
QUIC and
Furious

$ whoami

Bojan Ždrnja (@bojanz on Twitter)

CTO and penetration testing team lead at INFIGO IS

- https://www.infigo.is

SANS Certified Instructor

- Co-author SEC542 – Web application penetration testing and ethical hacking

Addicted to GIAC certificates

- GSE (GIAC Security Expert) #346
- GCIA, GCIH, GWAPT, GMOB, GXPN, GMON, GREM, GCFA, GCFE, GCPN, GCTI, GCSA

SANS Internet Storm Center Senior handler - https://isc.sans.edu

Name 10 Vin Diesel movies

1) The Fast and the Furious (2001)
2) The Fast and the Furious: Tokyo Drift (2006)
3) Fast and Furious (2009)
4) Fast Five (2011)
5) Fast & Furious 6 (2013)
6) Furious 7 (2015)
7) The Fate of the Furious (2017)
8) F9 (2021)
9) Fast X (2023)
10) Pitch Black

… see the pattern?

Name 10 QUIC app protocols

1) HTTP/3
2) DNS over QUIC (DoQ)
3) ???
4) Your
5) Favorite
6) Protocol
7) Is
8) ???
9) Almost
10) Anything

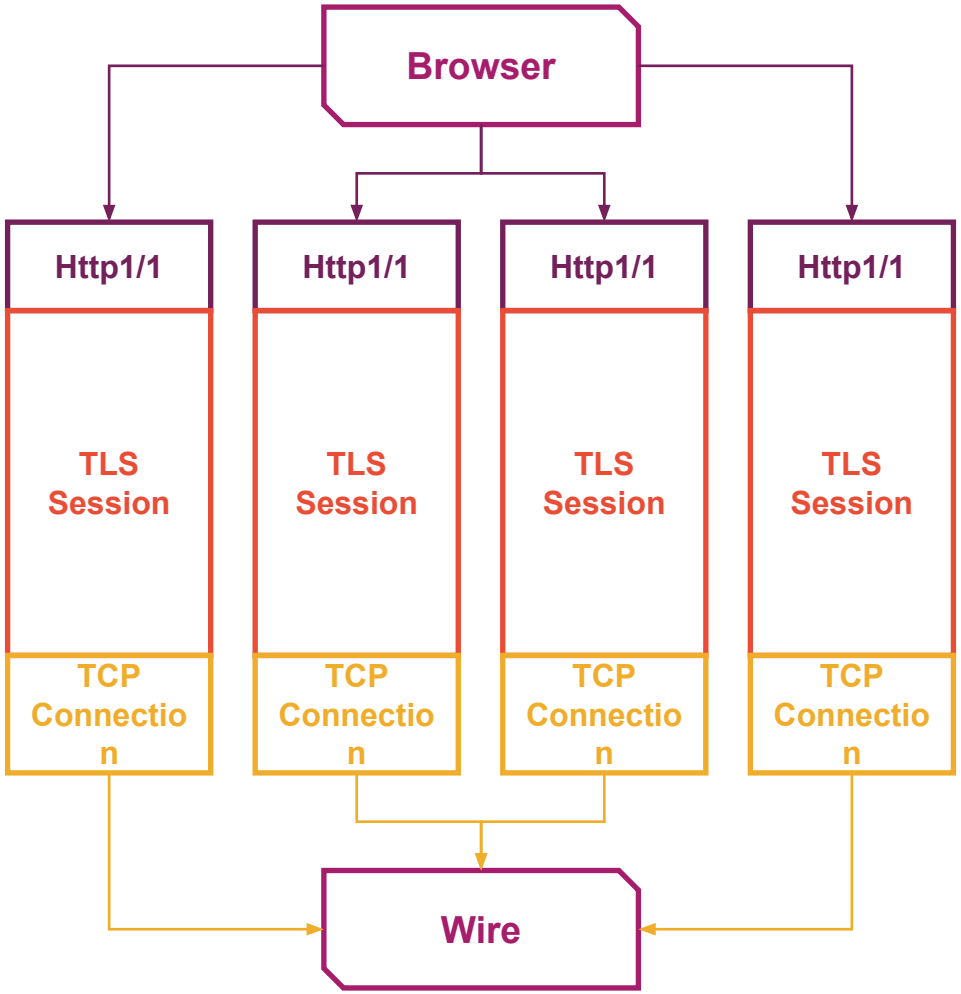
… see the pattern?

