Defending against automatization

using nginx

2022
ALMATY
KAZAKHSTAN

TOLTARYS
KAZHACKSTAN

The plan

Increase performance

Turn on cool stuff to improve speed

Traffic marking

Mark who is producing the parasitic traffic

Bot mitigation

Decide what to do with it

Configuring NGINX: Performance

Where to start and what to include:

- Gzip, brotli (both of them)
- HTTP/2
- Server push and preload
- SSL protocols

TL;DR: https://nginxconfig.io/

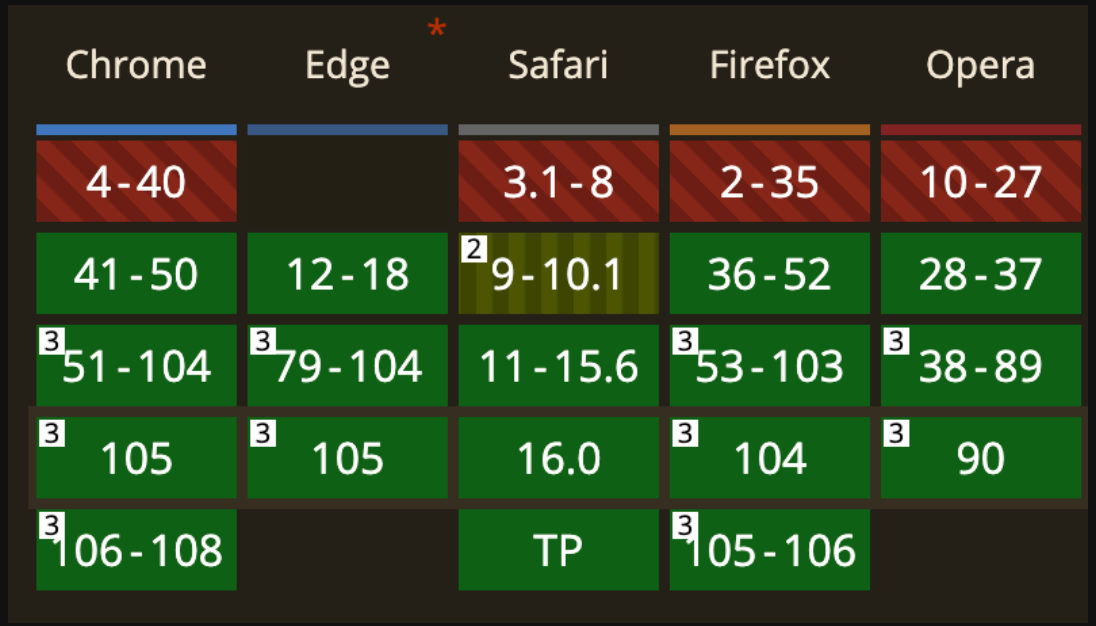
Configuring NGINX: Compress

```
# gzip
gzip              on;
gzip_vary         on;
gzip_proxied      any;
gzip_comp_level   6;
gzip_types        text/plain text/css text/xml application/json application/javascript application/

# brotli
brotli            on;
brotli_comp_level 6;
brotli_types      text/plain text/css text/xml application/json application/javascript application/
```

Brotli compresses 80% of the files on my server

Configuring NGINX: HTTP/2



Chrome	Edge *	Safari	Firefox	Opera
4-40		3.1-8	2-35	10-27
41-50	12-18	[2] 9-10.1	36-52	28-37
[3] 51-104	[3] 79-104	11-15.6	[3] 53-103	[3] 38-89
[3] 105	[3] 105	16.0	[3] 104	[3] 90
[3] 106-108		TP	[3] 105-106	

Support for major browsers. For example, the release of Chrome 41 was 7 years ago

Configuring NGINX: Mechanisms of prediction

Preload

The preload attribute specifies if and how the author thinks that the media file should be loaded when the page loads. The preload attribute allows the author to provide a hint to the browser about what he/she thinks will lead to the best user experience.

HTTP/2 Push

Server Push is a performance technique aimed at reducing latency by loading resources preemptively, even before the client knows they will be needed.

