Abstract

This document defines the 'Upgrade+Hello' handshake for the WebSocket protocol.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 18, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
Table of Contents

1. Introduction ........................................... ancho
2. Terminology ........................................... ancho
3. Upgrade+Hello Handshake ............................... ancho
4. Changes to 1.2. Protocol overview ................... ancho
5. Changes to 1.3. Opening handshake ..................... ancho
6. Changes to 1.9 Subprotocols using the WebSocket protocol . ancho
7. Changes to 3.1. Parsing WebSocket URLs ............... ancho
8. Changes to 4.2. Base Framing Protocol ................. ancho
9. Changes to 4.3. Fragmentation ........................ ancho
10. Changes to 4.4. Control Frames ....................... ancho
11. Changes to 4.6. Examples ............................ ancho
12. Changes to 5.1. Client Requirements ................. ancho
13. Changes to 5.2.1. Reading the client’s opening handshake . ancho
14. Security Considerations ............................. ancho
15. IANA Considerations ................................. ancho
16. Acknowledgments ..................................... ancho
17. References ........................................... ancho
    17.1. Normative References ........................... ancho
    17.2. Informative References ......................... ancho
1. Introduction

As of version 3 of the websocket (WS) protocol
[I-D.ietf-hybi-thewebsocketprotocol], the handshake used to
transition from HTTP to Websocket protocol is deficient with respect
to HTTP compatibility and security. Accordingly, much discussion has
ensued to try to define the replacement handshake for adoption by the
WG.

The handshake defined in this document maintains compatibility with
HTTP in accordance with [I-D.ietf-hybi-websocket-requirements] while
protecting against currently known vulnerabilities. It leverages the
‘GET+Upgrade’ approach in the current handshake, in addition to
several other mechanisms, as detailed below.

An initial version of this handshake has been previously shared with
the working group
(http://www.ietf.org/mail-archive/web/hybi/current/msg04534.html).

2. Terminology

This document uses HyBi-related terms as defined in
[I-D.ietf-hybi-websocket-requirements].

3. Upgrade+Hello Handshake

The Upgrade+Hello handshake implements the following mechanisms:

a. Use Websocket Hello from the server and a Websocket Hello from
the client, using Websocket frames to transport hashed nonces
(rather than unframed bytes). This fixes the issue with
intermediaries not forwarding the unframed bytes on the wire and
makes the handshake comply with the requirement to be HTTP/1.1
compliant up through the 101 response from the server.

b. Include a server nonce in the server-sent Hello, that the client
must hash and return in its Hello.

c. Use a Hello frame type instead of ping/pong. This does not
require an extra round trip. It just ensures that the first
frame in each direction is something that is entirely not HTTP
and uses the Websocket binary framing instead.

d. Replace the char/space encoding of the client nonce with simple
hex encoding.

e. Define tight restrictions on the punctuation that can be sent in
ws URLs and subprotocols (e.g., prohibit use of ‘:’ ) so that
they cannot be used to inject HTTP headers. While not as robust as encrypting the handshakes, these restrictions will provide substantial protection against user provided data being used as part of an attack.

f. Invert the framing MORE bit to be a FIN bit, so that WS control frames will start with a non-ascii character. This mitigates the attack vector affecting some intermediaries (transparent proxies, load balancers, etc) that have been known to continue parsing the traffic (incorrectly assuming it is still HTTP) even after the handshake.

g. TODO: Change use of MD5 to SHA1. Note the use in a "Challenge-Response" manner is not considered susceptible to recent attacks against these hash families per RFC4270, but still makes sense to migrate away from MD5 if nothing else for easier deployment.

The above summarizes the changes. The following sections show the text changes that implement the above mechanisms, in traditional "OLD" and "NEW" IETF notation.

