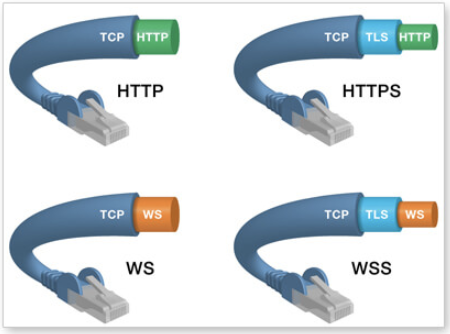
**[WebSocket协议理解-数据包格式解析](https://www.cnblogs.com/zhangmingda/p/12678630.html)**

**WebSocket 的诞生**

做客户端开发时，接触最多的应用层网络协议，就是 HTTP 协议，而今天介绍的 WebSocket，下层和 HTTP 一样也是基于 TCP 协议，这是一种轻量级网络通信协议，**也属于应用层协议。**

****

**WebSocket 与 HTTP/2 一样，其实都是为了解决 HTTP/1.1 的一些缺陷而诞生的，而 WebSocket 针对的就是「请求-应答」这种"半双工"的模式的通信缺陷。**

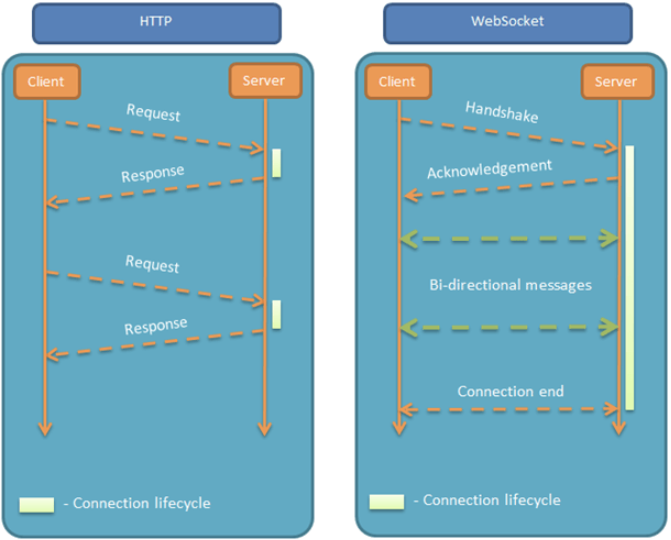
「请求-应答」是"半双工"的通信模式，数据的传输必须经过一次请求应答，这个完整的通信过程，通信的同一时刻数据只能在一个方向上传递。它最大的问题在于，HTTP 是一种被动的通信模式，服务端必须等待客户端请求才可以返回数据，无法主动向客户端发送数据。

这也导致在 WebSocket 出现之前，一些对实时性有要求的服务，通常是基于轮询（Polling）这种简单的模式来实现。轮询就是由客户端定时发起请求，如果服务端有需要传递的数据，可以借助这个请求去响应数据。**轮询的缺点也非常明显，大量空闲的时间，其实是在反复发送无效的请求，这显然是一种资源的损耗。**

**虽然在之后的 HTTP/2、HTTP/3 中，针对这种半双工的缺陷新增了 Stream、Server Push 等特性，但是「请求-应答」依然是 HTTP 协议主要的通信方式。**

**WebSocket 协议是由 HTML5 规范定义的，原本是为了浏览器而设计的，可以避免同源的限制，浏览器可以与任意服务端通信，现代浏览器基本上都已经支持 WebSocket。**

**虽然 WebSocket 原本是被定义在 HTML5 中，但它也适用于移动端，尽管移动端也可以直接通过 Socket 与服务端通信，但借助 WebSocket，可以利用 80（HTTP） 或 443（HTTPS）端口通信，有效的避免一些防火墙的拦截。**



