Comparisons and attacks on HTTP2 (English translation)

Comparisons and attacks on HTTP2 (Comparaisons et attaques sur HTTP2) - Georges Bossert, SSTIC.

- Title in English: Comparisons and attacks on HTTP2
- Published: date not stated
- Original: <https://www.sstic.org/media/SSTIC2016/SSTIC-actes/comparaisons_attaques_http2/SSTIC2016-Slides-comparaisons_attaques_http2-bossert.pdf>
- Preserved from: https://www.sstic.org/media/SSTIC2016/SSTIC-actes/comparaisons_attaques_http2/SSTIC2016-Slides-comparaisons_attaques_http2-bossert.pdf (live) on 2026-09-10
- Licence: unknown

Content (translated into English)

Machine translation of `comparisons-attacks-http2-comparaisons-et-attaques-sur-http2.md`, which holds the source's own words. Code, payloads, type names, URLs and CVE identifiers were masked before translating and restored after, so they are byte-identical to the original.

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Comparisons and attacks on HTTP2 Georges Bossert - June 3, 2016

@Lapeluche Agenda

1. Brief introduction to the HTTP/2 protocol
2. Comparison of server stacks
3. Exploitation of the results

@Lapeluche HTTP-What?

1990 1999 2009 HTTP 0.9 HTTP 1.1 SPDY 1.0

1996 2015 HTTP 1.0 HTTP 2.0

@Lapeluche 1990 1999 2009 SPDY HTTP 0.9 HTTP 1.1 1.0

SPDY 1.0 1996 2015 HTTP 1.0 HTTP 2.0

An improvement to HTTP proposed by Google

Main objective: Reduce latency

In HTTP/1.1

one HTTP request = one TCP connection (head-of-line blocking) the client ALWAYS initiates a data exchange headers are not compressed (and are sometimes unnecessary) the content of exchanges is not always compressed

@Lapeluche 1990 1999 2009 SPDY HTTP 0.9 HTTP 1.1 1.0

SPDY 1.0 1996 2015 HTTP 1.0 HTTP 2.0

An improvement to HTTP proposed by Google

Main objective: Reduce latency

Main features

an unlimited number of concurrent streams over a TCP connection request prioritization header compression "Server Push" and "Server Hint" SSL Required!

@Lapeluche 1990 1999 2009 SPDY HTTP 0.9 HTTP 1.1 1.0

HTTP/2 1996 2015 HTTP 1.0 HTTP 2.0

Standardization of SPDY

The first draft (Nov. 2012): a copy of SPDY

Then a few differences:

SSL NOT required! TLS extension: ALPN (not NPN) HPACK for compression (Oracle Attacks: BREACH, CRIME) Improved prioritization, (...)

@Lapeluche HTTP/2 A binary protocol that is encrypted almost all the time

Implementation of a state machine for stream multiplexing

A "stream" can be prioritized, reprioritized and canceled at any time A "stream" can have dependencies on other "streams" A "stream" has its own "control flow"

Headers are compressed

The server can "push" data to clients

Use of Upgrade-mode or TLS/ALPN negotiation

@Lapeluche Some websites using HTTP/2 Google, Facebook, Twitter, Wikipedia, Yahoo, Cloudflare, Amazon.com, ...

@Lapeluche A majority of browsers use HTTP/2 IE, Edge, Firefox, Chrome, Safari, Opera, IOS Safari, Firefox Android, ...

30/05/2016 @Lapeluche A large number of HTTP/2-compatible servers

> 40 implementations listed at

https://github.com/http2/http2-spec/wiki/Implementations

Aerys, Netty, Apache Httpd, Node-http2, Apache Tomcat, OpenLitespeed, Deuterium, ... H2O, Jetty, Nginx, Implémentations partielles du HTTP/2 "Push" pas toujours disponible "SSL not required" pas toujours respecté @Lapeluche HTTP/2 est complexe Protocole très récent Nombreux utilisateurs Nombreuses implémentations Nombreuses fonctionnalités

Mise en oeuvre de HPACK Passage d’un protocole stateless à statefull Repose sur une machine à états