4. Changes to 1.2. Protocol overview

OLD:

GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 4 @1 46546xW%01 1 5
Origin: http://example.com

The handshake from the server looks as follows:

HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Location: ws://example.com/demo
Sec-WebSocket-Protocol: sample

8jKS’y:G*Co,Wxa-
NEW:

GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Nonce: A23F2BCA452DDE01
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Origin: http://example.com
Sec-WebSocket-Draft: 3

The handshake from the server looks as follows:

HTTP/1.1 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Location: ws://example.com/demo
Sec-WebSocket-Protocol: sample

OLD:

Finally, after the last field, the client sends 10 bytes starting
with 0x0D 0x0A and followed by 8 random bytes, part of a challenge,
and the server sends 18 bytes starting with 0x0D 0x0A and followed by
16 bytes consisting of a challenge response. The details of this
challenge and other parts of the handshake are described in the next
section.

Once the client and server have both sent their handshakes, and if
the handshake was successful, then the data transfer part starts.
This is a two-way communication channel where each side can,
independently from the other, send data at will.

NEW:

Once the client and server have both sent their handshakes, and if
the handshake was successful, then the data transfer part starts.
This is a two-way communication channel where each side can,
independently from the other, send data at will.

The first frame sent in both directions must be a Hello frame
containing digests to prove the handshakes have been processed.
5. Changes to 1.3. Opening handshake

OLD:

The opening handshake is intended to be compatible with HTTP-based server-side software, so that a single port can be used by both HTTP clients talking to that server and WebSocket clients talking to that server. To this end, the WebSocket client’s handshake appears to HTTP servers to be a regular GET request with an Upgrade offer:

```
GET / HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
```

Fields in the handshake are sent by the client in a random order; the order is not meaningful.

NEW:

The opening handshake is intended to be compatible with HTTP-based server-side software, so that a single port can be used by both HTTP clients talking to that server and WebSocket clients talking to that server. To this end, the WebSocket client’s handshake appears to HTTP servers to be a regular HTTP request with an Upgrade offer:

```
GET / HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
```

Any valid URI may be passed, except that the client must not send a handshake with a URI path that contain the character ‘:’ or any control or whitespace characters in explicit or encoded form. This limits the possibility of using a websocket client to attack a vulnerable non WebSocket servers.

Fields in the handshake are sent by the client in an unspecified order; the order is not meaningful.

OLD:
To prove that the handshake was received, the server has to take three pieces of information and combine them to form a response. The first two pieces of information come from the |Sec-WebSocket-Key1| and |Sec-WebSocket-Key2| fields in the client handshake:

```
Sec-WebSocket-Key1: 18x 6]8vM;54 *(5: { U1}8 z [ 8
Sec-WebSocket-Key2: l_ tx7X d < nw 334J702) 7j0)' 0
```

For each of these fields, the server has to take the digits from the value to obtain a number (in this case 1868545188 and 1733470270 respectively), then divide that number by the number of spaces characters in the value (in this case 12 and 10) to obtain a 32-bit number (155712099 and 173347027). These two resulting numbers are then used in the server handshake, as described below.

The counting of spaces is intended to make it impossible to smuggle this field into the resource name; making this even harder is the presence of _two_ such fields, and the use of a newline as the only reliable indicator that the end of the key has been reached. The use of random characters interspersed with the spaces and the numbers ensures that the implementor actually looks for spaces and newlines, instead of being treating any character like a space, which would make it again easy to smuggle the fields into the path and trick the server. Finally, _dividing_ by this number of spaces is intended to make sure that even the most naive of implementations will check for spaces, since if the server does not verify that there are some spaces, the server will try to divide by zero, which is usually fatal (a correct handshake will always have at least one space).

The third piece of information is given after the fields, in the last eight bytes of the handshake, expressed here as they would be seen if interpreted as UTF-8:

```
Tm[K T2u
```

The concatenation of the number obtained from processing the |Sec-WebSocket-Key1| field, expressed as a big-endian 32 bit number, the number obtained from processing the |Sec-WebSocket-Key2| field, again expressed as a big-endian 32 bit number, and finally the eight bytes at the end of the handshake, form a 128 bit string whose MD5 sum is then used by the server to prove that it read the handshake.

