Scientific Research Publishing

Journal of Computer and Communications, 2017, 5, 98-111
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Steganography of Hypertext Transfer Protocol Version 2 (HTTP/2)

Biljana Dimitrova, Aleksandra Mileva

Faculty of Computer Science, University Goce Delčev, Štip, Republic of Macedonia
Email: aleksandra.mileva@ugd.edu.mk

How to cite this paper: Dimitrova, B. and Mileva, A. (2017) Steganography of Hypertext Transfer Protocol Version 2 (HTTP/2). *Journal of Computer and Communications*, **5**, 98-111.
https://doi.org/10.4236/jcc.2017.55008

Received: February 22, 2017
Accepted: March 28, 2017
Published: March 31, 2017

Abstract

Network steganography consists of different steganographic technics that utilize network protocols for hiding data. We present nine new covert channels which utilize the new standard, HTTP/2, and which can be used regardless its transport carrier (TLS or clear TCP). These covert channels use a protocol feature that has dual nature (for example, no padding can be represented in two ways); or a feature that is not mandatory (as streams prioritization and dependencies); or random value field (as PING frame payload field); or there is no strict rule how to obtain new values for some fields (as stream identifiers). As far as we know, this is the first research about hiding data in HTTP/2. Also, we give a small survey of existing covert channels that can be created using HTTP/1.x, with the analysis do they work or not work with the HTTP/2.

Keywords

Network Steganography, Network Security, Information Hiding, Covert Channels

1. Introduction

Network steganography is the art of hiding secret data in legitimate transmissions in communication networks without destroying the used hidden data carrier [1]. Usually it deploys different network protocols as carriers, while trying to conceal the presence of hidden data from network devices. The main area of study in the modern steganography, consequently are in the network steganography, also, are the covert channels. A covert channel is any communication channel that can be exploited by a process to transfer information in a manner that violates the systems security policy [2]. Network-based covert channels can be used illegally to coordinate distributed denial of service attacks or spreading of malware (e.g., the worm W32. Morto used DNS records to communicate with its command and control server), for secret communication between terrorists

and criminals, industrial espionage, but also legally, for circumvention of the limitation in using Internet in some countries (e.g., Infranet [3]), secure network management communication [4], copyright protection, etc.

One possible classification of network steganography methods is given in [5], where three broad groups are separated:

- Methods that modify protocol data unit (PDU), including fields with protocol control information from protocol header or/and the protocol payload.
- Methods that modify the structure of PDU streams, by PDU reordering, intentional losses, use of inter-packet delays, modification of timestamps, etc.
- Hybrid methods, which involve a combination of previous two types of methods.

The best choices of network protocols as a carrier of secret data are the most popular and most used protocols, so, the Hypertext Transfer Protocol (HTTP) arises as a natural choice. While the Web has dramatically evolved over the last two decades, in its bloodstream, the HTTP/1.1 till 2014 has remained without modification from its standardization in RFC 2068 in 1997 and its improvements in RFC 2616 in 1999. Previous years showed many shortcomings and inflexibilities in HTTP/1.1 when coping with new web technologies. In June 2014, the HTTPbis Working Group of the IETF released an updated specification from six parts (RFC 7230-5) and in May 2015 released a new major version HTTP/2 in RFC 7540. HTTP/2 is a binary protocol and it brings many improvements and benefits, compared to its predecessor, like:

a) Multiplexing and concurrency: Several HTTP requests can be sent on the same TCP connection as separate streams, and their responses can be received out of order in the same streams. This feature eliminates the need for multiple TCP connections between the client and the server;
b) Server push: If the server has a knowledge that some resources are needed and will be requested later for a given web site, the server can send these resources without a request, and the client will cache the resources till later;
c) Header compression: HTTP header size is drastically reduced using special frames and compression;
d) Stream dependencies and priorities: The client can indicate to the server, which of the streams are more important than the others, and need to be delivered first.

HTTP/2 is the newest member of the TCP/IP protocol suite built with security in mind, but still in his design there are many dualities, that can be used for building covert channels. These dualities come from the possibilities that some feature can be obtained in more than one way, or deployment of that feature is not mandatory. Interesting, designers of HTTP/2 learned how to deal with covert channels that use the header fields with random padding or reserved fields, by setting them to zero, or with covert channels that use PDU reordering, by making the order of the frames for header block to matter, but still they leave many ways how to build a covert channel. In this paper, we present several covert channels that can be created using the HTTP/2. The main HTTP/2 functio-

nalities and concepts are presented in Section 2. Section 3 presents different existing covert channels that can be created using HTTP/1.x, additionally with the explanation do they work/not work with the new version also. The main Section 4 describes nine groups of new covert channels in HTTP/2, that can be used regardless its transport carrier (TLS or clear TCP).

