BlueHat IL

Maor Abutbul

Why-So-QUIC!? Racing and Fuzzing HTTP/3 using QuicDraw-UI

paloalto NETWORKS | CYBERARK

BlueHat IL

Maor Abutbul

Why-So-QUIC!? Racing and Fuzzing HTTP/3 using QuicDraw-UI

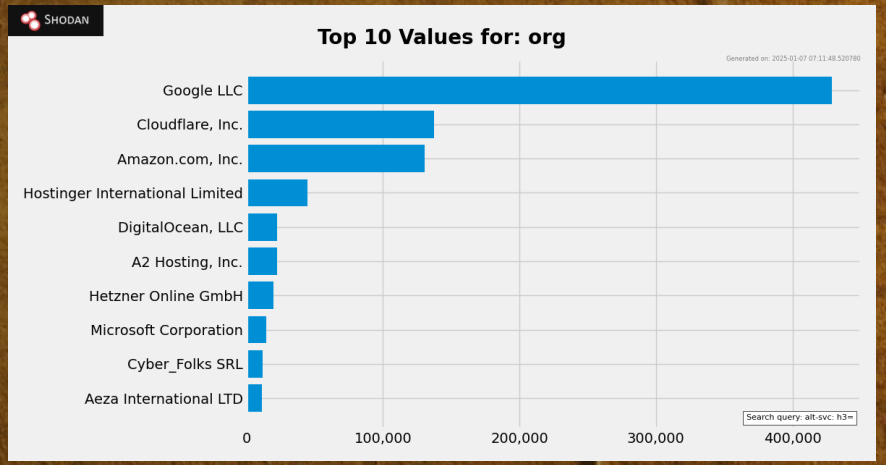
paloalto NETWORKS | CYBERARK

WHY HTTP/3?

UDP

HTTP/3? - Internet-wide support

- 39.8% (w3techs) of Internet Facing
  - ~13M+ HTTP3 Live Websites
- Major Browsers
- Major CDNs & Cloud Providers
- Major Websites



Alt-svc by organization - CDN's

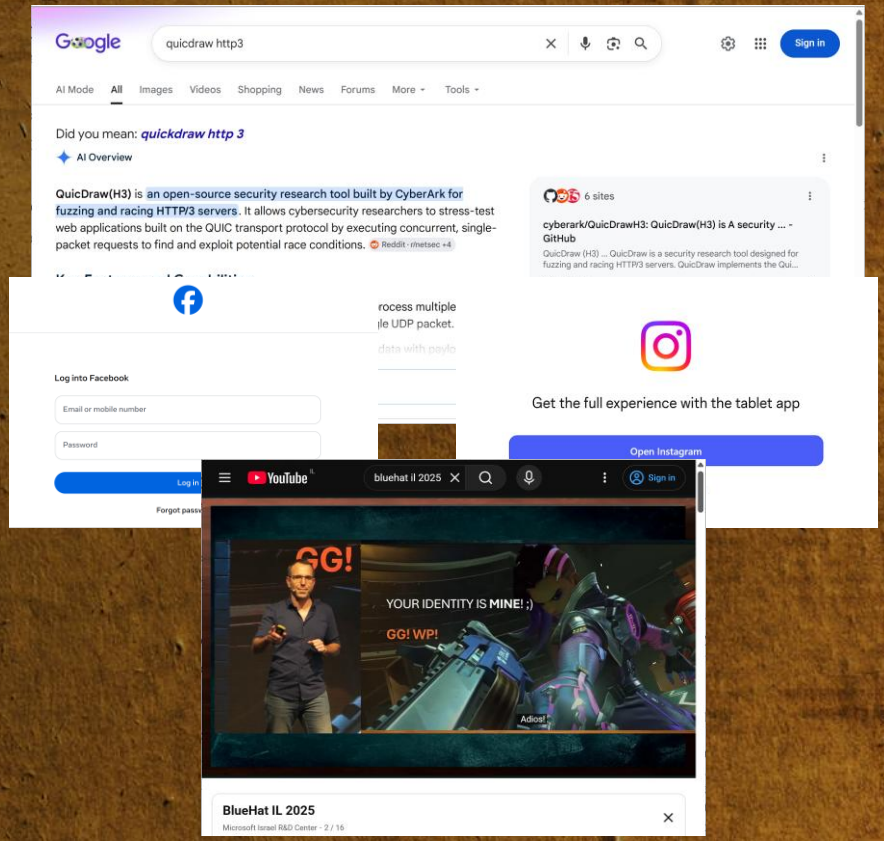
Browsers (caniuse.com/http3)

HTTP/3? - Internet-wide support

- 39.8% (w3techs) of Internet Facing
  - ~13M+ HTTP3 Live Websites
- Major Browsers
- Major CDNs & Cloud Providers
- Major Websites

Name	Method	Status	Protocol	Domain
www.youtube.com	GET	200	h3	www.youtube.com
www.instagram.com	GET	200	h3	www.instagram.com
www.google.com	GET	200	h3	www.google.com
www.facebook.com	GET	200	h3	www.facebook.com
www.cloudflare.com	GET	200	h3	www.cloudflare.com
www.bing.com	GET	200	h3	www.bing.com

Selected HTTP/3 domains



MISSION OBJECTIVE:

List HTTP/3 Features & Find security leads (scenarios)!

GET (:authority:) host :)

Name: Maor Abutbul

Background: Father, Engineer, Researcher, Gamer

Past: ~20 Years in Network & Security,
Engineering, AppSec, R&D, and Research

Current: Vulnerability Researcher @PANW Idira Labs

https://il.linkedin.com/in/maor-abutbul

Agenda

#1 Technical Background

#2 Introduction to HTTP/3

- Practical HTTP/3 Tips (&Tools)

#3 Our Research Journey

- ‘Quic-Fin-Sync’ (algorithm)

#4 QuicDraw & QuicDraw-UI (our open-source)

- QuicDraw Evaluation - Live Demo

#1 Technical Background

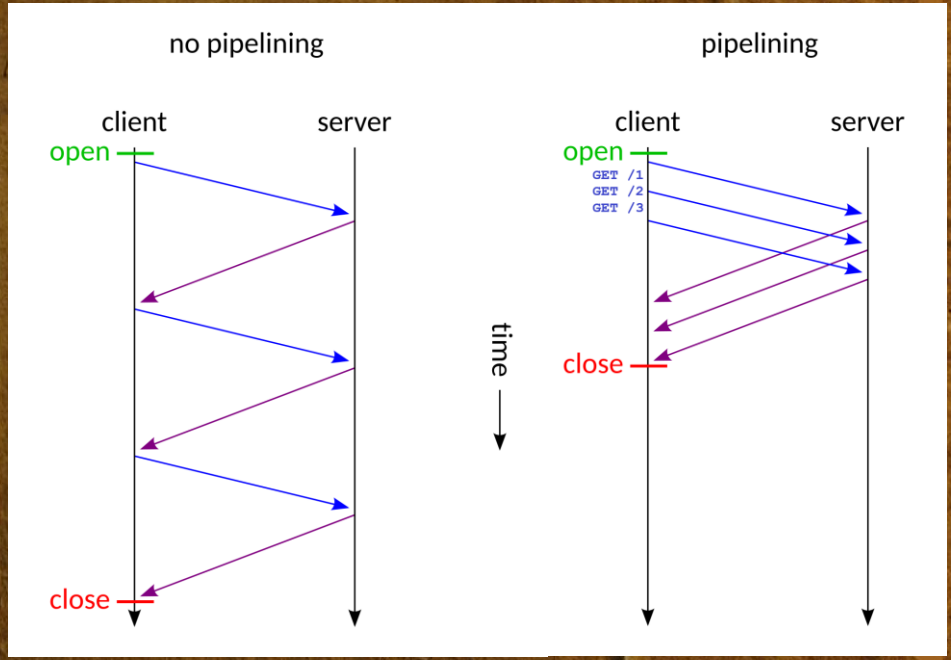
HTTP/1.1, HTTP/2 & HOL blocking

And why do we need **HTTP/3?**

HTTP/1.1 Pipelining

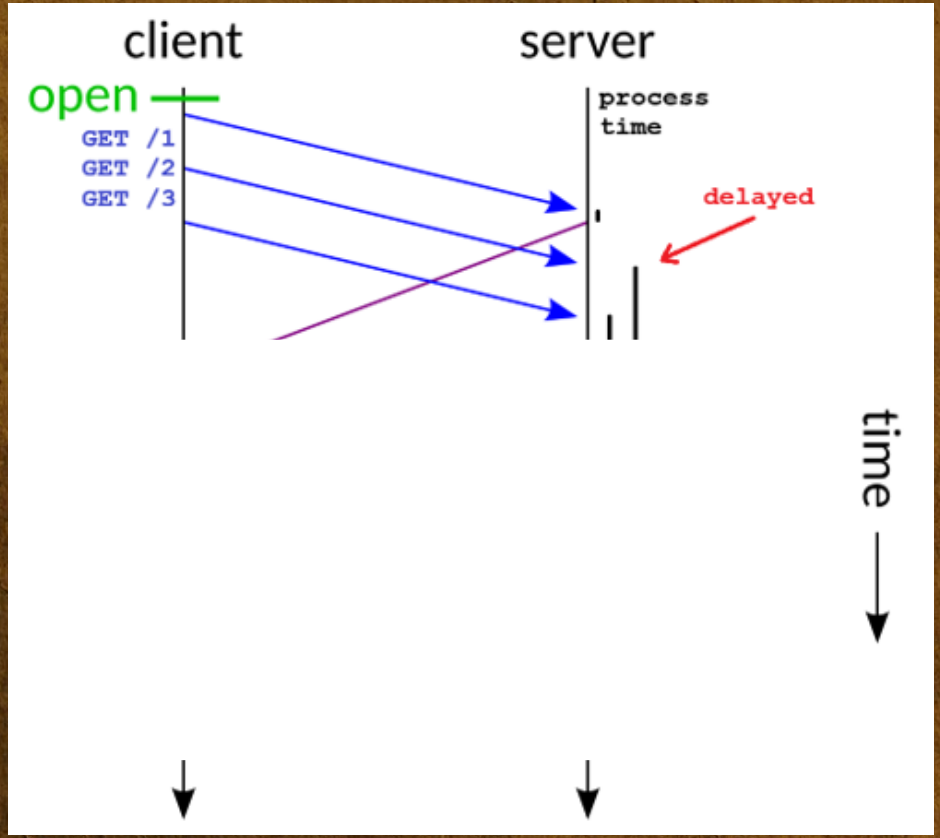
- HTTP/1.0 - response fully received
- HTTP/1.1 - introduced pipelining
- Limitation:
  - Response fully delivered (Ordered) before a new response can be sent.

- But What if a response is delayed?



HTTP/1.1 Pipelining

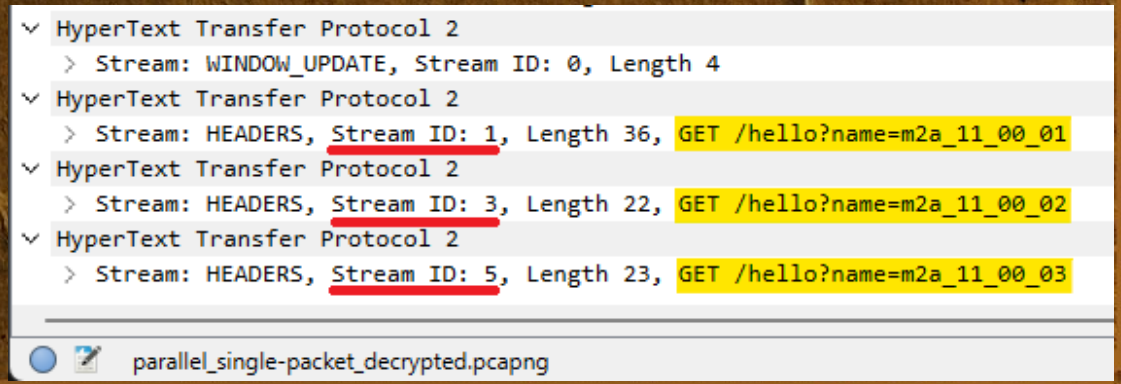
Pipelining – HTTP Head-Of-Line (HOL) Blocking