A little bit of history

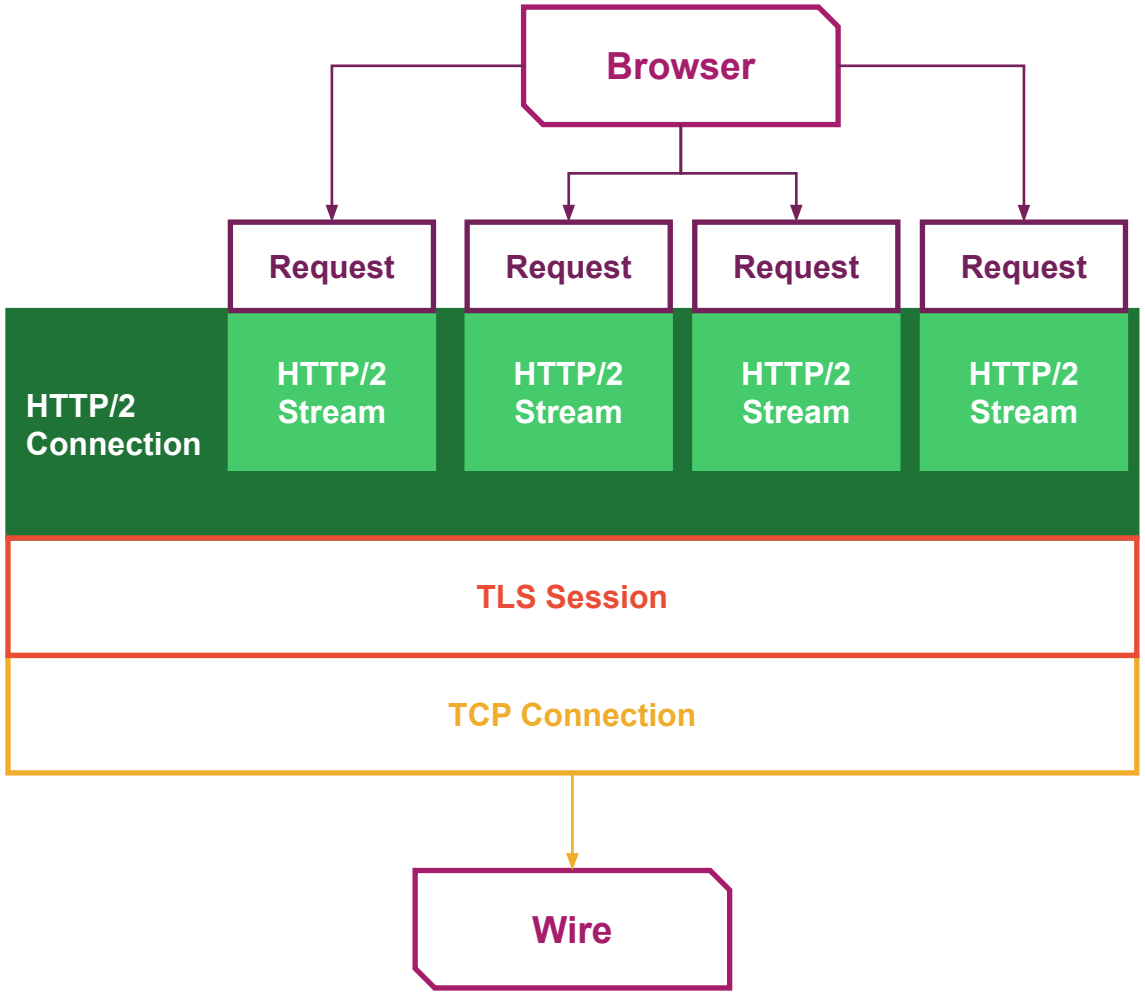
01 HTTP/1.1 vs HTTP/2

- HTTP/2 specification was published in 2015
  - Few years after, almost 50% of top 1000 web sites were running HTTP/2
- HTTP/2 brought a lot of changes!
  - It is a binary protocol, uses so–called binary framing
    - Better compression abilities
    - Parsing is handled in a more objective fashion
  - Multiplexed, not pipelined
    - With HTTP/1.1 each request requires its own TCP connection, or uses pipelining
      - Susceptible to Head-of-line blocking (HOL)
    - Streams are bi-directional sequences of frames exchanged in a single TCP connection
      - HTTP responses are split into frames, which can be simultaneously sent and prioritized