2. How HTTP/2 Works?

HTTP/2 retains the same semantics as HTTP/1.1 and does not make any changes in its basic concepts and functionality. It provides an optimized transportation mechanism for HTTP/1.1 requests/responses by changing the syntax how these semantics are conveyed. The main difference between the two protocols is that HTTP/2 is a binary protocol. HTTP/2 connection is a TCP connection between client and server which consists of three elements:

- Stream: A bidirectional flow that carries messages between the endpoints;
- Message: Logical HTTP message consisting of one or more frames;
- Frame: The smallest unit of communication that carries the specific type of data.

HTTP/2 connection can carry many independent bidirectional streams, where many streams can exchange messages in parallel. Each message is separated into smaller frames that are sent to the endpoint. Each frame transmitted via HTTP/2 is associated to a stream, and all streams are assigned to stream identifiers that are unique and cannot be used by other streams. There are 10 different types of frames consisting of fixed 9-octet header and variable-length payload which depends on the frame type. Each frame consists from a header and a payload. Each frame header further contains the following fields: 24-bit Length (of frame payload), 8-bit Type (of frame), 8-bit Flags, 1-bit reserved field (R) and 31-bit Stream Identifier. This is presented with connection lines between the header and its components on Figure 1.

HTTP/2 compresses header metadata in order of reducing overhead and improves performance using a new compressor HPACK. Types of frames that are used in HTTP/2 are: DATA, HEADERS, PRIORITY, RST STREAM, SETTINGS, PUSH PROMISE, PING, GOAWAY, WINDOWS UPDATE and CONTINUATION. RST STREAM and GOAWAY frames contain error codes that are used for indication of errors relating to any particular stream or for the entire connection. The order in which frames are sent from one stream is of great importance, because the recipient processes within the order in which they are received.

HTTP/2 connection is initiated by the client by first sending a request to the server to determine whether the server supports HTTP/2. This process is different for “http” and “https” URIs, where identification of protocols is made with different identifiers. String “h2” is used to identify use of HTTP/2 over TLS and string “h2c” for HTTP/2 over clear TCP. When client wants to use regular, non-encrypted channel, it must use an HTTP Upgrade mechanism to negotiate the protocol. The client first sends HTTP/1.1 request that contains Upgrade header field with the “h2c” token and a HTTP2-Setting header field (Figure 2).

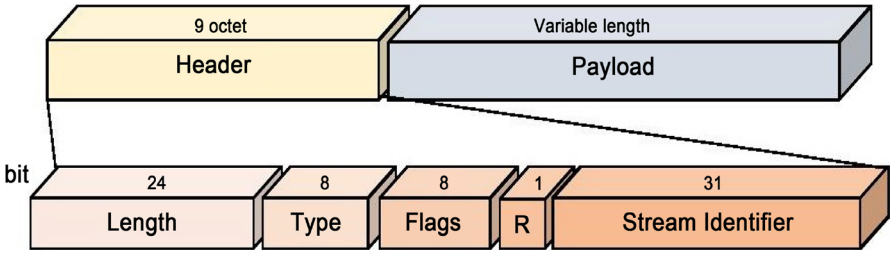

Figure 1. Frame layout.