**WebSocket 是真正意义上的全双工模式，也就是我们俗称的「长连接」。当完成握手连接后，客户端和服务端均可以主动的发起请求，回复响应，并且两边的传输都是相互独立的。**

**一、WebSocket 的特点**

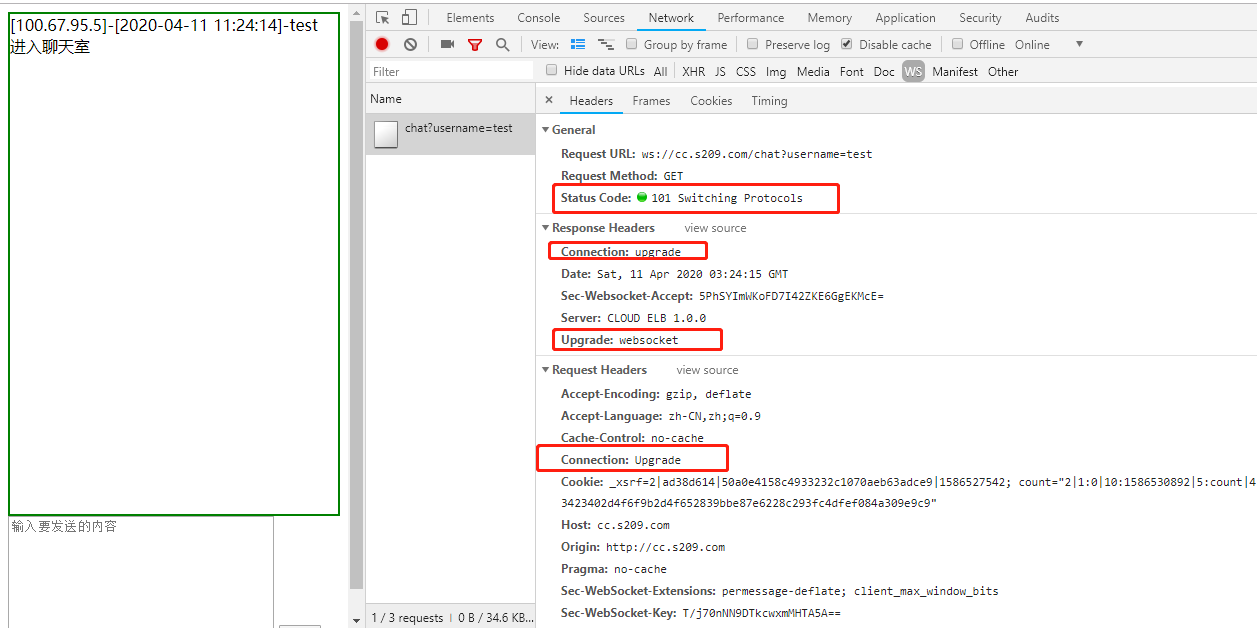
WebSocket 的数据传输，是基于 TCP 协议，但是在传输之前，还有一个握手的过程，双方确认过眼神，才能够正式的传输数据。

WebSocket 的握手过程，符合其 "Web" 的特性，是利用 HTTP 本身的 "协议升级" 来实现。

在建立连接前，客户端还需要知道服务端的地址，WebSocket 并没有另辟蹊径，而是沿用了 HTTP 的 URL 格式，但协议标识符变成了 "ws" 或者 "wss"，分别表示明文和加密的 WebSocket 协议，这一点和 HTTP 与 HTTPS 的关系类似。

 WebSocket连接url举例

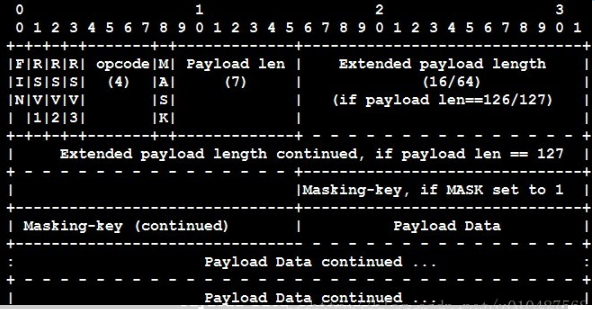
**二、由http/https---->WebSocket的切换浏览器调试模式可见**



