Hacking with WebSockets

Mike Shema
Sergey Shekyan
Vaagn Toukharian

A Trip into HTML5

- WebSockets background
- What makes them interesting
- What makes us worry
- What makes them better

Behold the Bidirectional Browser

Yay!

The WebSocket Protocol enables **two-way communication** between a client running **untrusted code** in a controlled environment to a remote host that has **opted-in** to communications from that code.

Hmm...

Uh oh

- RFC 6455

Wonky Workarounds

Forcing persistence on a non-persistent protocol with long-polling, cometd, etc.

...often at the server's expense of one **thread/request**

...while dealing with the browser's **per-domain connection limit**

...and trying to figure out a magic **polling frequency**

...just to know when the server has some **data** ready.

Speak to Me

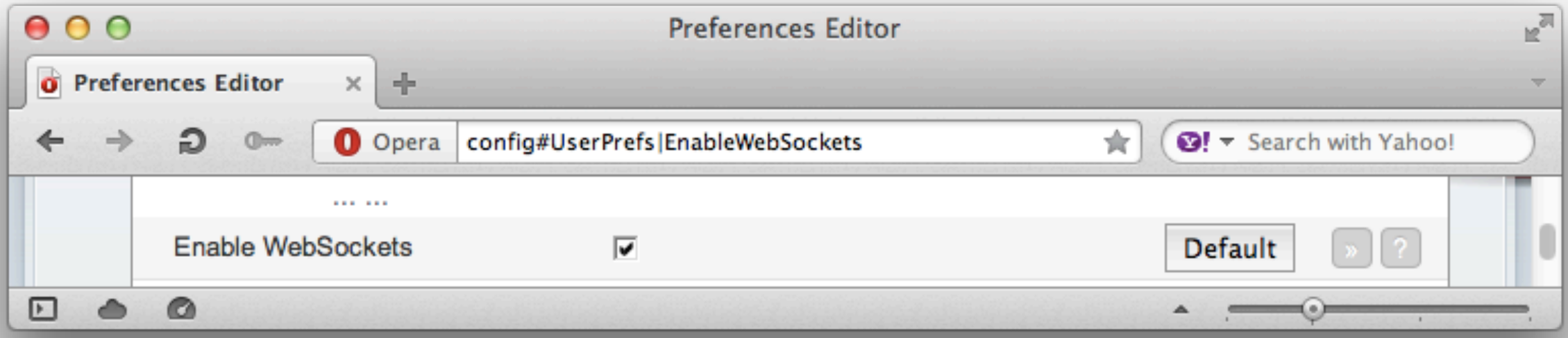
- Simple structure for transporting bytes: RFC 6455
- WebSockets API describes the JavaScript interface
  - receive with **`websocket.onmessage()`**
  - send with **`websocket.send()`**
  - transfer a **`String`**, **`Blob`**, **`ArrayBuffer`**
- Tunnel arbitrary data
  - JSON, XML, HTML
  - images, video, sound
  - another protocol

WebSockets in Action

DEMO

WebSockets Emulation

- **web-socket-js** -- The power of Flash's raw sockets with the benefits(?) of Flash's security
- **sockjs-client** -- Pure JavaScript, choose your poison: long-polling, XHR, etc.
- Forcing HTML5 on a non-HTML5 browser



WS = Works Superior

- Starts with an HTTP handshake
  - Transparent to proxies (well, it's supposed to be)
- "ping" / "pong" frames for keep-alive
- Data frames don't have HTTP overhead
  - No headers, cookies, authentication
- Data frames don't have HTTP security
  - No headers, cookies, authentication

Handshake Challenge

GET /?encoding=text HTTP/1.1

Host: echo.websocket.org

User-Agent: ...

Connection: Upgrade

Sec-WebSocket-Version: 13

Origin: http://www.websocket.org

Sec-WebSocket-Key: CjYoQD+BXC718rj3aiExxw==

Handshake Response

HTTP/1.1 101 Switching Protocols

Upgrade: WebSocket

Connection: Upgrade

Sec-WebSocket-Accept: c4RVZSknSoEHizZu6BKl3v+xUul=

[then the data frames begin]

