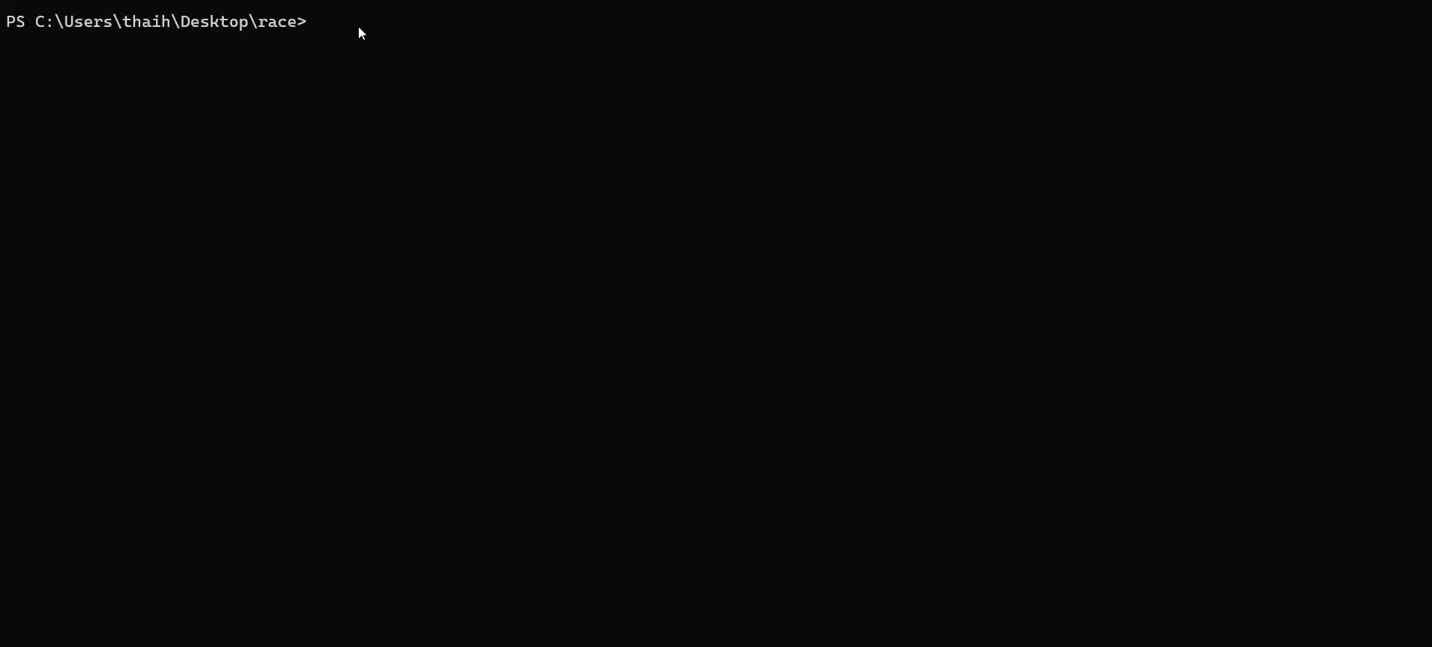
Từ phân tích thì mình nghĩ ra hai cách, áp dụng theo binh pháp tôn tử luôn cho học thức:

**Cách 1: Liên hoàn kế**

Với cách này, ta tính toán timestamp nhằm xác định tên file mới, upload file và truy cập vào file đó trước khi nó bị xóa. Từng thao tác click chuột, gõ phím phải nhanh, chuẩn và liên tiếp nhau không được phép sai sót nhưng mà đấy là nếu bạn có siêu tốc độ 🤡, mình thì không nên mình dùng python để hiện thực hóa điều này:



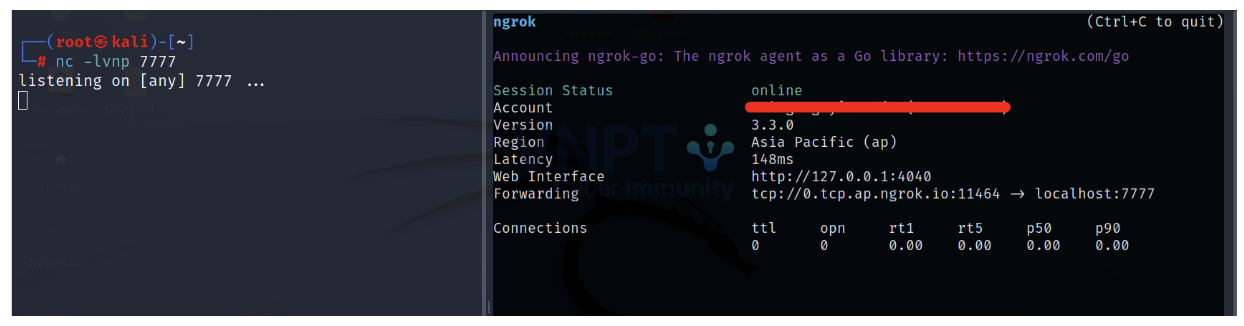
Đoạn code trên sẽ thực hiện các công việc sau:

* Lấy thời gian hiện tại và tạo một timestamp mới tăng giá trị lên 5 giây (Ở đây mình tăng lên khoảng 5s để đảm bảo file tồn tại).
* Xác định filename muốn tải lên hệ thống là “exploit.php”.
* Định nghĩa một proxies để sử dụng proxy ở địa chỉ “127.0.0.1:8080” cho request HTTP và HTTPS (xem log bằng Burp suite)
* Định nghĩa một hàm get(i) để thực hiện quá trình truy cập file nhằm kích hoạt revershell
* Trong vòng lặp, file “exploit.php” được mở để đọc dữ liệu dưới dạng binary ("rb").
* Sử dụng thư viện requests, gửi một yêu cầu POST đến URL mục tiêu với file được đính kèm trong phần body
* Định nghĩa đường dẫn download\_path để truy cập file, với tên file là "new\_timestamp" (vd: “/tmp/exploit\_1622025632.php”).
* Sử dụng requests để gửi một yêu cầu GET đến URL đã định nghĩa
* Quá trình lặp lại các bước và sẽ tiếp tục cho đến khi kết thúc chương trình.

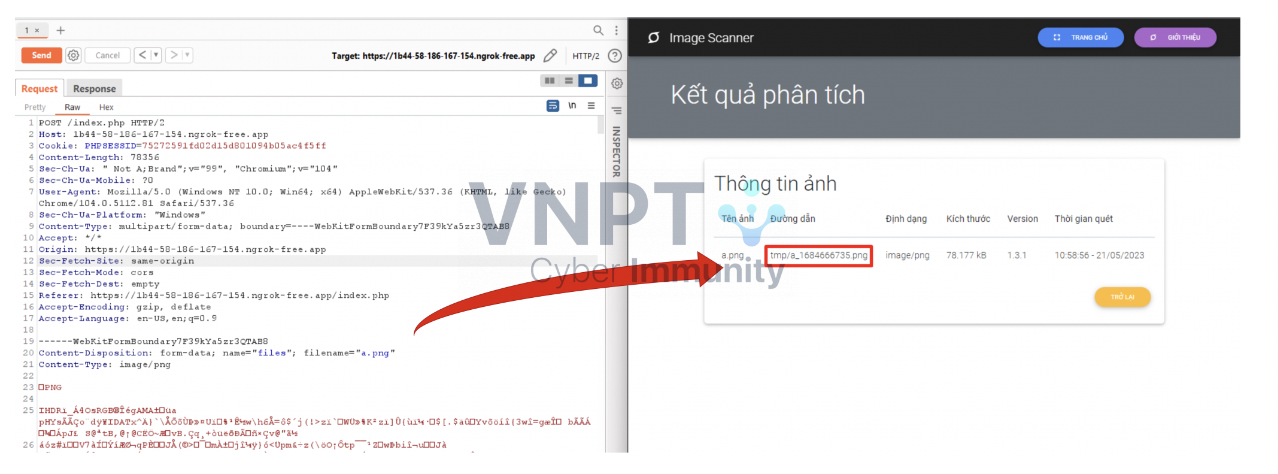
**Cách 2: Vô trung sinh hữu**

Cái này nghĩa là “Không có biến thành có”, cách này thì mình sẽ xác định một tên file ở thì tương lai và tạo request truy cập vào file đó đồng thời chúng ta cũng tiến hành upload file ⇒ cả hai diễn ra liên tục, khi đó thì file sẽ tồn tại trong một thời điểm và việc chúng ta phải làm là truy cập file trước khi nó bị xóa

Bước 1: Sử dụng netcat để lắng nghe kết nối trên cổng 7777 và dùng ngrok để tạo một kênh (tunnel) nhằm publish IP máy attack