HTTP/1.1 vs HTTP/2



Classic HTTP

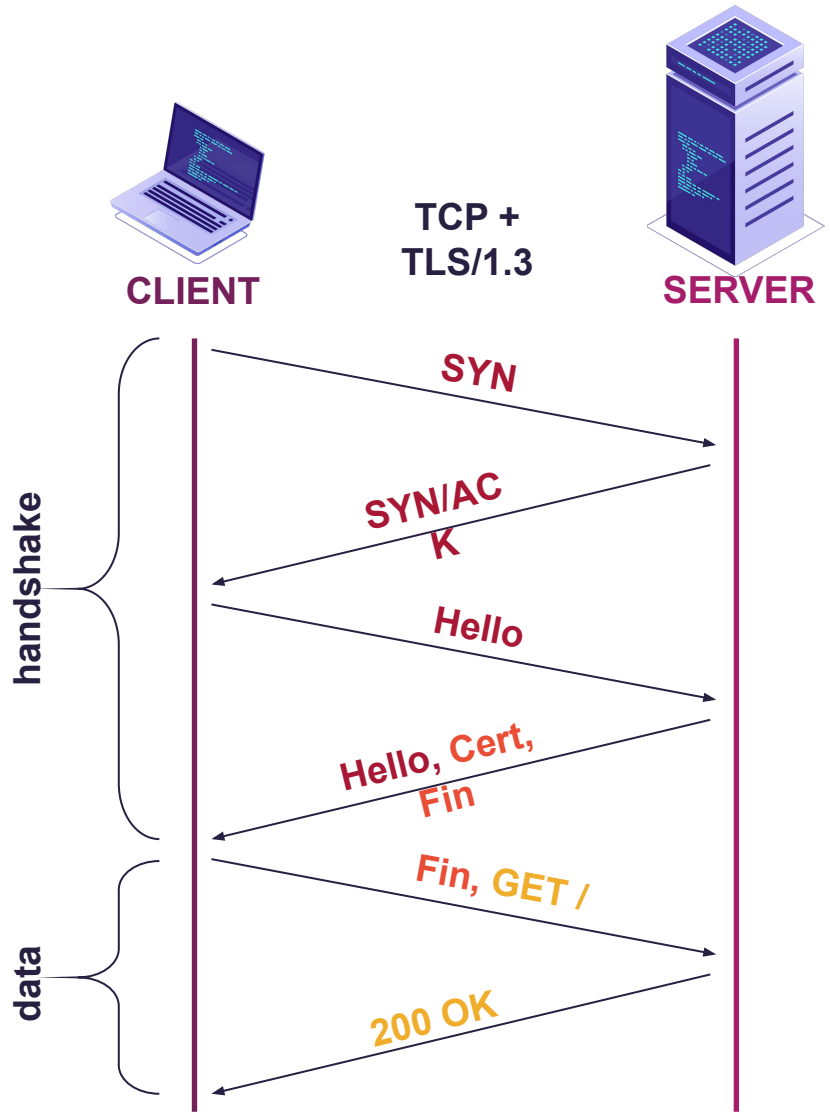


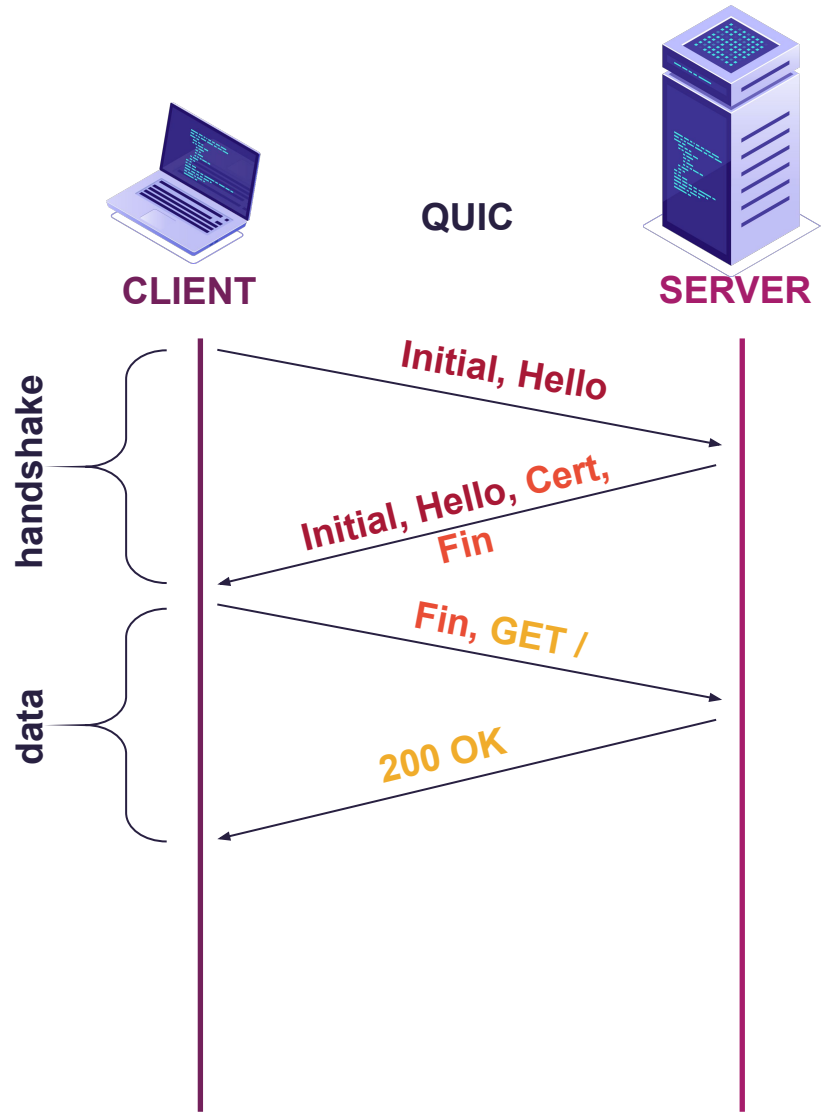
H2

03 HTTP/1.1 vs HTTP/2 vs HTTP/3

- With HTTP/1.1 browsers usually open 6 connections, with HTTP/2 one!
- One TCP connection is great for servers
  - … but not so great for users, if there is a lost packet
    - We solved HTTP head of line block, but now we got TCP head of line block
- QUIC introduced UDP as transport protocol
  - No more TCP head of line blocking
  - QUIC is secure – always encrypted
  - Published in 2021 as RFC 9000
  - HTTP-over-QUIC (HTTP/3) builds upon HTTP/2
    - Moves some of the specifics from the HTTP layer as they are covered by QUIC

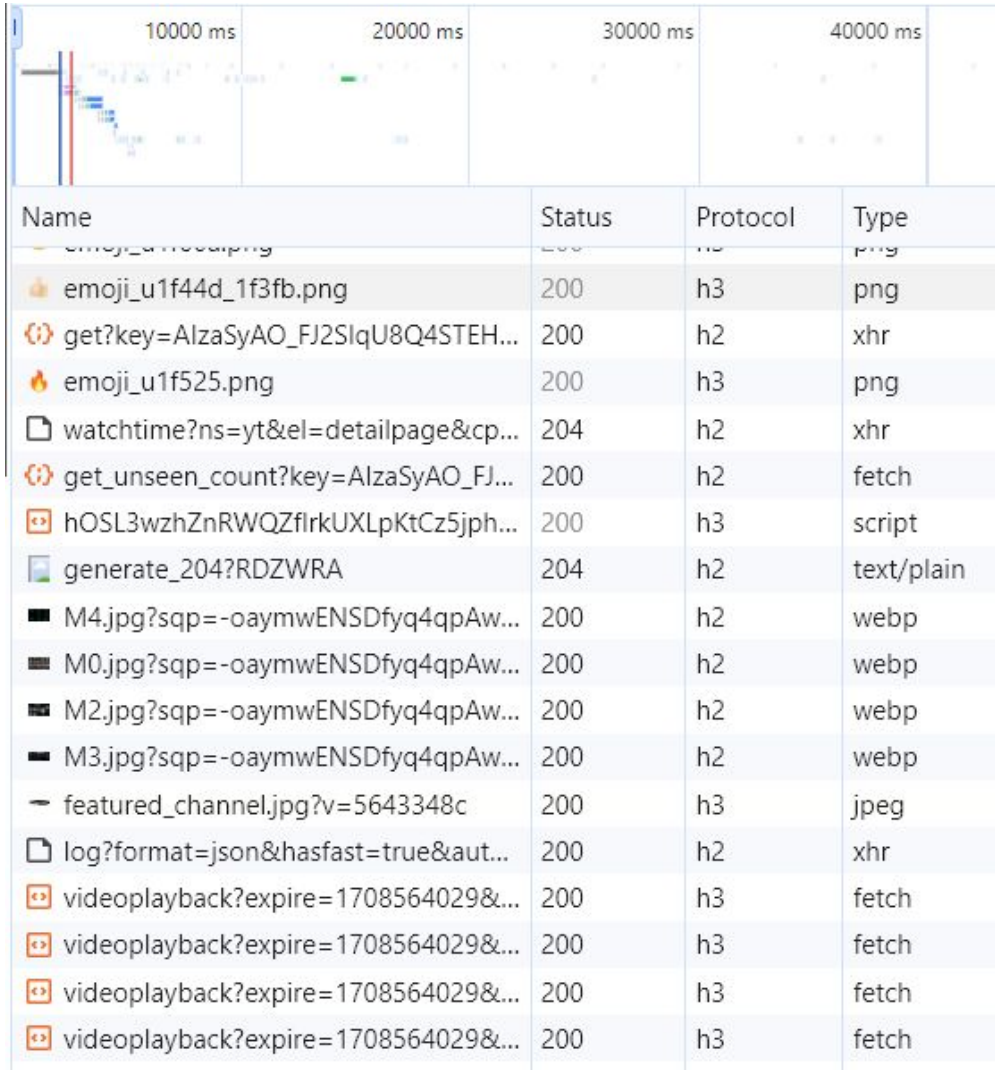
HTTP/2 vs HTTP/3





05 QUIC and HTTP/3 are everywhere

- Google services
  - Search, Youtube
- Facebook
- Instagram
- Uber
- Browsers
  - Chrome, from 2012!
  - Firefox, 2021
  - Safari