Us and Them

```
Sec-WebSocket-Key:
base64(16 random bytes)
```

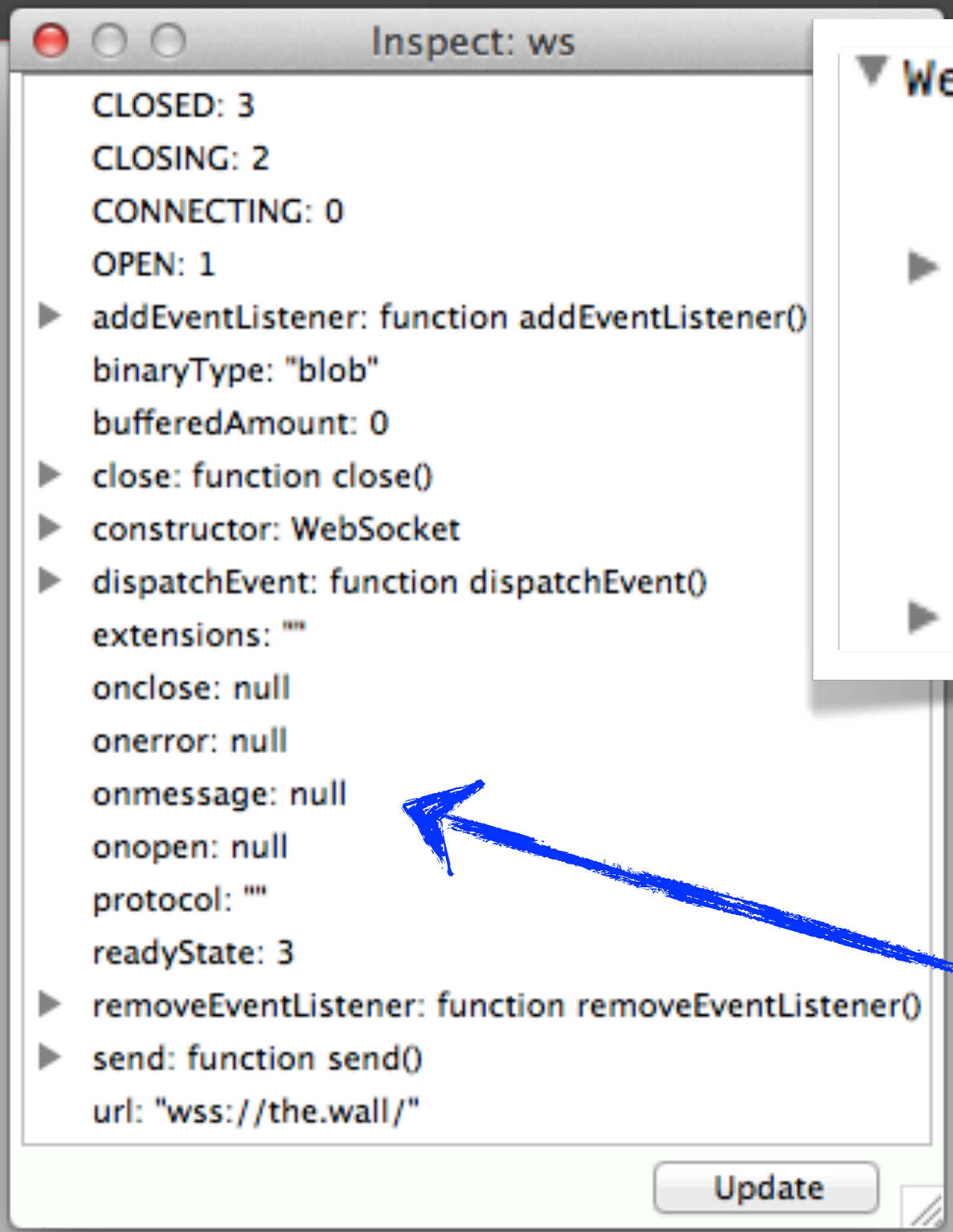
```
Sec-WebSocket-Accept:
base64(SHA1(challenge + GUID)
```

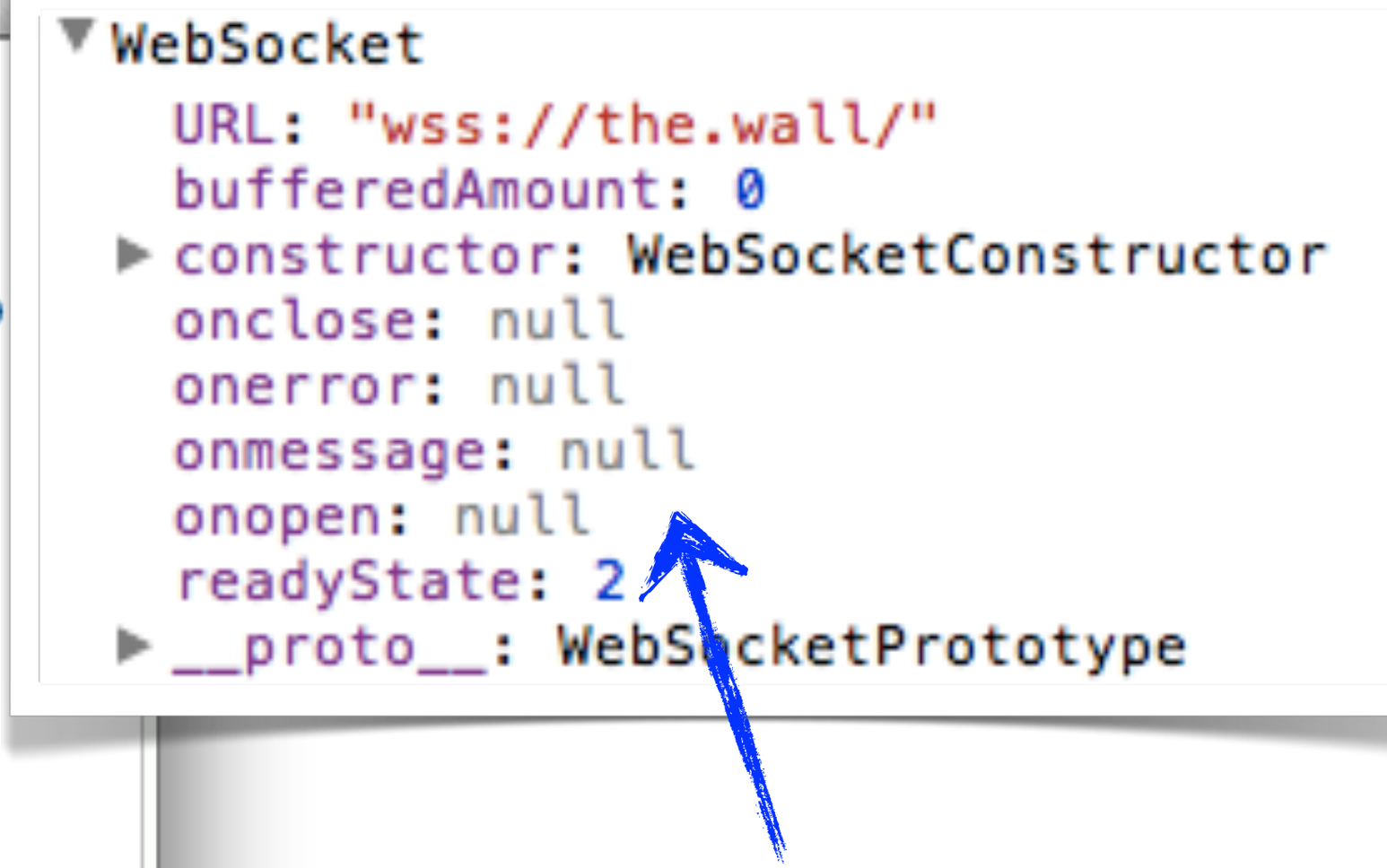
- Must finish the handshake before opening another connection to the same origin
- Success proves the endpoint speaks WebSocket
  - Does not prove identity or trust