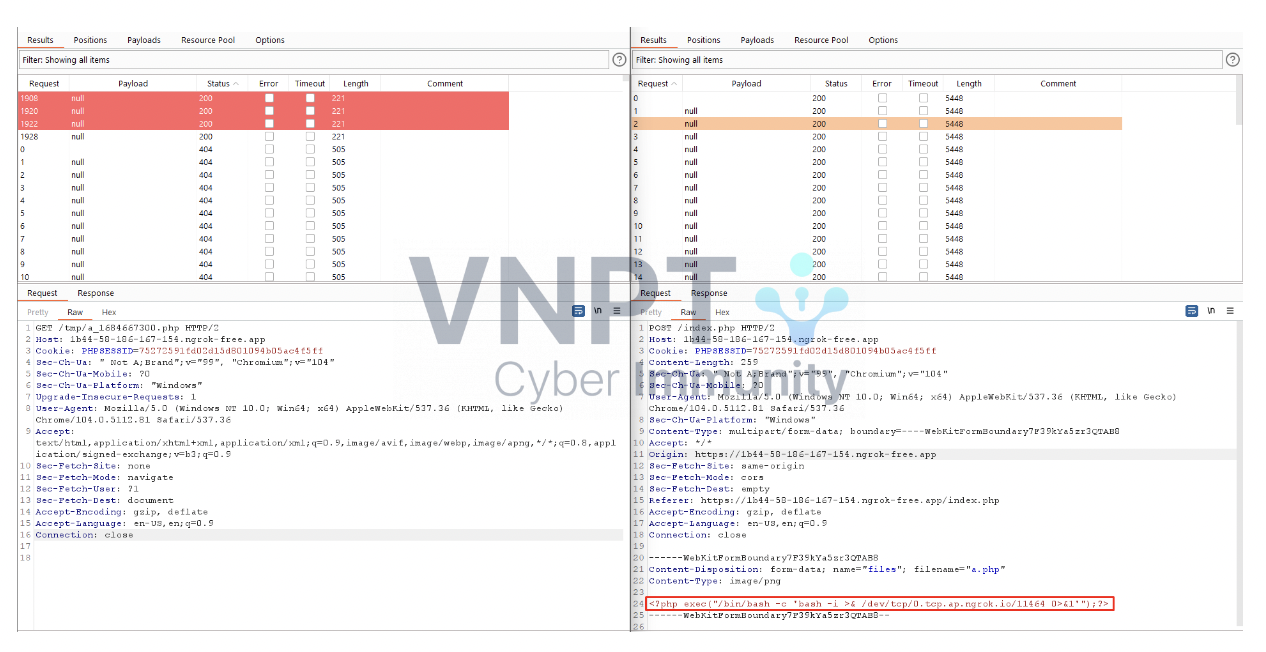


Bước 2: Lợi dụng việc upload ảnh hợp lệ để xác định new filename (Lấy tên của file ảnh, tăng lên vài đơn vị ở timestamp và sửa extension)

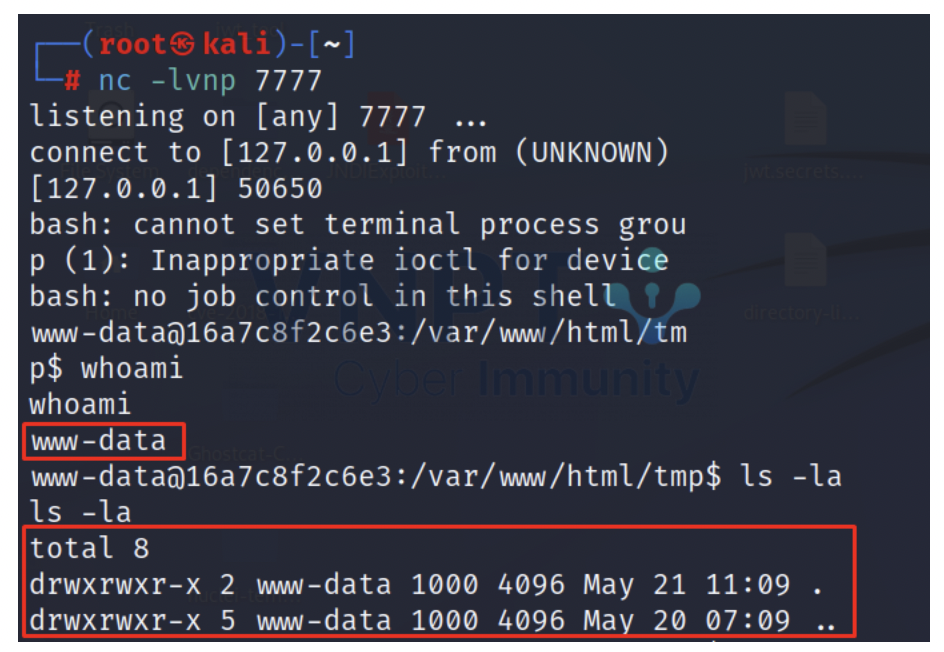


Bước 3: Sử dụng tính năng Intruder (mode null payload) của Burp suite cho 2 tiến trình:

* Truy cập liên tục vào file /tmp/a\_1684667300.php (Tên file php mà chúng ta upload)
* Upload file liên tục lên server, nội dung của file là một đoạn code php thiết lập kết nối (reverse shell) từ server tới địa chỉ IP đã publish bằng ngrok chỉ định trên cổng 11464.



Bước 4: RCE 🥳



Fact: Thật ra hai cách trên có ý tưởng base là giống nhau khác ở cách triển khai thôi:v

**Kết**

Trong đoạn code xử lý, mình đã sử dụng thêm lệnh sleep(1) để đảm bảo rằng quá trình Race condition đủ thời gian lý tưởng để diễn ra, còn trong thực tế thì các bạn sẽ phải sử dụng số lượng Thread tương đối lớn mới có thể khai thác được. Mong rằng các bạn sẽ có những trải nghiệm thú vị khi hí hoáy challenge này. Cya!