Configuring NGINX: Push? No, but yes

```
server {
    listen 443 ssl http2;
    server_name myserver;

    location / {
        add_header Link "</app.js>; rel=preload; as=script" always;
        http2_push /app.js;
        try_files $uri $uri/ =404;
        }
}
```

And terser to merge all js

Configuring NGINX: Push? No, but yes

Intent to Remove: HTTP/2 and gQUIC server push 26522 views

Brad Lassey

Primary eng (and PM) emails lassey@chromium.org, dschinazi@chromium.org, bnc@chromium.org, ianswett@

Morgaine

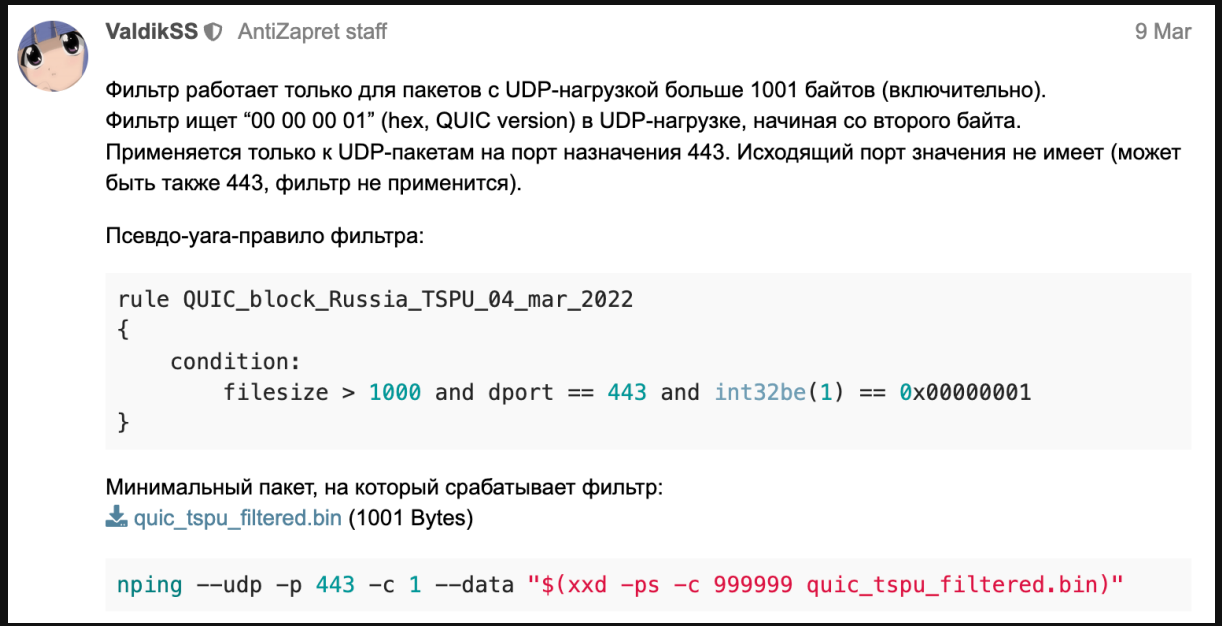
On Wednesday, November 11, 2020 at 5:00:39 PM UTC-5 las...@chromium.org wrote: Chrome currently

Tiago Cardoso

HTTP Pipelining all over again. A quinta-feira, 12 de novembro de 2020 à(s) 01:35:24 UTC, Morgaine

https://evertpot.com/http-2-push-is-dead/

Configuring NGINX: Quic? Yes, but no



ValdikSS AntiZapret staff 9 Mar

Фильтр работает только для пакетов с UDP-нагрузкой больше 1001 байтов (включительно).
Фильтр ищет “00 00 00 01” (hex, QUIC version) в UDP-нагрузке, начиная со второго байта.
Применяется только к UDP-пакетам на порт назначения 443. Исходящий порт значения не имеет (может быть также 443, фильтр не применится).

Псевдо-yara-правило фильтра:

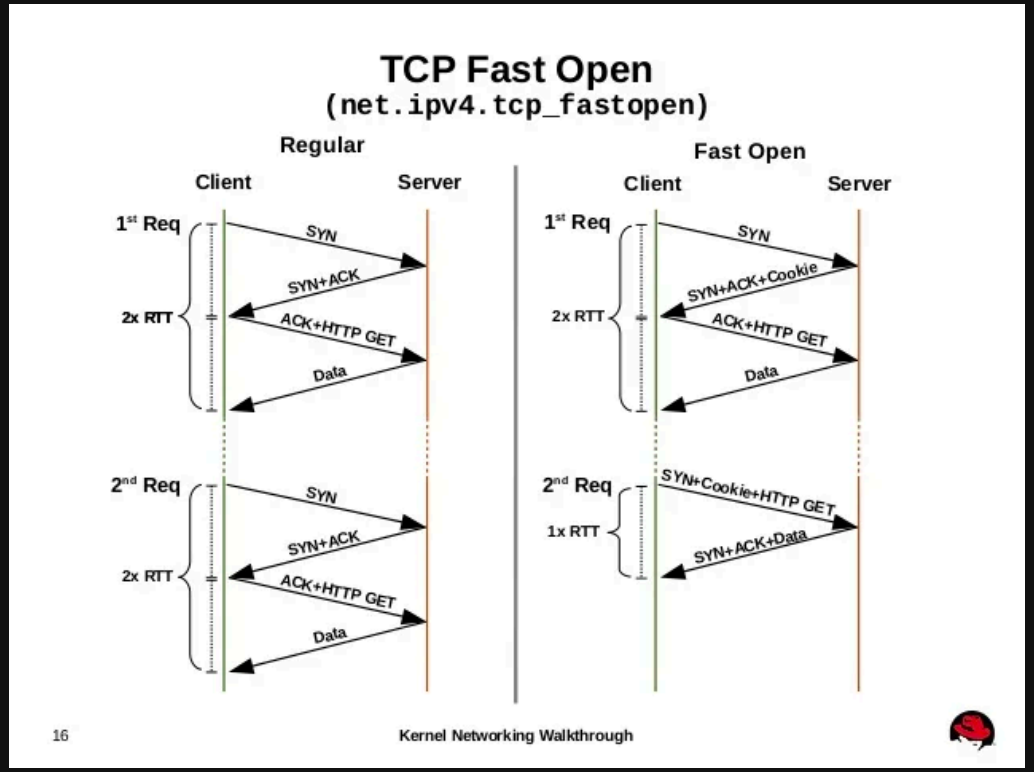
```
rule QUIC_block_Russia_TSPU_04_mar_2022
{
    condition:
        filesize > 1000 and dport == 443 and int32be(1) == 0x00000001
}
```

Минимальный пакет, на который срабатывает фильтр:
quic_tspu_filtered.bin (1001 Bytes)

```
nping --udp -p 443 -c 1 --data "$(xxd -ps -c 999999 quic_tspu_filtered.bin)"
```

Russian government partially blocks http3 due to censorship

Configuring NGINX: Fastopen

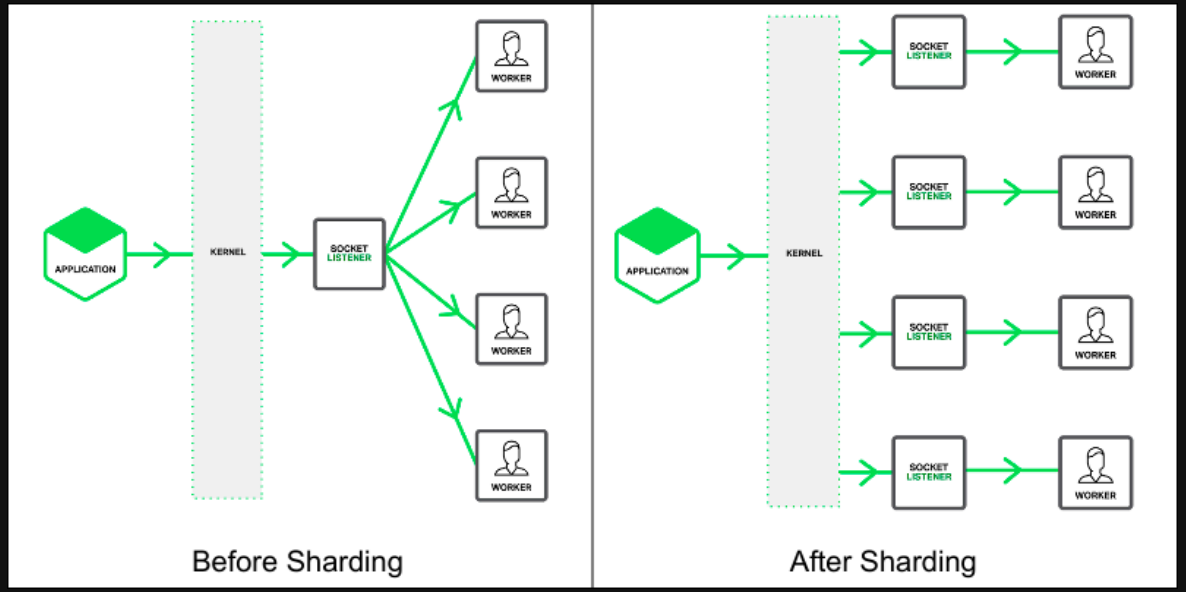


Configuring NGINX: Fastopen? No

> Remove TCP FastOpen support.
>
> We never enabled it by default, and have no plans to, so we should just remove it. QUIC also makes it less useful, and TLS 1.2 0-RTT session restore means it potentially mutates state.
>
> *It also leaks also leaks data between profiles*, including (but not limited to) incognito profiles, since it relies on global system state.

