HTTP Authentication trong Network Automation

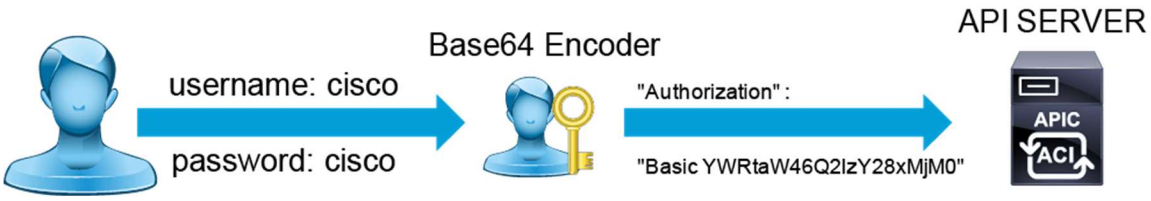
🔥 Bạn có đang "mở toang cánh cửa" API mà không biết?

Trong bài viết này, chúng ta sẽ đào sâu vào HTTP Authentication – một lớp bảo vệ cực kỳ quan trọng nhưng thường bị xem nhẹ khi triển khai RESTful API cho hệ thống tự động hoá mạng, SDN controller hoặc bất kỳ nền tảng nào có mở API công khai.

🔐 3 CƠ CHẾ XÁC THỰC HTTP MÀ KỸ SƯ TỰ ĐỘNG HOÁ PHẢI BIẾT

Khi bạn triển khai các RESTful API, điều quan trọng không chỉ là "trả về đúng dữ liệu", mà còn là ngăn chặn truy cập trái phép. Có 3 hình thức xác thực HTTP phổ biến:

1. Basic Authentication



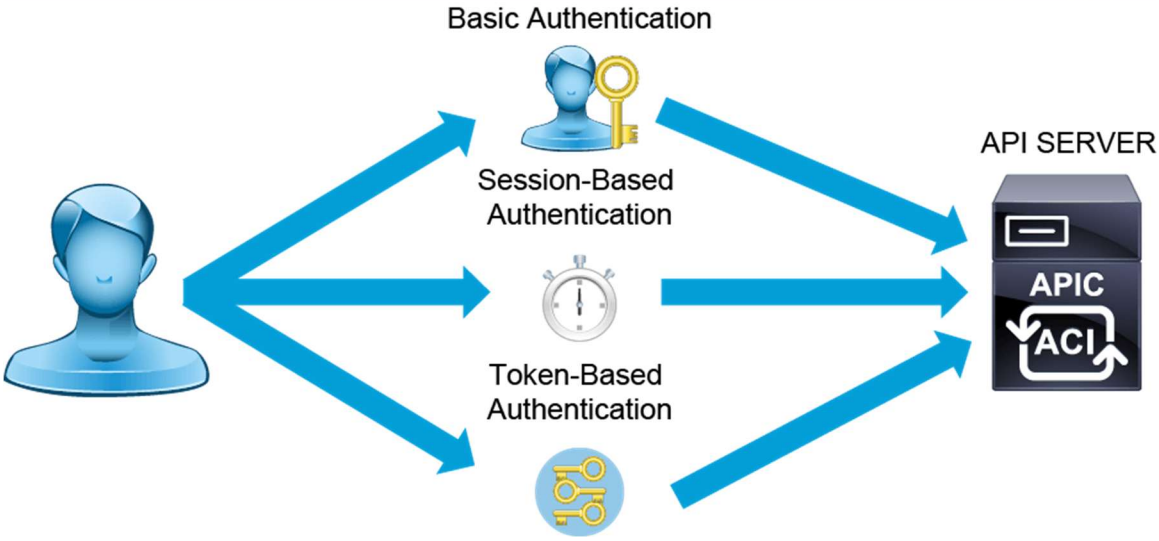
👉 Dễ triển khai, nhưng thiếu bảo mật.

Thông tin đăng nhập (username:password) được mã hoá bằng Base64, sau đó được gửi trong header HTTP.