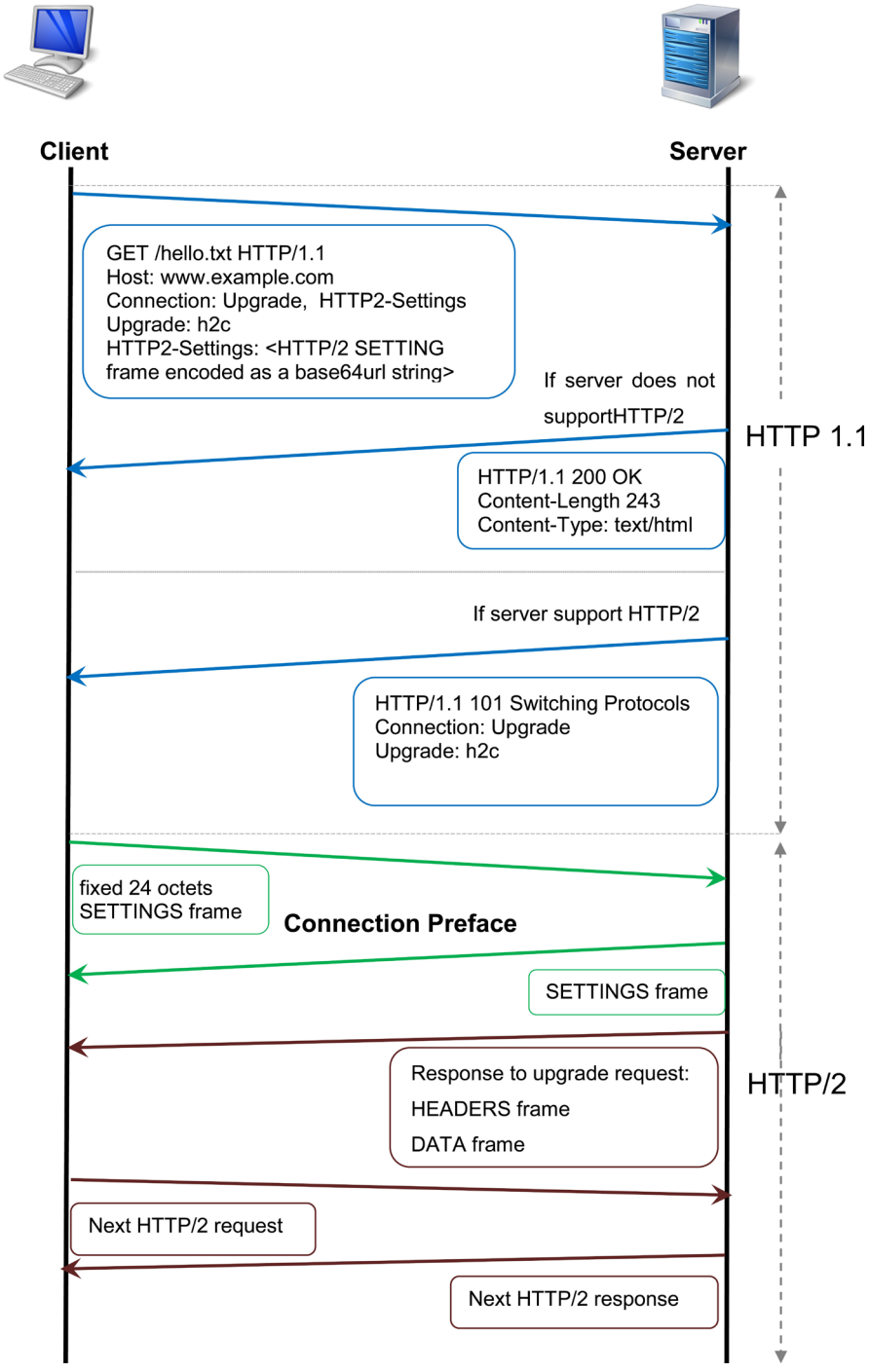

Figure 2. Upgrading from HTTP/1.1 to HTTP/2.

When the server does not support HTTP/2, it responds with an HTTP 1.1 response where the Upgrade header field is absent. On the contrary, by sending a 101 Switching Protocols response, the server confirms the upgrade and begins sending HTTP/2 frames. As a final confirmation of the protocol, the client and the server must establish the settings of the connection by sending different connection preface. The client connection preface starts with fix 24-octets sequence, followed by a potentially empty SETTINGS frame, and the server answers with connection preface which consists of a potentially empty SETTINGS frame. After establishment of the connection, the client and the server exchange frames of any type. By sending GOAWAY frame the endpoints gracefully close the connection and after receiving all frames from previously establish streams, data are no longer sent. For closing any particular stream, either endpoint can send: RST STREAM frame or frame that contains END STREAM flag.

3. Covert Channels in Previous Versions of the HTTP/2

One interesting feature of HTTP in the protocol description is the absence of limits on the sizes of the URI string, HTTP header or HTTP message body. Usually different limitations are introduced in different implementations. For example, Apache servers accept HTTP headers with size up to 8 KB, and IIS up to 8 KB or 16 KB depending on the version.

Many of the existing steganographic methods for HTTP/1.x are suppressed in HTTP/2, by design. For example, the methods presented in [6] [7] use the fact that HTTP/1.x treats any amount of consequent linear white space characters (optional line feed [CLRF], spaces [SP] and tabs [HT]) present in the header, in the same way as a single space character (e.g., [HT] can be a binary one and [SP] can be a binary zero). This is suppresed in HTTP/2 by requirement that requests and responses with invalid header names and with characters not permitted in a header field values, must be treated as malformed. Intermediary nodes must not forward a malformed request or response. Also, because header names are case-insensitive in the HTTP/1.x, one can use different capitalization for the header field values for covert channel [8]. This cannot be done in HTTP/2, because the header fields names must be converted to lowercase prior to their encoding in HTTP/2. Other three methods presented by Dyatlov and Castro [8] that use HTTP header fields reordering, their presence/absence when is possible (e.g., Accept-Encoding header field), and HTTP message body can be extended to HTTP/2 also.