Name	Status	Protocol	Type
emoji_u1f44d_1f3fb.png	200	h3	png
get?key=AIzaSyAO_FJ2SlqU8Q4STEH...	200	h2	xhr
emoji_u1f525.png	200	h3	png
watchtime?ns=yt&el=detailpage&cp...	204	h2	xhr
get_unseen_count?key=AIzaSyAO_FJ...	200	h2	fetch
hOSL3wzhZnRWQZflrkUXLpKtCz5jph...	200	h3	script
generate_204?RDZWRA	204	h2	text/plain
M4.jpg?sqp=-oaymwENSDfyq4qpAw...	200	h2	webp
M0.jpg?sqp=-oaymwENSDfyq4qpAw...	200	h2	webp
M2.jpg?sqp=-oaymwENSDfyq4qpAw...	200	h2	webp
M3.jpg?sqp=-oaymwENSDfyq4qpAw...	200	h2	webp
featured_channel.jpg?v=5643348c	200	h3	jpeg
log?format=json&hasfast=true&aut...	200	h2	xhr
videoplayback?expire=1708564029&...	200	h3	fetch
videoplayback?expire=1708564029&...	200	h3	fetch
videoplayback?expire=1708564029&...	200	h3	fetch
videoplayback?expire=1708564029&...	200	h3	fetch

06 Protocol advertising

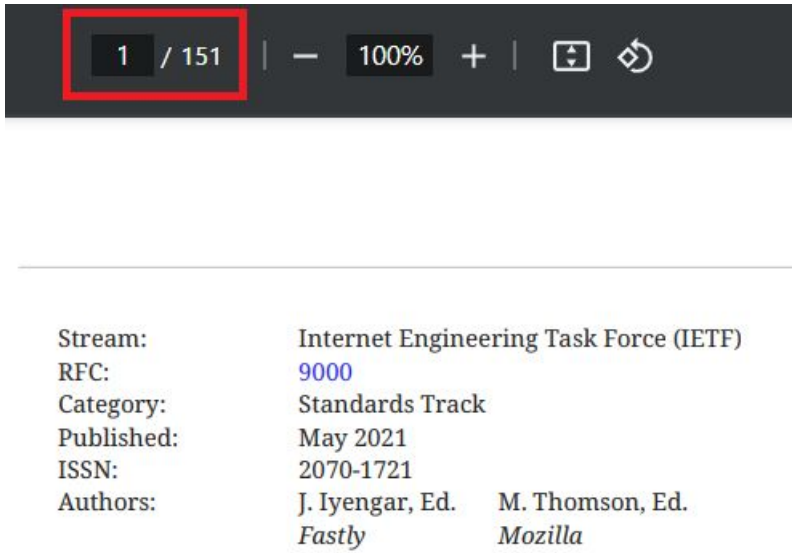
- Today browsers will still try HTTP/1.1 or HTTP/2 as default protocols
- A server will advertise its support for QUIC with a new response HTTP header:
  - Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
    - Google supports h3 and h3 draft 29 on port 443
    - Can be cached for 2592000 seconds (30 days)
- If we do not want QUIC to be used, we can delete the response header
  - This way the client will think the server does not support QUIC
  - The server will think the client does not support QUC

A QUIC deep dive

Parental Advisory: Rated X: Some CRYPTO is about to be shown

01 How QUIC actually works?

- QUIC is specified in RFC 9000: "*QUIC: A UDP-Based Multiplexed and Secure Transport*"
- It is not an easy read, or a simple protocol
- On the contrary – while reading all this severa times I thought certain things are insane
- Let's see why



02 Initialization

- QUIC makes security and privacy a first-class citizen
  - This means that authors try to encrypt as much as possible
  - While certain things can be decrypted, as we will show in a minute, this will inevitably cause problems for various IDS/IPS devices
    - Attackers wave to Darktrace, Vectra and ExtraHop
- QUIC relies on TLSv1.3
  - This helps with a shorter handshake (one of QUIC goals)
  - However certain metadata is still visible
    - QUIC tries (not successfully though!) to hide that

03 Initialization

- In TLSv1.3 we almost exclusively use elliptic curve (EC) cryptography for key exchange
  - Means – forget about passive analysis (another wave to previous slide devices)
- These are sent in TLS extension supported groups
  - x25519, secp256r1, secp384r1
- The first step for a client is to create a private/public keypair, usually with X25519, which does point operations on the Curve25519 elliptic curve

04 Initial keys calculation (QUIC v1)

- Now comes the fun, the client generates certain random data
- This will be used for various initial keys generation
- Initial salt is ALWAYS:
  38762cf7f55934b34d179ae6a4c80cadccbb7f0a
  - First SHA-1 collision found by Google researchers
- QUIC uses HKDF – HMAC Key Derivation Function
- HKDF consists of two important functions:
  - HKDF-Extract
    - Takes random data and salt, and generates a key used by HKDF-Expand
  - HKDF-Expand
    - Takes the key, some “info” and length, and produces out of desired length