HTTP/1.1 HOL Blocking

Brief overview of HTTP2

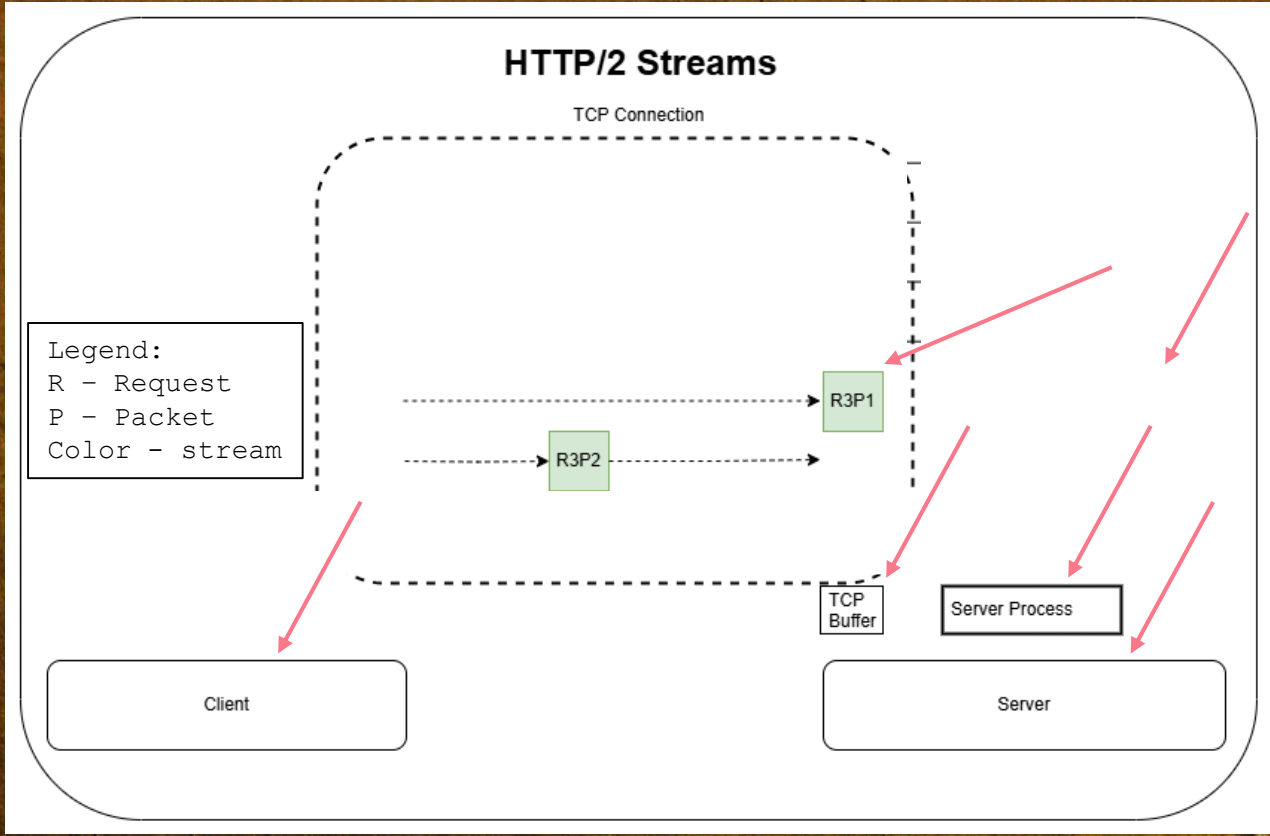
- Binary
- Encrypted (Browsers - De facto)
- Header compression (HPack)
- Support Multiplexing
  - (using frames and streams)



Brief overview of HTTP2

- Binary
- Encrypted (Browsers - De facto)
- Header compression (HPack)
- Support Multiplexing
  - (using frames and streams)
- Eliminates application-layer (HTTP/1.1) Head-Of-Line blocking

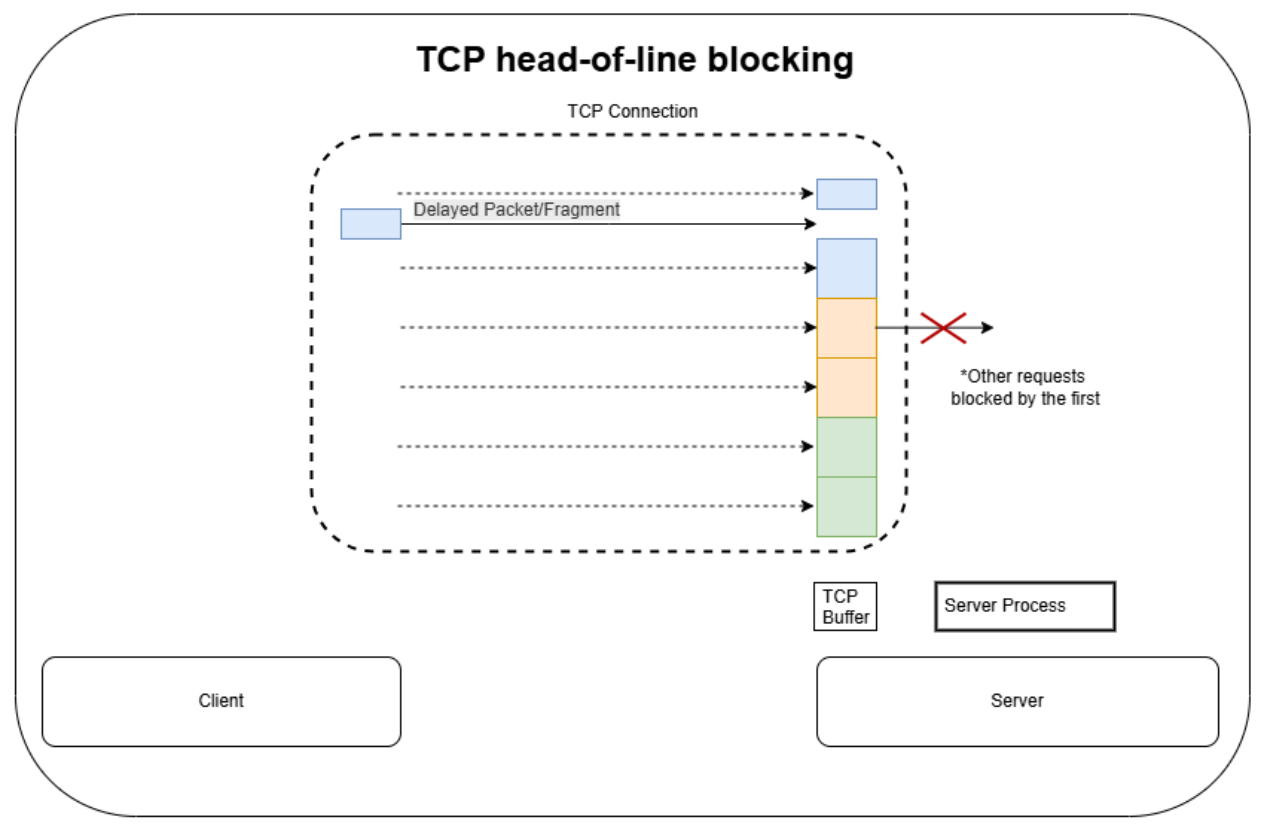
HTTP/2 Streams - HOL blocking Elimination



HTTP/2 Streams - sent immediately

BUT - NETWORK LAYER

TCP Head-Of-Line Blocking



HTTP Stack Perspective

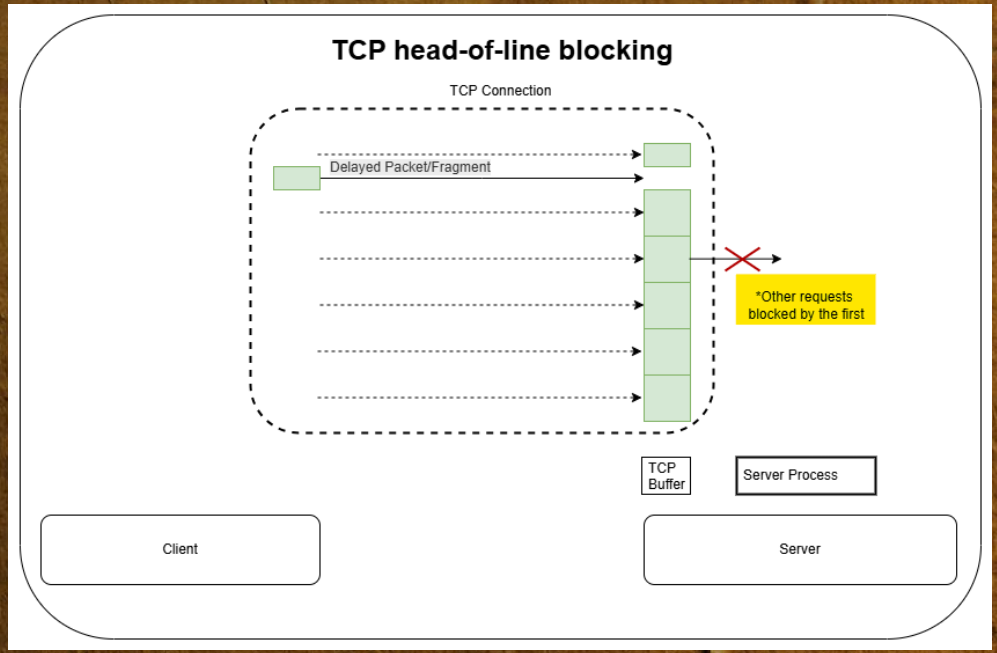
TCP Stack Perspective

TCP HOL blocking (HTTP / TCP stack)

Brief overview of HTTP2

HTTP/2

- Eliminates HTTP HOL blocking
- Suffers from TCP HOL blocking



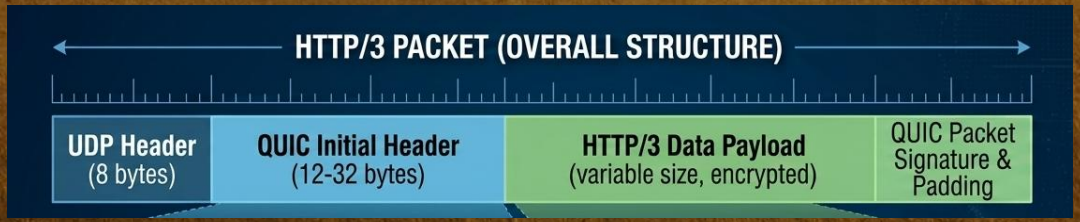
HTTP/2 Streams

SOLVING TCP HOL BLOCKING – PRIMARY MOTIVATIONS BEHIND HTTP/3

#2 Brief Introduction to HTTP/3

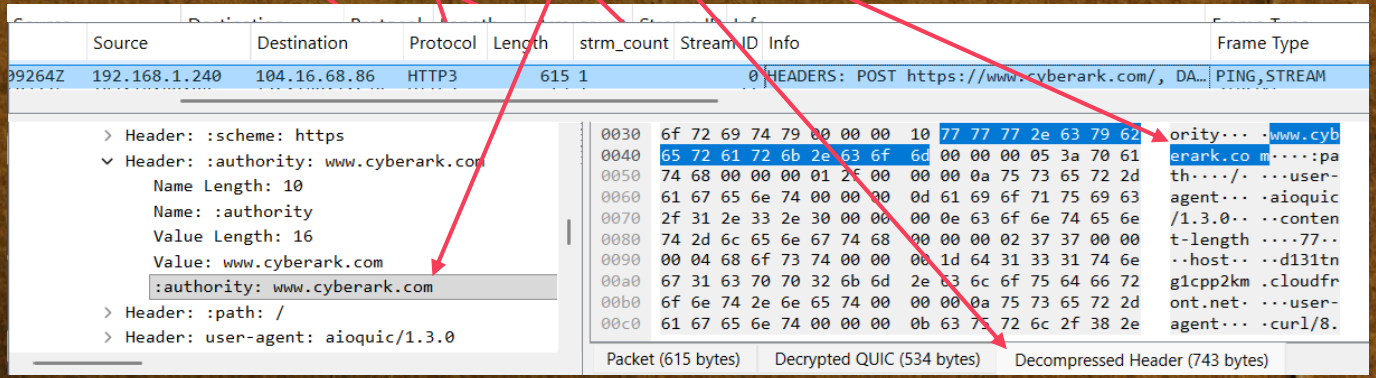
Note

HTTP/3 ~= HTTP/2 OVER QUIC



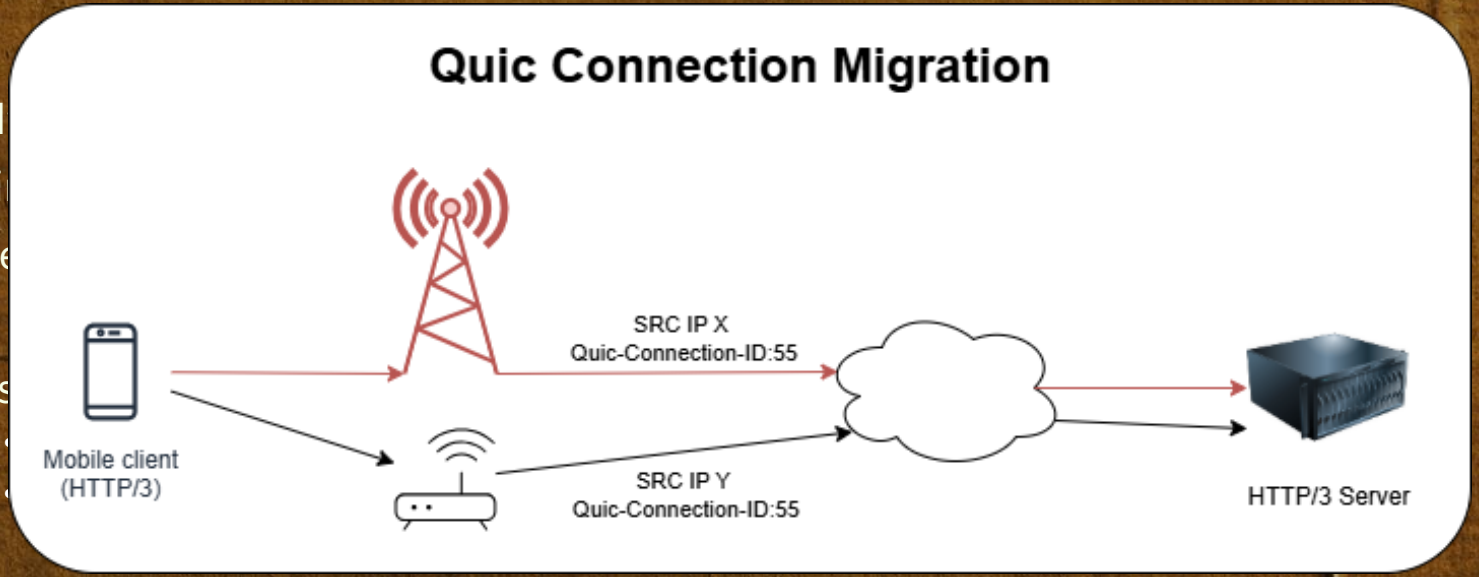
HTTP/3 (+Quic) Main Features

- Always Encrypted (TLS 1.3)
- Binary
- Header (Field) Compression (QPACK)