NEW:

To prove that the handshake was received, the server has to initiate websocket communication with a Hello frame containing the MD5 hash of a random 64 bit client nonce generated by the client and sent as a
hex encoded string in the Sec-WebSocket-Nonce field of the client handshake:

Sec-WebSocket-Nonce: A23F2B5A452DDE01

The server has to take 8 octets of the client nonce, append the octets of the ASCII string "WebSocket" and then append 8 octets of a randomly generated server nonce (eg 15F0D2278BCD457F). The server nonce and the 16 octet MD5 hash (eg Ee7eBe0c369cBe255a4451Fc58D7Cc4f) of the combined octets must be sent in a Hello frame as the first websocket frame from the server to the client:

8F18 15F0D2278BCD457F Ee7eBe0c369cBe255a4451Fc58D7Cc4f

OLD:
After the fields, the server sends the aforementioned MD5 sum, a 16 byte (128 bit) value, shown here as if interpreted as UTF-8:

fQJ,fN/4F4!~K~MH

This value depends on what the client sends, as described above. If it doesn’t match what the client is expecting, the client would disconnect.

Having part of the handshake appear after the fields ensures that both the server and the client verify that the connection is not being interrupted by an HTTP intermediary such as a man-in-the-middle cache or proxy.

NEW:
After the fields, the server sends the aforementioned Hello frame containing the 8 octets of the server nonce followed by the 16 octets of the MD5 digest.

After receiving the server handshake, the client must wait for the following Hello frame and verify that the contents. The digest value depends on both the client and server nonces, as described above. If it doesn’t match what the client is expecting, the client must disconnect. Only after verifying the contents of the Hello frame may the client application be notified of an open connection.

The first websocket frame sent by the client to the server must be a Hello frame containing the 128 bit MD5 digest of the 8 octets of the
server nonce followed by the octets of the ASCII string "WebSocket"
followed by the 8 octets of the client nonce:

8F10 33122035D11258Fd7c764314580e539c

6. Changes to 1.9  Subprotocols using the WebSocket protocol

OLD:

The client can request that the server use a specific subprotocol by
including the |Sec-Websocket-Protocol| field in its handshake. If it
is specified, the server needs to include the same field and one of
the selected subprotocol values in its response for the connection to
be established.

These subprotocol names do not need to be registered, but if a

NEW:

The client can request that the server use a specific subprotocol by
including the |Sec-Websocket-Protocol| field in its handshake. If it
is specified, the server needs to include the same field and one of
the selected subprotocol values in its response for the connection to
be established.

Subprotocol names are restricted to use the ASCII characters A-Z,
a-z, 0-9, '/', '_', and '-'.

These subprotocol names do not need to be registered, but if a

7. Changes to 3.1. Parsing WebSocket URLs

OLD:

11. If /url/ has a <query> component, then append a single U+003F
    QUESTION MARK character (?) to /resource name/, followed by the
    value of the <query> component.

12. Return /host/, /port/, /resource name/, and /secure/.

NEW:
11. If /resource name/ contains the character ‘:’ or any white
space characters or any characters with codes <= U+0020,
then the URL is invalid.

12. If /url/ has a <query> component, then append a single U+003F
QUESTION MARK character (?) to /resource name/, followed by the
value of the <query> component.

13. Return /host/, /port/, /resource name/, and /secure/.

8. Changes to 4.2. Base Framing Protocol

OLD:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

+-------+------------------+
| M | R | R | R | opcode | R | Payload len |
| O | S | S | S | (4) | S | (7) |
| R | V | V | V | (16/63) | V | (if payload len==126/127) |
| E | 1 | 2 | 3 | | 4 | |

+--------------------------+------------------+
| | Extended payload length |
| | (16/63) |

+--------------------------+------------------+
| | Extended payload length continued, if payload len == 127 |
| | +----------------------------------------------+
| | | Extension data |
| | | |
| | +----------------------------------------------+
| | | Application data |
| | | |
| | +----------------------------------------------+