**--------------------------------------------------------------------------------**

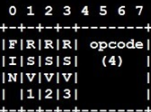
**Websocket的世界**

通信协议格式是WebSocket格式，服务器端采用Tcp Socket方式接收数据，进行解析，协议格式如下：



 　　首先我们需要知道数据在物理层，数据链路层是以二进制进行传递的，而在应用层是以16进制字节流进行传输的。

**第一个字节：**

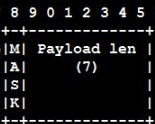
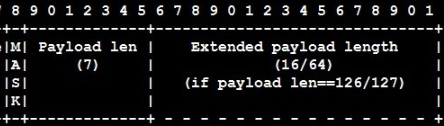


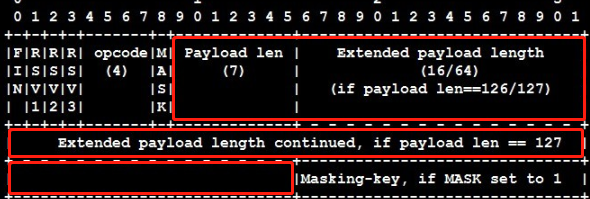
**FIN:1位，用于描述消息是否结束，**如果为1则该消息为消息尾部,如果为零则还有后续数据包;  
　　RSV1,RSV2,RSV3：各1位，用于扩展定义的,如果没有扩展约定的情况则必须为0  
　　**OPCODE:4位，用于表示消息接收类型，**如果接收到未知的opcode，接收端必须关闭连接。**长连接探活包就是这里标识的。**

**OPCODE定义的范围：**

**0x0表示附加数据帧**  
**0x1表示文本数据帧**  
**0x2表示二进制数据帧**  
　　　　0x3-7暂时无定义，为以后的非控制帧保留  
**0x8表示连接关闭**  
**0x9表示ping**  
**0xA表示pong**  
　　　　0xB-F暂时无定义，为以后的控制帧保留

**第二个字节以及以后字节：**

**或者**

****

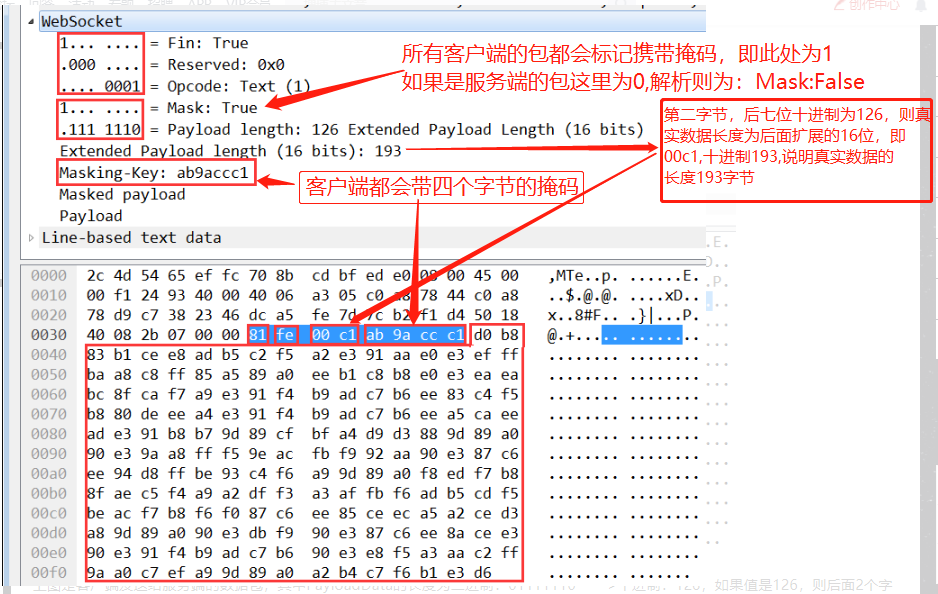
**MASK:1位**，用于标识PayloadData是否经过掩码处理，客户端发出的数据帧需要进行掩码处理，所以此位是1。数据需要解码。

