Homework for the lesson "Organizing Network Security (Part 1)"

Rate limit

Introductory

The nginx server is often used not only as a web server, but also as a reverse proxy. What does it mean? This means that his task is in front of complex web applications and allows him to solve the problem:

1. Give away "statics" (static files, with which it is quite good)
2. Proxy requests on the upstream server (i.e. the request comes to nginx, nginx sends it further to the server with the application)

Why is the second function needed? nginx can take care of slow clients, caching, etc.

In addition, it can serve as a software load balancer and limit the flow of incoming requests (this is what we will be interested in).

What do we want to do? We want to use nginx to find out the limit of flows coming from one IP address.

Description of execution

Two virtual machines:

- Ubuntu with nginx (10.0.0.1)
- Kali (10.0.0.2)

You should already have nginx configured, if it suddenly doesn't exist, then it happened [to view the lesson] (../10_internet).

No limit

Ubuntu

one. We make sure that the nginx service is running, port 80 is open.

2. If you don't install nginx then:

```
sudo suitable update
sudo apt install nginx
```

3. Edits the `sudo mcedit /etc/nginx/sites-enabled/default` configuration to look correct:

```
server {
        server_name netology.local;

        listen 80;
        root /var/www/html;

        index index.html index.htm index.nginx-debian.html;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

4. Saves changes

5. Check if the build is correct `sudo nginx -t`

6. The setting `sudo nginx -s reload` applies

Kali

We will use the Apache Benchmarks utility to generate source files.

```
ab --help
```

If you get a message that the program was not found (`bash: ab command not found`), then install it:

```
sudo suitable update
sudo apt install ab
```

Answer: 1000 shipments of 100 shipments at the same time:

```
ab -n 1000 -c 100 http://netology.local
```

We make sure that all requests occur:

```
kali@kali:~$ ab -n 1000 -c 100 http://netology.local/
This is ApacheBench, Version 2.3 <$Revision: 1874286 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking netology.local (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        nginx/1.18.0
Server Hostname:        netology.local
Server Port:            80

Document Path:          /
Document Length:        612 bytes

Concurrency Level:      100
Time taken for tests:   0.259 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      854000 bytes
HTML transferred:       612000 bytes
Requests per second:    3866.86 [#/sec] (mean)
Time per request:       25.861 [ms] (mean)
Time per request:       0.259 [ms] (mean, across all concurrent requests)
Transfer rate:          3224.90 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   2.4      0      11
Processing:     6   23   4.4     24      29
Waiting:        1   22   4.5     24      29
Total:         12   24   2.9     25      30

Percentage of the requests served within a certain time (ms)
  50%     25
  66%     26
  75%     26
  80%     26
  90%     27
  95%     28
  98%     29
  99%     29
 100%     30 (longest request)
```

Speed limit

Ubuntu

Change nginx settings:

```
sudo mcedit /etc/nginx/sites-enabled/default
```

Editing the appearance to make it look right:

```
limit_req_zone $binary_remote_addr zone=limit:10m rate=50r/s;

server {
        server_name netology.local;

        listen 80;
        root /var/www/html;

        index index.html index.htm index.nginx-debian.html;

        limit_req zone=limit burst=50 nodelay;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

Only the 1st and 17th lines have changed.

▶ Measurement description

```
limit_req_zone $binary_remote_addr zone=limit:10m rate=50r/s
```

Creating a zone for a limited number of departures:

- `$binary_remote_addr` - based on the ip address of the requester
- `zone=limit:10m` - zone with a name limit and 10 megabytes of memory (information about 16,000 addresses takes about 1 megabyte)
- `rate=50r/s` - 50 files per second

```
limit_req zone=limit Burst=50 nodelay
```

- `limit_req zone=limit` - enable limit for heavy server.
- `burst=50` - by default, 50 outcomes per second means that requests cannot be made more often than once every 20 ms, i.e. if requests come more often, then nginx will disable them with a 503 error. `burst` allows you to specify the number of links that are allowed in the context of the window set by `rate`, while the first one will be served, and the rest will wait for the set time limit (those who went beyond 50 gets 503)
- `nodelay` - allows you not to wait for the set limit, but send a request immediately if the set limits allow it

Testing climate and environmental changes:

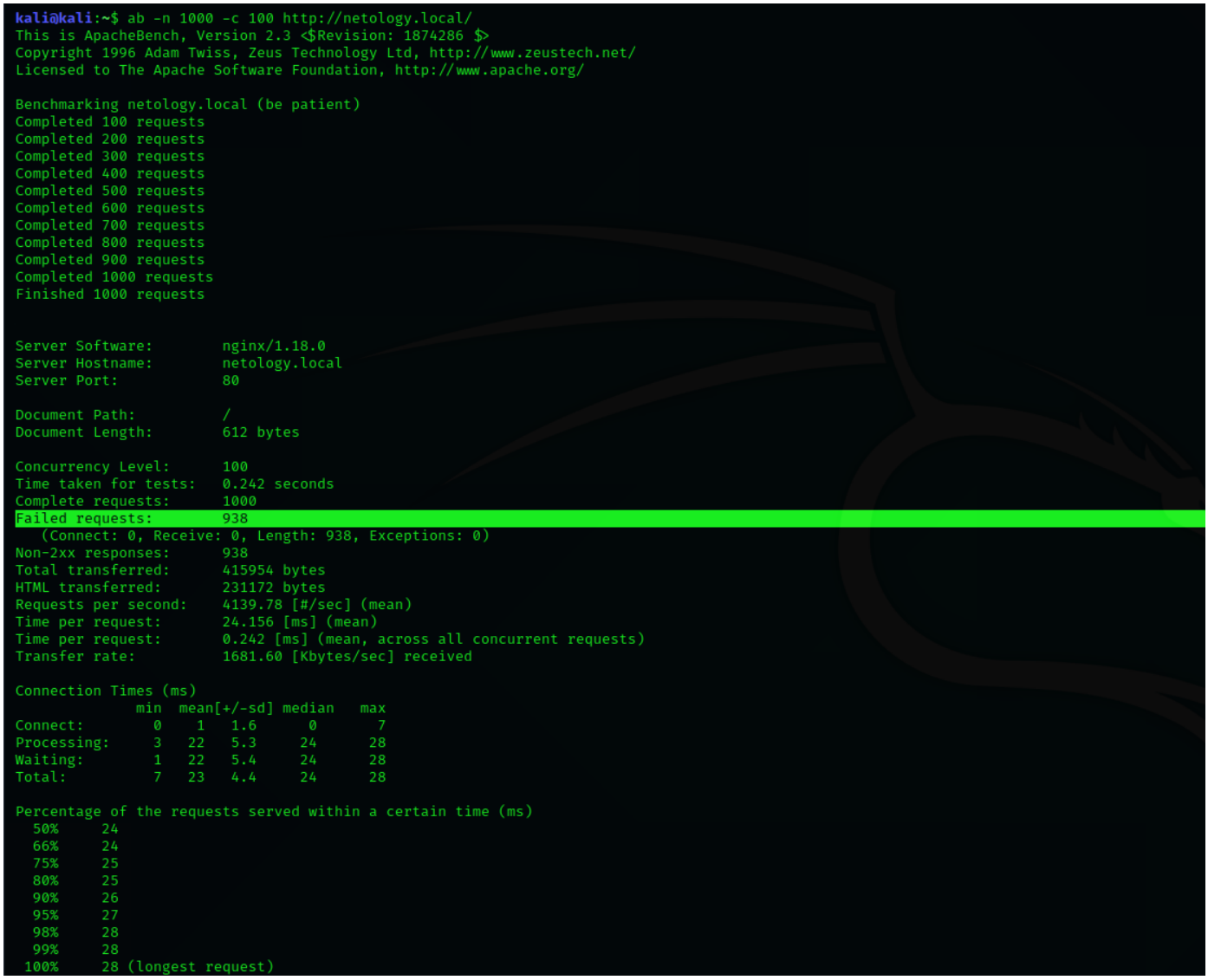
```
sudo nginx -t
sudo nginx -s reload
```

Cali

Once again we do 1000 events for 100 meetings at the same time:

```
ab-n 1000-s 100 https://netology.local
```

We make sure that some of them end with an error:



Fail2Ban (optional part)

Of course, you can go further and "block" the IP, Fail2Ban will help us with this, which we will discuss in detail in the lecture.

Ubuntu

```
sudo suitable update
sudo apt install fail2ban
```

Make sure the service is running:

```
systemctl fail2ban status
```

```
ubuntu@ubuntu:/etc/nginx/sites-enabled$ systemctl status fail2ban
● fail2ban.service - Fail2Ban Service
     Loaded: loaded (/lib/systemd/system/fail2ban.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2020-10-23 14:33:53 UTC; 39s ago
       Docs: man:fail2ban(1)
   Main PID: 12414 (f2b/server)
      Tasks: 5 (limit: 2282)
     Memory: 12.5M
     CGroup: /system.slice/fail2ban.service
             └─12414 /usr/bin/python3 /usr/bin/fail2ban-server -xf start

Oct 23 14:33:53 ubuntu systemd[1]: Starting Fail2Ban Service...
Oct 23 14:33:53 ubuntu systemd[1]: Started Fail2Ban Service.
Oct 23 14:33:53 ubuntu fail2ban-server[12414]: Server ready
```

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

We go down to the `[nginx-limit-req]` entry and enable it by adding the line `enabled = true`:

```
383 # To use 'nginx-limit-req' jail you should have `ngx_http_limit_req_module`
384 # and define `limit_req` and `limit_req_zone` as described in nginx documentation
385 # http://nginx.org/en/docs/http/ngx_http_limit_req_module.html
386 # or for example see in 'config/filter.d/nginx-limit-req.conf'
387 [nginx-limit-req]
388 enabled = true
389 port    = http,https
390 logpath = %(nginx_error_log)s
391
392 [nginx-botsearch]
393
394 port     = http,https
395 logpath  = %(nginx_error_log)s
396 maxretry = 2
```

Save the file and apply the settings:

```
sudo fail2ban-server restart
sudo fail2ban-server status
```

```
Status
|- Number of jail:      2
`- Jail list:   nginx-limit-req, sshd
```

Kali

Make sure you get "banned" after running the `ab` pair:

```
kali@kali:~$ ab -n 1000 -c 100 http://netology.local/
This is ApacheBench, Version 2.3 <$Revision: 1874286 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking netology.local (be patient)
apr_socket_recv: Connection refused (111)
```

Verify that the Kali address is indeed banned:

```
sudo zgrep 'Ban' /var/log/fail2ban.log
```

Result

Submit the following screenshots:

1. Screenshot with `ab` requests before enabling limit_req on nginx
2. Screenshot with `ab` after requests