Some Origin Policies

- Handshake includes Origin header
- User Agent should not establish plaintext WebSocket (**`ws:`**) from “secure” resource (**`https:`**)
- User Agent should minimize details for certain kinds of connection failures
  - “host/port scanning”
  - Still doesn’t affect timing analysis
- Web Workers might use WebSocket objects

WebSocket JavaScript Object

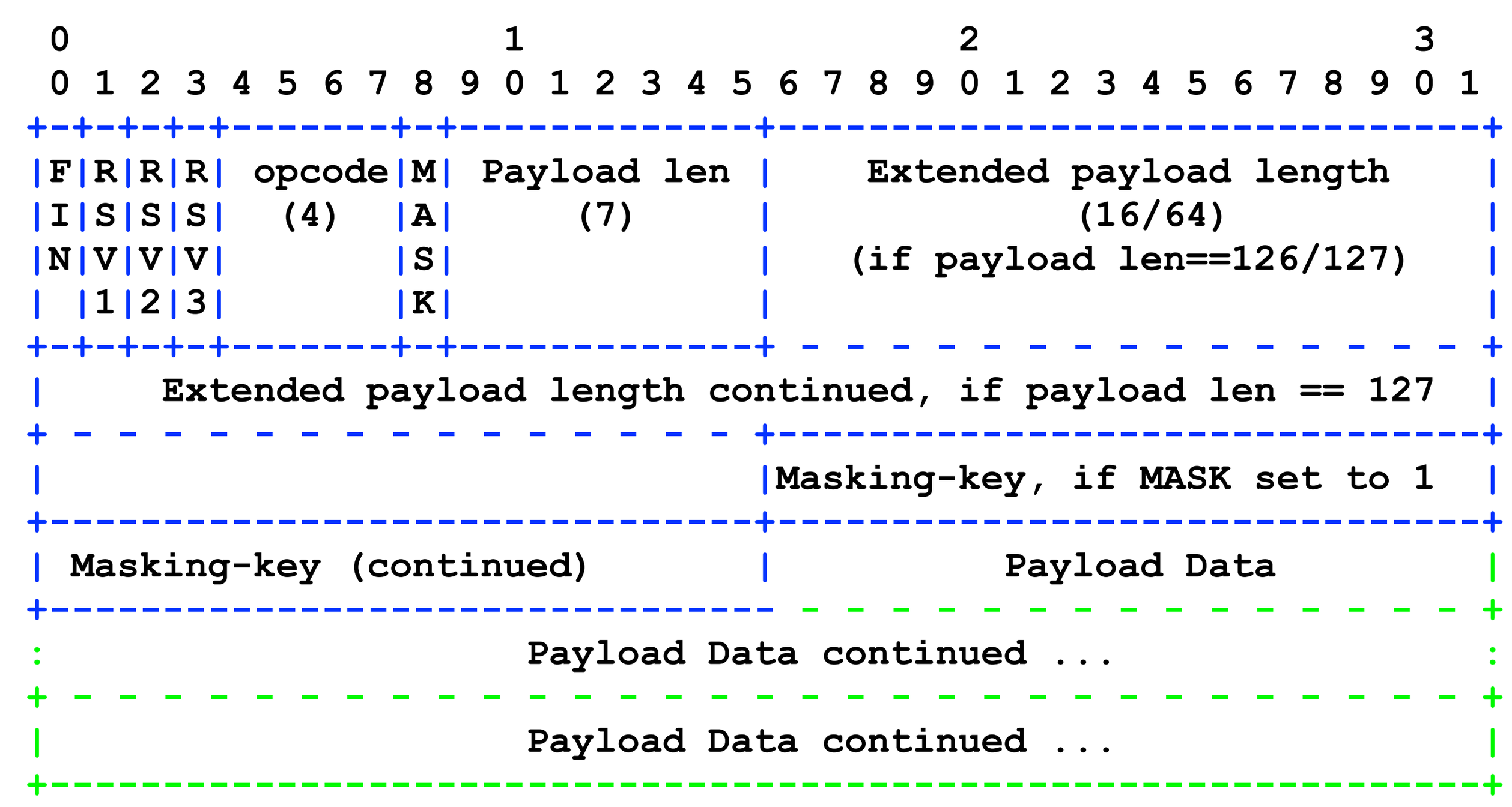





```
function(evt) {
    ...
}
```


13

Data Frame Details



Masking Data

- 32-bit pseudo-random value, XOR byte by byte
- Prevent the browser from being leveraged for cross-protocol attacks, cache poisoning

