Attacking HTTP/2 Implementations

Stuart Larsen and John Villamil
Yahoo! Pentest Team
Introduction

- HTTP/2
- Why HTTP/2 is hard
- http2fuzz
- ATS
- Firefox
- NodeJS
- Conclusion
whoami

Stuart Larsen
- https://c0nrad.io

John Villamil
- @day6reak
HTTP/1.1 came out in 1999

Since then…
- HTML and JS have evolved
- Sites are much more complex
- ISP speeds have improved and more bandwidth is available
- Far more content is served over the web
- SSL is now more common
HTTP/2 is about performance

- “Hello HTTP/2, Goodbye SPDY”

- HTTP/2 evolved from SPDY and SPDY/2

- Features:
  - Better use of TCP
  - Binary protocol
    - Frames and Streams
  - Multiplexing (Streams)
    - reducing perceived latency
  - Server Push (PUSH_PROMISE)
Upgrading from HTTP/1.1

- Keep the same URI scheme http:// and https://
- TLS is optional
  - Chrome and Firefox make it mandatory
  - Internet Explorer and other clients, including curl, keep it optional
- Upgrade header for http://
  - Causes a roundtrip but HTTP/2 connections are more robust
- No minor version numbers

GET /index.html HTTP/1.1
Host: example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <SETTINGS payload>

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

- HTTP is stateless
  - Request sizes (cookies and other headers) are significant in size
  - Compression is a natural solution
- “Hpack was designed to make it difficult for a conforming implementation to leak information, to make encoding and decoding very fast/cheap, to provide for receiver control over compression context size, to allow for proxy re-indexing (i.e. shared state between frontend and backend within a proxy), and for quick comparisons of huffman-encoded strings.”
- Receiver controls the maximum memory used
  - 0 at a minimum and 2^32 at a maximum
  - Specified in SETTINGS frame
- SPDY uses deflate algorithm
  - Vulnerable to CRIME
HPACK

- Maps name:value pairs to an index
  › Creates a dynamic table on both endpoints
- Differential Encoding
  › Yields big gains in terms of speed and size
  › Reference tables are built for the headers
  › Only encode the differences between current and previous headers
  › Eliminates redundancy
- Huffman Encoding
  › static huffman table
  › string literals

<table>
<thead>
<tr>
<th>Index</th>
<th>Header Name</th>
<th>Header Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>:authority</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>:method</td>
<td>GET</td>
</tr>
<tr>
<td>3</td>
<td>:method</td>
<td>POST</td>
</tr>
<tr>
<td>4</td>
<td>:path</td>
<td>/</td>
</tr>
<tr>
<td>5</td>
<td>:path</td>
<td>/index.html</td>
</tr>
<tr>
<td>6</td>
<td>:scheme</td>
<td>http</td>
</tr>
<tr>
<td>7</td>
<td>:scheme</td>
<td>https</td>
</tr>
<tr>
<td>8</td>
<td>:status</td>
<td>200</td>
</tr>
<tr>
<td>9</td>
<td>:status</td>
<td>204</td>
</tr>
</tbody>
</table>
HPACK Attack Surface

- Index table sizes and offsets
- String literal example
  - One bit determines whether its huffman encoded
  - Integer representing size
  - String data follows
  - Padding up to an octet boundary
- Context updates
- Header Table Size Changes
  - By default, the dynamic table size is 4k
Multiplexing

- HTTP/1.1 had pipelining, so multiple request can be sent
- Head-Of-Line blocking
  - Responses have to come back in order
### Frames

- **Fundamental unit of communication**

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>24</td>
</tr>
<tr>
<td>Type</td>
<td>8</td>
</tr>
<tr>
<td>Flags</td>
<td>8</td>
</tr>
<tr>
<td>Stream Identifier</td>
<td>31</td>
</tr>
<tr>
<td>Payload</td>
<td>0+</td>
</tr>
</tbody>
</table>

#### Types

- Headers
- Data
- Priority
- Reset
- Settings
- Push
- Ping
- Goaway
- Update
- Continuation
Push

- New feature for pushing resources onto a client
- For example, if a client requests index.html from a web server, the web server can probably assume the client will also want logo.png
  - Instead of waiting for the client to request logo.png, the server can preemptively push logo.png so the user doesn’t have to wait an entire request
HTTP/2 New Attack Surface

