mindthesec FORUM

HTTP/2 & InfoSec

Anderson Dadario

Topics

- HTTP Today
- Why HTTP/2
- How it works
- What is relevant to your InfoSec job

HTTP Today

- Using HTTP 1.1 since 1997 / 1999
  - Connection: keep-alive
  - Head of Line Blocking
- But we still use N TCP Connections per origin ...
- And Many Hacks because *requests are evil*
  - CSS Spriting
  - Inlining
  - Concatenation
  - Domain Sharding
- No Header Compression

So comes SPDY in 2009

- With some cool stuff
  - Header Compression (vulnerable to CRIME)
    - Now cookieless domains are useless
  - Multiplexing
    - Now sharding is harmful (1 TCP connection per origin)
    - Has prioritization (e.g., focus on JS and CSS files)
  - Server Push
    - Although some pushes may be wasteful, there is "Server Hint" for SPDY, and RST_STREAM for HTTP/2
  - HTTPS Only → there's a gotcha here: do you wonder why? avoid intermediaries

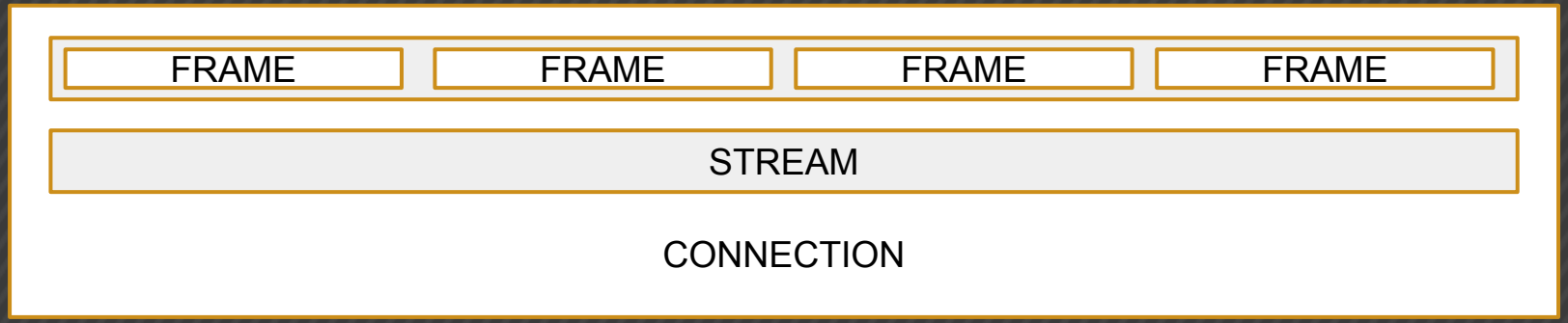
What about HTTP/2?

- Used SPDY 3 as its first draft
- Main Driven by Performance
- But also includes …
  - Security
  - Reliability

Key Differences

- Binary instead of ASCII
- Header Compression (HPACK - RFC 7541)
- Fully multiplexed - Means: Parallelism and Out of Order Req/Res
  - Stream Prioritization
  - 1 TCP Connection > N Streams > N Frames
    - Solves Head of Line Blocking
- Server Push what it thinks that the client will need (e.g., assets)

HTTP/2 Units



Frames have
- FLAGS,
- TYPE,
- STREAM IDENTIFIER,
- PAYLOAD and
- LENGTH

Streams have
- IDENTIFIER
- STATE
- PRIORITY
- FLOW CONTROL

Connections have
- FLOW CONTROL

HTTP/2 Frame Types

1. DATA
2. HEADERS
3. PRIORITY
4. RST_STREAM
5. SETTINGS
    a. SETTINGS_HEADER_TABLE_SIZE
    b. SETTINGS_ENABLE_PUSH
    c. SETTINGS_MAX_CONCURRENT_STREAMS
    d. SETTINGS_INITIAL_WINDOW_SIZE
    e. SETTINGS_MAX_FRAME_SIZE
    f. SETTINGS_MAX_HEADER_LIST_SIZE
6. PUSH_PROMISE
7. PING
8. GOAWAY
9. WINDOW_UPDATE
10. CONTINUATION

HTTP/2 GET

```
GET /resource HTTP/1.1              HEADERS
Host: example.org          ==>        + END_STREAM
Accept: image/jpeg                    + END_HEADERS
                                        :method = GET
                                        :scheme = https
                                        :path = /resource
                                        host = example.org
                                        accept = image/jpeg
```

```
HTTP/1.1 304 Not Modified           HEADERS
ETag: "xyzzy"              ==>        + END_STREAM
Expires: Thu, 23 Jan ...              + END_HEADERS
                                        :status = 304
                                        etag = "xyzzy"
                                        expires = Thu, 23 Jan ...
```

HTTP/2 POST Request

```
POST /resource HTTP/1.1
Host: example.org
Content-Type: image/jpeg
Content-Length: 123

{binary data}
```