05 Initial keys calculation

- So, a QUIC client basically calculates the following:
  - initial_salt = 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
  - initial_random = (random bytes)
  - initial_secret = HKDF-Extract(salt: initial_salt, key: initial_random)
  - client_secret = HKDF-Expand-Label(key: initial_secret, label: "client in", ctx: "", len: 32)
  - server_secret = HKDF-Expand-Label(key: initial_secret, label: "server in", ctx: "", len: 32)
  - client_key = HKDF-Expand-Label(key: client_secret, label: "quic key", ctx: "", len: 16)
  - server_key = HKDF-Expand-Label(key: server_secret, label: "quic key", ctx: "", len: 16)
  - client_iv = HKDF-Expand-Label(key: client_secret, label: "quic iv", ctx: "", len: 12)
  - server_iv = HKDF-Expand-Label(key: server_secret, label: "quic iv", ctx: "", len: 12)
  - client_hp_key = HKDF-Expand-Label(key: client_secret, label: "quic hp", ctx: "", len: 16)
  - server_hp_key = HKDF-Expand-Label(key: server_secret, label: "quic hp", ctx: "", len: 16)

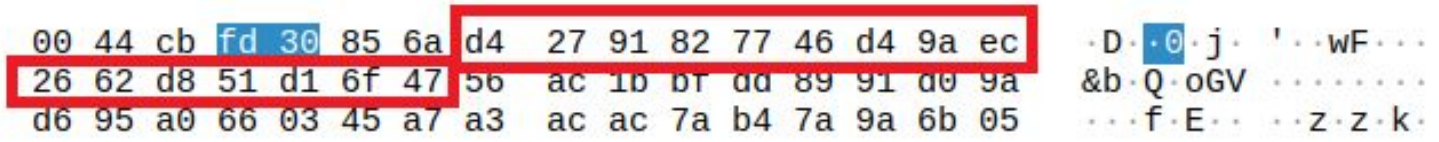
06 First packet

- These initial keys are used to encrypt the first packet!

```
▸ Frame 1: 1294 bytes on wire (10352 bits), 1294 bytes captured (10352 bits)
▸ Ethernet II, Src: VMware_73:f4:c7 (00:0c:29:73:f4:c7), Dst: VMware_72:b7:b0 (00:0c:29:72:b7:b0)
▸ Internet Protocol Version 4, Src: 192.168.100.130, Dst: 192.168.100.138
▸ User Datagram Protocol, Src Port: 57359, Dst Port: 443
▾ QUIC IETF
  ▸ QUIC Connection information
    [Packet Length: 1252]
    1... .... = Header Form: Long Header (1)
    .1.. .... = Fixed Bit: True
    ..00 .... = Packet Type: Initial (0)
    .... 00.. = Reserved: 0
    .... ..01 = Packet Number Length: 2 bytes (1)
    Version: 1 (0x00000001)
    Destination Connection ID Length: 15
    Destination Connection ID: 43e68568d3fe7fa83a8c68aeab2ab0
    Source Connection ID Length: 0
    Token Length: 0
    Length: 1227
    Packet Number: 0
    Payload: 856ad42791827746d49aec2662d851d16f4756ac1bbfdd8991d09ad695a0660345a7a3ac…
```

07 Header protection

- This gets even more insane
- The first byte's nibble and the Packet Number get "encrypted" by XOR-ing them with product of the following:
  - Take 16 bytes of the payload, but 4 bytes past the first byte of the packet number



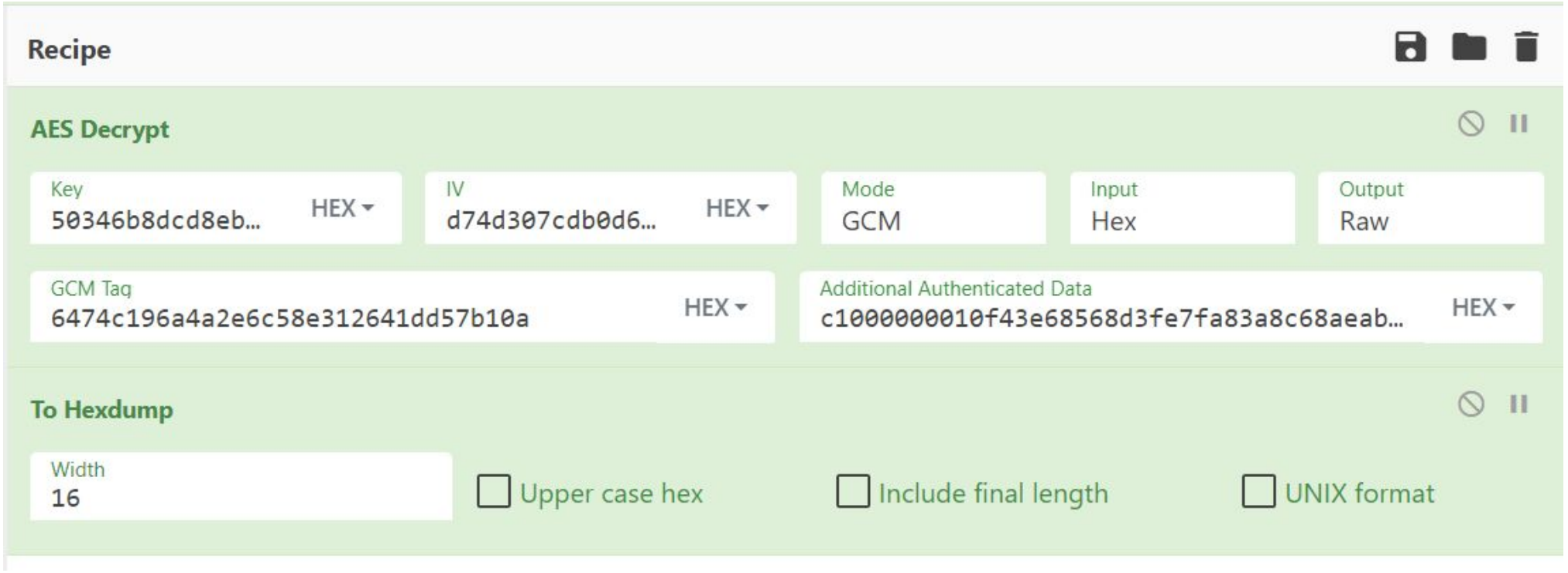
  - Now encrypt those 16 bytes with client_hp_key by using AES-128-ECB and use first 5 bytes of result for XOR-ing the first byte's nibble and the Packet Number
- We can calculate this, of course, and restore proper first byte and Packet Number values