Alman [9] showed that due to a weakness in the CONNECT method in the HTTP protocol, arbitrary connection can be made through an HTTP proxy server. These HTTP tunnels are not restricted only to the ports 80 and 443, but they are capable of passing any outbound traffic on any TCP port as long as the client warps the appropriate HTTP CONNECT header around the data stream. There are many tools for tunneling different protocols over HTTP, like Corkscrew [10], which tunnels SSH over HTTP proxy. These are examples of cross-protocol attacks, when an attacker causes a client to submit a request in

one protocol to a server that understands a different protocol, and this request to be valid in the second protocol also. The clear text version of HTTP/2 does not offer sufficient protection against these kinds of attacks, but in RFC 7540 is stated: "Completing a TLS handshake with an ALPN identifier for HTTP/2 can be considered sufficient protection against cross-protocol attacks".

Bauer [11] suggests a protocol "Muted Posthorn" that allows to create an anonymous overlay network by exploiting the web browsing activities of regular users. The protocol uses five HTTP/HTML mechanisms: redirects, cookies, Referer headers, HTML elements and Active contents.

Van Horenbeeck [12] implemented a tool Wondjina that creates a bidirectional covert channel using the HTTP ETag and If-None-Match header fields, which allows a client to verify whether its local cached copy is still current. When a specific document is served, the web server is allowed to include an ETag header field that contains a string which describes the page, without a specification how it should look like. Upon first retrieval, the client caches both the page and its ETag. If-None-Match is primarily used in conditional GET requests to enable efficient updates of cached information with a minimum amount of transaction overhead. When a client desires to update one or more stored responses that have entity-tags, the client should generate an If-None-Match header field containing a list of currently cached ETags, when making a GET request. The author suggests also a Content-MD5 header field to be used for sending 128 bits of secret data per HTTP message in one way. But this header field has been removed from the protocol specification from 2014 (RFC 7231).

Duncan and Martina [13] suggest modulating the least significant bits of the date-based fields such as Date and Last-Modified in HTTP response and use of Content-Location header field, which is designed to provide an alternative URL for the resource currently being accessed. Eßer and Freiling [14] suggest covert timing channel using HTTP, in which a web server sends covert data to a client by delaying a response (binary 1) or responding immediately (binary 0).

Infranet [3] is a framework which uses covert channels in HTTP to circumvent censorship in the Internet in some countries. Infranet's web servers receive covert requests for censured web pages encoded as a sequence of HTTP requests to harmless web pages and return their content hidden inside harmless images using steganography.

Another covert channel for HTTP 1.1 and up, given by Graniszewski, *et al* [15], uses Trailer field in the HTTP header for hiding data. The Trailer response header field allows the sender to include additional fields at the end of chunked messages in order to supply metadata that might be dynamically generated while the message body is sent, such as a message integrity check, digital signature, or post-processing status.

4. Covert Channels in the HTTP/2

There are several ways how one can create new covert channels in HTTP/2. For this purpose, usually we use a protocol feature that has dual nature, *i.e.*, the same

feature can be obtained in more than one way, the feature is not mandatory, there exist a random value field, or there is no strict rule how to obtain new values for some fields.

4.1. Covert Channel Using Padding

Three frames in HTTP/2, DATA, HEADERS and PUSH PROMISE frames, use padding as a security feature to obscure the size of messages. It is provided to mitigate specific attacks within HTTP, like BREACH, where compressed content includes both attacker-controlled plaintext and secret data. Another way to mitigate these attacks is by disabling or limiting the compression. Padding octets must be set to zero when used, to prevent other attacks. When padding is used, the third flag, PADDED (0×8), is set to 1, and at the beginning of Frame Payload there is a 8-bit field Pad Length containing the length of the frame padding (with position at the end of Frame Payload) in units of octets. Pad Length and Padding fields are present only if PADDED flag is set to 1. When no padding is used, there are two representations with the same effect:

- PADDED flag set to 0, and
- PADDED flag set to 1, together with Pad Length field set to 0.

These two representations can be used as binary zero and one (Figure 3). In the RFC 7540 one can find that: "Intermediaries SHOULD retain padding for DATA frames, but MAY drop padding for HEADERS and PUSH PROMISE frames. A valid reason for an intermediary to change the amount of padding of frames is to improve the protections that padding provides". So, for DATA frames, no intermediaries will change padding, and this can be used as bidirectional one-bit covert channel per DATA frame between client and server.