@Lapeluche HTTP/2 is complex Very recent protocol Many users Many implementations Many features Implementation of HPACK Transition from a stateless to a stateful protocol Relies on a state machine

How can we help developers? @Lapeluche Agenda

1. Brief introduction to the HTTP/2 protocol
2. Comparison of server stacks
3. Exploitation of the results

@Lapeluche Comparaison de piles protocolaires Piste 1 : Comparer les documentations Pas toujours à jour Pas suffisamment précises Piste 2 : Comparer les codes sources Manuellement (très lent) ou Automatiquement (très complexe) Pas toujours facile d’accéder au code source Piste 3 : Collecter puis analyser des traces d’exécutions Difficulté de mener une comparaison reproductible Complétude de la comparaison limitée au contenu des traces Piste 4 : Inférer l’automate en stimulant l’implémentation L’implémentation doit être instrumentable La machine à états doit être représentable par un IO-Automata

@Lapeluche Comparison of protocol stacks ACTIVE inference of an implementation's automaton via LSTAR (L*)

1. Initialize an observation table
2. Fill the observation table

  Construct a sequence of messages and send it to the implementation Store the responses in the observation table Reset the implementation

1. Transform the observation table into an automaton
2. Compare the automaton with that of the implementation

  Generate random sequences accepted by the automaton If an error is found, correct the observation table and return to 2)

1. The inferred automaton is equivalent to that of the implementation

@Lapeluche Comparison of protocol stacks ACTIVE inference of an implementation's automaton via LSTAR (L*)

  Creation of an open-source implementation in python: pylstar

start / stop send serveur_x.dot pylstar Wrapper.py Server X receive

Vocabulaire.py GDB Valgrind Pin @Lapeluche Comparison of protocol stacks The HTTP/2 servers compared

Name Info Selected version

Apache “mod_http2” module (experimental) 2.4.20 (latest)

- nghttp2 1.10.0 (latest)

Nginx http2 not available in stable 1.9.15 (mainline)

H2o latest stable (2.0.0 under development) 1.7.1 (latest release)

Tomcat 9 9.0.0.M4 (latest)

Versions identified on April 25, 2016 https://github.com/gbossert/http2_compare All source code All build parameters All configurations @Lapeluche Comparison of protocol stacks Construction of the input vocabulary (vocabulaire.py)

Read the HTTP/2 specifications Identify edge cases Negation of MUST, SHOULD, MAY Represent messages with Netzob

@Lapeluche Comparison of protocol stacks Construction of the input vocabulary (vocabulaire.py)

24 input messages 34 output messages

@Lapeluche Example result

@Lapeluche Example result Final state Streams Processing

Initial state

@Lapeluche Agenda

1. Brief introduction to the HTTP/2 protocol
2. Comparison of server stacks
3. Exploitation of the results

@Lapeluche Attacks With knowledge of the automata, we can

create a smart-fuzzer

If the automata are not strictly equivalent, we can

create a fingerprinting tool create IDS evasion rules

@Lapeluche Intelligent fuzzer Implementation of an HTTP/2 smart-fuzzer

Phase 1: Positioning within the automaton Random traversal of the automaton Respect for the vocabulary and grammar Random depth and Reset

@Lapeluche Intelligent fuzzer Implementation of an HTTP/2 smart-fuzzer

Phase 1 : Positionnement dans l'automate Parcours aléatoire de l'automate Respect du vocabulaire et de la grammaire Profondeur et Reset aléatoire Phase 2 : Fuzzing Grammatical + Vocabulaire Fuzzing Grammatical Choix aléatoire du prochain état Choix du message parmi les messages acceptés par les transitions menant à cet état Fuzzing Vocabulaire Modification du type et de la valeur des champs Modification des contraintes sur les champs, ... @Lapeluche Fuzzer intelligent Nécessite du CPU, beaucoup de CPU...