==>

```
HEADERS
  - END_STREAM
  - END_HEADERS
    :method = POST
    :path = /resource
    :scheme = https

CONTINUATION
  + END_HEADERS
    content-type = image/jpeg
    host = example.org
    content-length = 123

DATA
  + END_STREAM
{binary data}
```

HTTP/2 POST Response

```
HTTP/1.1 200 OK                          HEADERS
Content-Type: image/jpeg      ==>          - END_STREAM
Content-Length: 123                        + END_HEADERS
                                             :status = 200
{binary data}                                content-type = image/jpeg
                                             content-length = 123

                                         DATA
                                           + END_STREAM
                                         {binary data}
```

Request Reliability

8.1.4 Request Reliability Mechanisms in HTTP/2

In HTTP/1.1, an HTTP client is unable to retry a non-idempotent request when an error occurs because there is no means to determine the nature of the error. It is possible that some server processing occurred prior to the error, which could result in undesirable effects if the request were reattempted.

HTTP/2 provides two mechanisms for providing a guarantee to a client that a request has not been processed:

- The GOAWAY frame indicates the highest stream number that might have been processed. Requests on streams with higher numbers are therefore guaranteed to be safe to retry.
- The REFUSED_STREAM error code can be included in a RST_STREAM frame to indicate that the stream is being closed prior to any processing having occurred. Any request that was sent on the reset stream can be safely retried.

Requests that have not been processed have not failed; clients MAY automatically retry them, even those with non-idempotent methods.

Upgrade Request Anatomy

When you don't know if it supports HTTP/2

- "h2c" means no TLS connection
- "h2" means TLS connection [TLS-ALPN]

GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>

[Response]

HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c

Implicit acknowledgement of HTTP2-Settings

- A server MUST NOT upgrade the connection to HTTP/2 if this header field is not present or if more than one is present.
- A server MUST NOT send this header field.

InfoSec Overview 1-4

- Increased Attack Surface
  - Supporting HTTP/1 and HTTP/2
  - HTTP/2 extensions (e.g., new settings, frame type or error code)
  - Possibility to simulate bad implementations that results in DoS
    - e.g., reply RST_STREAM to a RST_STREAM frame.
- Non mature implementations == High probability to find Bugs
  - E.g., Yahoo fuzzing Apache HTTP/2
- DAST Market
  - Force scanners to support HTTP/2
  - Decrease scan time

InfoSec Overview 2-4

- Wireshark support (partially)
  - Support HPACK but missing continuation frame support...
- Frame Padding to obscure the size of messages
  - “Use of padding can result in less protection than might seem immediately obvious. At best, padding only makes it more difficult for an attacker to infer length information by increasing the number of frames an attacker has to observe.” RFC 7540
- TLS Cipher Blacklist (MAY trigger INADEQUATE_SECURITY ERR)
- TLS 1.2 or higher w/ SNI support is a MUST
- TLS MUST disable compression and renegotiation

InfoSec Overview 3-4

- TLS Implementations MUST support ephemeral key exchange sizes of at least 2048 bits for cipher suites that use ephemeral finite field Diffie-Hellman (DHE) [TLS12] and 224 bits for cipher suites that use ephemeral elliptic curve Diffie-Hellman (ECDHE) [RFC4492]. Clients MUST accept DHE sizes of up to 4096 bits.
- Opportunistic Security for HTTP (...)
  - "(...) serve http URIs over TLS without being required to support strong server authentication. (...)"

For pentesting:
- it is possible for server configurations to change;
- for configurations to differ between instances in clustered servers, or
- for network conditions to change.

InfoSec Overview 4-4

- (...) Opportunistic Security for HTTP (Opportunistic Encryption)
  - No padlock symbol
  - Won't verify X.509 certificate: "(...) The server certificate, if one is proffered by the alternative service, is not necessarily checked for validity, expiration, issuance by a trusted certificate authority or matched against the name in the URI. (...)"
  - Left out from HTTP/2 RFC
  - Polemic: does it prevents HTTPS adoption or help HTTP?
- ALMOST mandatory HTTPS as Google and Firefox said that their browsers will only allows HTTP/2 for HTTPS connections
- Many open TCP connections (persistent connections)