**Payload length** ===**x**，如果

　　如果**x**值在0-125，则是payload的真实长度。  
　　如果**x**值是126，则后面2个字节形成的16位无符号整型数的值是payload的真实长度。  
　　如果**x**值是127，则后面8个字节形成的64位无符号整型数的值是payload的真实长度。

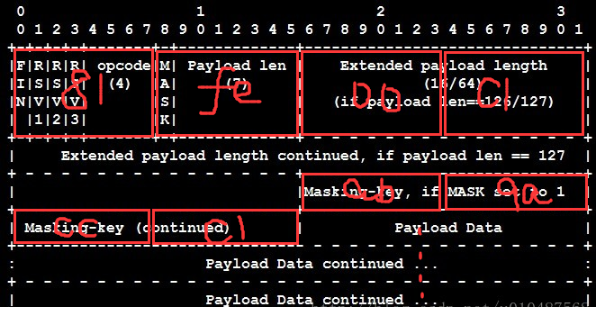
　　此外，如果payload length占用了多个字节的话，payload length的二进制表达采用网络序（big endian，重要的位在前）。

**举例抓到客户端返回包：**

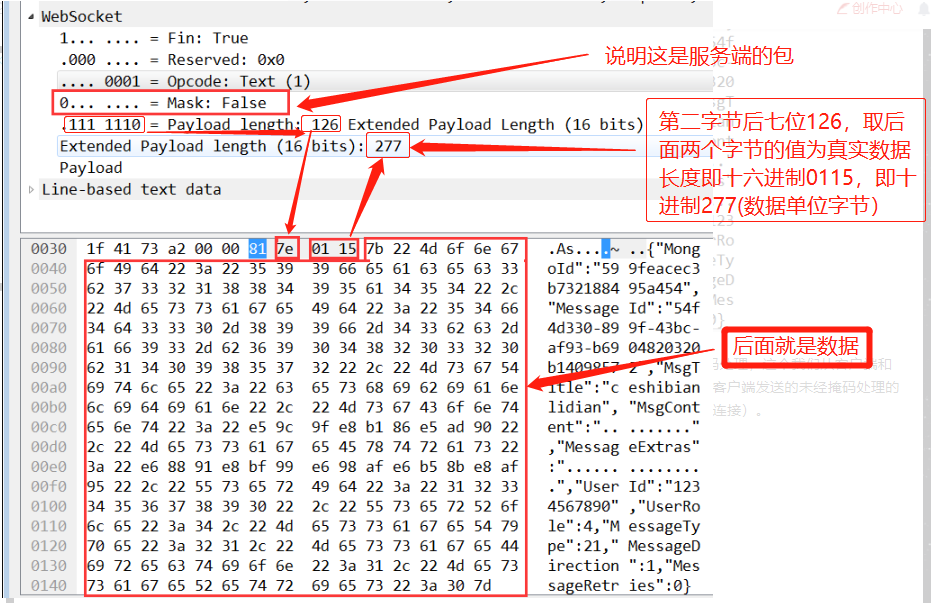
****

上图是客户端发送给服务端的数据包，其中PayloadData的长度为二进制：01111110——>十进制：126；如果值是126，则后面2个字节形成的16位无符号整型数的值是payload的真实长度。也就是圈红的十六进制：00C1——>十进制：193 byte。所以PayloadData的真实数据长度是193 bytes；