08 Decrypting the QUIC header

- Now we can decrypt the QUIC protected header, as we have all ingredients:
  - client_key
  - client_iv
  - Record number (0 in this case)
  - GCM authentication tag
    - Last 16 bytes in the packet
  - Additional authenticated data
    - This is the whole QUIC unprotected header, but with decrypted (XOR-ed) first nibble and packet number bytes
      - In other words, we need the original values

09 Decrypted QUIC header

- Here's the CyberChef recipe



10 QUIC CRYPTO frame

```
▸ PADDING Length: 938
▾ CRYPTO
    Frame Type: CRYPTO (0x0000000000000006)
    Offset: 0
    Length: 267
    Crypto Data
  ▾ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 263
        Version: TLS 1.2 (0x0303)
        Random: dcb842b1b3c87e2171c9362e7bbf5a4623ede11446ca03433bf36df3672af4fc
        Session ID Length: 0
        Cipher Suites Length: 6
      ▸ Cipher Suites (3 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 216
      ▸ Extension: server_name (len=20)
      ▸ Extension: status_request (len=5)
      ▸ Extension: supported_groups (len=10)
      ▸ Extension: ec_point_formats (len=2)
      ▸ Extension: signature_algorithms (len=26)
      ▸ Extension: renegotiation_info (len=1)
      ▸ Extension: extended_master_secret (len=0)
      ▸ Extension: application_layer_protocol_negotiation (len=5)
      ▸ Extension: signed_certificate_timestamp (len=0)
      ▸ Extension: supported_versions (len=3)
      ▸ Extension: key_share (len=38)
      ▸ Extension: quic_transport_parameters (len=58)
```

11 Important parameters

```
▸ Extension: ec_point_formats (len=2)
▸ Extension: signature_algorithms (len=26)
▸ Extension: renegotiation_info (len=1)
▸ Extension: extended_master_secret (len=0)
▾ Extension: application_layer_protocol_negotiation (len=5)
    Type: application_layer_protocol_negotiation (16)
    Length: 5
    ALPN Extension Length: 3
  ▾ ALPN Protocol
      ALPN string length: 2
      ALPN Next Protocol: h3
▸ Extension: signed_certificate_timestamp (len=0)
▸ Extension: supported_versions (len=3)
▸ Extension: key_share (len=38)
▾ Extension: quic_transport_parameters (len=58)
    Type: quic_transport_parameters (57)
    Length: 58
  ▸ Parameter: GREASE (len=0)
  ▸ Parameter: initial_max_stream_data_bidi_local (len=4) 524288
  ▸ Parameter: initial_max_stream_data_bidi_remote (len=4) 524288
  ▸ Parameter: initial_max_stream_data_uni (len=4) 524288
  ▸ Parameter: initial_max_data (len=4) 786432
  ▸ Parameter: initial_max_streams_bidi (len=1) 10
  ▸ Parameter: initial_max_streams_uni (len=2) 100
  ▸ Parameter: max_idle_timeout (len=4) 30000 ms
  ▸ Parameter: max_udp_payload_size (len=2) 1452
  ▸ Parameter: GREASE (len=1) 26
  ▸ Parameter: disable_active_migration (len=0)
  ▸ Parameter: active_connection_id_limit (len=1) 4
  ▸ Parameter: initial_source_connection_id (len=0)
  ▸ Parameter: max_datagram_frame_size (len=2) 16383
```

12 QUIC connections

- QUIC connection is a single conversation between two endpoints
  - Once a connection has been created, streams are used to send/receive data
- Each connection has its unique Connection ID
  - Ensures that packets are delivered to the correct endpoint
    - Allows us to migrate connections over IP addresses!
- Streams provide ordered byte-stream abstraction
  - Can be unidirectional and bidirectional
  - Can be interleaved with other streams
- 0-RTT can be achieved by a client that connected previously
  - They can cache certain data to achieve 0-RTT

13 And to spice things up

- Anyone can change/introduce new QUIC protocols
  - Change congestion, flow control, streams, 0-RTT, packet size
- Google has their own (3 are IANA registered)
  - 0x5130303[1-9] (Q001 – Q009) … Q059
- Facebook decided to have their own
  - Called MVFST (https://github.com/facebook/mvfst) - 0xfaceb00[0-f]
- Mozilla too: 0xf123f0c[0-f] (MozQuic)
- Microsoft said – hey us too: 0xabcd000[0-f] (MsQuic)
- Tencent too ☺: 0x0700700[0-f] (TencentQuic)

Applications that live on top of QUIC

01 HTTP/3

- The obvious candidate is HTTP/3
- Streams are provided by QUIC (compared to HTTP/2 which provides streams itself)
- Bootstrap is with Alt-Svc as already mentioned
- As with HTTP/2 we have server push
  - Mechanism that allows a server to send data that the client never asked for!
  - PUSH_PROMISE frame
- Is always encrypted
  - HTTP/2 can be in “plain text”, although not common