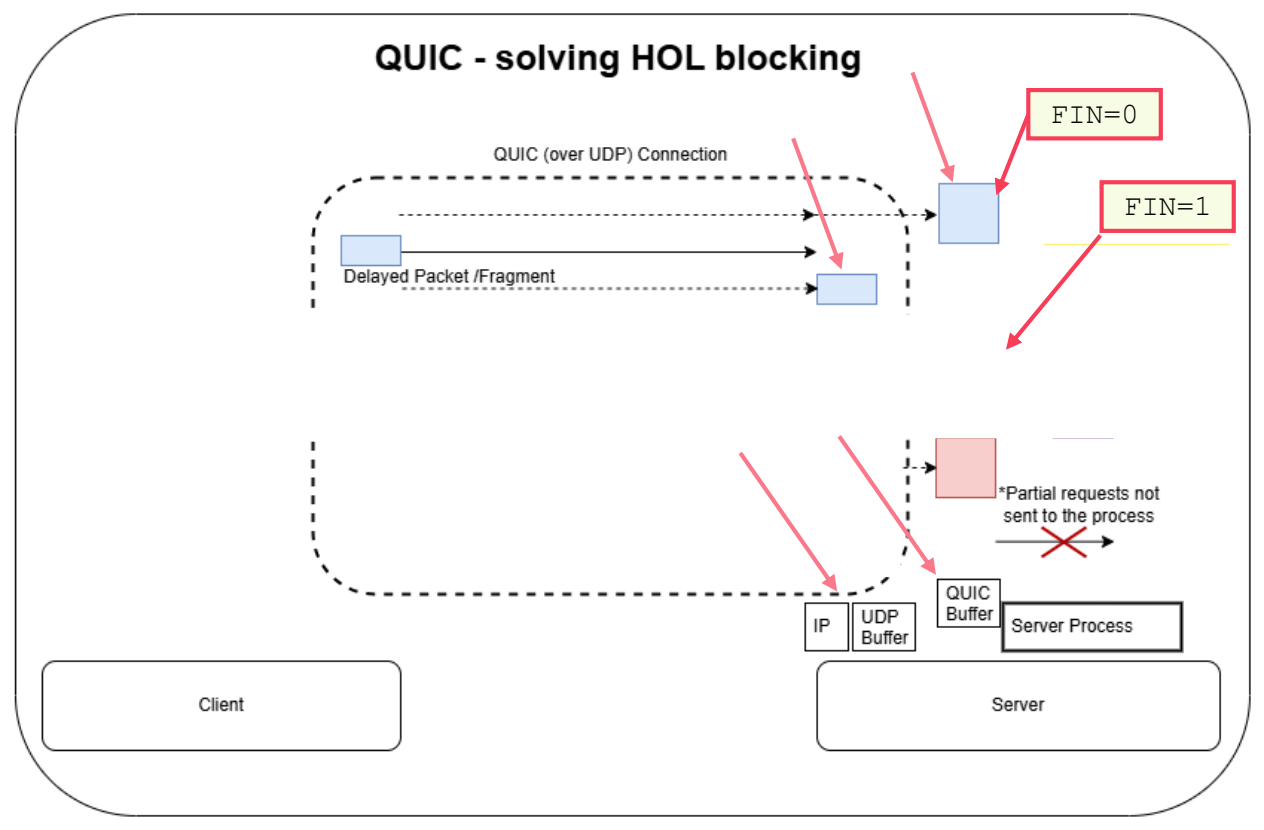


HTTP/3 (+Quic) Main Features

- Al
- Bi
- He
- Us
  - Connection Migration
  - Fast connection setup & 0-RTT
  - Multiplexing Without Head-of-Line Blocking



QUIC (multiplexing) Without Head-of-Line Blocking



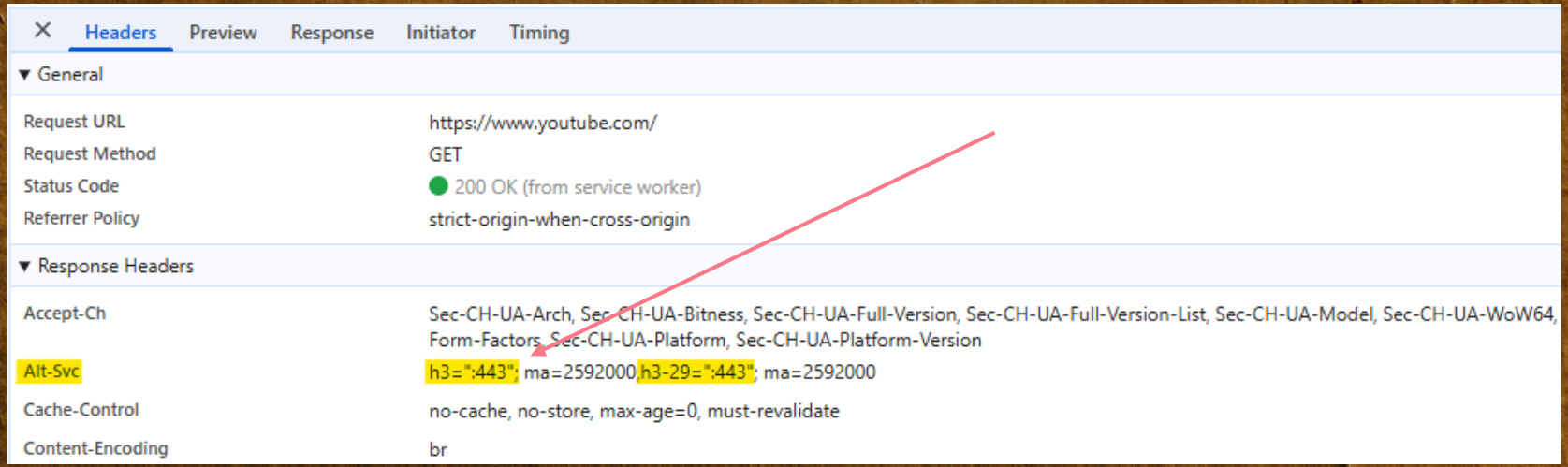
QUIC Solve TCP HOL Blocking

#2.1 HTTP/3 Practical Tips

Server supports HTTP/3?

HTTP/3 Support Discovery

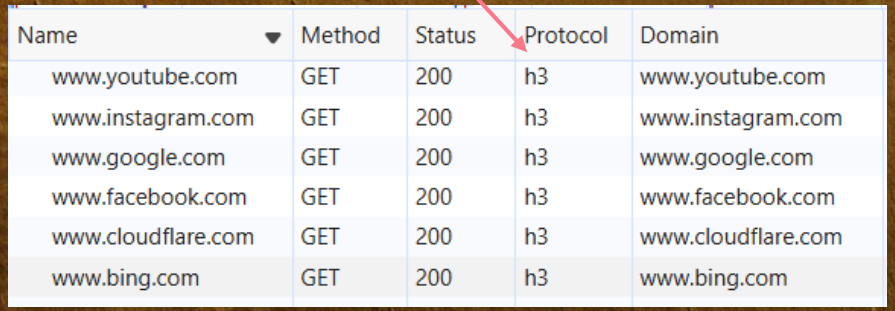
- The Alt-Svc header



The Alt-Svc header in a response indicates the server supports HTTP/3

HTTP/3 Support Discovery

- All Major web browsers support HTTP/3
  - Prefer using HTTP/3



Name	Method	Status	Protocol	Domain
www.youtube.com	GET	200	h3	www.youtube.com
www.instagram.com	GET	200	h3	www.instagram.com
www.google.com	GET	200	h3	www.google.com
www.facebook.com	GET	200	h3	www.facebook.com
www.cloudflare.com	GET	200	h3	www.cloudflare.com
www.bing.com	GET	200	h3	www.bing.com

TOOLS

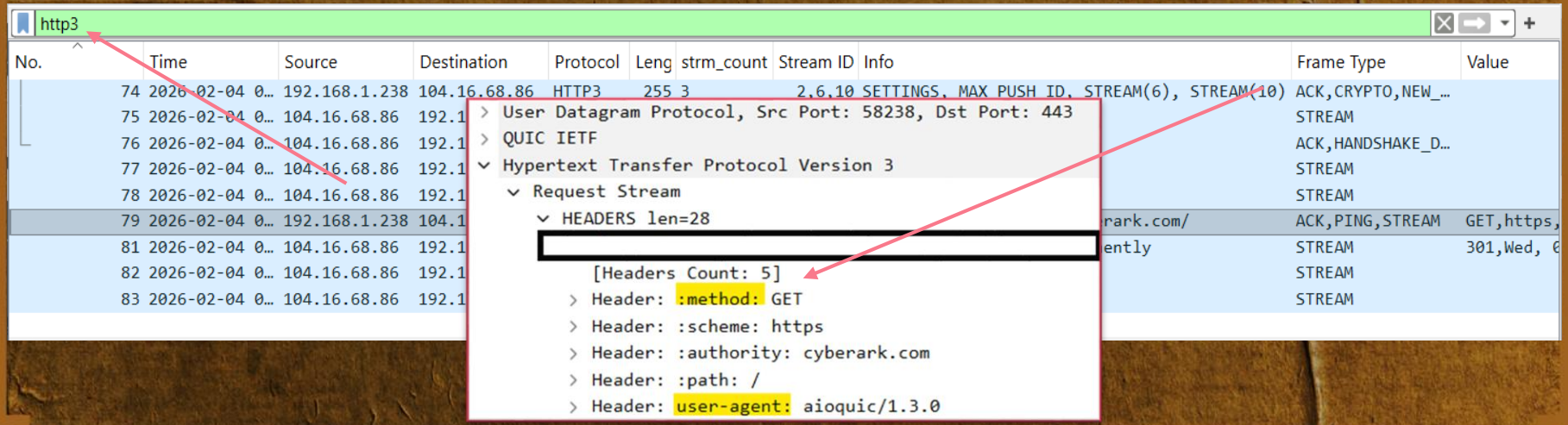
Tools supporting HTTP3 - browsers & curl

- All Major web browsers support HTTP/3.
  - *local proxy => not using HTTP/3
- Build curl with HTTP/3 Support
  - $ curl --http3-only https://cyberark.com
- (use “export SSLKEYLOGFILE=“ for SSL decryption)

```
$ curl3 --http3-only https://cyberark.com
<html>
<head><title>301 Moved Permanently</title></head>
<body><a href="https://cyberark.com/cdn-cgi/content?
```

Tools supporting HTTP3 - Wireshark

- All Major web browsers support HTTP/3.
- Build curl with HTTP/3 Support
- Wireshark ! (headers)
  - Edit -> Preferences -> Protocols -> TLS -> “Master Secret log filename”



Tools supporting HTTP3 - QuicDraw

- All Major web browsers support HTTP/3.
- Build curl with HTTP/3 Support
- Wireshark !
- QuicDraw & QuicDraw-UI ☺
  - $ pip install quicdraw[ui]
  - $ quicdraw https://cyberark.com

```
$ quicdraw https://cyberark.com
Response headers received (stream:0) GET  : status: (301)
2026-02-04 11:02:47,076 INFO QuicDraw Response data  received (stream:0) GET  : 162 bytes
```

Agenda

#1 Technical Background

#2 Introduction to HTTP/3

- Practical HTTP/3 Tips (&Tools)

#3 Our Research Journey

- ‘Quic-Fin-Sync’ (algorithm)

#4 QuicDraw & QuicDraw-UI (our open-source)

- QuicDraw Evaluation - Live Demo

#3 Our Research Journey

MISSION OBJECTIVE:

List HTTP/3 Features & Find security leads (scenarios)!