根据我们的分析，客户端到服务端数据包的websocket帧图应该为：

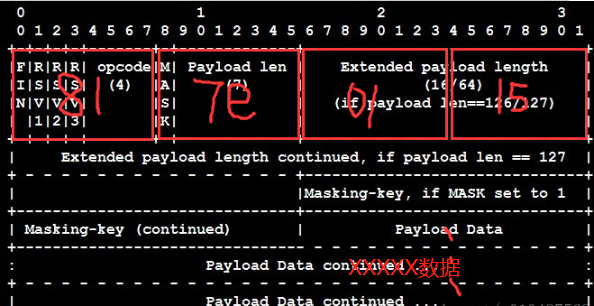


**举例抓到服务端返回包：========================================================**

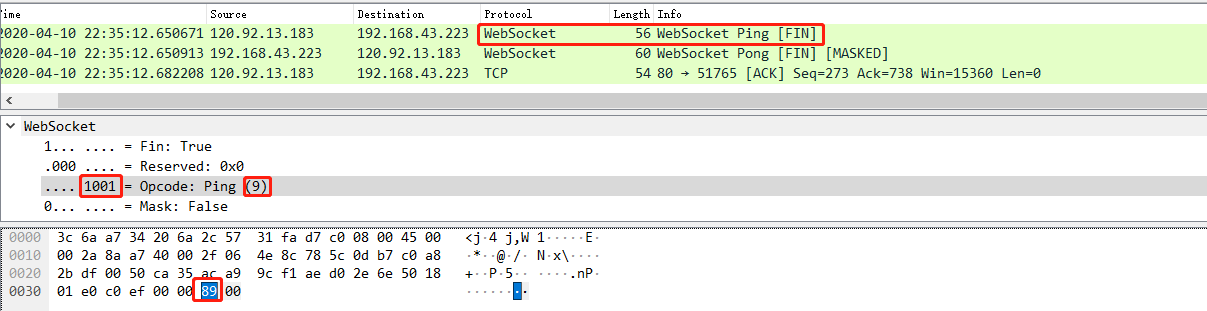


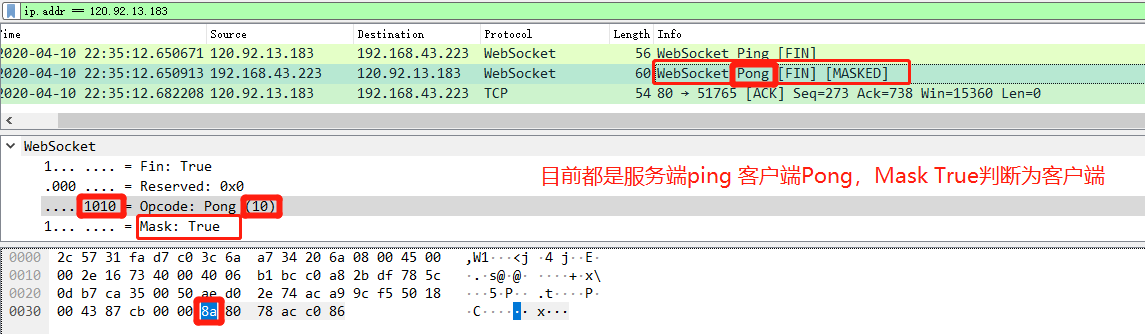
可以发现**服务器发送给客户端的数据包中第二个字节中MASK位为0**，这说明服务器发送的数据帧未经过掩码处理，这个我们从客户端和服务端的数据包截图中也可以发现，客户端的数据被加密处理，而服务端的数据则没有。（如果服务器收到客户端发送的未经掩码处理的数据包，则会自动断开连接；反之，如果客户端收到了服务端发送的经过掩码处理的数据包，也会自动断开连接）。

根据我们的分析，服务端到客户端数据包的websocket帧图应该为：



**====保活数据包ping/pong + TCP-ACK===============**

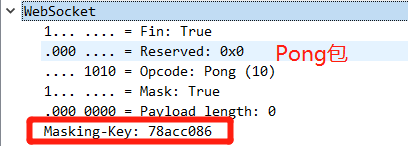
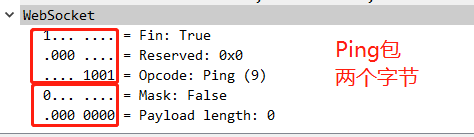
****