4.2. Covert Channel Using Stream Identifiers

A stream identifier is presented by an unsigned 31-bit integer. The value 0×0 is reserved for connection control messages, and the value 0×1 is reserved for HTTP/1.1 request, prior to upgrading to HTTP/2. Odd-numbered stream identifiers are used for streams initiated by a client, and even numbered stream identifiers are used for streams initiated by the server. Any new stream must have a stream identifier greater than the stream identifiers of all opened or reserved

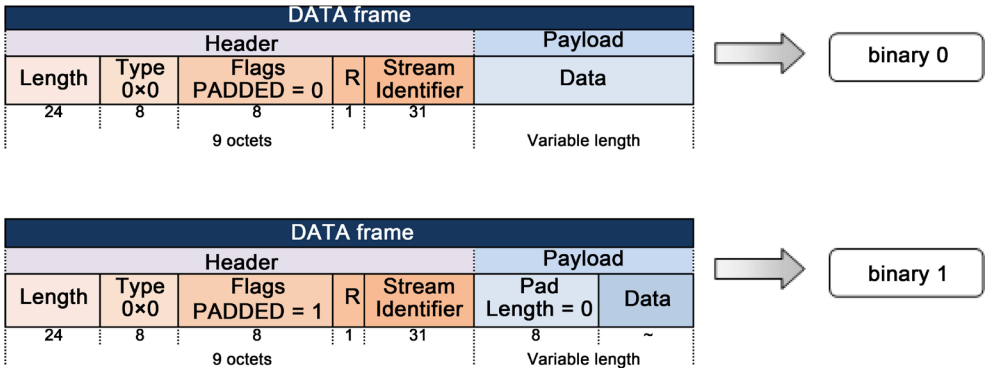

Figure 3. Representation of binary 0 and 1 with covert channel using padding.

streams by the endpoint. Stream identifiers cannot be reused, so, they can be exhausted by a long-lived connection, which results in establishing of a new connection. Streams can be in one of seven different states: "idle", "reserved (local)", "reserved (remote)", "open", "half closed (remote)", "half closed (local)" and "closed".

Let MAX CSI be the greatest used stream identifier for streams initiated by the client in a given moment, and let MAX SSI be the greatest used stream identifier for streams initiated by the server at the same moment. One bi-directional covert channel between the client and the server can be created in the following way (Figure 4):

- If the client wants to send binary 1 to the server, it initiates a new stream with stream identifier MAX CSI + 2, and for binary 0, with stream identifier MAX CSI + 4
- If the server wants to send binary 1 to the client, it initiates a new stream with stream identifier MAX SSI + 2, and for binary 0, with stream identifier MAX SSI + 4.

In this way, for a long-lived connection, one side can transmit maximum between $2^{29} - 1$ (for all bits-binary 0) and $2^{30} - 1$ bits (for all bits-binary 1), without rising any anomaly. One limit on number of concurrently active streams can be introduced be either communication site using the SETTINGS MAX CONCURRENT STREAMS parameter within a SETTINGS frame. Initially, there is no limit to the value of this parameter, only a recommendation to be no smaller than 100. As active streams appear only streams in "open", "half closed (remote)", or "half closed (local)" states.

4.3. Covert Channel Using PING Frame

The PING frames can be sent from both sides, from the client and from the server, and they are associated only with stream identifier 0 × 0. They are used for determining whether an idle connection is still functional, and for measuring a minimal round trip time from the sender. The PING frame without ACK flag, must be acknowledged by sending a PING frame as a response with ACK bit set,

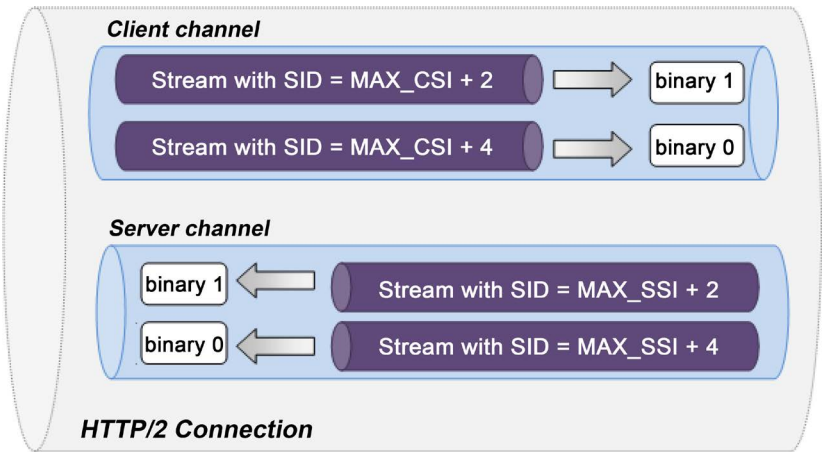

Figure 4. Representation of binary 0 and 1 with covert channel using stream identifiers.

with identical payload, and with higher priority. The payload of the PING frame can be any 64-bit number. There is no restriction, so one can create a covert channel by sending 64 bits per PING frame.