MORE: 1 bit

Indicates more fragments follow in the current message

NEW:
FIN: 1 bit

Indicates no more fragments follow in the current message

OLD:

ws-frame = frame-more
frame-rsv1
frame-rsv2
frame-rsv3
frame-opcode
frame-length
frame-extension
application-data;

frame-more = %x0 ; final frame of message
/ %x1 ; more frames of this message follow

frame-rsv1 = %x0 ; 1 bit, must be 0
frame-rsv2 = %x0 ; 1 bit, must be 0
frame-rsv3 = %x0 ; 1 bit, must be 0
frame-opcode = %x0 ; continuation frame
/ %x1 ; connection close
/ %x2 ; ping
/ %x3 ; pong
/ %x4 ; text frame
/ %x5 ; binary frame
NEW:

ws-frame = frame-fin
  frame-rsv1
  frame-rsv2
  frame-rsv3
  frame-opcode
  frame-rsv4
  frame-length
  frame-extension
  application-data;

frame-fin = %x1 ; final frame of message
  / %x0 ; more frames of this message follow

frame-rsv1 = %x0 ; 1 bit, must be 0
frame-rsv2 = %x0 ; 1 bit, must be 0
frame-rsv3 = %x0 ; 1 bit, must be 0

frame-opcode = %x0 ; continuation frame
  / %x1 ; connection close
  / %x2 ; ping
  / %x3 ; pong
  / %x4 ; text frame
  / %x5 ; binary frame
  / %x6-E ; reserved
  / %xF ; Hello frame

9. Changes to 4.3. Fragmentation

OLD:

  o An unfragmented message consists of a single frame with the MORE
    bit clear and an opcode other than 0.

  o A fragmented message consists of a single frame with the MORE bit
    set and an opcode other than 0, followed by zero or more frames
    with the MORE bit set and the opcode set to 0, and terminated by a
    single frame with the MORE bit clear and an opcode of 0. Its
    content is the concatenation of the application data from each of
    those frames in order.

NEW:
An unfragmented message consists of a single frame with the FIN bit set and an opcode other than 0.

A fragmented message consists of a single frame with the FIN bit clear and an opcode other than 0, followed by zero or more frames with the FIN bit clear and the opcode set to 0, and terminated by a single frame with the FIN bit set and an opcode of 0. Its content is the concatenation of the application data from each of those frames in order.

10. Changes to 4.4. Control Frames

OLD:

Control frames have opcodes of 0x01 (Close), 0x02 (Ping), or 0x03 (Pong). Control frames are used to communicate state about the websocket.

NEW:

Control frames have opcodes of 0x01 (Close), 0x02 (Ping), 0x03 (Pong) or 0x0f (Hello). Control frames are used to communicate state about the websocket.

Add a NEW section 4.4.4. after 4.4.3 for the new "Hello" command:

4.4.4. Hello

The Hello message contains an opcode of 0x0F and must be the first frame sent on any WebSocket connection.

A Hello message sent from the server to the client must contain 24 octets of data, comprising of the 8 octets of the server nonce and the 16 octets of the MD5 digest of the 41 octets of the client nonce, plus the ASCII string "WebSocket", plus the server nonce.

A Hello message sent from the client to the server must contain 16 octets of data, comprising the MD5 digest of the 41 octets of the server nonce, plus the ASCII string "WebSocket", plus the client nonce.

11. Changes to 4.6. Examples

OLD:
o A single-frame text message
  * 0x04 0x05 "Hello"

o A fragmented text message
  * 0x84 0x03 "Hel"
  * 0x00 0x02 "lo"

o Ping request and response
  * 0x02 0x05 "Hello"
  * 0x03 0x05 "Hello"

o 256 bytes binary message in a single frame
  * 0x05 0x7E 0x0100 [256 bytes of binary data]

o 64KiB binary message in a single frame
  * 0x05 0x7F 0x0000000000010000 [65536 bytes of binary data]

NEW:

o A single-frame text message
  * 0x84 0x05 "Hello"

o A fragmented text message
  * 0x04 0x03 "Hel"
  * 0x80 0x02 "lo"

o Ping request and response
  * 0x82 0x05 "Hello"
  * 0x83 0x05 "Hello"

o 256 bytes binary message in a single frame
  * 0x85 0x7E 0x0100 [256 bytes of binary data]

o 64KiB binary message in a single frame
12. Changes to 5.1. Client Requirements

OLD:

15. Add the string "Sec-WebSocket-Draft: 2" to /fields/.

NEW:

15. Add the string "Sec-WebSocket-Draft: 3" to /fields/.

Note: In the following changes, renumbering is not shown. That should be taken care of by the diff patch to the specification source file.

