5.7. VẤN ĐỀ XÁC THỰC TRONG KHI SỬ DỤNG RESTFUL API

Khi chúng ta gọi API, một số hệ thống sẽ yêu cầu chúng ta thực hiện một tiến trình xác thực (authentication). Vấn đề xác thực này cũng tương tự khi chúng ta truy cập một thiết bị nào đó, thiết bị đó sẽ thực hiện vài bước kiểm tra như kiểm tra tên người dùng và mật khẩu. Khi một người dùng đã hoàn thành việc xác thực, bất kỳ thay đổi nào mà người dùng thực hiện thông qua API sẽ ảnh hưởng đến toàn bộ ứng dụng. Ví dụ, nếu một RESTful API được dùng để xóa dữ liệu, dữ liệu đó sẽ bị xóa ra khỏi hệ thống. Kết quả của gọi API để xóa dữ liệu sẽ giống y như là người dùng login vào hệ thống từ giao tiếp dòng lệnh CLI và thực hiện tác vụ xóa dữ liệu.

Vì giao tiếp giữa *API client* và *API server* diễn ra bằng cách sử dụng các chu kỳ *HTTP request/response* cho nên nếu tiến trình xác thực của API tận dụng các cơ chế sẵn có của HTTP thì đó cũng là điều tự nhiên. Ngày nay, phần lớn các REST API sử dụng các trường có trong *HTTP header* để lưu trữ các dữ liệu khác nhau dùng cho tiến trình xác thực.

Mặc dù các cách hiện thực có thể khác nhau, một cách tổng thể, chúng ta có ba cách tiếp cận:

- Kiểu xác thực cơ bản dùng HTTP: dùng các cơ chế có sẵn của HTTP.
- Kiểu xác thực dùng khóa API: phía *client* sẽ thêm vào các khóa tạo trước để gán vào *HTTP header*.
- Kiểu xác thực dùng custom token: Một token tạo ra động sẽ được dùng.

5.7.1. Kiểu xác thực cơ bản

Đây là kiểu xác thực đơn giản và phổ biến nhất được dùng trong API. Nhược điểm của kiểu xác thực này là các thông tin xác thực được gửi đi ở dạng không mã hóa. Điều này có nghĩa là, nếu chúng ta dùng HTTP như là giao thức lớp truyền vận (transport), các dữ liệu này có thể bị nghe lén và thông tin người dùng (*username, password*) có thể bị lộ dễ dàng. Các thông tin thường được hiển thị ở dạng *base64* trong định dạng header của HTTP.

Trong lĩnh vực lập trình, định dạng base64 là một nhóm của các thuật toán chuyển đổi từ dạng nhị phân binary sang dạng văn bản bằng cách hiển thị dữ liệu ở dạng chuỗi khối 64bit.

Để giải quyết yếu điểm trên, kiểu xác thực cơ bản thường được sử dụng với SSL/TLS.

Một vấn đề lớn với kiểu xác thực cơ bản này là mật khẩu của người dùng được gửi qua lại trong từ cuộc gọi API (API request). Cơ chế này càng làm gia tăng khả năng kẻ tấn công bắt được các lưu lượng mạng, trong đó có chứa mật khẩu. Vì vậy, chúng ta có thêm một lý do nữa để mã hóa các kiểu gọi API dùng xác thực cơ bản.

5.7.2. Xác thực dùng các khóa API (API Keys)

Một kiểu xác thực khác mà một số API có thể sử dụng là *API key*. Một *API key* là một chuỗi ký tự được xác định trước mà *client* sẽ gửi đến API server. *API key* này giống như là một khóa bí mật, hoạt động giống như mật khẩu. Bất kỳ ai hoặc phần mềm nào với khóa này có thể thực hiện việc gọi API và có thể gây ra các gián đoạn hệ thống hay có thể truy cập các dữ liệu quan trọng. Khóa API thường được mã hóa ở dạng *base64*, vì vậy nó có thể được truyền bên trong HTTP header. Trong lúc truyền, các khóa API này có thể được lưu trữ ở dạng cookie hoặc lưu trong một *HTTP header* riêng lẻ. Sau đó, bên máy chủ, giá trị này sẽ được so sánh với các giá trị đã lưu trong cơ sở dữ liệu của các khóa hợp lệ.