Ví dụ Python:

```python
import base64
username = "Cisco"
password = "Cisco"
message = f"{username}:{password}"
encoded = base64.b64encode(message.encode()).decode()
headers = {"Authorization": f"Basic {encoded}"}
```

🛑 Rủi ro: Base64 không phải là mã hoá, dễ bị giải mã nếu không dùng HTTPS.



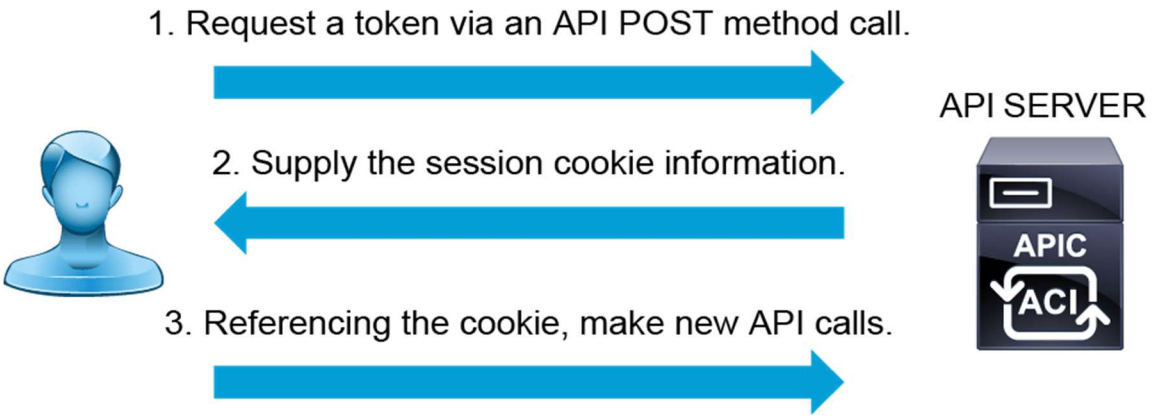
2. Session-Based Authentication

👉 Một lựa chọn phổ biến cho các hệ thống web truyền thống.

Sau khi đăng nhập thành công, server tạo một session cookie.

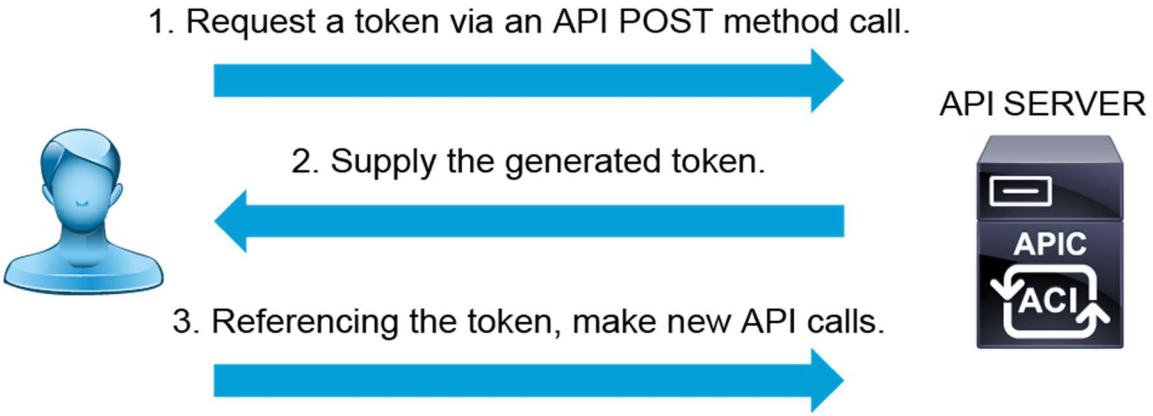
```http
Set-Cookie: sessionid=xyz12345abc; HttpOnly
```

Session dễ bị hijack nếu không có HTTPS. Không chia sẻ context giữa các hệ thống.