F

Configuring NGINX: SO_REUSEPORT



https://blog.n0p.me/2018/02/2018-02-20-portsharding/

https://blog.cloudflare.com/the-sad-state-of-linux-socket-balancing/

Configuring NGINX: SO_REUSEPORT

```
server {
        listen 443 ssl http2 default_server reuseport;
        server_name mysite;

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

https://blog.cloudflare.com/the-sad-state-of-linux-socket-balancing/

Configuring NGINX: Rate limit

```
limit_req_zone $binary_remote_addr zone=ip:10m rate=5r/s;
server {
    #...
    location /login/ {
        limit_req zone=ip burst=10 delay=8;
        #...
    }
    location /search/ {
        limit_req zone=ip burst=20 nodelay;
        #...
    }
    #...
}
```

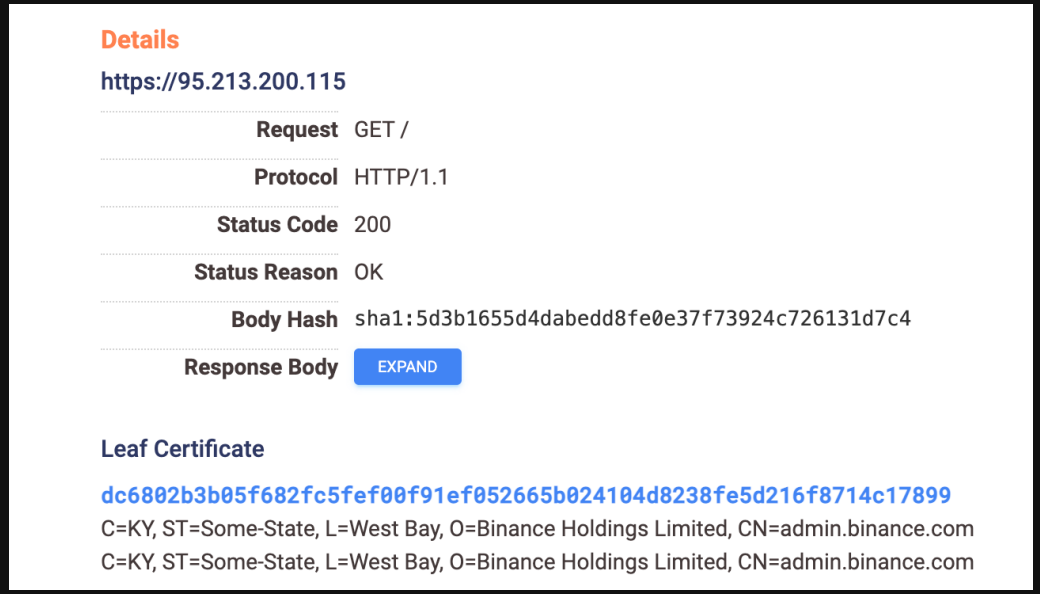
https://www.nginx.com/blog/rate-limiting-nginx/

Configuring NGINX: Security

What must be done

- Cache
- CORS
- CSP (with report-uri)
- Hardening

Configuring NGINX: Security



Example from censys.io, how you can find a certificate by ip (or vice versa)

Configuring NGINX: Security

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;
        listen 443 ssl default_server;
        listen [::]:443 ssl default_server;
        ssl_reject_handshake on; # nginx ≥ 1.19.4
        server_name _;

        location / {
                return 444;
        }
}
```

Return 444 or 204

80% of the effects come from 20% of the causes.

NGINX leveling up: SSL Fingerprint

Client Hello ×

The session begins with the client saying "Hello". The client provides the following:

- protocol version
- client random data (used later in the handshake)
- an optional session id to resume
- a list of cipher suites
- a list of compression methods
- a list of extensions

Annotations

```
16 03 01 00 a5 01 00 00 a1 03 03 00 01 02 03 04 05 06 Cipher Suites 0c 0d 0e
0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 00 00 20 cc a8 cc a9 c0 2f
c0 30 c0 2b c0 2c c0 13 c0 09 c0 14 c0 0a 00 9c 00 9d 00 2f 00 35 c0 12 00 0a
01 00 00 58 00 00 00 18 00 16 00 00 13 65 78 61 6d 70 6c 65 2e 75 6c 66 68 65
69 6d 2e 6e 65 74 00 05 00 05 01 00 00 00 00 00 0a 00 0a 00 08 00 1d 00 17 00
18 00 19 00 0b 00 02 01 00 00 0d 00 12 00 10 04 01 04 03 05 01 05 03 06 01 06
03 02 01 02 03 ff 01 00 01 00 00 12 00 00
```