Khóa API có thể được gửi từ *client* đến máy chủ *API* theo ba cách:

- Dạng chuỗi *(string*).
- Dùng *request header* của HTTP.
- Dùng *cookie*.

Đây là một ví dụ của khóa API dùng dạng chuỗi.

```
GET /something?api_key=abcdef12345
```

Khóa API dạng này sẽ luôn được gửi bên trong từng cuộc gọi API. Tuy nhiên nếu chúng ta thực hiện nhiều lần gọi API, sẽ rất bất tiện nếu chúng ta phải nhập vào từng chuỗi ký tự này. Lúc này, để gọi API, chúng ta có thể dùng kiểu *request header* hay kiểu *cookie*.

Kiểu *request header* thường được thấy khi chúng ta sử dụng công cụ Postman (Postman là một công cụ cho phép chúng ta thực hiện gọi API và nhiều chức năng hữu ích khác) hoặc khi chúng ta gọi API bằng cách viết các đoạn *Python scripts*. Trong trường *header* của API request, lúc này chúng ta cần bao gồm thêm các chuỗi hoặc token.

```
GET /something HTTP/1.1
X-API-Key: abcdef12345
```

Dưới đây là một ví dụ khác mô tả cách thức bạn tạo ra một người dùng mới, sử dụng *API key*. Đoạn chương trình dùng Python và thư viện requests. Lưu ý cách thức API key được thêm vào hàm gọi request như là *HTTP header*.

```
def createUser(url, name, age):

    headers = {'X-Api-Key': '28fnhu783hb39bb0'}
    payload = {'user': {'name': name, 'age': age}}

    try:
        response = requests.post(url + '/users', headers=headers, data=payload)

        if response.status_code == 401:
            raise Exception('API key invalid!')
        elif response.status_code == 403:
            raise Exception('API key not authorized!')
        elif response.status_code != 200:
            raise Exception('API error: {}'.format(response))
    except Exception as e:
        print('User creation failed! Error: {}'.format(e))

    print('User {} created successfully!'.format(name)
```

Trong trường hợp một khóa API key không còn hợp lệ, một mã lỗi *401 Unauthorized* sẽ bị gửi trả về. Hoặc nếu khóa vẫn còn hợp lệ nhưng lại không đủ quyền truy cập tài nguyên đang cần, mã lỗi *403 Forbidden* sẽ xuất hiện.

Cuối cùng, một trong những cách phổ biến nhất để gọi API nhiều lần là sử dụng *cookies*. Một *HTTP cookie* (hay còn được gọi là *web cookie*, *browser cookie*) là một mẫu dữ liệu nhỏ được lưu trên máy tính của người dùng bên trong trình duyệt web. Cookie được thiết kế như là một cơ chế tin cậy cho các website nhớ các trạng thái của một phiên giao dịch (chẳng hạn như các hàng hóa thêm vào một rổ mua hàng) hay ghi lại các hoạt động của người dùng (các nút được nhấn, các trang web đã xem trong quá khứ). Trong trường hợp của xác thực API, một cookies sẽ lưu các giá trị chuỗi khóa *API keytring* và có thể được tái sử dụng nhiều lần.

Hình bên dưới mô tả một API key cookie với cùng một khóa trong ví dụ trước.

```
GET /something HTTP/1.1
Cookie: X-API-KEY=abcdef12345
```