WebSocket

flags	FIN
opcode	text_frame
mask_flag	1L
length	37L
mask	0xbdccefe0
frame_data	'\xe9\xa4\x8a\x99\[...]

```
81 a5 bd cc ef e0 e9 a4 8a 99 9a be 8a c0
de a3 82 89 d3 ab cf 94 d2 ec 88 85 c9 ec 96 8f
c8 e0 cf a2 dc be 8d 81 cf ad c1 ce 93
```

```
bd cc ef e0 | bd cc ef e0 | bd ...
e9 a4 8a 99 | 9a be 8a c0 | de ...
 T  h  e  y |  `  r  e  |  c
```

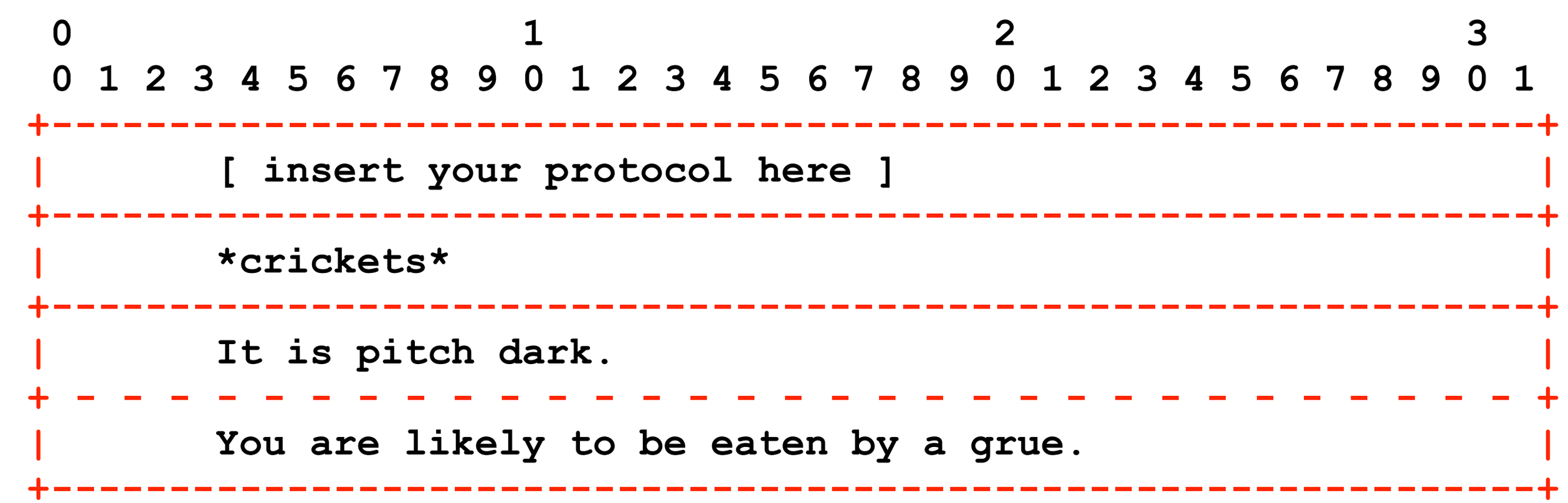
Variable Lengths

Decimal	Length (7 bits)	Variable Length (16- or 64-bit)
1	1 0 0 0 0 0 0	n/a
128	0 1 1 1 1 1 1	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
65535	0 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
65536	1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 . . .
2^64 - 1	1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1
19	1 1 0 0 1 0 0	n/a
19	0 1 1 1 1 1 1	1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
19	1 1 1 1 1 1 1	1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 . . .

Scapy Dissection

```
class WebSocket(Packet):
  name = "WebSocket"
  fields_desc = [ FlagsField("flags", 0, 4, ["RSV3", "RSV2", "RSV1",
"FIN"]),
                  BitEnumField("opcode", 0, 4, _ws_opcode_names),
                  BitField("mask_flag", 0, 1),
                  BitField("length", 0, 7),
                  ConditionalField(BitField("length16", None, 16),
lambda pkt:pkt.length == 126),
                  ConditionalField(BitField("length64", None, 64),
lambda pkt:pkt.length == 127),
                  ConditionalField(XIntField("mask", 0), lambda
pkt:pkt.mask_flag == 1),
                  StrLenField("frame_data", None,
length_from=lambda pkt:(pkt.length64 if pkt.length64 else
pkt.length16 if pkt.length16 else
                                                  pkt.length))
                ]
```