OLD:

19. Let /spaces_1/ be a random integer from 1 to 12 inclusive.
    Let /spaces_2/ be a random integer from 1 to 12 inclusive.

    EXAMPLE: For example, 5 and 9.

20. Let /max_1/ be the largest integer not greater than 4,294,967,295 divided by /spaces_1/.
    Let /max_2/ be the largest integer not greater than 4,294,967,295 divided by /spaces_2/.

    EXAMPLE: Continuing the example, 858,993,459 and 477,218,588.

21. Let /number_1/ be a random integer from 0 to /max_1/ inclusive.
    Let /number_2/ be a random integer from 0 to /max_2/ inclusive.

    EXAMPLE: For example, 777,007,543 and 114,997,259.

22. Let /product_1/ be the result of multiplying /number_1/ and /spaces_1/ together.
    Let /product_2/ be the result of multiplying /number_2/ and /spaces_2/ together.

    EXAMPLE: Continuing the example, 3,885,037,715 and 1,034,975,331.

23. Let /key_1/ be a string consisting of /product_1/, expressed in
base ten using the numerals in the range U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9).

Let /key_2/ be a string consisting of /product_2/, expressed in base ten using the numerals in the range U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9).

EXAMPLE: Continuing the example, "3885037715" and "1034975331".

24. Insert between one and twelve random characters from the ranges U+0021 to U+002F and U+003A to U+007E into /key_1/ at random positions.

Insert between one and twelve random characters from the ranges U+0021 to U+002F and U+003A to U+007E into /key_2/ at random positions.

NOTE: This corresponds to random printable ASCII characters other than the digits and the U+0020 SPACE character.

EXAMPLE: Continuing the example, this could lead to "P388O503D& u17{K%gX(%715" and "1N?kUT0or3o4I97N5-S3O31".

25. Insert /spaces_1/ U+0020 SPACE characters into /key_1/ at random positions other than the start or end of the string.

Insert /spaces_2/ U+0020 SPACE characters into /key_2/ at random positions other than the start or end of the string.

EXAMPLE: Continuing the example, this could lead to "P388 O503D& u17 (K%gX ( %7 15" and "1 N ?k UT0or 3o 4 I97N 5-S3O 31".

26. Add the string consisting of the concatenation of the string "Sec-WebSocket-Key1:", a U+0020 SPACE character, and the /key_1/ value, to /fields/.

Add the string consisting of the concatenation of the string "Sec-WebSocket-Key2:", a U+0020 SPACE character, and the /key_2/ value, to /fields/.

NEW:
19. Let /client nonce/ be a random 64 bit integer.

   EXAMPLE expressed as a hexadecimal number:

   A23F2BCA452DDE01

26. Add the string consisting of the concatenation of the string
   "Sec-WebSocket-Nonce: ", a U+0020 SPACE character, and the
   /client nonce/ value expressed as a base 16 using numerals
   in the range U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9)
   and letters in the range U+0041 LATIN CAPITAL LETTER A to
   U+0046 LATIN CAPITAL LETTER F, to /fields/.

Delete these OLD items:

29. Let /key_3/ be a string consisting of eight random bytes (or
   equivalently, a random 64 bit unsigned integer encoded in big-
   endian order).

   EXAMPLE: For example, 0x47 0x30 0x22 0x2D 0x5A 0x3F 0x47 0x58.

30. Send /key_3/ to the server.

OLD:

45. Let /challenge/ be the concatenation of /number_1/, expressed as
   a big-endian 32 bit integer, /number_2/, expressed as a big-
   endian 32 bit integer, and the eight bytes of /key_3/ in the
   order they were sent on the wire.

   EXAMPLE: Using the examples given earlier, this leads to the 16
   bytes 0x2E 0x50 0x31 0xB7 0x06 0xDA 0xB8 0x0B 0x47 0x30 0x22
   0x2D 0x5A 0x3F 0x47 0x58.