HTTP/2 Adoption Rate

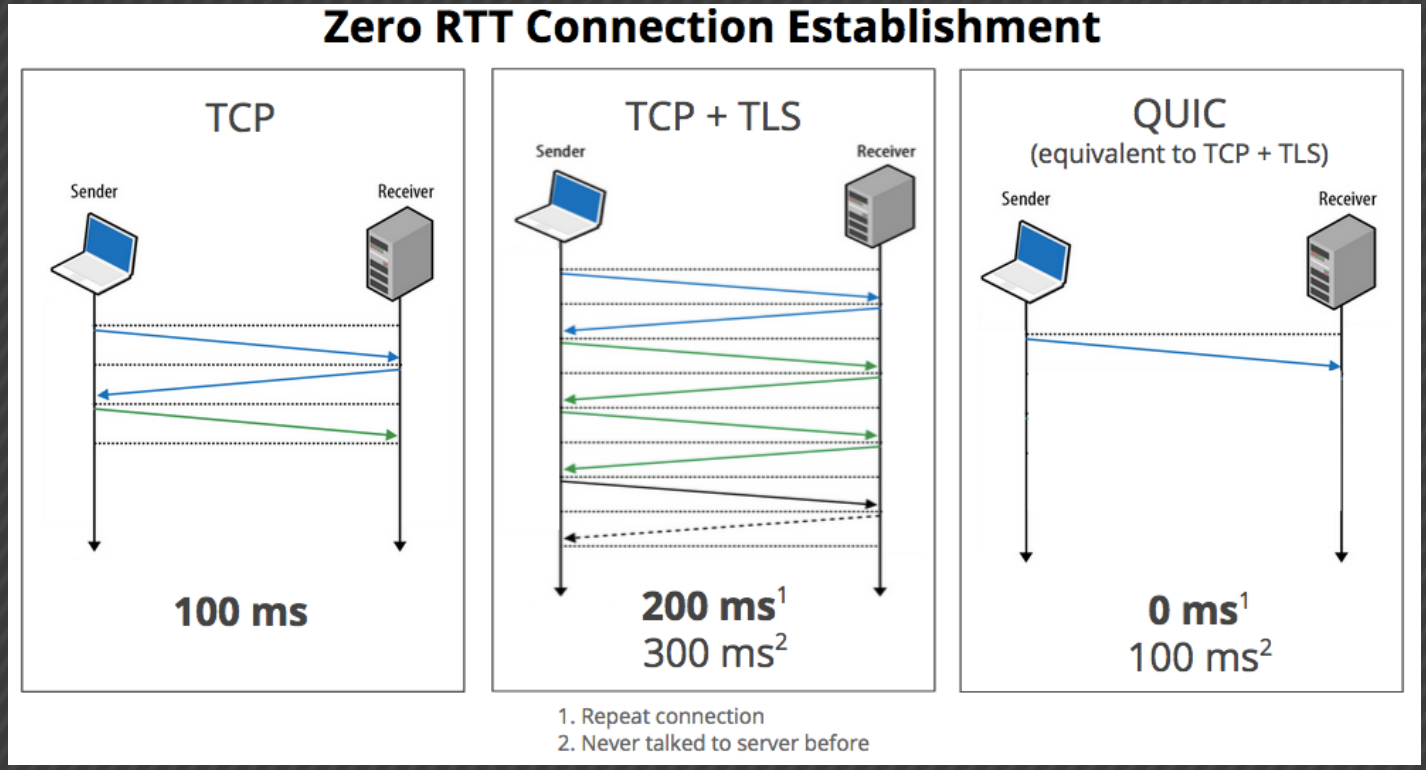
- Browsers: Chrome and Firefox latest versions support already
- Servers: Apache (mod_h2), jetty, Apache Traffic Server
- Services: Google, Twitter
- Proxy: Squid
- CDN
  - Akamai said in the end of the year and
  - CloudFlare when ‘nginx supports HTTP/2’

References 1-2

1. HTTP/2 - RFC 7540 - http://www.rfc-editor.org/rfc/rfc7540.txt
2. HPACK - RFC 7541 - http://www.rfc-editor.org/rfc/rfc7541.txt
3. ALPN - RFC 7301 - https://tools.ietf.org/html/rfc7301
4. HTTP/2 Home - https://http2.github.io/
5. Daniel's Blog - http://daniel.haxx.se/blog/
6. SPDY & HTTP 2 with Akamai CTO Guy Podjarny https://www.youtube.com/watch?v=WkLBrHW4NhQ
7. An overview of HTTP/2 with Daniel Sommermann https://www.youtube.com/watch?v=-yxQIRl6Qic
8. HTTP/2 (Mark Nottingham) https://www.youtube.com/watch?v=OQ158bJPvx4

References 2-2

9. Pervasive Monitoring - RFC 7258 - http://www.rfc-editor.org/info/rfc7258
10. Opportunistic Security for HTTP - http://httpwg.github.io/http-extensions/encryption.html
11. HTTP/2 Book - http://daniel.haxx.se/http2/
12. TLS in HTTP/2 - http://daniel.haxx.se/blog/2015/03/06/tls-in-http2/
13. HTTP BIS mailing list



QUIC: UDP-based transport protocol for the modern Internet

Today, roughly half of all requests from Chrome to Google servers are served over QUIC

Thanks!

Anderson Dadario

anderson@gauntlet.io | Twitter: @andersonmvd

http://Gauntlet.io