Data Frame Security Features



What makes them interesting

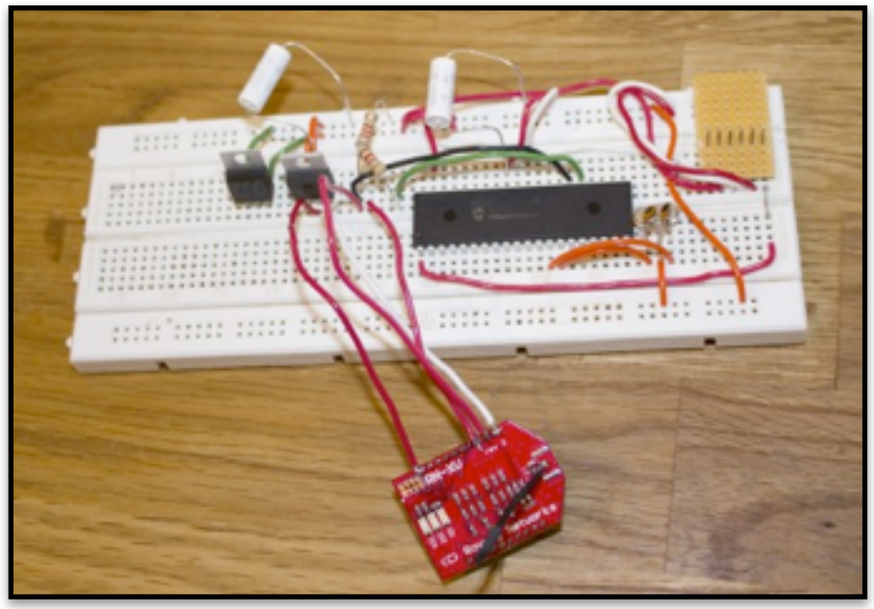
Hacking with WebSockets

WebSockets in the Wild

- Micro-SCADA
- Web apps
- Cool games
- Mobile apps

Embedded Devices

WebSocket server with PIC microcontroller allows control of electronics on the board from the browser



4 port HDMI switch controlled by embedded I/O controller with WebSocket server running in embedded linux kernel