 -------------------------------------------------------------------------------------------------------------------------------------------

**详细分析：**

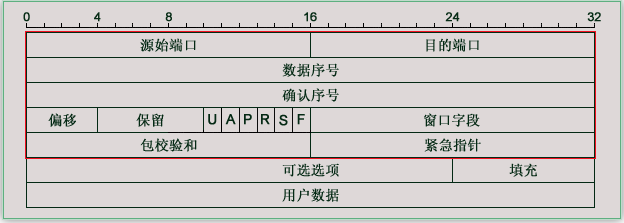
**WebSocket的ping包总长度2个字节；**

**WebSocket的Pong包总长度6个字节，包括4个字节的**

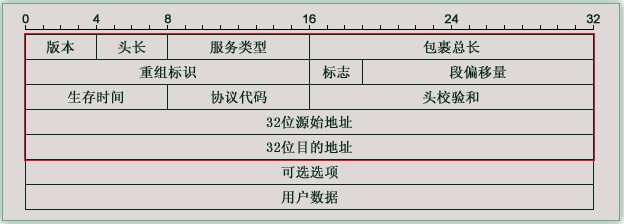


　　而一个ping包WireShark抓包的Length总长度为56个字节。其中WebSocket包长度2字节；TCP头20字节；IP头20字节；以太网帧长度14个字节；