4.4. Covert Channels Using Stream Priorities and Dependencies

HTTP/2 use streams with or without assigning priorities, making them to be or not to be dependent on the competition of other streams. When there is a limited capacity for sending, the sender will make a selection of a stream for transmitting frame, based on priorities. Additionally, on each dependency is assigned a relative weight between 1 and 256. If several streams depend on the same stream, the weight will be used to determine the relative proportion of available resources that are assigned to them. All streams are initially non- exclusively dependent on the stream 0 × 0, and pushed streams initially depend on their associated stream. Default weight in both cases is 16.

Opening a stream is done by one HEADERS frame, which additionally can assign a priority to the stream. After that, the priority of a given stream can be changed, at any time, by PRIORITY frame.

For the new covert channels, other details about prioritisation and dependencies are not important. Only important is that the prioritization process is only advisory, and not mandatory, for the other communication endpoint - there is no guarantee that it will be fulfilled. So, we can use this for creating new covert channels.

If the PRIORITY (0 × 20) flag of the HEADERS frame is set to 1, then in the frame payload, the Exclusive Flag (E), Stream Dependency, and Weight fields are present. Making the new stream to be dependant on the previously created one (other previous streams also can come into account), one can use 9 bits per a HEADERS frame, 1 bit from the E flag, and 8 bits from the Weight field, for creating a bidirectional covert channel, without rising a suspicious situation. So, if one HTTP/2 connection has n streams in one direction over its lifetime, one can send 9n bits per HTTP/2 connection in one direction.

The PRIORITY frame has an advisory role and specifies the sender-advised priority of a stream. It can be sent in any stream state and any time, except between consecutive frames that comprise a single header block. Again, we can use 9 bits per a PRIORITY frame, 1 bit from the E flag, and 8 bits from the Weight field, for creating a covert channel. The number of PRIORITY frames sends during one HTTP/2 connection, should not be an anomaly by itself.

4.5. Covert Channels Using Different Number of Specific Kind of Frames

An HTTP request consists of:

- one HEADERS frame, followed by zero or more CONTINUATION frames, containing the header block;
- zero or more DATA frames containing the payload body;
- optionally, one HEADERS frame, followed by zero or more CONTINUATION

frames containing the trailer-part.

The structure of the HTTP response is similar, with additional zero or more HEADERS frames at the beginning, each followed by zero or more CONTINUATION frames, containing the header blocks of informational (1xx) HTTP responses. Additionally, when PUSH PROMISE frame is used, the header block starts inside this frame, and it can be followed by zero or more CONTINUATION frames. Each header block is transmitted as a contiguous sequence of frames, with no interleaved frames of any other type or from any other stream, and with preservation of frame order. An important property that can be deployed for covert channel creation, is the fact that DATA and CONTINUATION frames are variable-length sequences of octets and they can be sent in different number. So, we can create one covert channel using:

- odd number of DATA frames to be binary 1, and
- even number of DATA frames to be binary 0.

In this way, we can send at most one bit per stream in each direction, or at most n bits per HTTP/2 connection consisting of n streams, in each direction. Similarly, we can create another covert channel using:

- odd number of CONTINUATION frames to be binary 1, and
- even number of CONTINUATION frames to be binary 0.

In this way, we can send at most two bits to the server and at most k bits to the client per stream, if there are k different HEADERS and PUSH PROMISE frames with CONTINUATION frames after, per stream. Or, at most $2n$ bits to the server, and at most kn bits to the client, per HTTP/2 connection consisting of n streams.

4.6. Covert Channel Using Cookie Header Field

For better compression efficiency, the HTTP/2, differently from the rules in HTTP/1.x, can allow separation of cookie-pairs from one Cookie header field into several Cookie header fields, each with one or more cookie-pairs. So, because of this duality, we can create a one directional covert channel from a client (which supports HTTP/2) to the server by:

- only one present Cookie header field to be binary 1, and
- more than one present Cookie header fields to be binary 0.

Normally, to do this, there must be at least two cookie-pairs. In this way, we can send at most n bits to the server, per HTTP/2 connection consisting of n streams.