Other places

http://labsocket.com/example.html

Current implementations

RFC 6455

unitt

Autobahn|Python

Tufão

XSockets

ServerFramework.com

resin®

websocketpp

Others

libwebsockets

jWebSocket

wt

apache-websocket

pywebsocket

jetty://

WebSocket-Node

JavaScript

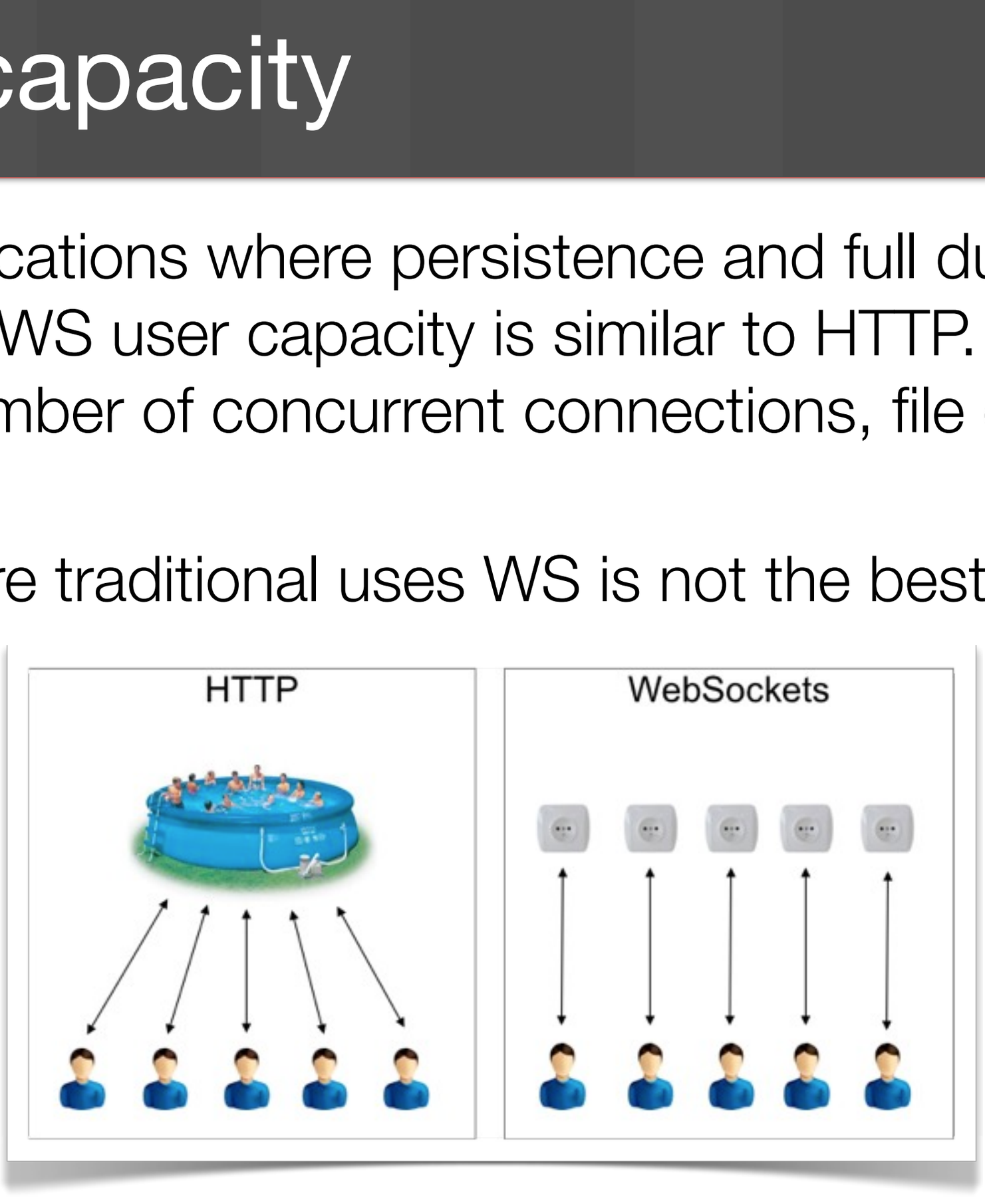
jQuery

SOCKET IO

node.js

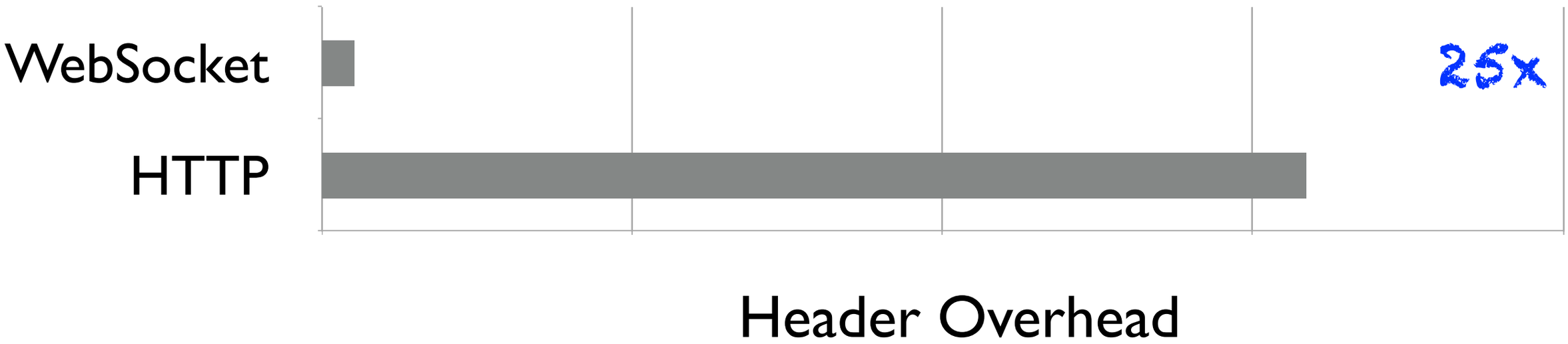
User capacity

- For applications where persistence and full duplex is required, WS user capacity is similar to HTTP. Both limited to the number of concurrent connections, file descriptors, C10K?
- With more traditional uses WS is not the best solution



Performance & bandwidth usage

- Good news here, if WS is not facing limitations of number of connections, it will outperform XHR/Long-Poll.
- WS handshake is done only once, and consecutive messages can have overhead of as low as 2 bytes.
- No compression support by default



Is there anybody out there?

- We wrote a QtWebKit-based crawler with overloaded WebSocket ctor; whenever it's called - we get a record in the DB. As simple as:

```
window._WebSocket = window.WebSocket;
window.WebSocket = function(u, p) {
    cpp_accessible_obj.ws_url = u;
    cpp_accessible_obj.dumpToDB();
    return new window._WebSocket(u, p);
}
```

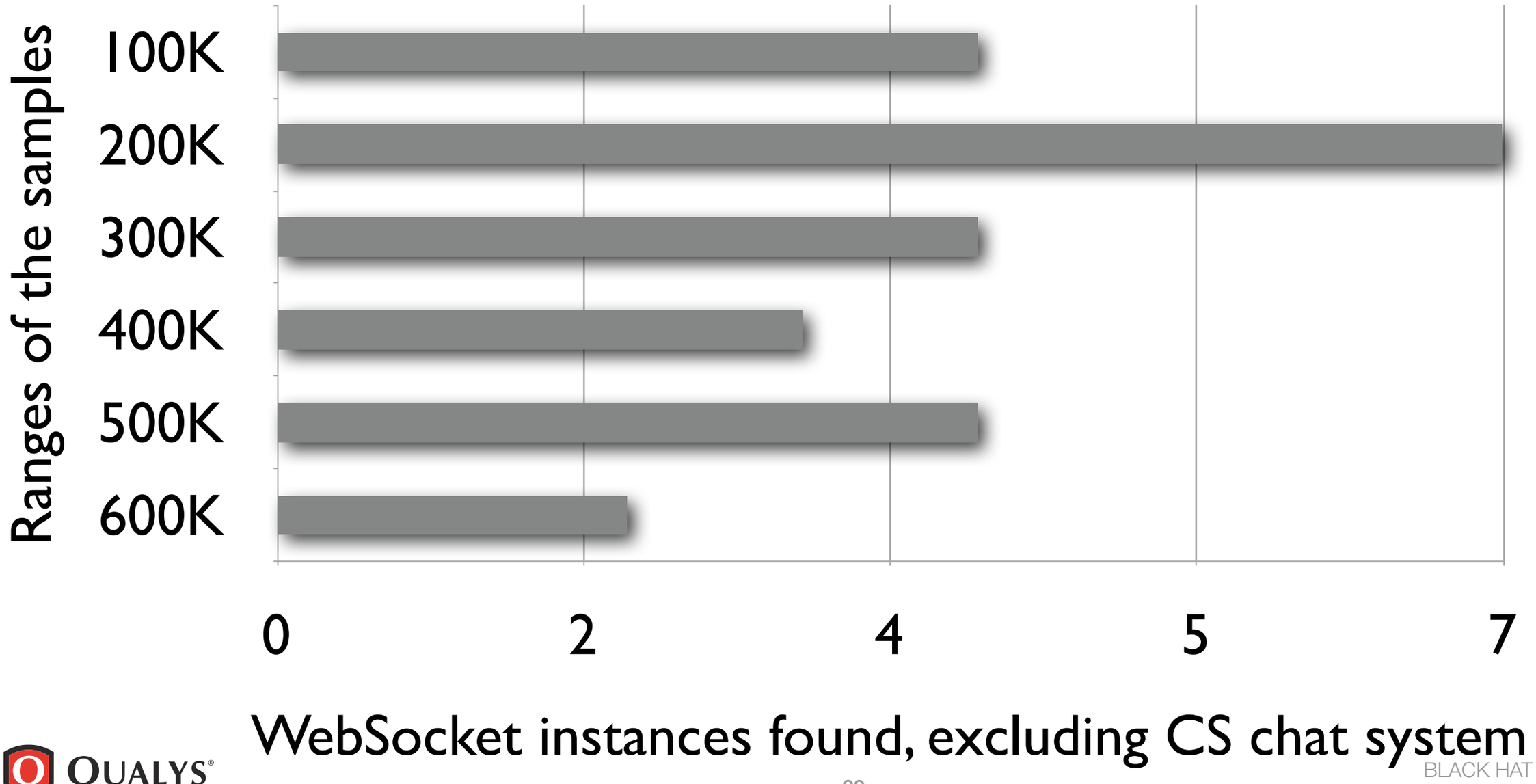
Not really...