02 DNS over QUIC (DoQ)

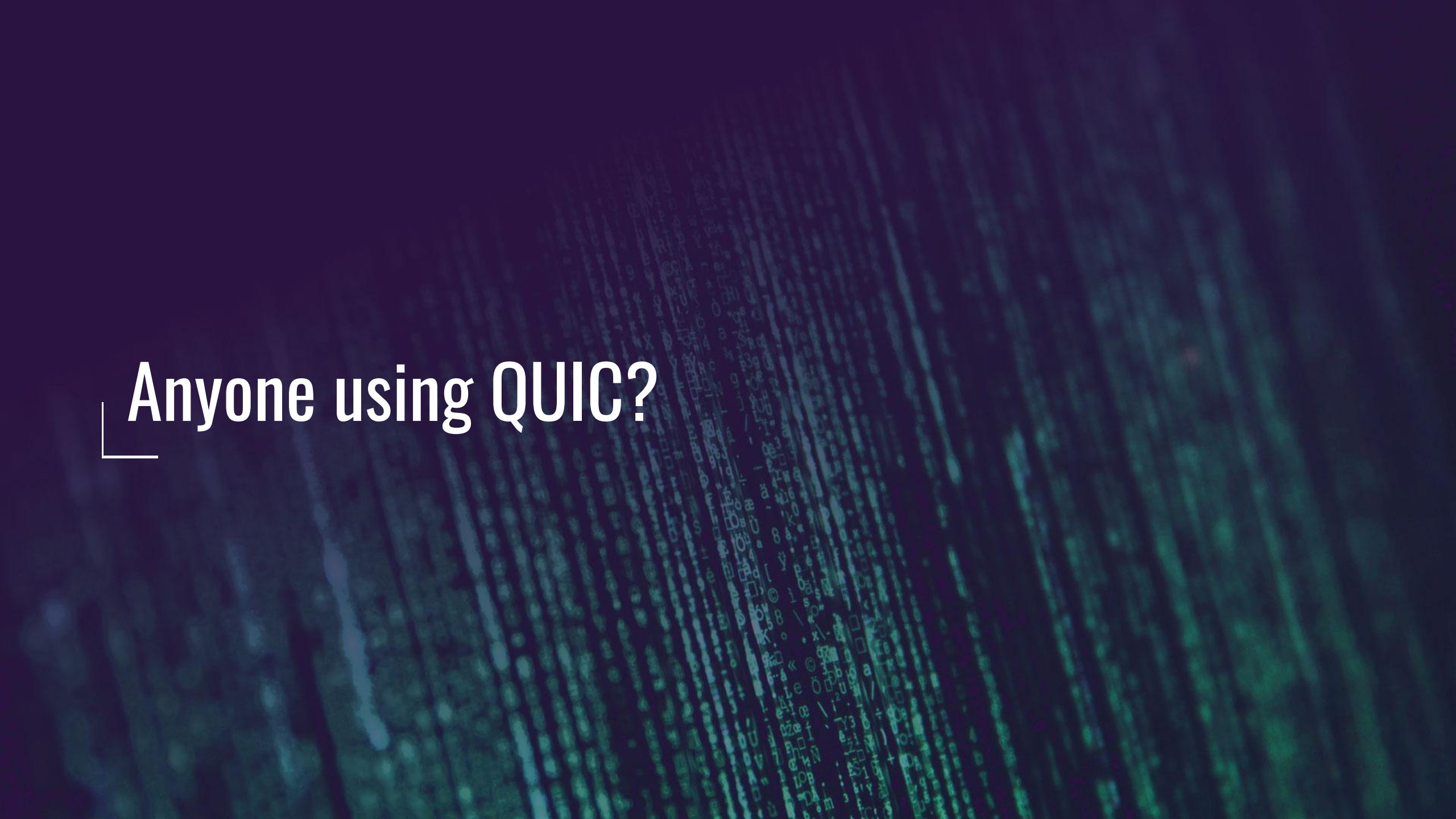
- Became a standard in 2022
- Encrypts everything, again, so your ISP cannot see what you resolve
- As before solves the head of line blocking issue
  - Usually visible for those using DNS-over-TLS
- Fast so we get to resolve hostnames even faster
- Uses UDP port 853
  - Same port as DNS-over-TLS
- Do not mix it with DNS over HTTP/3
- Still not widely supported

03 Samba over QUIC

- ~~Simon~~ Microsoft says: we'll push everything over QUIC and we'll start with Samba
- Initially available only Windows Server 2022 Datacenter Azure Edition
  - Now in Windows Server 2025 on premise, client in Windows 11
- Samba over QUIC – UDP port 443
- Requires properly setup certificates
- Still uses TCP by default
  - QUIC tried if TCP fails, or if manually set
- Microsoft calls this "SMB VPN". Any issues here?

04 SSH over QUIC (SSH3)

- And for the final abomination: SSH3
  - Well, it's perhaps not that bad
- Released at https://github.com/francoismichel/ssh3
  - While not production ready, works surprisingly well
- Complete revisit of the SSH protocol
  - Semantics of the protocol are mapped on top of HTTP3
  - QUIC+TLSv1.3 used for server authentication
  - HTTP Authorization used for user authentication
- Cool feature: it can be made (almost) invisible


Anyone using QUIC?