3. Token-Based Authentication

👉 Phương pháp hiện đại và linh hoạt nhất.



Client gửi POST để lấy token và dùng token đó trong các request sau.
```http
Authorization: Bearer eyJhbGciOi...
```
Ưu điểm: Stateless, bảo mật cao, dễ tích hợp với OAuth2, API Gateway, CI/CD.

So sánh nhanh:

- Basic: Cấu hình nhanh, bảo mật yếu, không linh động.
- Session: Dễ triển khai với web, bảo mật phụ thuộc cookies.
- Token: API hiện đại, bảo mật cao, dễ quản lý.

CCNA Automation Tip:

Trong môi trường network automation (Cisco DNA Center, ACI, NSO), token-based authentication là hình thức được khuyến nghị vì có thể tích hợp tốt với Python scripts, Postman, Ansible, Terraform và CI/CD pipeline.

Ví dụ thực tế:

Postman tương tác API Cisco DNA Center:
1. Gửi POST `/auth/token` để lấy token.
2. Dùng token cho các request như `/network-device`, `/site`.

Tóm lại cho anh em DevOps/Automation:

- Đừng để API trống trơn mà không xác thực.
- Luôn dùng HTTPS.
- Ưu tiên token-based authentication để phát triển hệ thống API chuẩn hóa và an toàn.

Token-Based và Session-Based Authentication trong Tự động hóa mạng

Trong thời đại Automation và AI lên ngôi, việc tương tác với API của các hệ thống như Cisco DNA Center hay Cisco ACI là chuyện hằng ngày ở huyện. Nhưng nếu không nắm rõ cách "xác thực" (authentication) đúng chuẩn, bạn sẽ vấp ngã đau đớn ngay từ cú request đầu tiên.

Trong bài này, mình chia sẻ hai phương pháp xác thực Token-Based và Session-Based, cực kỳ phổ biến khi làm việc với các controller như Cisco DNA Center và APIC (ACI). Và quan trọng là: có ví dụ Python cụ thể để anh em copy-paste làm lab ngay!

Token-Based Authentication – Dành cho DNA Center

Ý tưởng chính: Gửi thông tin username/password → Nhận lại Token → Dùng Token cho các API call tiếp theo.

1. Chuẩn bị biến Python

```
import requests
DNA_URL = "https://sandboxdnac.cisco.com"
HEADERS = {"content-type":"application/json"}
USER = "devnetuser"
PASS = "Cisco123!"
AUTH = (USER, PASS)
LOGIN_URL = DNA_URL + "/api/system/v1/auth/token"
```

2. Gửi POST request để nhận Token

```
result = requests.post(url=LOGIN_URL, auth=AUTH, headers=HEADERS, verify=False)
```

3. Trích xuất Token từ response JSON

```
TOKEN = result.json()['Token']
```

```
HEADERS['X-Auth-Token'] = TOKEN
```

4. Thực hiện các API Call

```
INVENTORY_URL = DNA_URL +"/dna/intent/api/v1/network-device"
response = requests.get(url=INVENTORY_URL, headers=HEADERS, verify=False)
```

🍪 Session-Based Authentication – Dành cho APIC (ACI)

Ý tưởng chính: Gửi thông tin đăng nhập để nhận cookie phiên → Tự động gắn cookie cho các request tiếp theo.

1. Tạo Session và định nghĩa login body

```
import requests
session = requests.Session()

APIC_URL = "https://sandboxapicdc.cisco.com"
USER = "admin"
PASS = "ciscopsdt"
AUTH_BODY = {
   "aaaUser": {
      "attributes": {"name": USER, "pwd": PASS}
   }
}
LOGIN_URL = APIC_URL + "/api/aaaLogin.json"
```

2. Gửi POST để đăng nhập và nhận cookie

```
response = session.post(url=LOGIN_URL, json=AUTH_BODY, verify=False)
```

3. Kiểm tra cookie tồn tại

```
print(response.cookies)
```

4. Gửi API Call với Session đã đăng nhập

```
TENANT_URL = APIC_URL + "/api/node/class/fvTenant.json"
```

```
response = session.get(url=TENANT_URL, verify=False)
print(response.json())
```

Tổng kết nhanh

Token-Based: Cisco DNA Center – dùng X-Auth-Token trong header
Session-Based: Cisco APIC (ACI) – dùng cookie tự động
→ Mỗi API platform sẽ có kiểu xác thực riêng, đọc kỹ tài liệu trước khi làm!