HTTP/3 ATTACK SCENARIOS - EASY

Comment

HTTP/3 Attack Scenarios – EZ (selected Scenarios)

- ~~Amplification~~
- ~~Slow Loris Attacks~~
- ~~User Tracking?~~
- ~~Scenarios #4+~~

THIS IS THE END

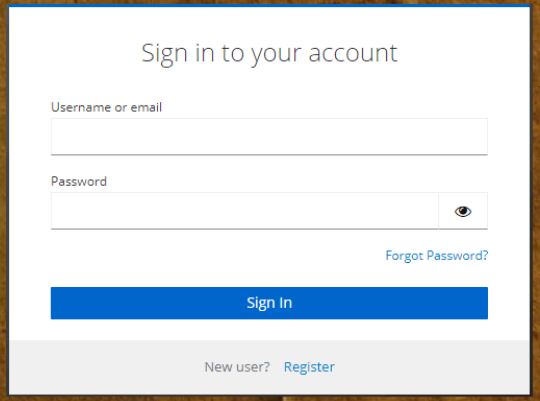
Or is it?

One more lead

HTTP/3 Attack Scenarios - One more lead

- Keycloak (Race condition finding)
  - Single-Packet Attack (Portswigger)
  - (More details later)

- HTTP/3 - Race conditions !?

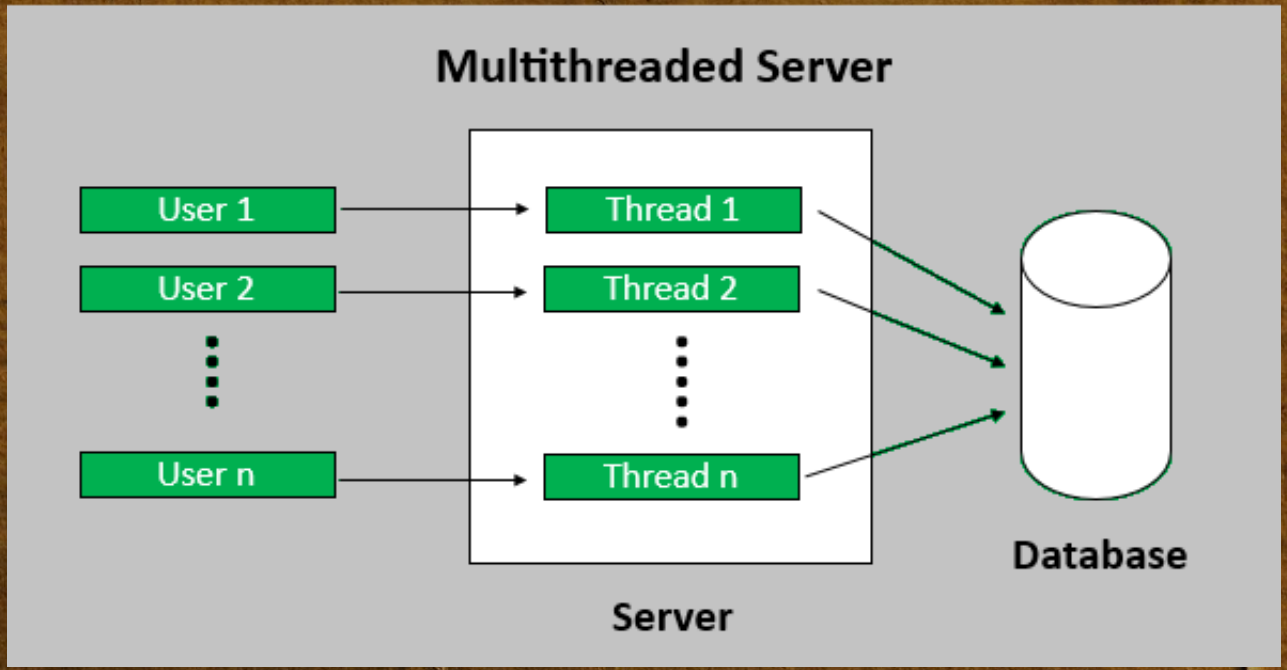


KEYCLOAK

Is it possible to make race conditions work in HTTP/3?

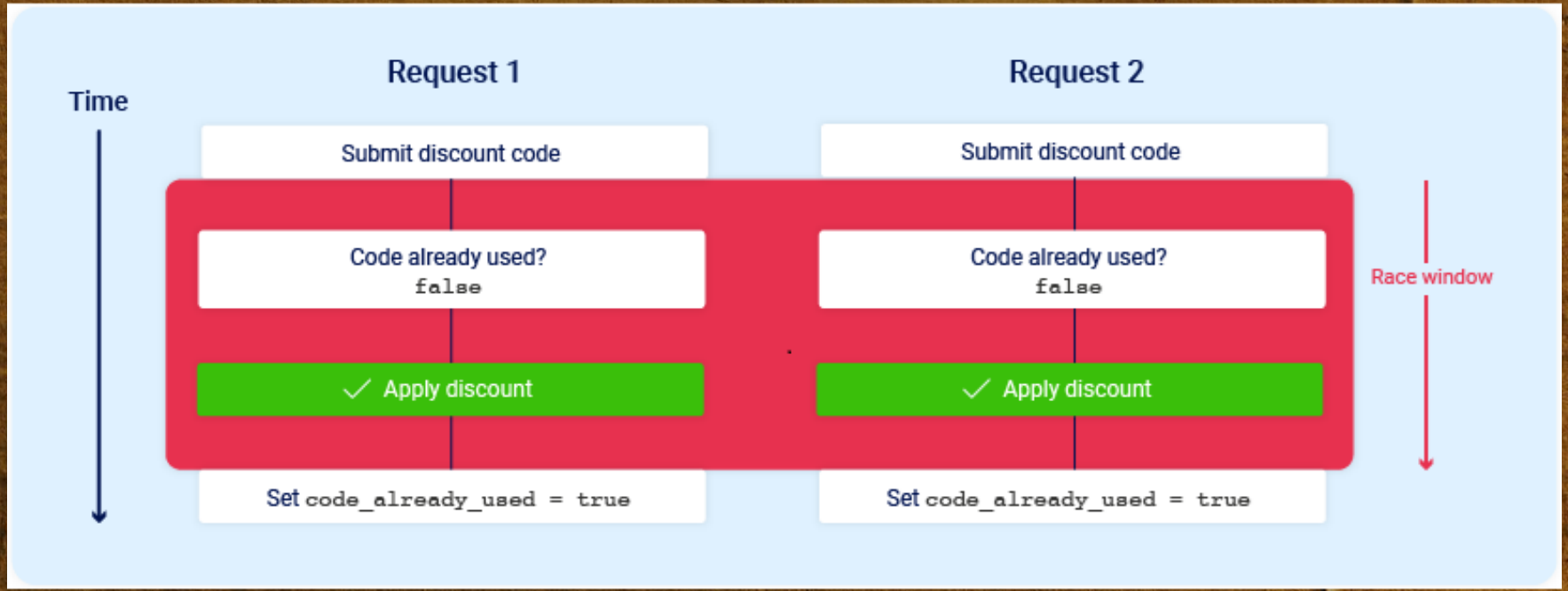
On Race Conditions

Multithreaded Server



"A race condition vulnerability requires a 'collision' of two concurrent operations on a shared resource." (PortSwigger Research)

The classic discount code - Race Flow



Race Conditions - Race Flow

QUIC on the RaceTrack

paloalto NETWORKS | CYBERARK

QUIC on the RaceTrack

Is it possible to make race conditions work in HTTP/3?

A way to send / release requests in bulk

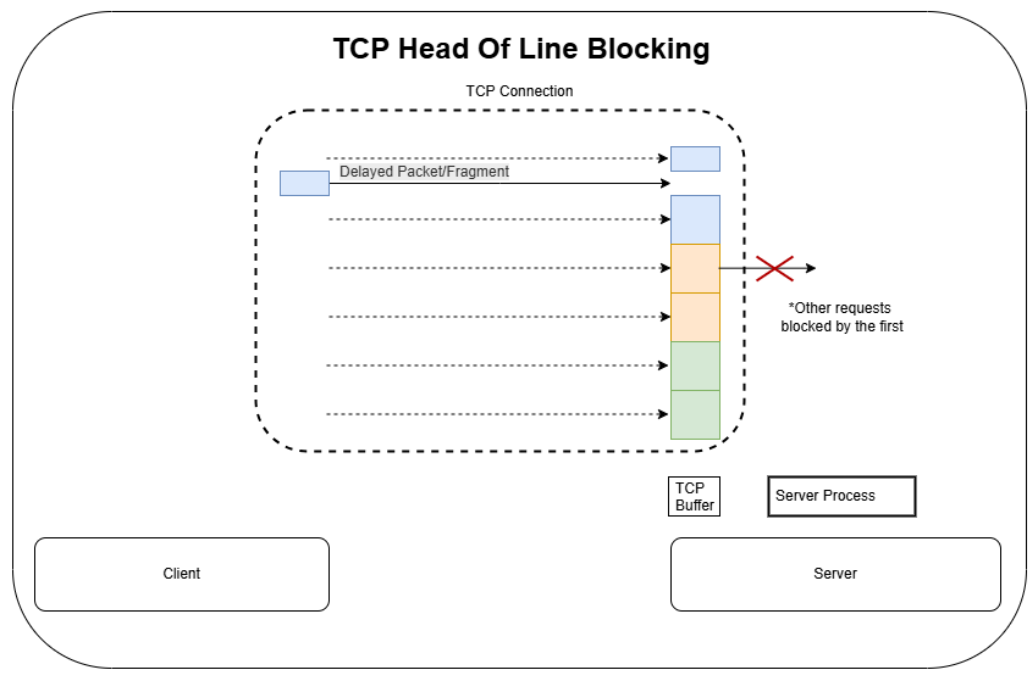
QUIC on the RaceTrack – First Attempt: Fragmentation

- HTTP/3 is ‘just’ HTTP2 over QUIC
  - (QUIC implements "the good parts of TCP")
- QUIC runs over UDP
- UDP runs over IP
  - IP has Fragmentation!

“Let’s just Fragment the HTTP/3 traffic”

Racing on HTTP/3 – “Let’s just fragment the HTTP/3 traffic”

Using TCP HOL Blocking (HTTP/2)



TCP HOL Blocking Fragmentation

Fragmentation – EZ ;)

Found a solution
EZ GG

"Humm…"

What?
We did! … Didn't we?

"NOP"

Why?

"RTFM!"

"RTFM (RTF RFC)"

NOP! Read The RFC

QUIC RFC (9000), “UDP datagrams MUST NOT be fragmented at the IP layer.”

> UDP datagrams **MUST NOT** be fragmented at the IP layer. In IPv4 [IPv4], the Don't Fragment (DF) bit **MUST** be set if possible, to prevent fragmentation on the path.

RFC9000 - QUIC don't fragment

¯_(ツ)_/¯

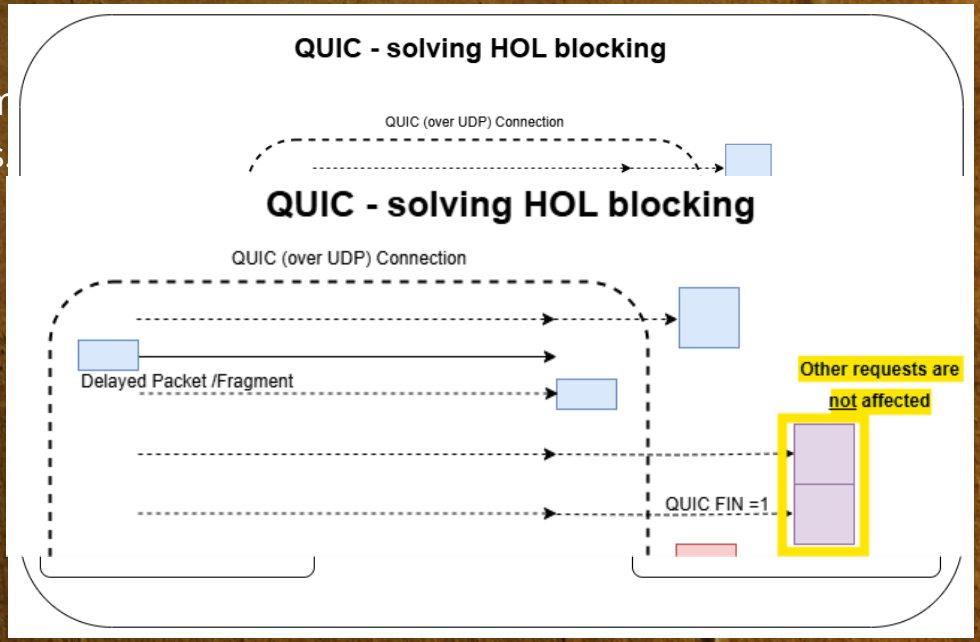
Racing on HTTP/3 - “Let’s just fragment the HTTP/3 traffic”

Anyway. Each request has its own stream!

Racing on HTTP/3 - “Let’s just fragment the HTTP/3 traffic”

Anyway. Each request has its own stream

- Should not interfere with other streams.