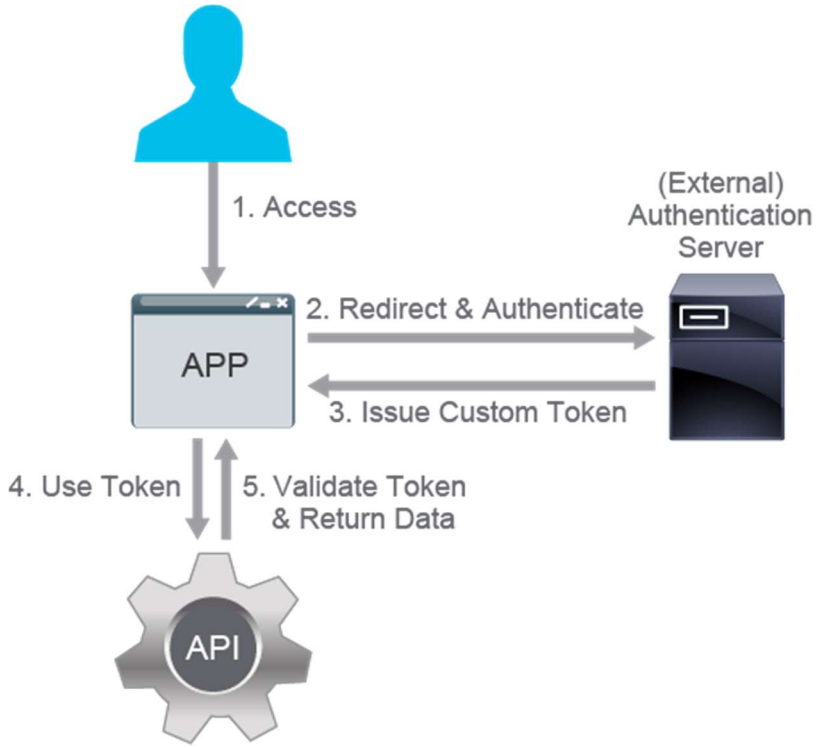
5.7.3. Xác thực dùng custom tokens

Kiểu xác thực custom token cho một một người dùng nhập vào tên người dùng và mật khẩu. Sau đó server sẽ tạo ra duy nhất một token đã được mã hóa. Người dùng có thể sử dụng token mã hóa này để truy cập đến các trang đã được bảo mật hay các tài nguyên khác mà không cần phải nhập vào các thông tin truy cập (tên người dùng và mật khẩu) nhiều lần.

Tiến trình xác thực diễn ra như sau:

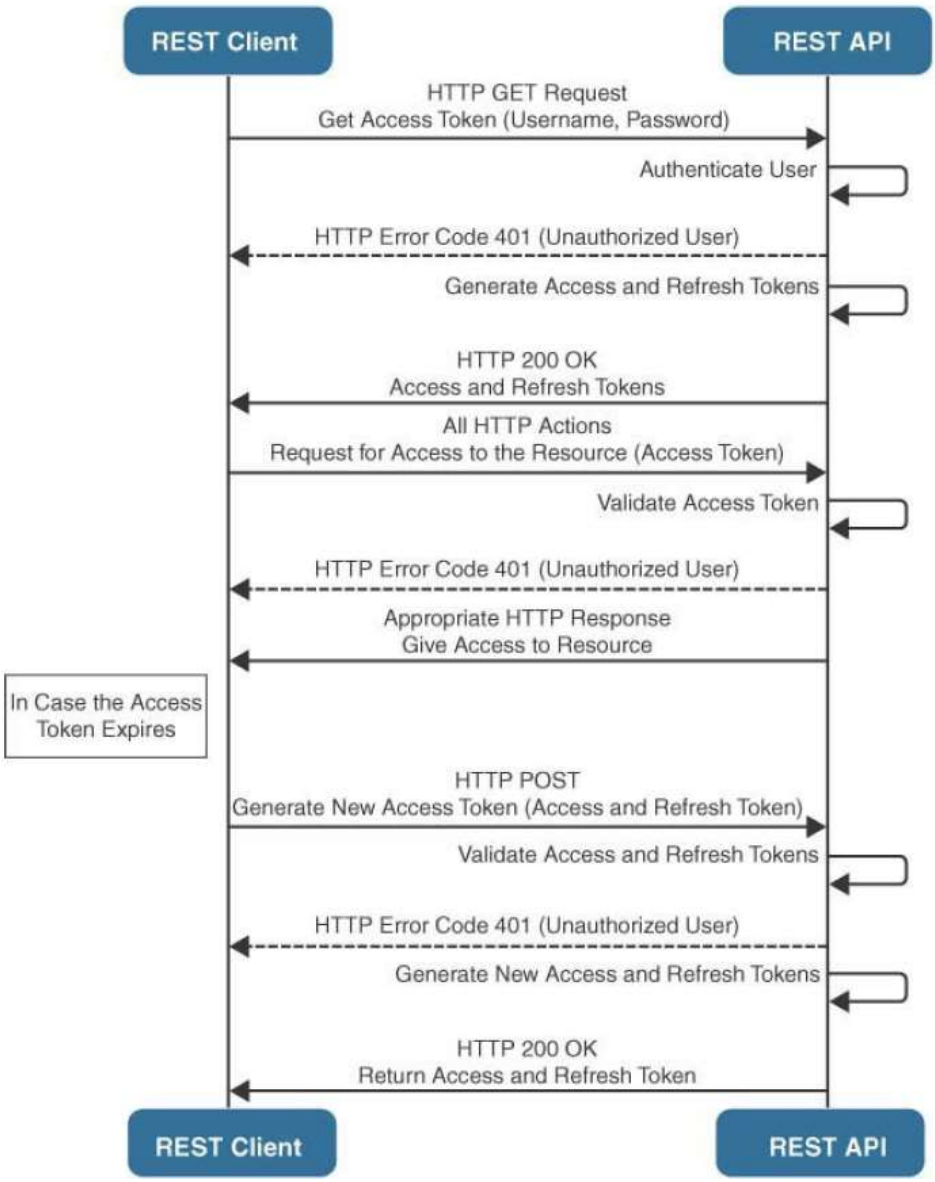
1. *Client* cố gắng truy cập ứng dụng.
2. Vì *client* chưa được xác thực, nó sẽ bị điều hướng *redirect* đến máy chủ xác thực, ở đó người dùng sẽ phải cung cấp *username* và *password*.
3. Máy chủ xác thực sẽ kiểm tra tính hợp lệ của thông tin. Nếu thông tin là hợp lệ, một *token* sẽ được cấp và trả ngược về cho người dùng.