**TCP头20字节：**

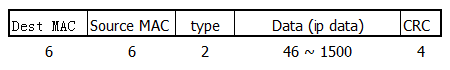
****

**IPV4头20字节：**

****

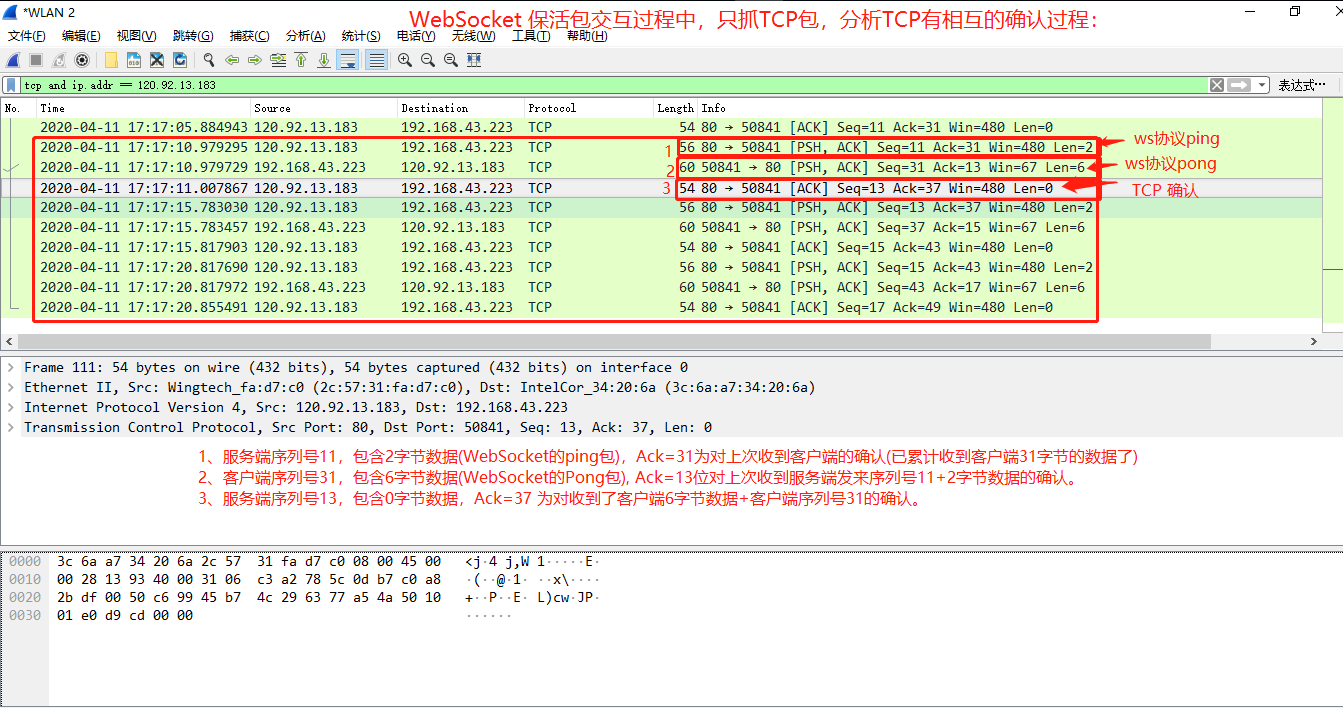
**以太网帧头14字节：**

　　目的/源MAC地址共计12字节，类型2字节。共计14字节。

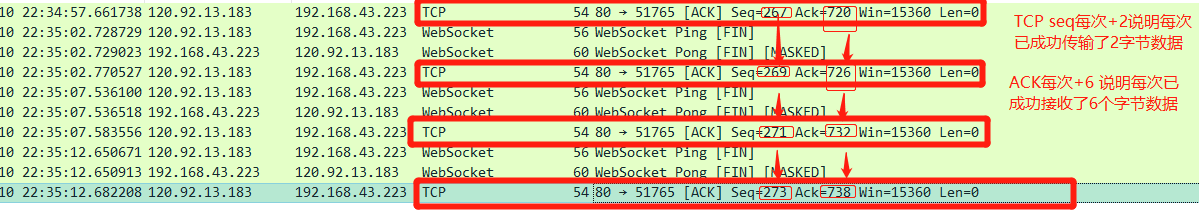


**TCP ACK 确认包：**

**Seq每次+2 说明WebSocket的2字节ping数据包发送成功，Ack每次+6说明成功接收了WebSocket的6字节Pong回复包。**



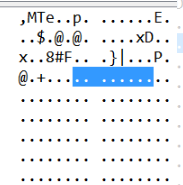
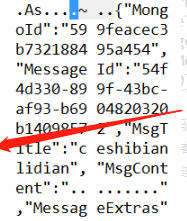
 　　捕获WebSocket包



**关于掩码：**

**=======客户端加密和服务端数据不加密，抓包区别明显==========**

**看客户端有掩码的数据                                   服务端无掩码的数据**

**                    **

**掩码算法了解**

掩码键（Masking-key）是由客户端挑选出来的32位的随机数。掩码操作不会影响数据载荷的长度。掩码、反掩码操作都采用如下算法：

首先，假设：

* + original-octet-i：为原始数据的第i字节。
  + transformed-octet-i：为转换后的数据的第i字节。
  + j：为i mod 4的结果。
  + masking-key-octet-j：为mask key第j字节。

算法描述为： original-octet-i 与 masking-key-octet-j 异或后，得到 transformed-octet-i。

**掩码的诞生---------**

　　WebSocket协议中，数据掩码的作用是增强协议的安全性。但数据掩码并不是为了保护数据本身，因为算法本身是公开的，运算也不复杂。除了加密通道本身，似乎没有太多有效的保护通信安全的办法。

　　那么为什么还要引入掩码计算呢，除了增加计算机器的运算量外似乎并没有太多的收益（这也是不少同学疑惑的点）。

答案还是两个字：**安全。但并不是为了防止数据泄密，而是为了防止早期版本的协议中存在的代理缓存污染攻击（proxy cache poisoning attacks）等问题。**

1、代理缓存污染攻击

下面摘自2010年关于安全的一段讲话。其中提到了代理服务器在协议实现上的缺陷可能导致的安全问题。猛击出处。

“We show, empirically, that the current version of the WebSocket consent mechanism is vulnerable to proxy cache poisoning attacks. Even though the WebSocket handshake is based on HTTP, which should be understood by most network intermediaries, the handshake uses the esoteric “Upgrade” mechanism of HTTP [5]. In our experiment, we find that many proxies do not implement the Upgrade mechanism properly, which causes the handshake to succeed even though subsequent traffic over the socket will be misinterpreted by the proxy.”