QUIC Solve HOL Blocking (Fragmentation)

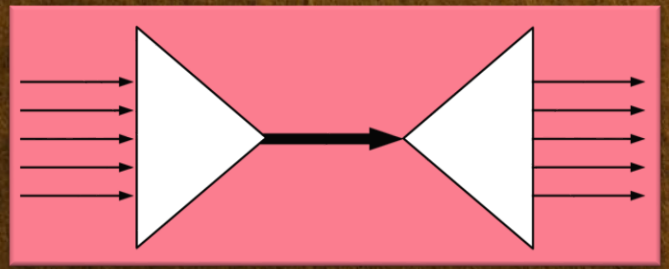
“Yes, But!”

“Streams can be Multiplexed!”

Amazing! But What's Multiplexing?

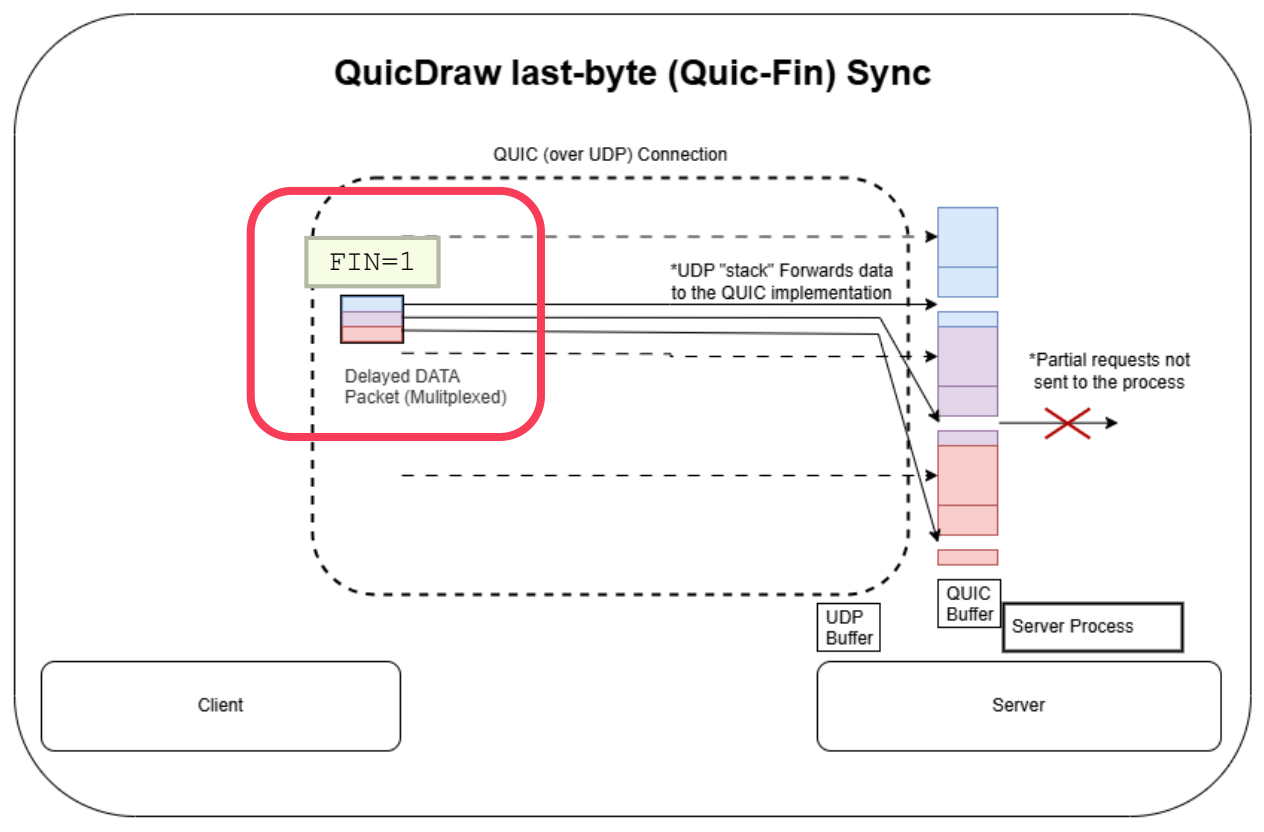
Current filter: http3

Request Versi	Info	Stream ID	Frame Type
	Protected Payload (KP0), DCID=33e641d76a95e0d4, PKN: 15, STREAM(0), HEADERS: 200 OK	0	STREAM
	Protected Payload (KP0), DCID=33e641d76a95e0d4, PKN: 16, STREAM(0), DATA	0	STREAM
	Protected Payload (KP0), DCID=33e641d76a95e0d4, PKN: 17, STREAM(7), STREAM(4), HEADERS: 200 OK	7,4	STREAM,STREAM
	Protected Payload (KP0), DCID=33e641d76a95e0d4, PKN: 18, STREAM(4), DATA	4	STREAM
	Protected Payload (KP0), DCID=33e641d76a95e0d4, PKN: 19, STREAM(8), HEADERS: 200 OK	8	STREAM
	Protected Payload (KP0), DCID=33e641d76a95e0d4, PKN: 20, STREAM(8), DATA	8	STREAM



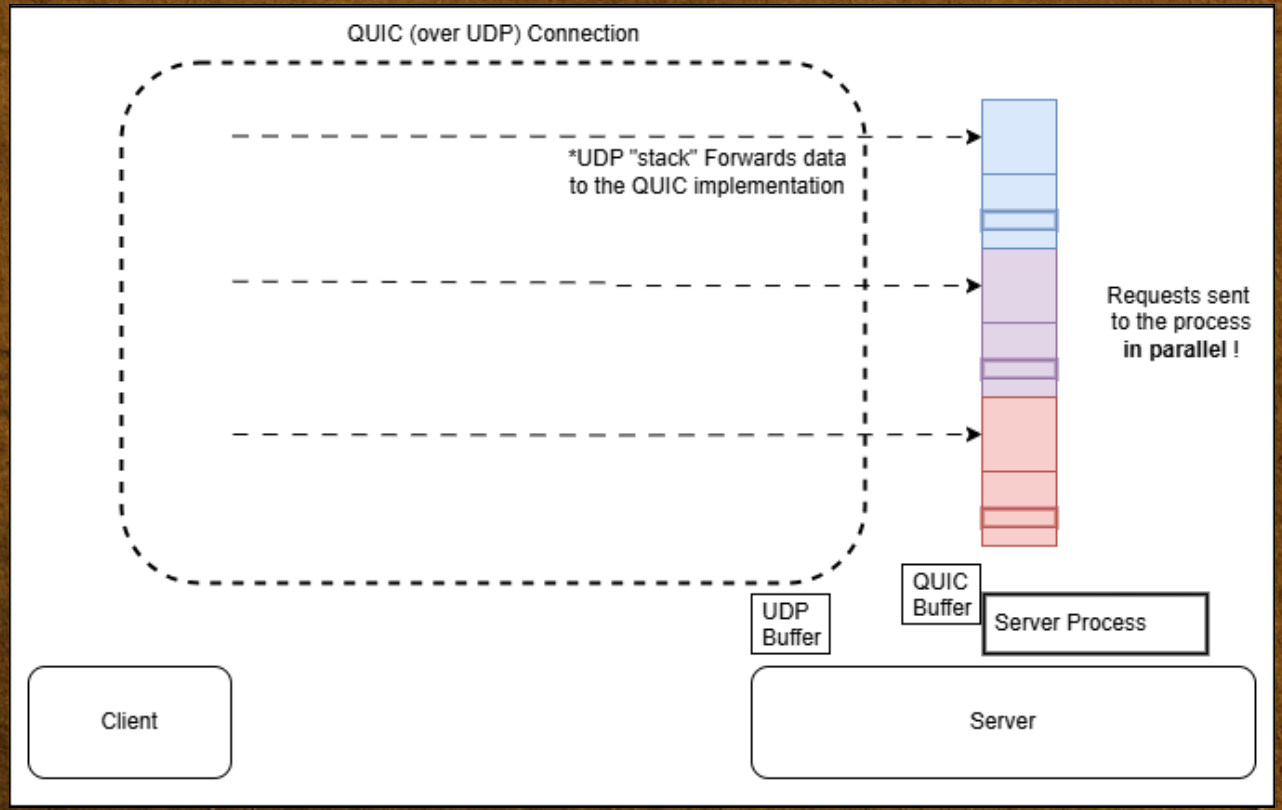
#3.2 Quic-Fin-Sync - Concept

QUIC on the Racetrack - Multiplexing - Concept



QuicDraw Network (Concept)

QUIC on the Racetrack - Quic-Fin-Sync - Concept



QuicDraw Network (Concept)

Decided to build our own tool

Decided to build our own tool

Why reinvent the wheel?

Agenda

#1 Technical Background (HTTP/2)

#2 Introduction to HTTP/3

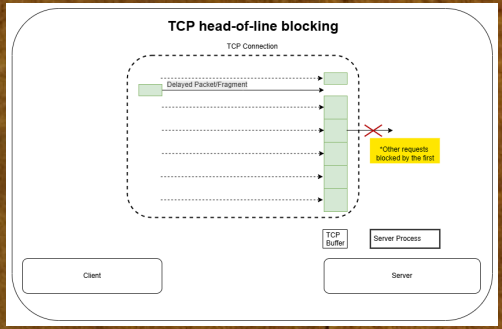
- Practical HTTP/3 Tips (&Tools)

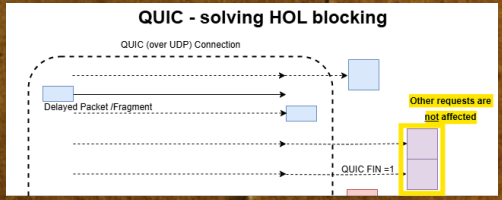
#3 Our Research Journey

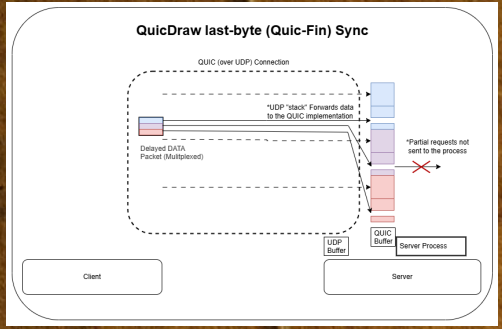
- ‘Quic-Fin-Sync’ (algorithm)

#4 QuicDraw & QuicDraw-UI (open-source)

- QuicDraw Evaluation - Live Demo



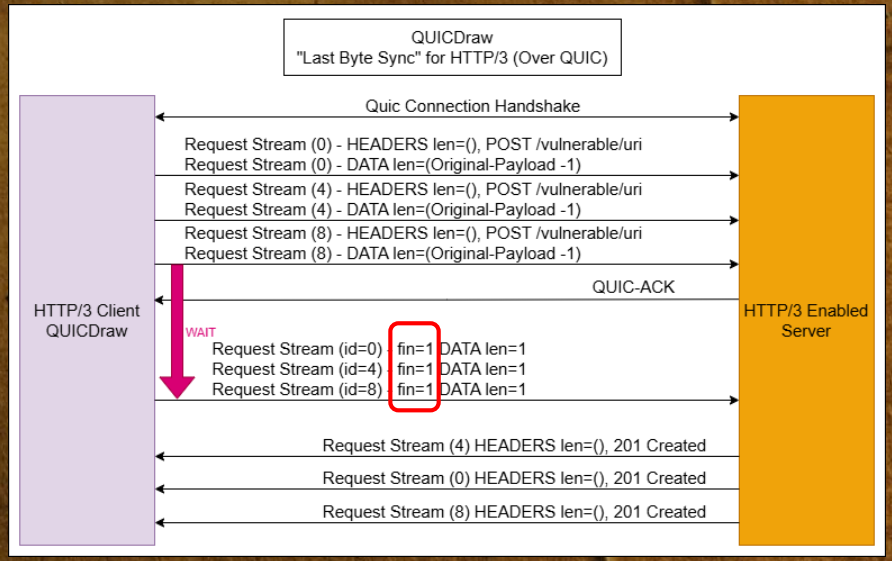




#4 QuicDraw & QuicDraw-UI

Our Quic-Fin-Sync Implementation - The Algorithm and Traffic Diagram

```
For each Request
    queue Request-Headers
    queue Request-Data[:-1]

Transmit

wait 1000 ms

For each Request
    queue RequestData[-1:]& end_stream=1
                          (QUIC-FIN = 1)

Transmit
```



QuicDraw Quic-Fin-Sync (HTTP3) traffic diagram

QuicDraw & QuicDraw-UI

```
-----------
QuicDraw: HTTP/3 Fuzzing and Racing (Client)
-----------

        |QFS____| -------------------- HTTP/3

GitHub: https://github.com/cyberark/QuicDrawH3
License: Apache-2.0 License
Author: Maor Abutbul <CyberArk Labs>
Version: 0.8.27
-----------
```