Câu hỏi nhanh kiểm tra kiến thức

❓ Lớp nào dùng để quản lý session authentication?
✅ Session()

❓ Username và password cần định dạng thế nào trước khi mã hóa Base64?
✅ "username:password"

TÓM TẮT

Không phải API nào cũng xác thực giống nhau. Có API yêu cầu Token kiểu Bearer, có API cần Cookie phiên, có API chơi luôn cả 2. Vậy nên, luôn đọc kỹ tài liệu API của controller bạn đang làm việc, và viết hàm Python chuẩn xác. Trong các bài lab DevNet hay tự động hóa mạng, hãy dành thời gian làm thật kỹ phần xác thực API – vì nếu bước này sai, tất cả các bước tiếp theo… vô nghĩa.

Linux Networking Kiến Thức Thực Chiến – Cấu Hình Mạng Vĩnh Viễn

1. Cấu hình IP tĩnh với file /etc/network/interfaces (Debian cũ)

Trên các bản Debian/Ubuntu cũ (pre-Netplan), cấu hình mạng được lưu tại:
/etc/network/interfaces

Ví dụ:

```
auto eth0
iface eth0 inet static
  address 192.168.1.50
  netmask 255.255.255.0
  up ip route add 172.16.10.0/24 via 192.168.1.25 dev eth2
  up ip route add 10.10.10.0/24 via 192.168.1.35 dev eth3

auto eth1
iface eth1 inet dhcp
```

Lưu ý:
- Dòng auto dùng để khởi động giao diện khi boot.
- Dòng up cho phép thêm static route vĩnh viễn ngay sau khi interface hoạt động.

Muốn sửa file: sudo nano /etc/network/interfaces

2. Cấu hình với Netplan (Debian mới, Ubuntu)

Trên các bản mới (Ubuntu 18.04+), cấu hình dùng Netplan, file thường nằm ở: /etc/netplan/*.yaml

Ví dụ nội dung:

```
network:
 version: 2
 renderer: NetworkManager
 ethernets:
  ens192:
   dhcp4: no
   addresses: [172.21.55.10/24]
   gateway4: 172.21.55.1
   nameservers:
    addresses: [100.65.0.51, 100.65.0.52]
```

Các bước:
1. sudo nano /etc/netplan/01-network-manager-all.yaml
2. Sửa nội dung
3. sudo netplan apply

renderer: NetworkManager dùng cho Desktop GUI, còn networkd dùng cho server.

3. RedHat / CentOS: Cấu hình với ifcfg-*

Các hệ RedHat-based dùng cấu hình ở: /etc/sysconfig/network-scripts/ifcfg-ens33

Ví dụ:

```
DEVICE=ens33
BOOTPROTO=none
ONBOOT=yes
PREFIX=24
IPADDR=172.21.1.55
```

GATEWAY=172.21.1.1

Chỉnh sửa với: sudo nano /etc/sysconfig/network-scripts/ifcfg-ens33

4. Cập nhật DNS vĩnh viễn

DNS được cấu hình tại /etc/resolv.conf nhưng thực tế file này được sinh từ /etc/resolvconf/resolv.conf.d/

Muốn sửa DNS không bị mất sau reboot:
1. Sửa file: sudo nano /etc/resolvconf/resolv.conf.d/head
2. Thêm dòng nameserver
3. sudo resolvconf -u

5. Restart Networking

Sau khi thay đổi IP, subnet, gateway, hoặc thêm NIC:
sudo service networking restart
hoặc:
sudo systemctl restart networking

Review nhanh kiến thức

Câu hỏi: 2 phát biểu nào sau đúng về cấu hình DNS trong Linux?