It runs in the background on my server @home

Several bugs found

two security bugs identified in h2o nginx and apache crashes not yet characterized

TODO

improve exploitation of the bugs use a cloud (AWS, Azure, ...) implement a feedback loop @Lapeluche HTTP2 fingerprinting Web server fingerprinting relies

on the banner on the value of certain fields

Proposal: use the automaton as a fingerprint

Identify differences between the automata Use probabilities

@Lapeluche Connection_preface

WTF (1) ! Settings_small_max_header_list_size Settings_small_max_header_list_size Settings_small_max_header_list_size, Window_update_size_inc_stream_0

Settings_initial_header_table_size Nginx H2o Empty_settings_ack, Windows_update Empty_settings_ack Tomcat Empty_settings_ack Empty_settings Empty_settings_ack Empty_settings_ack Empty_settings_ack Settings_invalid_name Empty_settings_ack Empty_settings_ack Empty_settings_ack Priority_stream_2_average_weight_dp_stream_0

@Lapeluche IDS evasion The theory

If WEB servers behave differently, it is difficult for the IDS to track the streams.

In practice

HTTP/2 is not yet supported by the main IDSs BUT, it is likely to be complicated for IDS developers!

@Lapeluche Connection_preface

WTF (2) ! Settings_small_max_header_list_size Settings_small_max_header_list_size Settings_small_max_header_list_size, Window_update_size_inc_stream_0

Settings_large_header_table_size Nginx H2o Empty_settings_ack, Windows_update Empty_settings_ack Tomcat Empty_settings_ack Priority_stream_1_maximal_weight_depends_stream_1 Empty_symbol GO-AWAY Rst_stream_1_protocol_error Empty_settings_unknown_stream

Empty_settings_ack Ping TCP_CLOSED TCP_CLOSED Ping_ack @Lapeluche Connection_preface

WTF (n) ! Settings_small_max_header_list_size Settings_small_max_header_list_size Settings_small_max_header_list_size, Window_update_size_inc_stream_0

Windows_update_size_inc_average_stream_0 Nginx H2o Tomcat

Settings_disable_push

Empty_settings_ack Empty_settings_ack Ping_stream_1

Go_away_protocol_error Ping_ack Settings_minimal_max_header_list_size TCP_CLOSED

Empty_settings_ack @Lapeluche Connection_preface

WTF (n+1) ! Settings_small_max_header_list_size Settings_small_max_header_list_size Settings_small_max_header_list_size, Window_update_size_inc_stream_0

Windows_update_size_inc_0_stream_0 Nginx H2o Go_away_protocol_error Tomcat Empty_symbol Ping

Ping_ack Settings_normal_initial_window_size TCP_CLOSED TCP_CLOSED Empty_settings_ack Headers_stream_3_end_stream_end_header_big_get_huffman

Data (content of the web page) @Lapeluche Conclusion RFCs do not allow a protocol's automaton to be formally defined a few initiatives (see Cosmogol by S. Bortzmeyer)

The tools presented here are available under the GPLv3 license:

https://github.com/gbossert/pylstar

https://github.com/netzob/netzob

https://github.com/gbossert/http2_compare

Feel free to come and discuss your use cases (and your questions)

@Lapeluche 1990 1999 2009 SPDY HTTP 0.9 HTTP 1.1 1.0

HTTP 0.9 1996 2015 HTTP 1.0 HTTP 2.0

First “documented” version

Only one method available: GET No concept of headers or metadata Only one file type: text/plain RFC 7230: “The expectation to support HTTP/0.9 requests has been removed”

@Lapeluche 1990 1999 2009 SPDY HTTP 0.9 HTTP 1.1 1.0

HTTP 1.0 1996 2015 HTTP 1.0 HTTP 2.0

First standardized version (RFC 1945)

Welcome to data transfer (POST method) Concept of MIME types: ability to transport several file types Solution for cache and congestion management Authentication: BASIC and DIGEST methods (...)

@Lapeluche 1990 1999 2009 SPDY HTTP 0.9 HTTP 1.1 1.0

HTTP 1.1 1996 2015 HTTP 1.0 HTTP 2.0

Modernization of HTTP

Enables virtual hosting (Virtual-hosting) OPTION method + UPGRADE header TCP connections are persistent by default Pipelining Cache ++ (Cache-Control, age, max-age, ...) (...)

@Lapeluche