Nitter server - Pass Rate Limiting

- Skip Twitters Ratelimiting

Skip Twitters Ratelimiting

Mitigating Unauthorized Web Scraping Bot Traffic

The current design of Nitter, along with its methodology for accessing the Twitter service, necessitates heightened vigilance on the part of instance operators to manage unwarranted access by web scraping bots. This guide is intended to provide operators with essential information to effectively mitigate unauthorized web scraping bot traffic.

Prerequisites

Before proceeding with the rate-limiting setup, ensure that you have:

1. A functional Nitter installation located at `/opt/nitter`.
2. Nginx as your web server, with the server block as outlined in Nginx Configuration. In this guide, we refer to this server block as 'nitter.nginx.'

Rate Limiting Configuration

Navigate to `/etc/nginx` and create two necessary files with the following content:

shared_cache.conf

```
proxy_buffers 64 16k;
proxy_buffer_size 4k;
expires 90d;
access_log off;
proxy_pass http://127.0.0.1:8080;
```

shared_static.conf

```
expires 90d;
access_log off;
root /opt/nitter/public
```

These files ensure that normal usage, such as serving images, videos, and site data, does not trigger rate limiting. Logging for these locations is disabled to prevent entries in access or error logs.

Next, add the rate-limiting rules within the nginx.conf file, located within the http block:

nginx.conf

```
http {

    limit_req_zone $binary_remote_addr zone=nitter.tld_sec:10m rate=1r/s;
    limit_req_zone $binary_remote_addr zone=nitter.tld_min:10m rate=45r/m;
}
```

These settings limit users to one request per second and 45 requests per minute, a natural browsing rate for the site.

Now, in your 'nitter.nginx' server block:

nitter.nginx

```
server {
    location /pic/ { include shared_cache.conf; }
    location /video/ { include shared_cache.conf; }

    location /css/ { include shared_static.conf; }
    location /js/ { include shared_static.conf; }
    location /fonts/ { include shared_static.conf; }
    location = /apple-touch-icon.png { include shared_static.conf; }
    location = /apple-touch-icon-precomposed.png { include shared_static.conf; }
    location = /android-chrome-192x192.png { include shared_static.conf; }
    location = /favicon-32x32.png { include shared_static.conf; }
    location = /favicon-16x16.png { include shared_static.conf; }
    location = /favicon.ico { include shared_static.conf; }
    location = /logo.png { include shared_static.conf; }
    location = /site.webmanifest { include shared_static.conf; }
}
```

nitter.nginx

```
        location / {
                proxy_pass http://localhost:8080;
                limit_req zone=nitter.tld_sec burst=3 nodelay;
                limit_req zone=nitter.tld_min burst=4;
        }
```

The 'burst' parameter allows for temporary bursts of traffic while maintaining the overall rate limit, ensuring a smoother user experience while preventing server overload and misuse.

Reload Nginx configuration files:

```
nginx -s reload
```

Nginx will now rate limit IP for excessive usage.

Fail2ban

To implement rate limiting and address repeat offenders, a functional install of Fail2ban (https://github.com/fail2ban/fail2ban) is required. Typically, Fail2ban configuration files are located at /etc/fail2ban. Make a copy of jail.conf named jail.local, as Fail2ban will prioritize jail.local by default when both are present:

```
cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

Within jail.local, ensure that bantime.increment is uncommented and set to true:

```
bantime.increment = true
```

Additionally, enable the rate-limiting block in jail.local by setting 'enabled' to true:

```
[nginx-limit-req]
enabled = true
port    = http,https
logpath = %(nginx_error_log)s
```

Restart Fail2ban:

```
systemctl restart fail2ban
```

To verify that the jail is running:

```
fail2ban-client status nginx-limit-req
```

You'll receive a summary of failures triggered by the filter and the number of active actions.

```
Status for the jail: nginx-limit-req
|- Filter
|  |- Currently failed:	0
|  |- Total failed:	0
|  `- File list:	%(nginx_error_log)s
`- Actions
   |- Currently banned:	0
   |- Total banned:	0
   `- Banned IP list:
```

Fail2ban will now enforce rate limits, with incremental punishments for repeat infractions when bans expire.

Switch to guest_accounts, compile it as normal

```
git clone https://github.com/zedeus/nitter
cd nitter
git branch guest_accounts
nimble build -d:release
nimble scss
nimble md
```

```
touch guest_accounts.jsonl
vim guest_accounts.jsonl
```

paste `[]` inside it

Finally apply for guest accounts here:

https://twitterminator.x86-64-unknown-linux-gnu.zip/

Create cronjob

```
0 */4 * * * curl ...guest_accounts.jsonl
```