Đáp án đúng:
- DNS được cấu hình động từ /etc/resolvconf/resolv.conf.d/
- DNS được lưu tại /etc/resolv.conf

Ứng dụng thực tế

- Cấu hình gateway dự phòng tĩnh
- Định tuyến mạng nội bộ riêng ra các interface cụ thể
- Chỉnh sửa DNS cố định cho server trong nội bộ doanh nghiệp

Bạn đang làm hạ tầng server Linux hay thi CCNA/DevOps? Đây là kỹ năng bắt buộc!

Tổng hợp các lệnh mạng cơ bản trong Linux – DevNetOps cần thuộc lòng!

1. Kiểm tra kết nối và định tuyến

`ping`: Gửi ICMP echo request để kiểm tra kết nối giữa hai node. Trên Linux, lệnh này chạy vô hạn cho đến khi bạn nhấn Ctrl + C.

→ Ví dụ: ping 8.8.8.8

`traceroute`: Xem các hop Layer 3 mà gói tin đi qua để đến đích.

→ Ví dụ: traceroute google.com

2. Quản lý giao diện mạng

`ifconfig`: Xem thông tin hoặc cấu hình các interface.

→ Ví dụ: ifconfig eth0, ifconfig eth1 down

⚠️ Tuy nhiên lệnh này dần được thay thế bằng `ip`.

`ip link`: Liệt kê các interface đang có trên máy.

`ip addr`: Hiển thị thông tin IP của các interface.

→ Cấu hình IP: sudo ip addr add 192.168.1.50/24 dev eth1

❗ Lưu ý: Không tồn tại sau khi reboot.

`ip route`: Thêm static route.

→ Ví dụ: sudo ip route add 10.10.10.0/24 via 192.168.1.1 dev eth1

`ifup` / `ifdown`: Bật/tắt một interface nhanh chóng.

→ sudo ifdown eth1, sudo ifup eth1

3. Kiểm tra bảng định tuyến và ARP

`ip route list`: Hiển thị bảng định tuyến.

→ Xu hướng dùng lệnh này thay vì `route`.

`route`: Lệnh cũ để xem bảng định tuyến kernel.

`arp`: Xem bảng ánh xạ địa chỉ IP–MAC.

→ Thêm tĩnh: sudo arp -s 192.168.1.10 00:11:22:33:44:55

4. Thống kê và kiểm tra kết nối

`netstat`: Hiển thị thông tin kết nối, bảng định tuyến, thống kê interface.

→ netstat -r: bảng định tuyến

→ netstat -i: thông tin thống kê interface

5. Truy vấn DNS

`dig`: Lấy thông tin DNS như A, MX, CNAME.

→ dig cisco.com

`host`: Resolve IP sang hostname hoặc ngược lại.

→ host google.com

6. Kết nối từ xa

`ssh`: Đăng nhập bảo mật đến thiết bị từ xa.

→ ssh admin@192.168.1.1 -p 2222

→ ssh -l user 10.0.0.1

Lưu ý cho dân DevOps/NetOps

`ip` là lệnh mạnh hơn `ifconfig`, đang dần trở thành tiêu chuẩn.

Kết hợp với Ansible, bạn có thể tự động hóa cấu hình interface hoặc kiểm tra kết nối qua module ping, uri, command.

Dùng trong các lab kiểm tra routing, DNS, SSH hoặc thu thập dữ liệu mạng để debug nhanh.

Gợi ý ứng dụng thực tế

- ✅ Lab kiểm tra kết nối sau khi cấu hình VRF: dùng ping, traceroute, ip route list
- ✅ Debug lỗi DNS cho container: dùng dig, host
- ✅ Xác minh SSH kết nối tự động hóa với GitLab runner
- ✅ Thu thập interface stats trước và sau khi deploy ứng dụng