- `00 20` - 0x20 (32) bytes of cipher suite data
- `cc a8` - assigned value for `TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`
- `cc a9` - assigned value for `TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`
- `c0 2f` - assigned value for `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `c0 30` - assigned value for `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `c0 2b` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `c0 2c` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `c0 13` - assigned value for `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`
- `c0 09` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`
- `c0 14` - assigned value for `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `c0 0a` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
- `00 9c` - assigned value for `TLS_RSA_WITH_AES_128_GCM_SHA256`
- `00 9d` - assigned value for `TLS_RSA_WITH_AES_256_GCM_SHA384`
- `00 2f` - assigned value for `TLS_RSA_WITH_AES_128_CBC_SHA`
- `00 35` - assigned value for `TLS_RSA_WITH_AES_256_CBC_SHA`
- `c0 12` - assigned value for `TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `00 0a` - assigned value for `TLS_RSA_WITH_3DES_EDE_CBC_SHA`

https://tls12.xargs.org/

NGINX leveling up: SSL Fingerprint

$ssl_protocol

SSLv2, SSLv3, **TLSv1, TLSv1.1, TLSv1.2,** TLSv1.3

$ssl_cipher

ECDHE-RSA-AES128-GCM-SHA256/TLSv1.2

https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_protocols

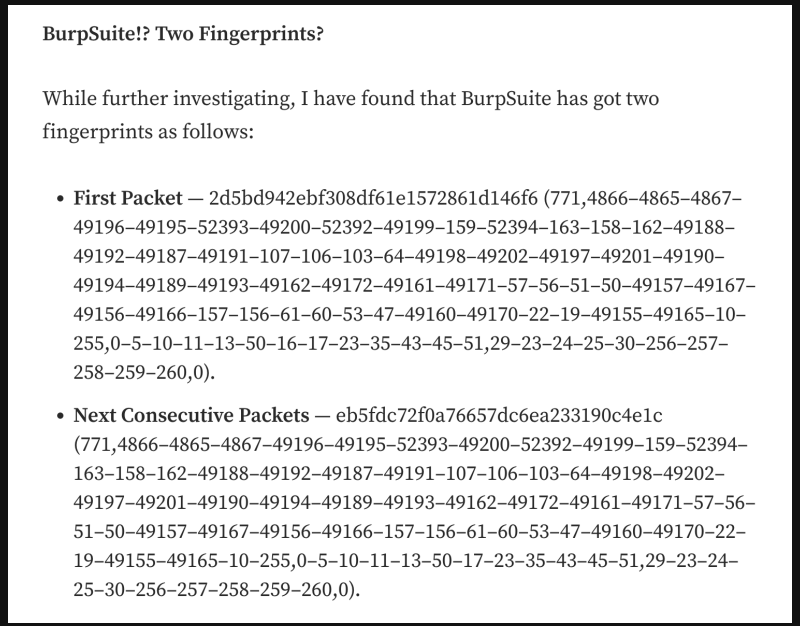
NGINX leveling up: SSL Fingerprint

```
log_format all '$remote_addr - $remote_user [$time_local] '
               '$ssl_cipher/$ssl_protocol '
               '"$request" $status $body_bytes_sent '
               '"$http_referer" "$http_user_agent"';
```

ECDHE-RSA-AES128-GCM-SHA256/TLSv1.2
ECDHE-RSA-AES128-GCM-SHA256/TLSv1.2
ECDHE-RSA-AES256-GCM-SHA384/TLSv1.2
ECDHE-RSA-CHACHA20-POLY1305/TLSv1.2
ECDHE-RSA-CHACHA20-POLY1305/TLSv1.2

TLS_AES_128_GCM_SHA256/TLSv1.3
TLS_AES_128_GCM_SHA256/TLSv1.3
TLS_AES_256_GCM_SHA384/TLSv1.3
TLS_CHACHA20_POLY1305_SHA256/TLSv1.3
TLS_CHACHA20_POLY1305_SHA256/TLSv1.3

NGINX leveling up: SSL Fingerprint



BurpSuite!? Two Fingerprints?

While further investigating, I have found that BurpSuite has got two fingerprints as follows:

- **First Packet** — 2d5bd942ebf308df61e1572861d146f6 (771,4866–4865–4867–49196–49195–52393–49200–52392–49199–159–52394–163–158–162–49188–49192–49187–49191–107–106–103–64–49198–49202–49197–49201–49190–49194–49189–49193–49162–49172–49161–49171–57–56–51–50–49157–49167–49156–49166–157–156–61–60–53–47–49160–49170–22–19–49155–49165–10–255,0–5–10–11–13–50–16–17–23–35–43–45–51,29–23–24–25–30–256–257–258–259–260,0).
- **Next Consecutive Packets** — eb5fdc72f0a76657dc6ea233190c4e1c (771,4866–4865–4867–49196–49195–52393–49200–52392–49199–159–52394–163–158–162–49188–49192–49187–49191–107–106–103–64–49198–49202–49197–49201–49190–49194–49189–49193–49162–49172–49161–49171–57–56–51–50–49157–49167–49156–49166–157–156–61–60–53–47–49160–49170–22–19–49155–49165–10–255,0–5–10–11–13–50–17–23–35–43–45–51,29–23–24–25–30–256–257–258–259–260,0).