Distribution of Alexa Top 600K websites that use WebSockets
Ranges of the samples
100K
200K
300K
400K
500K
600K
0
38
75
113
150
WebSocket instances found

Details?

- **0.15%** of websites use WebSockets on landing page.
- **Less than 4%** of captured WebSockets are using plain **`ws:`**
  - 95% of total WebSockets connect to a single vendor's customer support chat system
  - among remaining 5%, **less than 1% are using encryption**

True picture is...



More details?

- A few websites are using WebSockets as news feed (e.g. one way communication)
- A few send away every mouse click and keystroke
- Q&A website with real-time updates
- More sophisticated UX reporting
- Stock Price Push
- Chat!

{"username":"", "key":"2h3jk9"}

<prop name="ix">ZHVtbXk=</prop>

d(12, 835, 232, 34, 6);

User.UserID=507bcef0aa510038ef&transID=143619443609

But why?

- Recently created, draft still changing
- Lack of educational resources
- No debugging tools
- Lack of browser support
- Hard to choose the right server
- Lack of scalability research
- Hard to setup **`wss:`**
- New things are evil
- No one cares

What makes us worry

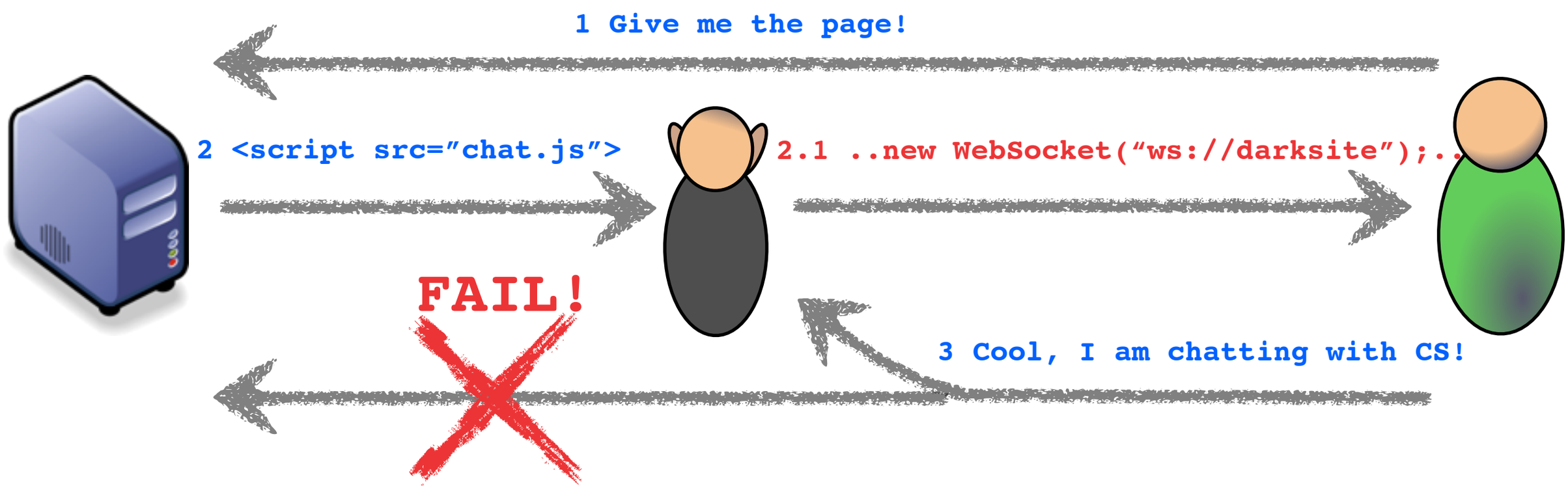
Hacking with WebSockets

(Don’t) Blame the Messenger

- WebSockets still fall victim to “old” threats
- WebSockets still have interesting things to discuss

Mixed content handling

- If you can sniff **`http:`** you can sniff **`ws:`**
- If you can intercept or inject you can overtake **`ws:`**/**`wss:`**
- It should be impossible to mix **`ws:`** with **`https:`** by RFC
  - only Firefox implements the policy



Denial of Service - Client

- WebSockets connection limit is different than HTTP connection limit
- Malicious content can exhaust browser by grabbing max. allowed number of WebSocket connections
  - *"..Yes, WebSocket is the first way to open an unlimited number of connections to a single server, so it indeed likely needs additional protection to prevent DOS attacks. But we don't really have a way to implement this correctly..."*