QuicDraw(&QuicDraw-UI) GitHub Repo

Racing HTTP/3 applications

$ pip install quicdraw[ui]

```
quicdraw <url>
```

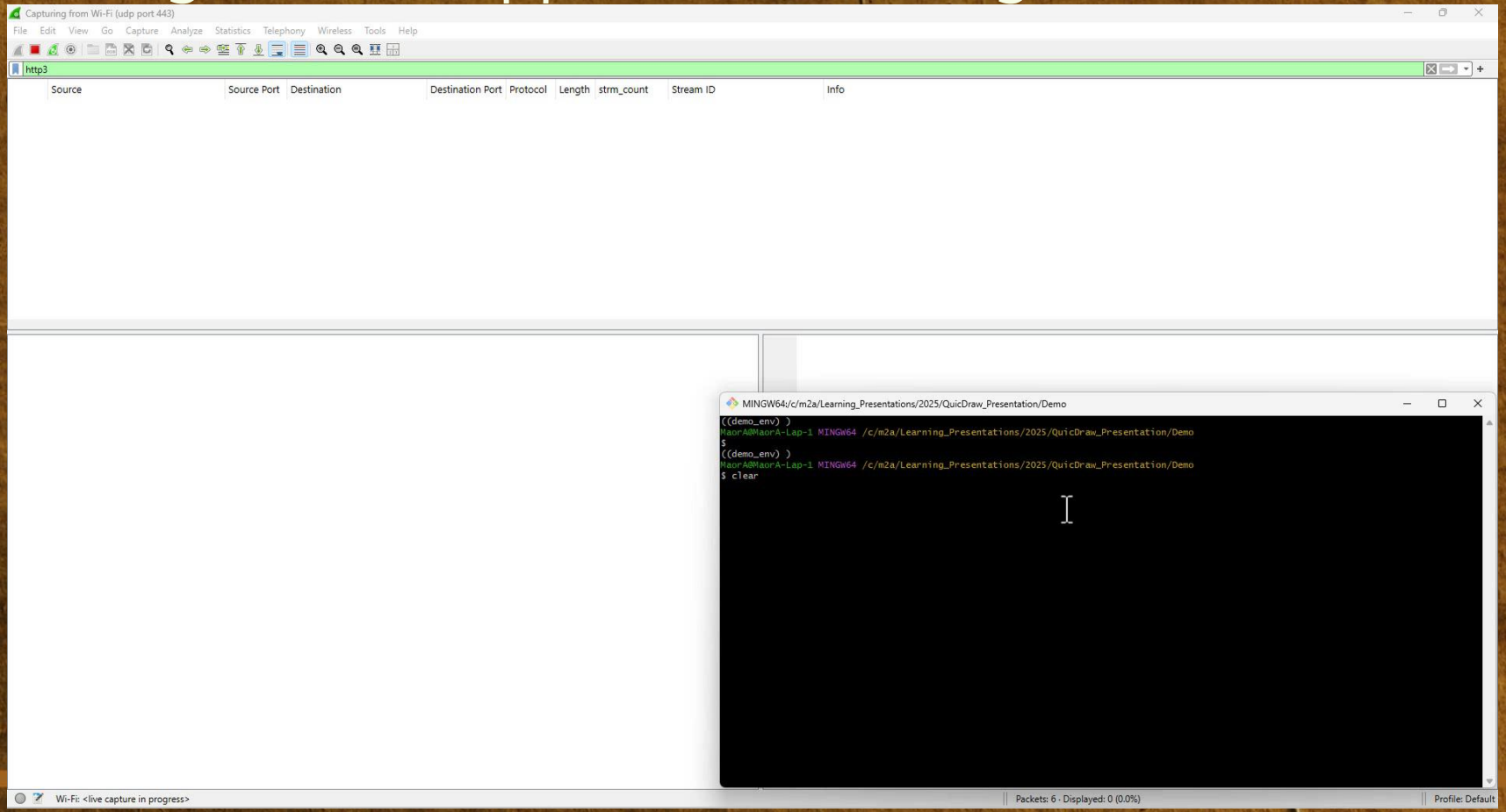
To use the same request multiple times

- (using the Quic-Fin-Sync)
- Use the `-tr/--total-requests` argument.

```
quicdraw <url> -tr TOTAL_REQUESTS
```

```
quicdraw https://google.com -tr 7
```

Racing HTTP/3 applications – usage demo



QuicDraw (HTTP3) Quic-Fin-Sync traffic

Destination Port	Protocol
50504	HTTP3
50504	HTTP3
443	HTTP3
443	HTTP3
50504	HTTP3
50504	HTTP3
50504	HTTP3

```
> User Datagram Protocol, Src Port: 50504, Dst Port: 443
v QUIC IETF
  > QUIC Connection information
    [Packet Length: 172]
  > QUIC Short Header DCID=f12f7e04d88b9f7b PKN=13
  > PING
  > STREAM id=6 fin=0 off=5 len=25 dir=Unidirectional origin=Client-initiated
  > STREAM id=0 fin=1 off=0 len=32 dir=Bidirectional origin=Client-initiated
  > STREAM id=4 fin=1 off=0 len=9 dir=Bidirectional origin=Client-initiated
  > STREAM id=8 fin=1 off=0 len=9 dir=Bidirectional origin=Client-initiated
  > STREAM id=12 fin=1 off=0 len=9 dir=Bidirectional origin=Client-initiated
  > STREAM id=16 fin=1 off=0 len=9 dir=Bidirectional origin=Client-initiated
  > STREAM id=20 fin=1 off=0 len=9 dir=Bidirectional origin=Client-initiated
  > STREAM id=24 fin=1 off=0 len=9 dir=Bidirectional origin=Client-initiated
> Hypertext Transfer Protocol Version 3
> Hypertext Transfer Protocol Version 3
> Hypertext Transfer Protocol Version 3
```

```
e8cbfa5038bc, PKN: 9,
e8cbfa5038bc, PKN: 12,
7e04d88b9f7b, PKN: 10,
7e04d88b9f7b, PKN: 13,
e8cbfa5038bc, PKN: 15,
e8cbfa5038bc, PKN: 18,
e8cbfa5038bc, PKN: 20,
```

QuicDraw –tr 7(HTTP3 Quic-Fin-Sync)

(Inspecting TLS traffic)

Fuzzing HTTP/3 applications

- Fuzzing concept, like other web fuzzers: (Ffuf, Wfuzz).
  - Go over the data section (-d) and
  - Replace any FUZZ with (line in) wordlist (-w)

```
quicdraw <url> -w WORDLIST -d DATA_with_FUZZ_keyword
```

```
quicdraw <https://http3_server.com/path> -w path/to/wordlist
-d '{"jsonkey":"FUZZ"}'
```

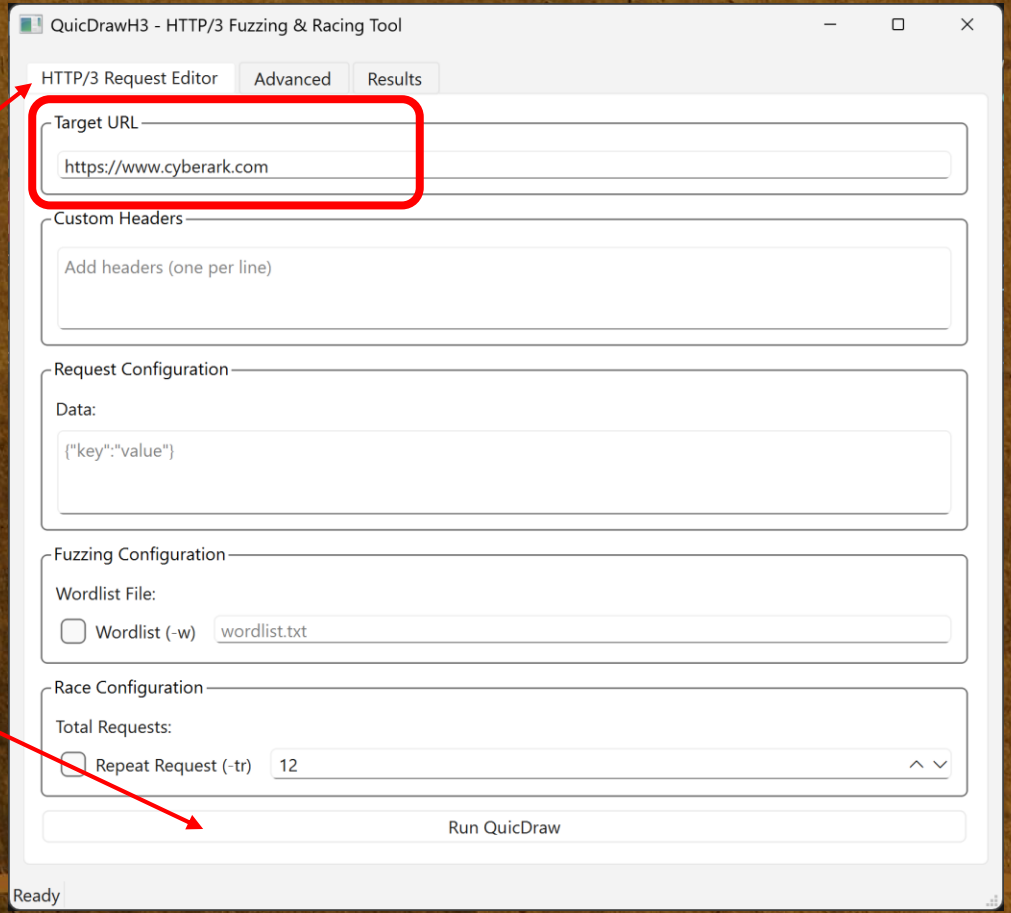
WE HAVE **FUZZING** AND **RACE-CONDITION** TESTING IN **HTTP/3**

Introducing QuicDraw-UI

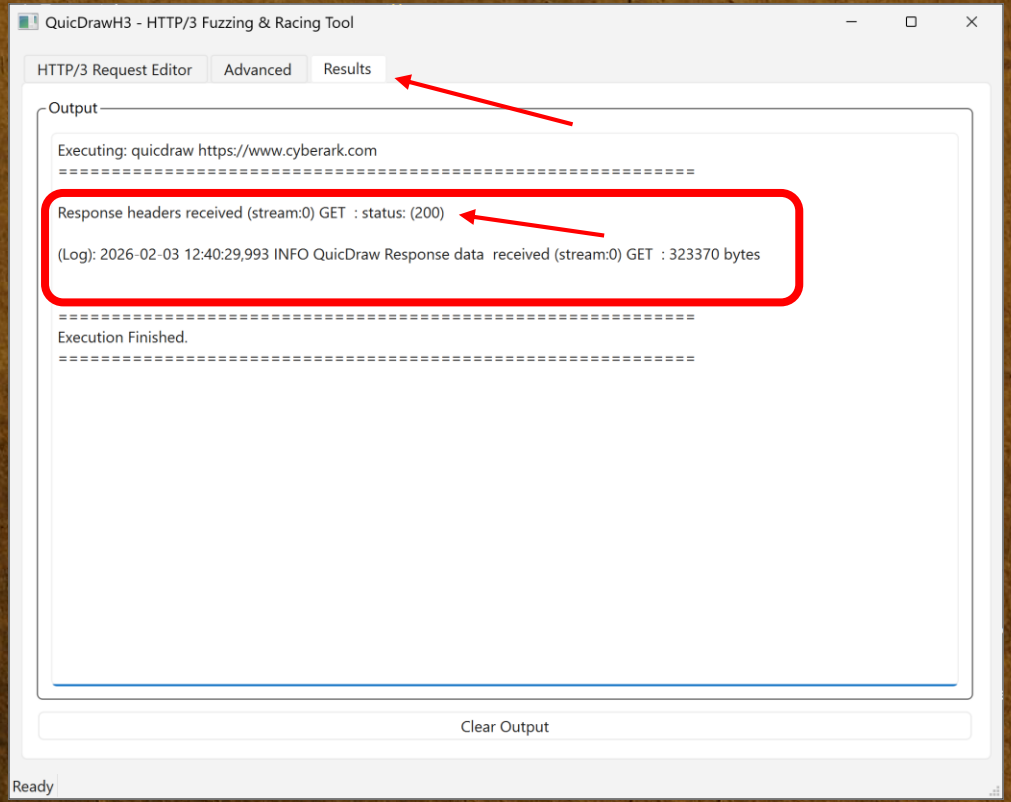
A GUI for QuicDraw(H3)