- HPACK
- Upgrades / Downgrades
- Inconsistent Multiplexing
- Malformed Frames
- Pushing arbitrary data to client
- Pushing arbitrary data to server
- Stream Dependencies
- Invalid Frame States
Fuzzing

- Sending random data (binary/ascii) to an application and monitoring for unexpected behavior
Fuzzing HTTP

**POST** /somepath?query=abc#fragment

Host: yahoo.com
Accept: text/plain
User-Agent: Chrome
Content-Length: 200

{ data: 10 }
http2fuzz

- first public http2 fuzzer
- built in golang
- operates as either client or server

replay mode for testcase minification
highly concurrent
http2fuzz: client mode

1. the client establishes an https connection with the server
2. in the ALPN section, h2 or h2-14 is specified
3. initial settings frames are sent back and forth
4. randomly generated frames are sent to the server
5. upon crashing, a new connection is started
http2fuzz: server mode

1. the server loads a localhost certificate / key for TLS
2. the server binds to a port and waits for an incoming connection
3. upon connection, random frames are sent in response to the browser till the browser closes the connection

- To fuzz browsers, I use a script like:
Strategies

- Mixture of valid, semi-valid, and completely invalid frames
- Each strategy generates frames a specific type
- Strategies are grouped together to create mini-fuzzers
Strategies

SettingsFuzzer:
- Picks a random number between 0-5
- Appends that many random settings with random values to a SettingsFrame

HeaderFuzzer:
- Picks a random number between 0-5
- Appends that many random HTTP headers with random values to a HeadersFrame
Strategies

PriorityFuzzer:
  ▪ Sends Priority frames with a random streamDependency, steamId, weight, and exclusive value

PingFuzzer:
  ▪ Sends a ping frame with a random 8 byte payload

ResetFuzzer:
  ▪ Sends a RST Frame with a random streamId and errorCode
Strategies

WindowUpdateFuzzer:
- Sends a Window Update Frame with a random streamId, and incr value.

RawFrameFuzzer:
- Generates a random frameType (0-12), randomFlags (0-256), and streamId(2**31), and a random byte array of length 0-10000.
- Sends the invalid frame
Strategies

DataFuzzer:
- Sends a Data Frame with a random streamId, endStream bool, and random payload between 0-10000 bytes

PushPromiseFuzzer:
- Sends a PushPromise Frame with a random payload of 0-10000 bytes, streamId, promiseId, endHeaders bool, and padlengthh (0-256)
Strategies

ContinuationFuzzer:
- Sends a Continuation Frame with a random streamId, endStream bool and payload of length 0-10000 bytes.

RawTCPFuzzer:
- Establishes a TLS connection, and sends complete garbage to it. The payload is a byte array of length 0-10000.
Replay Mode

- Once the fuzzer finds a crash, the cause needs to be determined
- When fuzzing, each payload is saved in replay.json
- After a crash the fuzzer stops and the file is closed
- Running ./http2fuzz --replay send the frames back to the server in the same order
- So you can delete some frames, and see if the payload is still crashing the server, reducing the set of frames till only the necessary frames remain
It started with ATS

- Yahoo Pentest Team decided to look at ATS (Apache Traffic Server)

Http2ConnectionState.cc

577) static const http2_frame_dispatch
578) frame_handlers[HTTP2_FRAME_TYPE_MAX] = {
579)   rccv_data_frame, // HTTP2 FRAME TYPE_DATA
580)   rccv_headers_frame, // HTTP2 FRAME TYPE_HEADERS
581)   rccv_priority_frame, // HTTP2 FRAME TYPE_PRIORITY
582)   rccv_rst_stream_frame, // HTTP2 FRAME TYPE_RST_STREAM
583)   rccv_settings_frame, // HTTP2 FRAME TYPE_SETTINGS
584)   rccv_push_promise_frame, // HTTP2 FRAME TYPE_PUSH_PROMISE
585)   rccv_ping_frame, // HTTP2 FRAME TYPE_PING
586)   rccv_goaway_frame, // HTTP2 FRAME TYPE_GOAWAY
587)   rccv_window_update_frame, // HTTP2 FRAME TYPE_WINDOW_UPDATE
588)   rccv_continuation_frame, // HTTP2 FRAME TYPE_CONTINUATION
589)};