46. Let /expected/ be the MD5 fingerprint of /challenge/ as a big-
   endian 128 bit string. [RFC1321]

   EXAMPLE: Using the examples given earlier, this leads to the 16
   bytes 0x30 0x73 0x74 0x33 0x52 0x6C 0x71 0x32 0x05A 0x55 0x5E 0x77 0x65 0x75. In UTF-8, these bytes correspond to
   the string "0st3Rl&q-2ZU\"weu".

47. Read sixteen bytes from the server. Let /reply/ be those bytes.

If the connection closes before these bytes are received, then
fail the WebSocket connection and abort these steps.

48. If /reply/ does not exactly equal /expected/, then fail the
WebSocket connection and abort these steps.

NEW:

47. Read twenty six bytes from the server. Let /server hello/ be those bytes.

   If the connection closes before these bytes are received, then fail the WebSocket connection and abort these steps.

48. If the first octet of /server hello/ is not 0x8f then fail the WebSocket connection and abort these steps.

   If the second octet of /server hello/ is not 0x18 then fail the WebSocket connection and abort these steps.

49. Let /server nonce/ be the 8 octets starting from the third octet of /server hello/

   EXAMPLE expressed as a hexadecimal number:
   
   15F0D2278BCD457F

   Let /server digest/ be the 16 octets starting from the eleventh octet of /server hello/

50. Let /server expected/ be the 16 octets of MD5 digest of the concatenation of /client nonce/, the ASCII string "WebSocket" and /server nonce/

   EXAMPLE expressed as a hexadecimal number:
   
   Ee7eBe0c369cBf255a4451Fc58D7Cc4f

51. If /server digest/ does not equal /server expected/ then fail the WebSocket connection and abort these steps.

52. Let /client digest/ be the 16 octets of MD5 digest of the concatenation of /server nonce/, the ASCII string "WebSocket" and /client nonce/.

53. Send the octets 0x8f, 0x10 and /client digest/

13. Changes to 5.2.1. Reading the client’s opening handshake

OLD:
1. The three-character UTF-8 string "GET".

2. A UTF-8-encoded U+0020 SPACE character (0x20 byte).

3. A string consisting of all the bytes up to the next UTF-8-encoded U+0020 SPACE character (0x20 byte). The result of decoding this string as a UTF-8 string is the name of the resource requested by the server. If the server only supports one resource, then this can safely be ignored; the client verifies that the right

NEW:

1. The three-character UTF-8 string "GET".

2. A UTF-8-encoded U+0020 SPACE character (0x20 byte).

3. A string consisting of all the bytes up to the next UTF-8-encoded U+0020 SPACE character (0x20 byte). The result of decoding this string as a UTF-8 string is the name of the resource requested by the server and must be a valid resource name according to Section 3.3. If the server only supports one resource, then this can safely be ignored; the client verifies that the right

OLD:

|Sec-WebSocket-Key1|

|Sec-WebSocket-Key2|

NEW:

|Sec-WebSocket-Nonce|

OLD:

/key_1/

The value of the "Sec-WebSocket-Key1" field in the client’s handshake.
The value of the "Sec-WebSocket-Key2" field in the client’s handshake.

The eight random bytes sent after the first 0x0D 0x0A 0x0D 0x0A sequence in the client’s handshake.

3. Let /location/ be the string that results from constructing a WebSocket URL from /host/, /port/, /resource name/, and /secure flag/.

4. Let /key-number_1/ be the digits (characters in the range U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9)) in /key_1/, interpreted as a base ten integer, ignoring all other characters in /key_1/.

Let /key-number_2/ be the digits (characters in the range U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9)) in /key_2/, interpreted as a base ten integer, ignoring all other characters in /key_2/.

EXAMPLE: For example, assume that the client handshake was:

GET / HTTP/1.1
Connection: Upgrade
Host: example.com
Upgrade: WebSocket
Sec-WebSocket-Key1: 3e6b263  4 17 80
Origin: http://example.com
Sec-WebSocket-Key2: 17  9 G'ZD9   2 2b 7X 3 /r90
WjN]|M(6

The /key-number_1/ would be the number 3,626,341,780, and the /key-number_2/ would be the number 1,799,227,390.