paloalto NETWORKS | CYBERARK

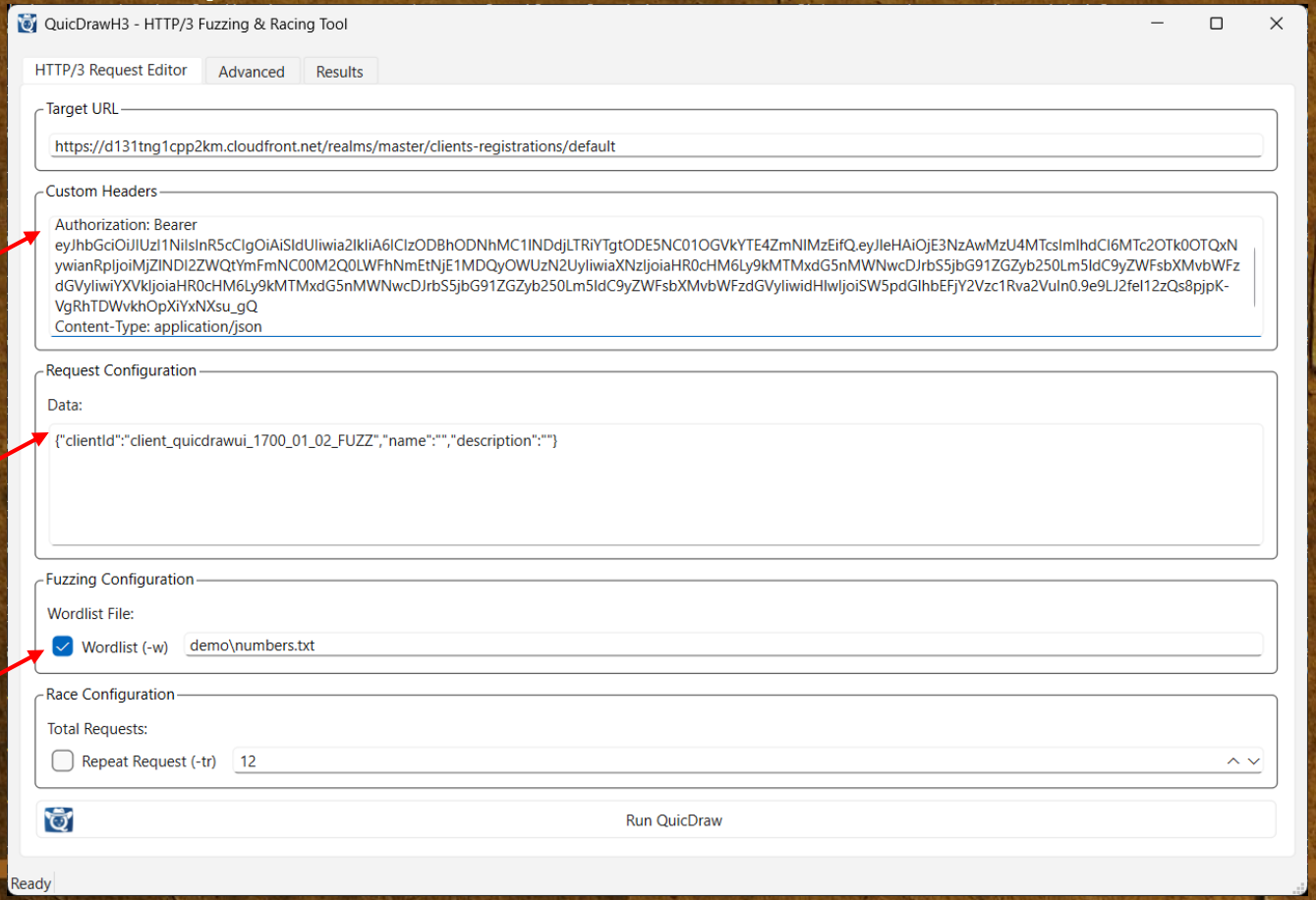
QuicDraw-UI HTTP/3 Request Editor (GET Request)



QuicDraw-UI HTTP/3 – (Execution) Results Tab

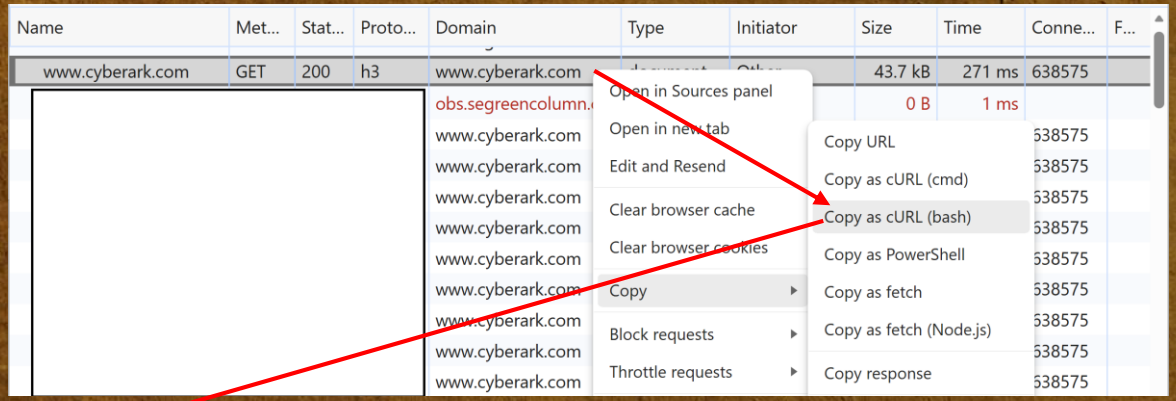


HTTP/3 Request Editor (Post + Headers)



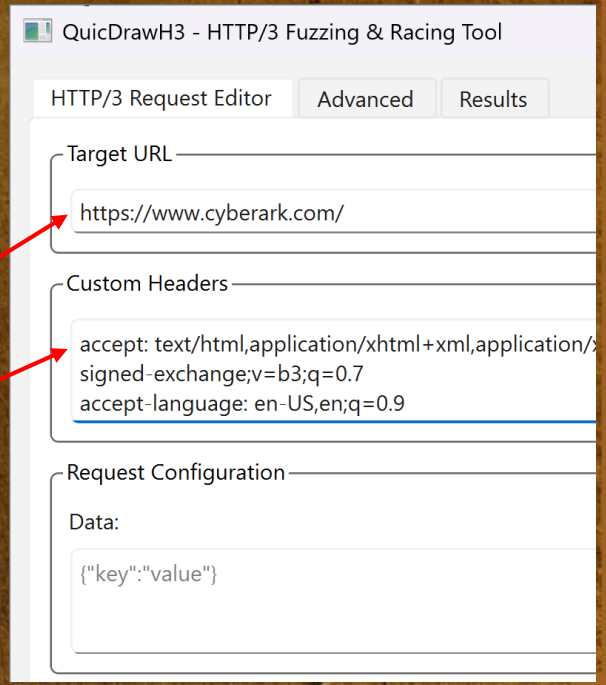
“It’s nice and all but I use other tools..”

QuicDraw-UI arguments import (curl compatibility)




```
$ curl 'https://www.cyberark.com/' \
  -H 'accept: text/html,application/xhtml+xml,app
bp,image/apng,*/*;q=0.8,application/signed-exchan
  -H 'accept-language: en-US,en;q=0.9' \
  -H 'cache-control: no-cache' \
  -H 'pragma: no-cache' \
```

```
$ quicdraw-ui 'https://www.cyberark.com/' \
  -H 'accept: text/html,application/xhtml+xml,app
bp,image/apng,*/*;q=0.8,application/signed-exchan
  -H 'accept-language: en-US,en;q=0.9' \
  -H 'cache-control: no-cache' \
  -H 'pragma: no-cache' \
```



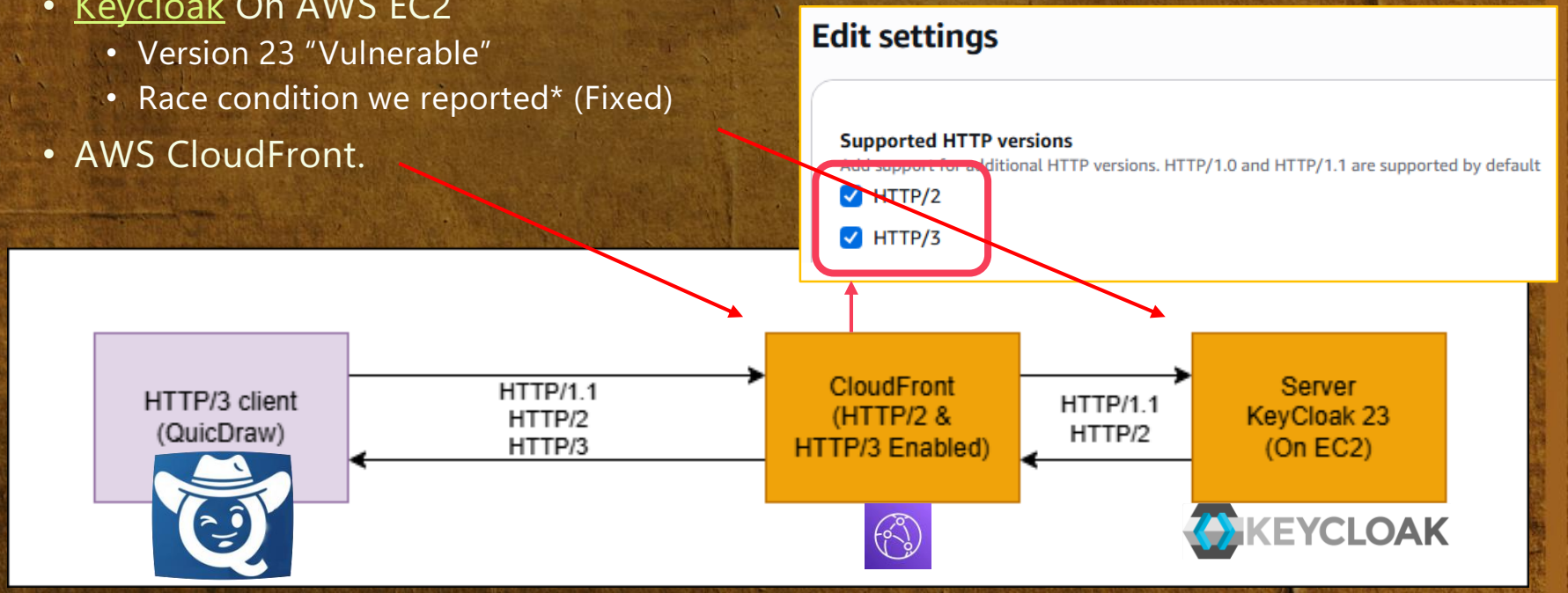
WE HAVE (REQUEST EDITOR) **FUZZING** AND **RACE-CONDITION** TESTING IN **HTTP/3**

QuicDraw Evaluation

but does it really work?

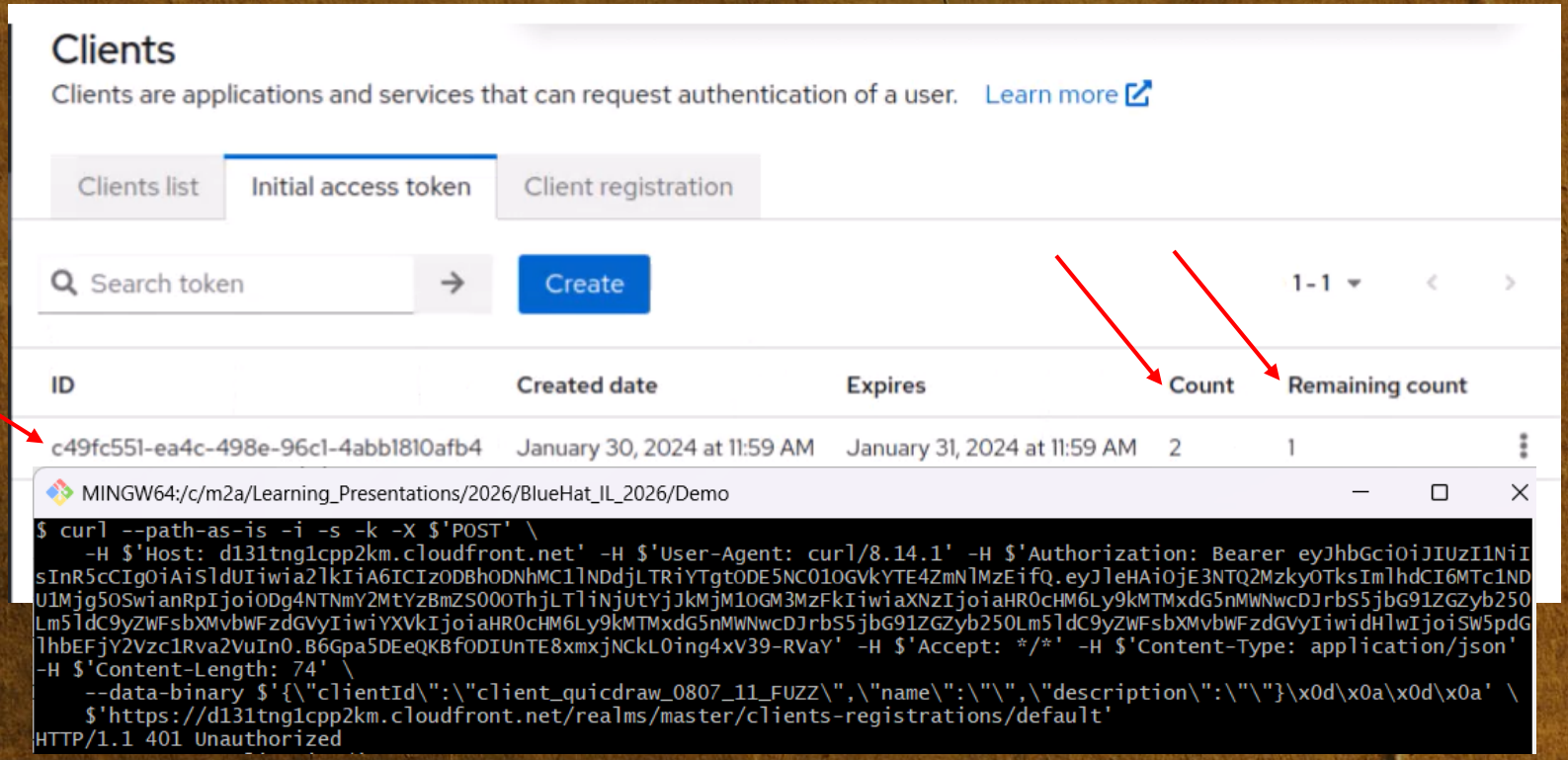
Our test setup

- Keycloak On AWS EC2
  - Version 23 “Vulnerable”
  - Race condition we reported* (Fixed)
- AWS CloudFront.