4. Lần gọi API kế tiếp của *client* đã có *token*, do đó yêu cầu được truyền đến API.
5. Dịch vụ API sẽ kiểm tra *token* và phục vụ các dữ liệu theo yêu cầu.



Các *token* có thể có thời gian hiệu lực có giới hạn. Khi hết hạn, người dùng sẽ phải thực hiện lại tiến trình xác thực bằng cách nhập lại thông tin tên người dùng và mật khẩu. Một *token* được thiết kế như là một bằng chứng cho thấy người dùng đã xác thực thành công. *Token* giúp đơn giản hóa quá trình truy cập và giảm số lần người dùng phải nhập thông tin truy cập.

Token thường được lưu trong trình duyệt web của người dùng và được kiểm tra mỗi lần người dùng cần truy cập các thông tin bảo mật. Khi người dùng thoát ra khỏi trang web, *token* sẽ bị hủy.



Các máy chủ xác thực thường là các dịch vụ bên ngoài (ví dụ như OpenID) được dùng kết hợp với các cơ chế cấp quyền (ví dụ như OAuth 2.0). Các *token* thường cũng chứa các dữ liệu về phân quyền (authorization). Thường thì các máy chủ xác thực được dùng để cấp các token có thể được sử dụng cho các dịch vụ khác nhau của các nhà sản xuất khác nhau. Ví dụ chúng ta có thể dùng dịch vụ xác thực của Google để truy cập những site có triển khai Google Sign-In.

Thuật ngữ phổ biến được dùng cho loại xác thực này (và cả phần phân quyền) được gọi là SSO (Single Sign On). SSO càng ngày càng phổ biến vì nó rất hữu ích cho các doanh nghiệp cung cấp nhiều loại dịch vụ khác nhau nhưng muốn dùng cùng một ID cho tất cả các dịch vụ.

Các ưu điểm của kiểu xác thực dùng *custom token* bao gồm:

- Thời gian trả lời nhanh hơn vì chỉ có *token* cần phải được xác thực, thay vì là tất cả các thông tin *username/password*. Các phiên được giữ trong cơ sở dữ liệu trong bộ nhớ tốc độ cao.
- Đơn giản hóa việc sử dụng các dịch vụ khác nhau. Giảm thiểu số lượng hệ thống xác thực. Vấn đề xác thực và phân quyền được hợp nhất.
- Gia tăng tính bảo mật. Các thông tin truy cập không bị truyền nhiều lần trong các lần gọi API. Các thông tin chỉ được gửi một lần lúc bắt đầu và định kỳ những lần sau, mỗi khi token bị hết hạn. Cơ chế này giảm thiểu nguy cơ tấn công MITM.

Kiểu xác thực dùng *token* cũng có nhược điểm là phức tạp. Chúng ta ít có kiểm soát chi tiết từng *token* riêng lẻ trong trường hợp máy chủ xác thực bị chiếm đoạt.

Viết cho cộng đồng SDWAN, DEVNET

Đặng Quang Minh