4.7. Covert Channel Using SETTINGS Frames

SETTINGS frames are used in connection preface phase for configuring different connection specific parameters by both sides, but also, they can be sent at any time during HTTP/2 connection. The values of parameters in the SETTINGS frame replace any existing values for those parameters, and they are acknowledged by empty SETTINGS frame with ACK bit set to 1. The payload of a SETTINGS frame consists of zero or more parameters, defined by unsigned

16-bit Identifier field, and unsigned 32-bit Value field. Parameters are processed in the order in which they appear in the payload. There are six defined parameters in the protocol specification:

- SETTINGS HEADER TABLE SIZE (0 × 1)-acceptable maximum size of the header compression table used to decode header blocks, in octets. The initial value is 4096 octets.
- SETTINGS ENABLE PUSH (0 × 2)-enabling/ disabling server push. The initial value is 1, which indicates that server push is allowed.
- SETTINGS MAX CONCURRENT STREAMS (0 × 3)-maximum number of concurrent streams that the sender will allow. Initially, there is no limit to this value.
- SETTINGS INITIAL WINDOW SIZE (0 × 4)-sender's initial window size, in octets, for stream-level flow control. The initial value is $2^{16} - 1$ octets.
- SETTINGS MAX FRAME SIZE (0 × 5)-size of the largest frame payload that the sender is willing to receive, in octets. The initial value is 2^{14} octets.
- SETTINGS MAX HEADER LIST SIZE (0 × 6)-maximum size of header list that the sender is prepared to accept, in octets. Initially, there is no limit to this value.

We can define a covert channel using different values of these parameters. We can exclude the SETTINGS ENABLE PUSH parameter because of his Boolean nature and influence on the connection. For each other parameter we can define that

- even Value field for a given parameter to be binary 0, and
- odd Value field for a given parameter to be binary 1.

Additionally, we can define an interval for changing these values, if this is necessary, for correct protocol functioning. So, we can send 5 bits per SETTINGS frame in one direction.

4.8. Covert Channel Using Flow Control

Flow control in HTTP/2 can be made on each individual stream or on the entire connection. This is made in hop-by-hop manner, not over the entire end-to-end path. Receiver, using credit based scheme, sends information to the sender about the amount of data that is prepared to receive, and the sender must respect these limits. The initial window size for new streams can be adjusted by including a value for SETTINGS INITIAL WINDOW SIZE in the SETTINGS frame in the connection preface. After that, the window size can be changed at any time by sending WINDOW UPDATE or SETTINGS frames. The connection flow-control window can only be changed using WINDOW UPDATE frames. The default value of stream and connection flow control window is 65,535 octets. Subject of flow control are only DATA frames.

The payload of WINDOW UPDATE frame contains one reserved bit field and Windows Size Increment field, which is an unsigned 31-bit integer (0 value is not allowed) that indicates the number of octets that the sender can transmit in addition of the existing flow-control window. There is no strictly define method

of how or when the endpoints can advertise the size of the frame, so this can be used for making new covert channel. Additionally, separate WINDOW UPDATE frames are sent for different streams, and there are no limits on the stream state for which these frames can be sent. One bi-directional covert channel between two neighboring hops can be done using the following:

- even value for Windows Size Increment field to be binary 0, and
- odd value for Windows Size Increment field to be binary 1.

In this way, if there are *k* streams in one HTTP/2 connection, one can send *k* bits in one direction, by sending a separate WINDOW UPDATE frame per each stream, at any time without raising any anomaly.

4.9. Covert Channels Using HPACK

HTTP/2 comes with a new compressor which eliminates the redundancy present in the header fields, HPACK. It threats header fields as an ordered collection of name-value pairs, considered as sequences of octets, with the possibility of their duplication. One header field can be encoded using static or dynamic tables into indexed values (references), or it can be represented as a literal value by specifying its name and value. The header field value is represented always literally. Additionally, literal values can be encoded directly or using static Huffman code. RFC 7541 describes only how a HPACK decoder is expected to operate. The HPACK decoder processes a header block sequentially to reconstruct the original header list.

String literal representation has three fields: one bit flag H, which indicates whether or not the Huffman encoding is used, a String Length field, which is the number of octets used to encode the string literal and a String Data field with encoded data of the string literal. So, we can create one covert channel using the following:

- string literal with no encoding (field H = 0) to be binary 0, and
- string literal with encoding (field H = 1) to be binary 1.

If there are *k* string literals in the header block, one can send *k* bits per header block. For each TCP stream, there can be at most two header blocks in the HTTP request (at the beginning and at the end), and there can be at most two header blocks in the HTTP response (at the beginning and at the end).

There are three different representations of the literal header field: with incremental indexing (starts with binary sequence 01), without indexing (starts with binary sequence 0000), and never indexed (starts with binary sequence 0001). A literal header field with incremental indexing representation adds a new entry into the dynamic table, if the field name is absent from the dynamic table, and a literal header field without indexing representation does not alter the dynamic table. A literal header field never indexed representation does not alter the dynamic table, but also all intermediaries must use the same representation for encoding this header field. We can create another covert channel using the following:

- literal header field with incremental indexing or without indexing represen-

tation to be binary 0, and

- literal header field never indexed representation to be binary 1.