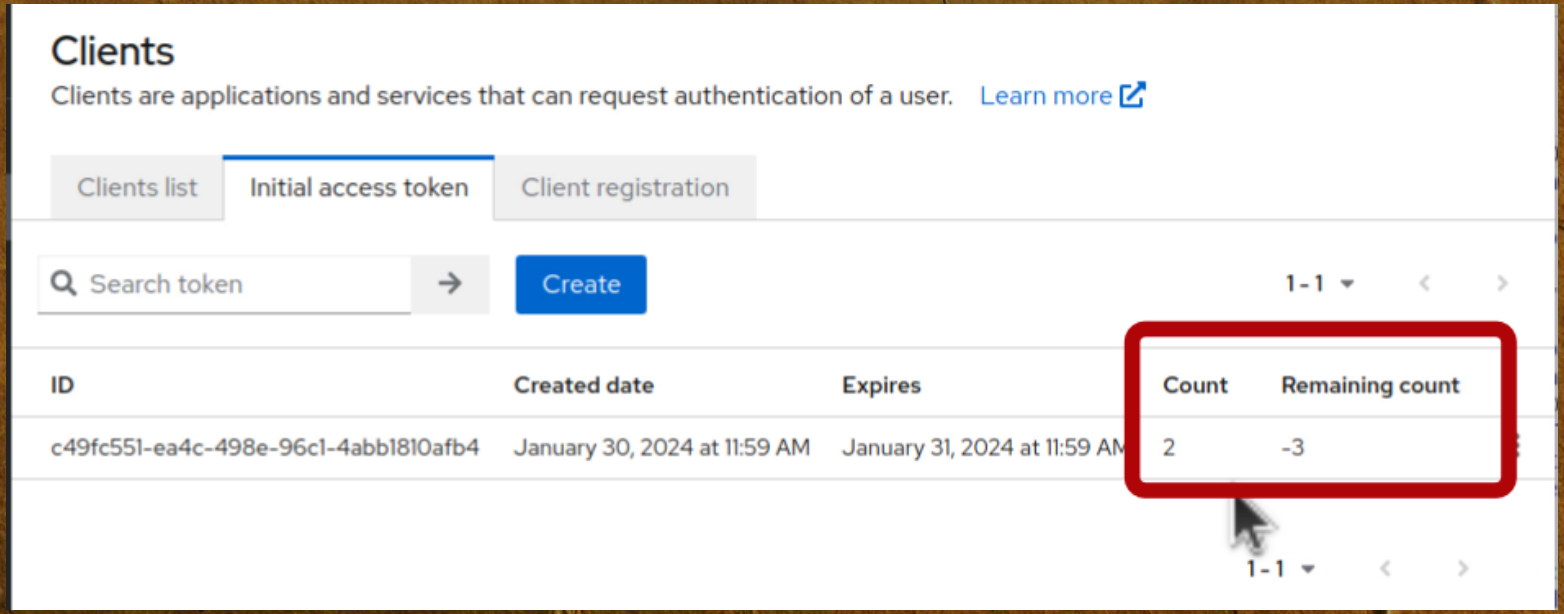


Keycloak 23 Setup on AWS

Keycloak Issue Recap (Demo Background)



Keycloak Issue Recap



IAT Token View 5 of 2

A NUMBER **BELOW ZERO** CAUSED A **RACE CONDITION** ON THE SERVER SIDE.

QuicDraw Performance Demo

PoC Or It Didn't Happen!

Demo Time! (PoC Or It Didn't Happen!)

(Using Mobile Network)

quicdraw-ui

https://d131tng1cpp2km.cloudfront.net/realms/master/clients-registrations/default

-l "demo\secrets.log"

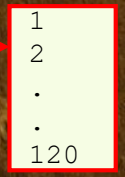
-H "Host: d131tng1cpp2km.cloudfront.net"

-H "Authorization: Bearer eyJhb...gQ"

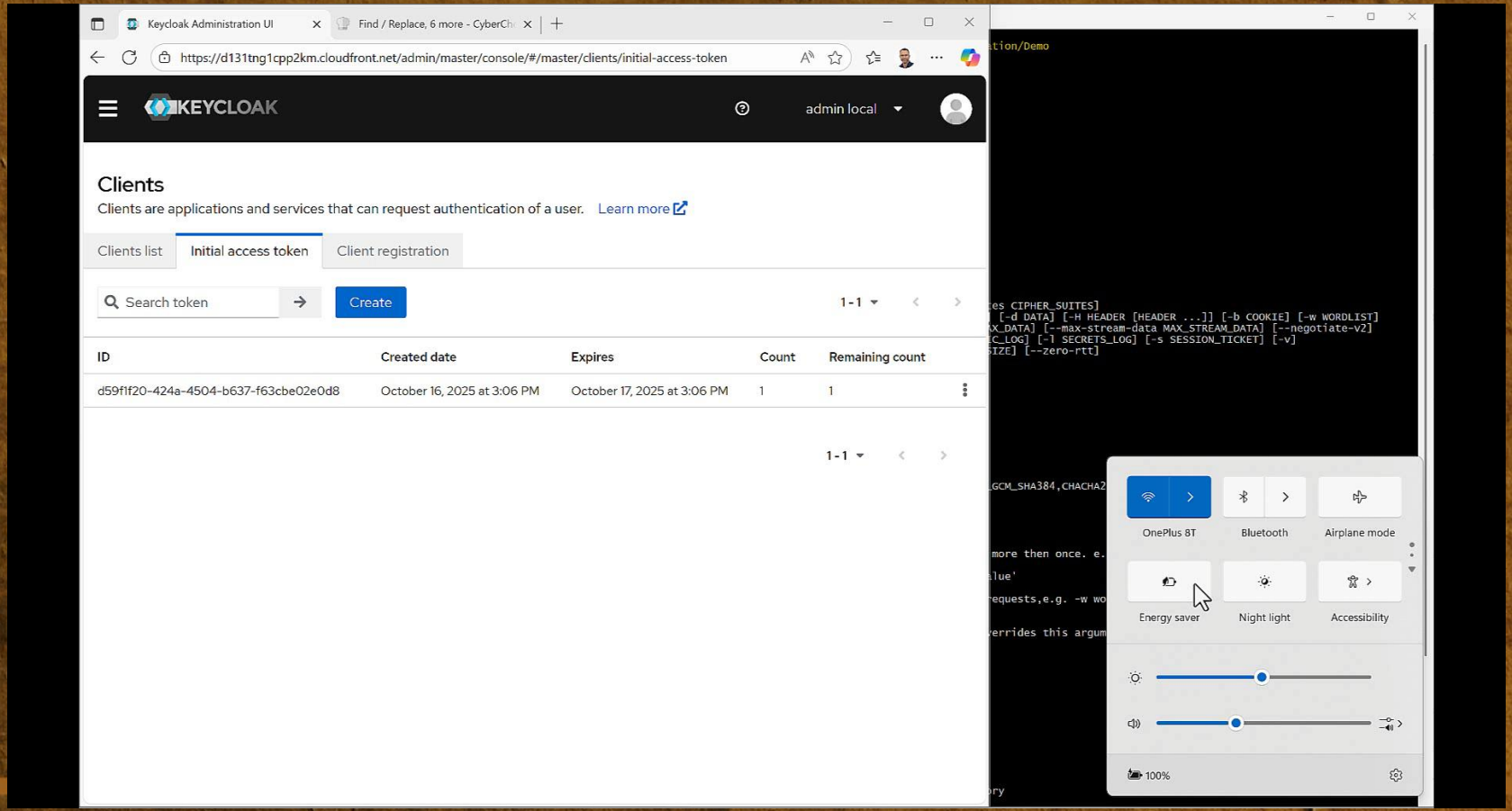
-H "Content-Type: application/json"

--data-raw $'{"clientId":"client_quicdrawui_1000_28_02_FUZZ","name":"","description":""}'

-w "demo\numbers.txt"



Demo Time! – Fuzzing using QuicDraw (Mobile Network)



Guess?

120 sent using QuicDraw (Quic-Fin-Sync)

No race (1)

2-20

20-50

50-80

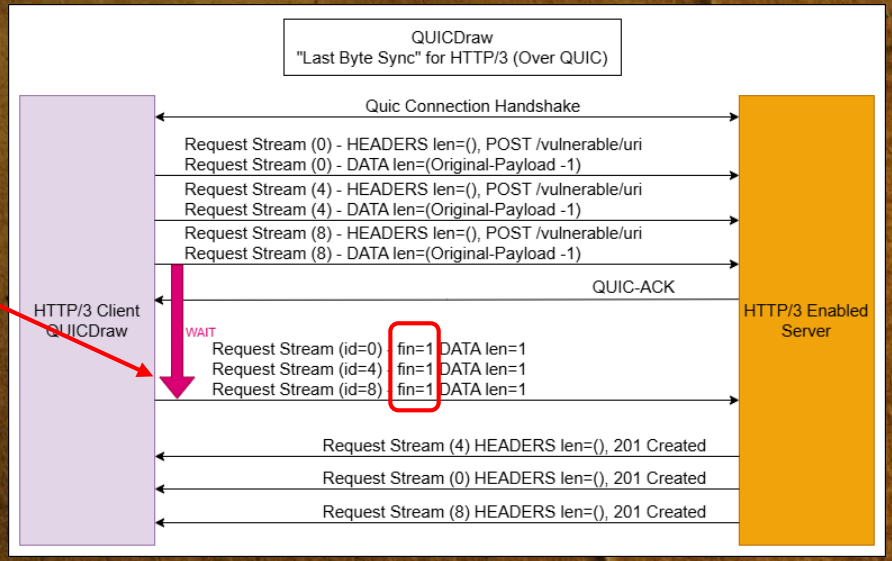
80+

Race Window

Quic-Fin-Sync - Can we do better?

```
For each Request
    queue Request-Headers
    queue Request-Data[:-1]

Transmit

wait 1000 ms

For each Request
    queue RequestData[-1:]& end_stream=1
                          (QUIC-FIN = 1)

Transmit
```

-sd <seconds>
(sync-delay)
*version 0.9.1+



QuicDraw Quic-Fin-Sync (HTTP3) traffic diagram

Summary, Takeaways & Further research

References

- https://webtechsurvey.com/technology/http-3
- https://caniuse.com/http3
- http://w3techs.com/
- RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport
- RFC 9114: HTTP/3
- The Ultimate Guide To The HTTP/3 And QUIC Protocols | DebugBear
- Why HTTP/3 is eating the world | APNIC Blog
- 6 ways HTTP/3 benefits security (and 7 serious concerns) | CSO Online
- README | HTTP/3 explained
- https://flatt.tech/research/posts/beyond-the-limit-expanding-single-packet-race-condition-with-first-sequence-sync/
- https://www.cyberark.com/resources/threat-research-blog/you-cant-always-win-racing-the-keycloak (our keycloak reaserch)
- https://aioquic.readthedocs.io/en/latest/h3.html
- https://aioquic.readthedocs.io/en/latest/asyncio.htmlHttp3Session and Streams — Firefox Source Docs documentation
- https://github.com/m2a2/awesome-http3
- HTTP pipelining – Wikipedia (image)
- https://ideogram.ai/ (cover image)

Takeaways

HTTP/3

- Wide Internet adoption
- Using QUIC
  - Connection Migration
  - Eliminates TCP HOL blocking
- Not supported by security tools / interception proxies

QuicDraw & QuicDraw-UI (Open-source tool)

- Can fuzz and race HTTP/3 web servers (using Quic-Fin-Sync)
- QD-UI HTTP/3 Request Editor
- In our test case - Exploited a race condition 80+? times!?

Multiplexed protocols

- can be susceptible to a last-byte-sync-like attacks

Further Research & Contribution

- QUIC and HTTP/3 research.
  - Many “independent” implementations
  - (https://github.com/m2a2/awesome-http3#implementations)
- Racing other multiplexed protocols
- Contribute to QuicDraw & QuicDraw-UI ☺ (We need you!)
  - (https://github.com/cyberark/QuicDrawH3)
  - Extend for other attacks over HTTP/3

Questions?

ONE MORE SLIDE…

paloalto NETWORKS | CYBERARK

Read more at our blog posts & Thank you

"Fuzzing and Racing HTTP/3: Open-Sourcing QuicDraw(H3)"

https://www.cyberark.com/resources/threat-research-blog/racing-and-fuzzing-http-3-open-sourcing-quicdraw

Contact (LinkedIn):

https://il.linkedin.com/in/maor-abutbul

QuicDraw & QuicDraw-UI Repo:

https://github.com/cyberark/QuicDrawH3