https://infosecwriteups.com/demystifying-ja3-one-handshake-at-a-time-c80b04ccb393

NGINX leveling up: SSL Fingerprint - module

Variables

Name	Default Value	Comments
http_ssl_greased	0	Chrome grease flag
http_ssl_ja3	NULL	The ja3 fingerprint for a SSL connection for a HTTP server.
http_ssl_ja3_hash	NULL	ja3 md5 hash

Example

```
http {
    server {
        listen                127.0.0.1:8443 ssl;
        ssl_certificate       cert.pem;
        ssl_certificate_key   priv.key;
        error_log             /dev/stderr debug;
        return                200 "$http_ssl_ja3";
    }
}
```

https://github.com/phuslu/nginx-ssl-fingerprint

NGINX leveling up: SSL Fingerprint - bypass

```
const initCycleTLS = require('cycletls');
// Typescript: import initCycleTLS from 'cycletls';

(async () => {
  // Initiate CycleTLS
  const cycleTLS = await initCycleTLS();

  // Send request
  const response = await cycleTLS('https://ja3er.com/json', {
    body: '',
    ja3: '771,4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-51-57-47-5
    userAgent: 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0',
    proxy: 'http://username:password@hostname.com:443'
  }, 'get');

  console.log(response);

  // Cleanly exit CycleTLS
  cycleTLS.exit();

})();
```

https://github.com/Danny-Dasilva/CycleTLS

NGINX leveling up: TCP Fingerprint

A snippet of typical p0f output may look like this:

```
.-[ 1.2.3.4/1524 -> 4.3.2.1/80 (syn) ]-
|
| client   = 1.2.3.4
| os       = Windows XP
| dist     = 8
| params   = none
| raw_sig  = 4:120+8:0:1452:65535,0:mss,nop,nop,sok:df,id+:0
|
`----

.-[ 1.2.3.4/1524 -> 4.3.2.1/80 (mtu) ]-
|
| client   = 1.2.3.4
| link     = DSL
| raw_mtu  = 1492
|
`----
```

https://github.com/p0f/p0f (and other implementations)

NGINX leveling up: TCP Fingerprint

Application level

- Run p0f in daemon mode
- Retrieve data by using a programming language
- Get a json file as output and work with the data

https://github.com/ValdikSS/p0f-mtu-script/blob/master/index.php

NGINX leveling up: TCP Fingerprint - bypass

You can also emulate the full p0f OS, using '-' with the OS Genre and '-d' with custom details:

```
$ osfooler-ng -o "Windows" -d "XP bare-bone"
 [+] Mutating to p0f:
     WWW:65520|TTL:128|D:1|SS:48|OOO:M*,N,N,S|QQ:.|OS:Windows|DETAILS:XP bare-bone
[+] Activating queues
     [->] Process-1: p0f packet processor
```

https://github.com/segofensiva/OSfooler-ng

Configuring NGINX: Fast recipe

Download the generated config: **nginxconfig.io-example.com.tar.gz**
and **upload** it to your server's `/etc/nginx` directory.

or, **Copy a base64 string of the compressed config**, paste it in your server's command line and execute it.

Navigate to your NGINX **configuration directory** on your server:

```
cd /etc/nginx
```

Create a **backup** of your current NGINX configuration:

```
tar -czvf nginx_$(date +'%F_%H-%M-%S').tar.gz nginx.conf sites-available/ sites-enabled/ nginxconfig.io/
```

Extract the new compressed configuration archive using tar:

```
tar -xzvf nginxconfig.io-example.com.tar.gz | xargs chmod 0644
```