577) case HTTP2_SESSION_EVENT_RECV: {
578)   Http2Frame *frame = (Http2Frame *)edata;
579)
580)   if (frame->header().type > countof(frame_handlers)) {
581)     return 0;
582)   }
583)
584)   error =
585)     frame_handlers[frame->header().type](this->ua_session, *this,
586)     *frame);
ATS 2

- From the RFC - [http://tools.ietf.org/html/rfc7541#section-4.3](http://tools.ietf.org/html/rfc7541#section-4.3)
  
  **4.3. Entry Eviction When Dynamic Table Size Changes**
  
  Whenever the maximum size for the dynamic table is reduced, entries are evicted from the end of the dynamic table until the size of the dynamic table is less than or equal to the maximum size.

- Bug occurs due to an unexpected packet ordering, updating the table size when the “headers” table is empty

HPACK.cc

[214] Http2DynamicTable::set_dynamic_table_size(uint32_t new_size)
[219] MIMEField *last_field = _headers.last();

_headers is an empty vector

Vec.h

[131] C &
[132] last() const
[133] {
[134] return v[n - 1];
[135] }

Program received signal SIGSEGV, Segmentation fault.

0x00000000006427de in Http2DynamicTable::set_dynamic_table_size (this=0x318e650, new_size=25) at HPACK.cc:219
219   HPACK.cc: No such file or directory.
(gdb) bt
#0 0x00000000006427de in Http2DynamicTable::set_dynamic_table_size (this=0x318e650, new_size=25) at HPACK.cc:219

The dereferenced rax register is set to a negative value.

(gdb) i r
rax 0xffffffffffffff8 -8
Firefox HTTP/2 Malformed Header Frame DoS

- A malformed http2 header frame is sent to the browser
- Normally a header frame consist of a pad length, steam dependency identifier, weight, header block fragment, and padding
- But only a single byte is sent
- This eventually results in an integer underflow, which causes nsCString to try to allocate nearly $2^{32}$ bytes of memory
Firefox HTTP/2 Malformed Header Frame DoS

On line 1226 of file Http2Session.cpp, a call is made to append the decompressed frame onto a decompressed frame buffer:

```c++
self->mDecompressBuffer.Append(self->mInputFrameBuffer + kFrameHeaderBytes + paddingControlBytes + priorityLen, self->mInputFrameDataSize - paddingControlBytes - priorityLen - paddingLength);

self->mDecompressBuffer.Append(ptr + unit_8(9) + uint8_t(0) + uint16_t(0),
   |uint32_t(1) - uint8_t(0) - uint32_t(5) - uint16_t(0)|);
```
Firefox HTTP/2 Malformed PushPromise Underflow DoS

On line 1634 of Http2Session.cpp, a call is made to append the decompressed frame onto a decompressed frame buffer:

```c
self->mDecompressBuffer.Append(self->mInputFrameBuffer + kFrameHeaderBytes + paddingControlBytes + promiseLen,
    self->mInputFrameDataSize - paddingControlBytes - promiseLen - paddingLength);
```

```c
self->mDecompressBuffer.Append(
    nsAutoArrayPtr<char>(ptr) + uint_8(9) + uint8_t(1) + uint32_t(4),
    uint32_t(76) - uint8_t(1) - uint32_t(4) - uint16_t(75));
```

= 4294967292
node-http2

- **index out of range in parsing small payload**
  
  #148 opened on Sep 1 by c0nrad

- **illegal frame HEADERS crash from Stream._pushUpstream**

  #147 opened on Sep 1 by c0nrad

- **unspecified "error" event in stream.js:265**

  #146 opened on Sep 1 by c0nrad

- **illegal frame crasher http2/node_modules/http2/lib/protocol/stream.js:101**

  #145 opened on Sep 1 by c0nrad

- hard to determine state
  - Which frame can be sent when
- all fields must be validated
  - value and size
Conclusion

- HTTP2
- Why HTTP2 is hard
- http2fuzz
- ATS
- Firefox
- NodeJS
The End

Questions?