In this example, incidentally, /key_3/ is "WjN]|M(6", or 0x57 0x6A 0x4E 0x7D 0x7C 0x4D 0x28 0x36.

5. Let /spaces_1/ be the number of U+0020 SPACE characters in /key_1/.

Let /spaces_2/ be the number of U+0020 SPACE characters in /key_2/.

If either /spaces_1/ or /spaces_2/ is zero, then abort the WebSocket connection. This is a symptom of a cross-protocol attack.
EXAMPLE: In the example above, /spaces_1/ would be 4 and /spaces_2/ would be 10.

6. If /key-number_1/ is not an integral multiple of /spaces_1/, then abort the WebSocket connection.

   If /key-number_2/ is not an integral multiple of /spaces_2/, then abort the WebSocket connection.

   NOTE: This can only happen if the client is not a conforming WebSocket client.

7. Let /part_1/ be /key-number_1/ divided by /spaces_1/.

   Let /part_2/ be /key-number_2/ divided by /spaces_2/.

   EXAMPLE: In the example above, /part_1/ would be 906,585,445 and /part_2/ would be 179,922,739.

8. Let /challenge/ be the concatenation of /part_1/, expressed as a big-endian 32 bit integer, /part_2/, expressed as a big-endian 32 bit integer, and the eight bytes of /key_3/ in the order they were sent on the wire.

   EXAMPLE: In the example above, this would be the 16 bytes 0x36 0x09 0x65 0x65 0x0A 0xB9 0x67 0x33 0x57 0x6A 0x4E 0x7D 0x7C 0x4D 0x28 0x36.

9. Let /response/ be the MD5 fingerprint of /challenge/ as a big-endian 128 bit string. [RFC1321]

   EXAMPLE: In the example above, this would be the 16 bytes 0x6E 0x60 0x39 0x65 0x42 0x6B 0x39 0x7A 0x24 0x52 0x38 0x70 0x4F 0x74 0x56 0x62, or "n`9eBk9z$R8pOtVb" in UTF-8.

NEW:
/client nonce/
   The value of the "Sec-WebSocket-Nonce" field in the client’s handshake.

/server nonce/
   An 64 bit random integer generated by the server.

   EXAMPLE expressed in hexadecimal:

       15F0D2278BCD457F

3. Let /location/ be the string that results from constructing a WebSocket URL from /host/, /port/, /resource name/, and /secure flag/.

4. Let /server digest/ be the 16 octets of MD5 digest of the concatenation of /client nonce/, the ASCII string "WebSocket" and /server nonce/.

   EXAMPLE expressed in hexadecimal:

       Ee7eBe0c369cBf255a4451Fc58D7Cc4f

OLD:

   13. Send /response/.

NEW:
13. Send the octets 0x8F, 0x18, /server nonce/ and /server digest/.

14. The *WebSocket connection is accepted*. Now the server may send messages to the connection as described in the next section.

15. Read eighteen octets from the server. Let /client hello/ be those bytes. If the connection closes before these bytes are received, then fail the WebSocket connection and abort these steps.

48. If the first octet of /client hello/ is not 0x8f then fail the WebSocket connection and abort these steps. If the second octet of /client hello/ is not 0x10 then fail the WebSocket connection and abort these steps.

50. Let /client expected/ be the 16 octets of MD5 digest of the concatenation of /server nonce/, the ASCII string "WebSocket" and /client nonce/

EXAMPLE expressed as a hexadecimal number:

33122035D11258Fd7c764314580e539c

Let /client digest/ be the 16 octets starting from the third octet of /client hello/

51. If /client digest/ does not equal /client expected/ then fail the WebSocket connection and abort these steps.

52. The *WebSocket connection is established*. Now the server may send and receive to and from the connection as described in the next section.

14. Security Considerations

15. IANA Considerations

This requirements document does not mandate any IANA actions.

16. Acknowledgments

Thanks to Scott Ferguson, Willy Tarreau, Dave Cridland, among others.

17. References
17.1. Normative References


17.2. Informative References


Authors’ Addresses

Greg Wilkins
WebTide

EMail: gregw@webtide.com
Gabriel Montenegro
Microsoft Corporation

EMail: gabriel.montenegro@microsoft.com