https://www.digitalocean.com/community/tools/nginx

Configuring NGINX: Fast recipe

```
git clone https://github.com/google/ngx_brotli.git
git clone https://github.com/phuslu/nginx-ssl-fingerprint
git clone https://github.com/leev/ngx_http_geoip2_module.git

git clone -b OpenSSL_1_1_1-stable --depth=1 https://github.com/openssl/openssl
git clone -b release-1.23.1 --depth=1 https://github.com/nginx/nginx

patch -p1 -d openssl < nginx-ssl-fingerprint/patches/openssl.1_1_1.patch
patch -p1 -d nginx < nginx-ssl-fingerprint/patches/nginx.patch

cd nginx

ASAN_OPTIONS=symbolize=1 ./auto/configure --prefix=/usr/share/nginx --conf-path=/etc/nginx/nginx.conf --http-log-path=/va
lock-path=/var/lock/nginx.lock --pid-path=/run/nginx.pid --modules-path=/usr/lib/nginx/modules --http-client-body-temp-pa
--http-proxy-temp-path=/var/lib/nginx/proxy --http-scgi-temp-path=/var/lib/nginx/scgi --http-uwsgi-temp-path=/var/lib/ngi
http_realip_module --with-http_auth_request_module --with-http_v2_module --with-http_slice_module --with-threads --with-h
http_sub_module --with-stream --with-stream_ssl_module --with-stream --with-openssl=$(pwd)/../openssl --add-module=$(pwd)
module=$(pwd)/../ngx_http_geoip2_module && make && ./objs/nginx -t && objs/nginx -p . -c $(pwd)/../nginx-ssl-fingerprint/
```

https://gist.github.com/Bo0oM/83fb14de8bf70e08de1131f74ec8cd65

Defense is an increase in the cost of attack

Bot mitigation: HTTP requests

- We enabled brotli and someone is still asking for gzip / deflate
- Value of the headers is not the same as the browser (e.g. without space)
- You have push set up, but the file is still requested by a separate request

Bot mitigation: HTTP requests

- HTTP headers are not in order
- Requests come from HTTP/1.0, HTTP/1.1, HTTP/1.2 (yes, that happens too)
- No headers (or vice versa), Origin, Referer, Accept, Sec-Ch-Ua

Bot mitigation: Active checks from JS

1. Create a domain with a self-signed certificate
2. Try to download an image from there
3. Did the onload event work? We have a villain in front of us!

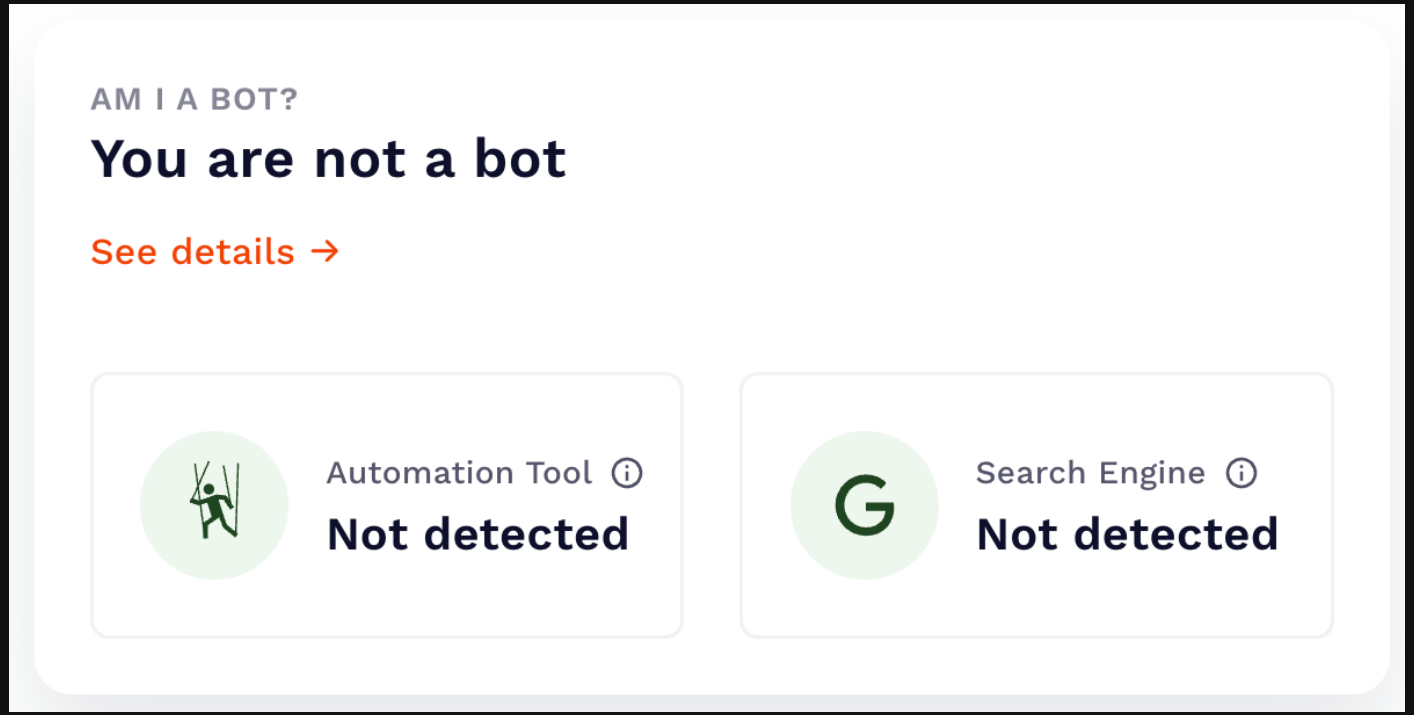
Utilities like burp suite ignore the error if the certificate is not valid.