If there are *k* literal header fields in the header block, one can send *k* bits per header block.

5. Conclusions

HTTP/2, like many other network protocols, is prone to hiding data in it. Additionally, it belongs to the group of network protocols that will be used a lot in the following years, and its traffic will not raise any suspicions. So, it is important to identify possible ways of hiding data in it, and try to mitigate them. This paper deals with the first part, leaving others to try to find a solution for mitigating presented covert channels.

Further, people involved in protocol standardization process can work on the possibility to eliminate these covert channels. For example, one best practice in protocol design should be elimination of features with dual nature, *i.e.* not to leave anything to be done in more than one way.

Implementation of these covert channels and their testing in lab or real environment is leaved as future work.

References

[1] Lubacz, J., Mazurczyk, W. and Szczypiorski, K. (2014) Principles and Overview of Network Steganography. *IEEE Communication Magazine*, **52**, 225-229. https://doi.org/10.1109/MCOM.2014.6815916

[2] Department of Defence (1985) Department of Defence Trusted Computer System Evaluation Criteria. Technical Report DoD 5200.28-STD. Supersedes CSC-STD-001-83. http://csrc.nist.gov/publications/history/dod85.pdf

[3] Feamster, N., Balazinska, M., Harfst, G., Balakrishnan, H. and Karger, D. (2002) Infranet: Circumventing Web Censorshipand Surveillance. *Proceedings of the* 11*th USENIX Security Symposium*, San Francisco, 8-12 August 2002, 247-262.

[4] Forte, D.V. (2005) SecSyslog: An Approach to Secure Logging Based on Covert Channels. *Proceedings of the FirstInternational Workshop of Systematic Approaches to Digital Forensic Engineering* (*SADFE* 2005), Taipei, 7-9 November 2005, 248-263. https://doi.org/10.1109/SADFE.2005.21

[5] Mazurczyk, W., Lubacz, J. and Szczypiorski, K. (2008) Hiding Data in VoIP. *Proceedings of the* 26*th Army Science Conference* (*ASC* 2008), Orlando, 1-4 December 2008.

[6] Kwecka, Z. (2006) Application Layer Covert Channel Analysis and Detection. Technical Report, Napier University Edinburgh. https://pdfs.semanticscholar.org/f740/ca7afcb75d9c90c50894396dcfc08f824a91.pdf

[7] Heilman, S., Williams, J. and Johnson, D. (2016) Covert Channel in HTTP User-Agents. *Proceedings of the* 11*th Annual Symposium on Information Assurance* (*ASIA* 16), Albany, 8-9 June 2016, 68-73.

[8] Dyatlov, A. and Castro, S. (2003) Exploitation of Data Streams Authorized by a Network Access Control System for ArbitraryData Transfers: Tunneling and Covert Channels over the HTTP Protocol. Gray-World, USA. http://gray-world.net/projects/papers/covert_paper.txt

[9] Alman, D. (2003) HTTP Tunnel Though Proxies. SANS Institude. https://www.sans.org/reading-room/whitepapers/covert/http-tunnels-proxies-1202

[10] Padgett, P. (2001) Corkscrew. https://www.mankier.com/1/corkscrew

[11] Bauer, M. (2003) New Covert Channels in HTTP: Adding Unwitting Web Browsers to Anonymity Sets. *Proceedings of the Workshop on Privacy Electronic Society* (*WPES* 2003), Washington DC, 30 October 2003, 72-78. https://doi.org/10.1145/1005140.1005152

[12] Van Horenbeeck, M. (2006) Deception on the Network: Thinking Differently about Covert Channels. *Proceedings of the Australian Information Warfare and Security Conference*, Perth, Western Australia, 4-5 December 2006, 174-184.

[13] Duncan, R. and Martina, J.E. (2010) Steganographic Message Broadcasting Using Web Protocols. *Proceedings of the Simposio Brasilerio de Seguranca* (*SBSeg* 2010), Fortaleza, Brasil, 11-15 October 2010, 61-70.

[14] Eßer, H.G. and Freiling, F.C. (2005) Kapazitätsmessung eines verdeckten Zeitkanals ber HTTP. Technical Report TR-2005-10, Universität Mannheim. https://ub-madoc.bib.uni-mannheim.de/1136/1/tr_2005_10.pdf

[15] Graniszewski, W., Krupski, J. and Szczypiorski, K. (2016) The Covert Channel over HTTP Protocol. *Proceedings of the SPIE* 10031, *Photonics Applications in Astronomy*, *Communications*, *Industry*, *and High-Energy Physics Experiments* 2016, 100314Z, 28 September 2016.