[TALKING] Huang, L-S., Chen, E., Barth, A., Rescorla, E., and C.

Jackson, "Talking to Yourself for Fun and Profit", 2010,

在正式描述攻击步骤之前，我们假设有如下参与者：

* + 攻击者、攻击者自己控制的服务器（简称“邪恶服务器”）、攻击者伪造的资源（简称“邪恶资源”）
  + 受害者、受害者想要访问的资源（简称“正义资源”）
  + 受害者实际想要访问的服务器（简称“正义服务器”）
  + 中间代理服务器

攻击步骤一：

* 1. 攻击者浏览器 向 邪恶服务器 发起WebSocket连接。根据前文，首先是一个协议升级请求。
  2. 协议升级请求 实际到达 代理服务器。
  3. 代理服务器 将协议升级请求转发到 邪恶服务器。
  4. 邪恶服务器 同意连接，代理服务器 将响应转发给 攻击者。

由于 upgrade 的实现上有缺陷，代理服务器以为之前转发的是普通的HTTP消息。因此，当协议服务器同意连接，代理服务器以为本次会话已经结束。

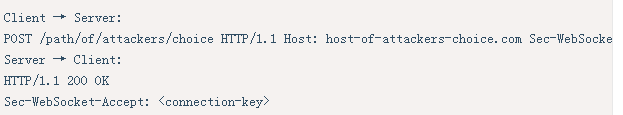
攻击步骤二：

* 1. 攻击者 在之前建立的连接上，通过**WebSocket的接口**向 邪恶服务器 发送数据，且数据是**精心构造的HTTP格式的文本**。其中包含了 正义资源 的地址，以及一个伪造的host（指向正义服务器）。（见后面报文）
  2. 请求到达 代理服务器 。虽然复用了之前的TCP连接，**但 代理服务器 以为是新的HTTP请求。**
  3. 代理服务器 向 邪恶服务器 **请求 邪恶资源。**
  4. 邪恶服务器 返回 邪恶资源。**代理服务器 缓存住 邪恶资源（url是对的，但host是 正义服务器 的地址）。**

到这里，受害者可以登场了：

* 1. **受害者 通过 代理服务器 访问 正义服务器 的 正义资源。**
  2. **代理服务器 检查该资源的url、host，发现本地有一份缓存（伪造的）。**
  3. **代理服务器 将 邪恶资源 返回给 受害者。**
  4. **受害者 卒！**

附：前面提到的精心构造的“HTTP请求报文”。



2、当前解决方案

最初的提案是对数据进行加密处理。基于安全、效率的考虑，最终采用了折中的方案：**对数据载荷进行掩码处理。**

需要注意的是，这里**只是限制了浏览器对数据载荷进行掩码处理**，但是坏人完全可以实现自己的WebSocket客户端、服务端，不按规则来，攻击可以照常进行。

**但是对浏览器加上这个限制后，可以大大增加攻击的难度，以及攻击的影响范围。**如果没有这个限制，只需要在网上放个钓鱼网站骗人去访问，一下子就可以在短时间内展开大范围的攻击。

总结：  
WebSocket 是一个独立的基于 TCP 的协议，它与 HTTP 之间的唯一关系就是它的握手请求可以作为一个升级请求（Upgrade request）经由 HTTP 服务器解释。再严谨一点:WebSocket是一个网络通讯协议, 只要理解上面的数据帧格式和握手流程, 都可以完成基于websokect的即时通讯。

**-------请关注原著，感谢！-------------------------**

**博客园   https://www.cnblogs.com/plokmju/p/okhttp\_weisocket.html**

CSDN　https://blog.csdn.net/u014252478/article/details/84380643

CSDN   https://blog.csdn.net/xiongping\_/article/details/47746953

搜狐  　https://www.sohu.com/a/227600866\_472869