Bot mitigation: Active checks from JS

1. Create subdomain (or other domain)
2. Try to download a third-party resource where CORS is interfering
3. Download successful? There's a villain in front of us!

Chrome Headless may be running with the "--disable-web-security" argument

Bot mitigation: Checking JS environment

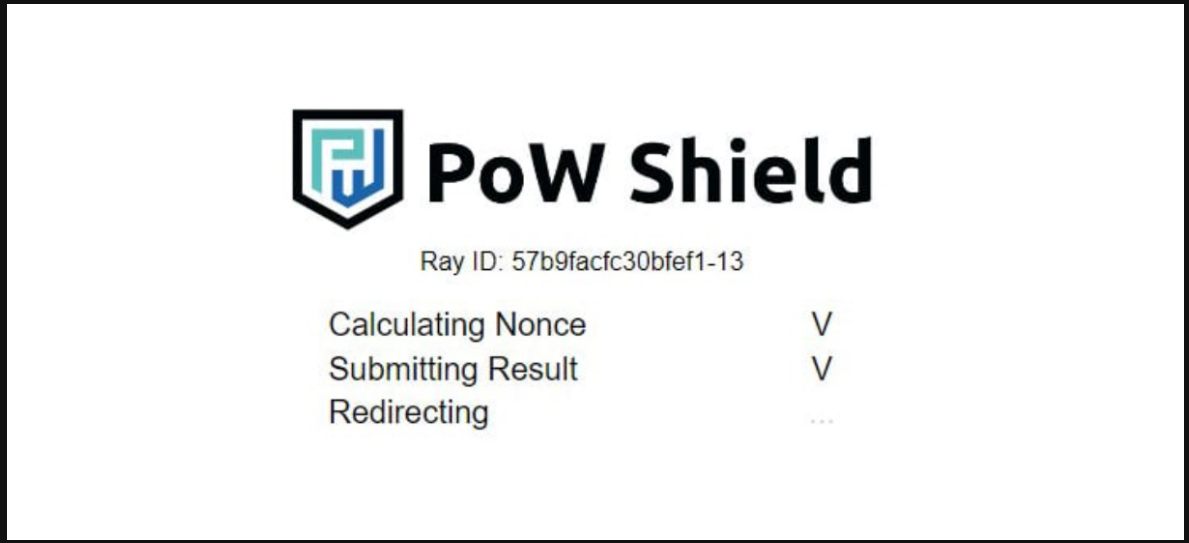


https://github.com/fingerprintjs

Bot mitigation: Useragents

https://github.com/mitchellkrogza/nginx-ultimate-bad-bot-blocker

Bot mitigation: Proof of work



https://github.com/kyprizel/testcookie-nginx-module

https://github.com/mCaptcha/mCaptcha

https://github.com/RuiSiang/PoW-Shield

Bot mitigation: SSHguard / Fail2ban

```
[nginx-limit-req]

enabled  = true
port     = http,https
logpath = %(nginx_error_log)s


[nginx-4xx]

enabled = true
port = http,https
logpath = /var/log/nginx/access.log
maxretry = 10
findtime = 300

[nginx-botsearch]

enabled  = true
port     = http,https
logpath  = %(nginx_access_log)s
maxretry = 3
```

< Block if rate limit are exceeded

< Block if a dirbuster

< Block if sensitive paths or honeypot (.git, dump, etc)

But is it necessary to block the intruder?

Bot mitigation: A/B

```
http {
    # ...
    # application version 1a
    upstream version_1a {
        server 10.0.0.100:3001;
        server 10.0.0.101:3001;
    }

    # application version 1b
    upstream version_1b {
        server 10.0.0.104:6002;
        server 10.0.0.105:6002;
    }

    split_clients "${arg_token}" $appversion {
        95%     version_1a;
        *       version_1b;
    }

    server {
        # ...
        listen 80;
```

https://www.nginx.com/blog/performing-a-b-testing-nginx-plus/

Bot mitigation: A/B

```
location ~* "^/(archive|auth|backup|clients|com|dat|dump|engine|files|home|html|index|
  access_log /var/log/nginx/dummy.log;
  default_type application/zip;
  root /usr/www/dummy;
  rewrite ^(.*)$ /r.zip break;
  max_ranges 0;
  limit_rate 2;
}
```

Maybe you want to ask a question?