01 Scanning for QUIC services

- Since QUIC uses UDP it is not trivial to scan for QUIC services
- nmap, my favorite tool actually fails
  - It's quite bad in fingerprinting UDP services

```
bojanz@mnemosyne:~$ sudo /usr/local/bin/nmap -sV -sU 142.250.74.206 -p 443 --version-all
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-21 21:09 CET
Nmap scan report for fra24s02-in-f14.1e100.net (142.250.74.206)
Host is up (0.00089s latency).

PORT    STATE SERVICE VERSION
443/udp open  https?
1 service unrecognized despite returning data. If you know the service/version, please submit
SF-Port443-UDP:V=7.94SVN%I=9%D=2/21%Time=65D658A4%P=x86_64-unknown-linux-g
SF:nu%r(RPCCheck,27,"d6\x82\xe1\xa5\x04\xc1tD\xa4\(@\xe7N\xd0\x1av\xf3\x01
SF:\x1f\xea}\x9f\x9f\xc79\xe2\xa4\xfe\xfc\xb6\xd3\x98e\xdf\x80@\xeb\x97")%
SF:r(SIPOptions,2A,"q\x90\xa1\xc9\xcb\xf8\x81\x14--\xd2\[\x8e\x20\xc4\xf1e
SF:\xfa\xec{\xa5z\xd5\x87\xb1q\x03\xb7\x20\x17\xfdc\x93k\xff\x9ae\xb3\x20w
SF:\xf9\xd1")%r(Citrix,3C,"\x0e\0\x010\x02\xfd\xa8\xe3\0PRST\x03\0\0\0EPID
SF:\x03\0\0\0RNON\x0b\0\0\0CADR\x13\0\0\0GFE\xb5i\x0f\0\0\0\0\0\x02\0m<\x0
SF:bOI\xd6")%r(Kerberos,2A,"\[\x0c\xfdj\x06Y>\x02\xe2h2\xa9\xe4\xa7%\x12r;
SF:\xcc\x82N\xc9\x8b\xf1\xeen\(\x83\xb3\x11\xaccP\x93\xb5\x96e\xa6\(\x97F\
SF:xe3")%r(SqueezeCenter,24,"@\xd0w\x93\xb1\xc0j\x01\xea\xff\xde\x0c\x06u\
SF:xa9\x87\xf5lk\x8a\xa5,\xe7v7\xfe\x0f@~\x9ae\xdf\xd9\xec\x9b\xce")%r(ser
SF:ialnumberd,1D,"\]\xf2\xa6\xbeI\x17FCe\xc3\x80wU\xb52\x16\xf5l\xa5\xc4Mv
SF:\x9ae\xb5\xce\xf1\xdd6")%r(ONCRPC_CALL,3C,"\x0e\xec\xe3\xca\0\0\0\0PR
SF:ST\x03\0\0\0EPID\x03\0\0\0RNON\x0b\0\0\0CADR\x13\0\0\0GFE\xb5i\x0f\0\0\
SF:0\0\0\x02\0m<\x0bO'\xda")%r(UPNP_MSEARCH,2A,"A\x81\x1b\x81,\x0e4\x12\x8
SF:b\xb6\xda\xa3\x8bi\xb6\xb5\xc9\x9cw\xebS\xa1_z\xe7\xbc\xdc\x1e2\?\xdf\x
SF:bbT\xfb\xbe\x99e\x8b\xc4\xc3\]\x04")%r(AMANDA_NOOP,2A,"L\xd1\xe6\x9d\xc
SF:1\xc2\$\xa5\xf3\xa9i\xc5A\x8c2;\xbf\x9a\xc9\x08m,\x04\xa1\[\)xu\xf8\xdb
SF:x\xf97\x98\x10\x9be\xed\xf6\x08\xdc\xfc")%r(WDB_TARGET_CONNECT,2A,"F\]\
SF:xab\xd4\xfc\xc4\x8f\xca\x90\xac\xcf\xd7\x9e\xd8=K\xfb\x8a\xfe\xe9\xdeC/
SF:L\xa3\x82\xab\x17\?\xf9\xe406\x87G\x9ae\x8c\(msl")%r(TS3INIT1,21,"G\xb2
SF:i#Ps\xfao\)\xfde\xa1Y<h\xc9\xae\xcc\]9R\x9b\xb5\xc8@\x05\x9be'\xf7\xc2\
SF:x03\xd5");

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 424.42 seconds
```

02 zmap to rescue

- Several researchers, with Johannes Zirngibl extended the ZMap fork with a QUIC module
  - This module sends QUIC initial packets with version 0x1a1a1a1a to force a Version Negotiation
  - Modules also pad packets to 1200 bytes as required by RFC9000
    - This means that an order of magnitude more traffic is produced than while performing a simple TCP SYN scan
      - But it works!
- The fork is available at https://github.com/tumi8/qscanner

03 QUIC-ing Croatia

- I used this modified version of zmap to scan all Croatian IP space
  - Contains ~2 million IP addresses
- Here's what QUIC looks like in Croatia
  - 230 IP addresses identified with QUIC services
    - 59 used by Google
    - 38 used by Facebook
    - 4 used by WhatsApp
    - 121 used by Akamai
    - 2 weird TRAEFIK devices
    - 5 used by Plus hosting / mojsite.com
    - rudar.rgn.hr – I was surprised but it just points to Plus hosting

04 QUIC-ing Croatia

- We can now scan identified sites to grab their HTTP/3 certificates etc
- Another tool was produced by same authors – Qscanner
  - Available at https://github.com/tumi8/qscanner
- It is written in Go, and thus quite fast
- Uses zmap's CSV output as an input file
- Supports logging of keys, QUIC transport parameters, TLS handshake information and X.509 certificates
  - A bit clumsy output in JSON
- Supports only HTTP/3 though

05 Release: quicmap

- Decided to make a new QUIC scanner that supports any/all ALPN's
- No need to use zmap/Qscanner anymore
  - Although this is in Python, but still relatively fast
  - Thanks to my colleague Fran Čutura
  - We will be adding new features in upcoming days/weeks
- Get it while it's hot!

- **https://github.com/bojanisc/quicmap**

06 Scanning ‘round the Internetz

```
bojanz@mnemosyne:~/work/quicmap$ python3 quicmap.py youtube.com,1.1.1.1,facebook.com -p 80,443
100%|                                                                                | 6/6 [00:05<00:00,  1.20it/s]
endpoint        : youtube.com
port            : 443
server_versions : 0x9a8aea9a, 0x1, 0xff00001d
ALPN            : h3 (HTTP/3)

endpoint        : 1.1.1.1
port            : 443
server_versions : 0x1
ALPN            : h3 (HTTP/3)

endpoint        : facebook.com
port            : 443
server_versions : 0xfaceb002, 0xfaceb00e, 0xfaceb011, 0xfaceb013, 0xfaceb010, 0x1, 0xfaceb003
ALPN            : h3 (HTTP/3)
```

07 Port knocking

- QUIC allows for almost ideal implementation of port knocking
  - Technique that allows us to start a backdoor service by sending a specific fingerprint/packet to the target server
- Or, we can invent our own protocol
  - Respond only if the proper ALPN string was supplied
    - We can use an existing, but not used protocol (i.e. irc)
    - Or use our own
    - How about using this for C&C?

```
bojanz@mnemosyne:~$ python3 quic2c.py -k 192.168.17.130 443
2024-02-24 11:32:03,017 INFO quic [907fc535587d2ac6] ALPN negotiated protocol infigo
2024-02-24 11:32:03,017 INFO client C&C request sent
2024-02-24 11:32:03,018 INFO client received: Welcome to Infigo C&C
2024-02-24 11:32:03,019 INFO quic [907fc535587d2ac6] Connection close sent (code 0x0, reason )
```

Time for questions?

YOUR DATA.
OUR RESPONSIBILITY.