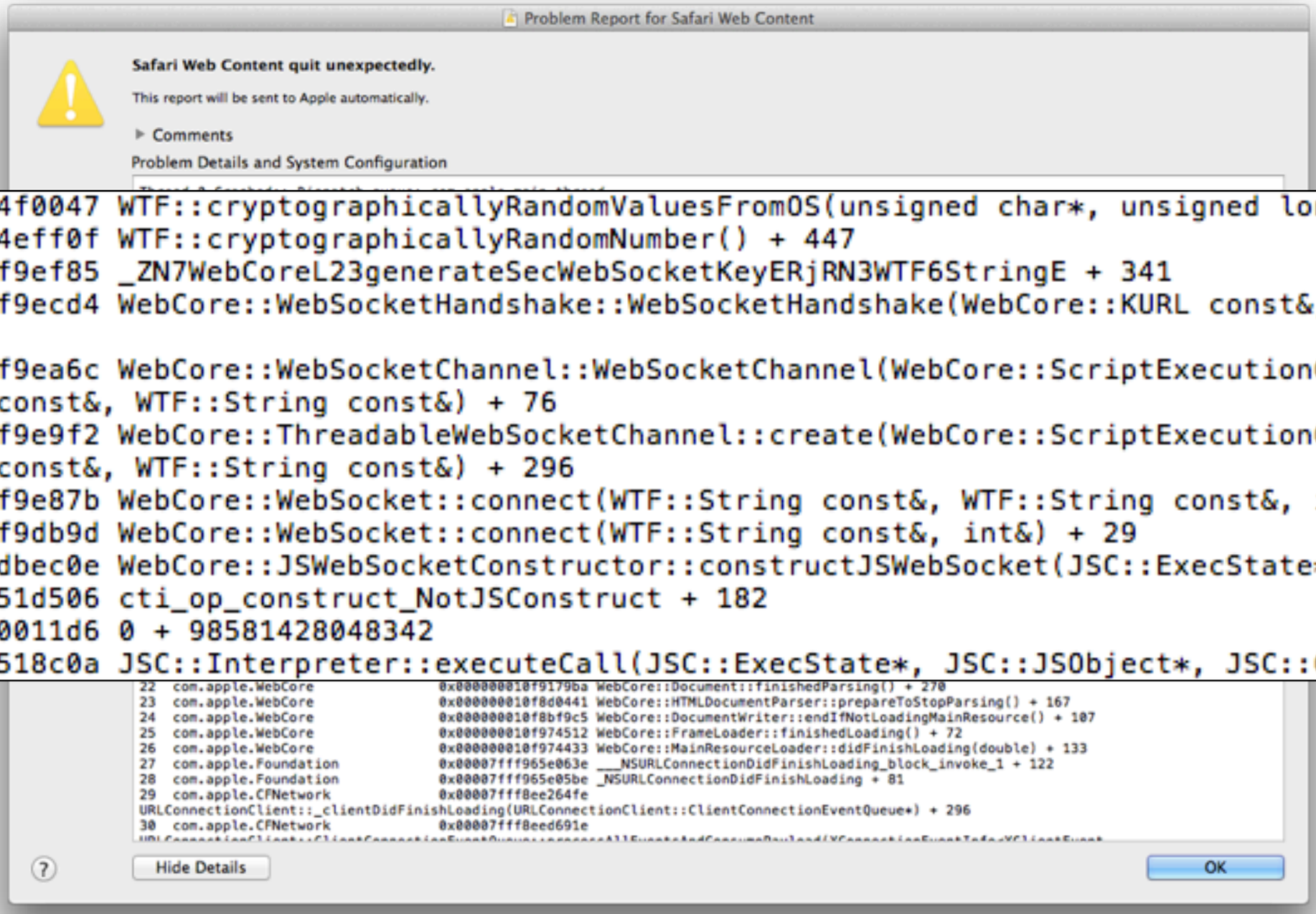
https://bugs.webkit.org/show_bug.cgi?id=32246

Chromium	Chrome	Safari	Firefox	Opera
924	3237	2970	200	900

Denial of Service - Server

- Malicious content can create large number of WebSocket connections to victim WebSocket server
- Attacks like SlowLoris strive to maintain persistent connections thus draining server resources. WebSockets are naturally like that

Stability?




```
 0x000000010f4f0047 WTF::cryptographicallyRandomValuesFromOS(unsigned char*, unsigned long) + 119
 0x000000010f4eff0f WTF::cryptographicallyRandomNumber() + 447
 0x000000010ff9ef85 _ZN7WebCoreL23generateSecWebSocketKeyERjRN3WTF6StringE + 341
 0x000000010ff9ecd4 WebCore::WebSocketHandshake::WebSocketHandshake(WebCore::KURL const&, WTF::String
text*) + 244
 0x000000010ff9ea6c WebCore::WebSocketChannel::WebSocketChannel(WebCore::ScriptExecutionContext*,
ebCore::KURL const&, WTF::String const&) + 76
 0x000000010ff9e9f2 WebCore::ThreadableWebSocketChannel::create(WebCore::ScriptExecutionContext*,
ebCore::KURL const&, WTF::String const&) + 296
 0x000000010ff9e87b WebCore::WebSocket::connect(WTF::String const&, WTF::String const&, int&) + 3233
 0x000000010ff9db9d WebCore::WebSocket::connect(WTF::String const&, int&) + 29
 0x000000010fdbec0e WebCore::JSWebSocketConstructor::constructJSWebSocket(JSC::ExecState*) + 466
 0x000000010f51d506 cti_op_construct_NotJSConstruct + 182
 0x000059a8c70011d6 0 + 98581428048342
 0x000000010f518c0a JSC::Interpreter::executeCall(JSC::ExecState*, JSC::JSObject*, JSC::CallType,
```

Are Browsers OK?

- Still **no mixed content handling policy** implemented by WebKit-based and Opera
- Firefox still doesn't let WebWorkers create WebSockets
- Message sizes handled differently

Waldo demo

- Waldo is a simple tool based on websocketpp server built to demonstrate why WebSockets as a transport are better

Transparent Proxy if ws:

Follow TCP Stream

Stream Content

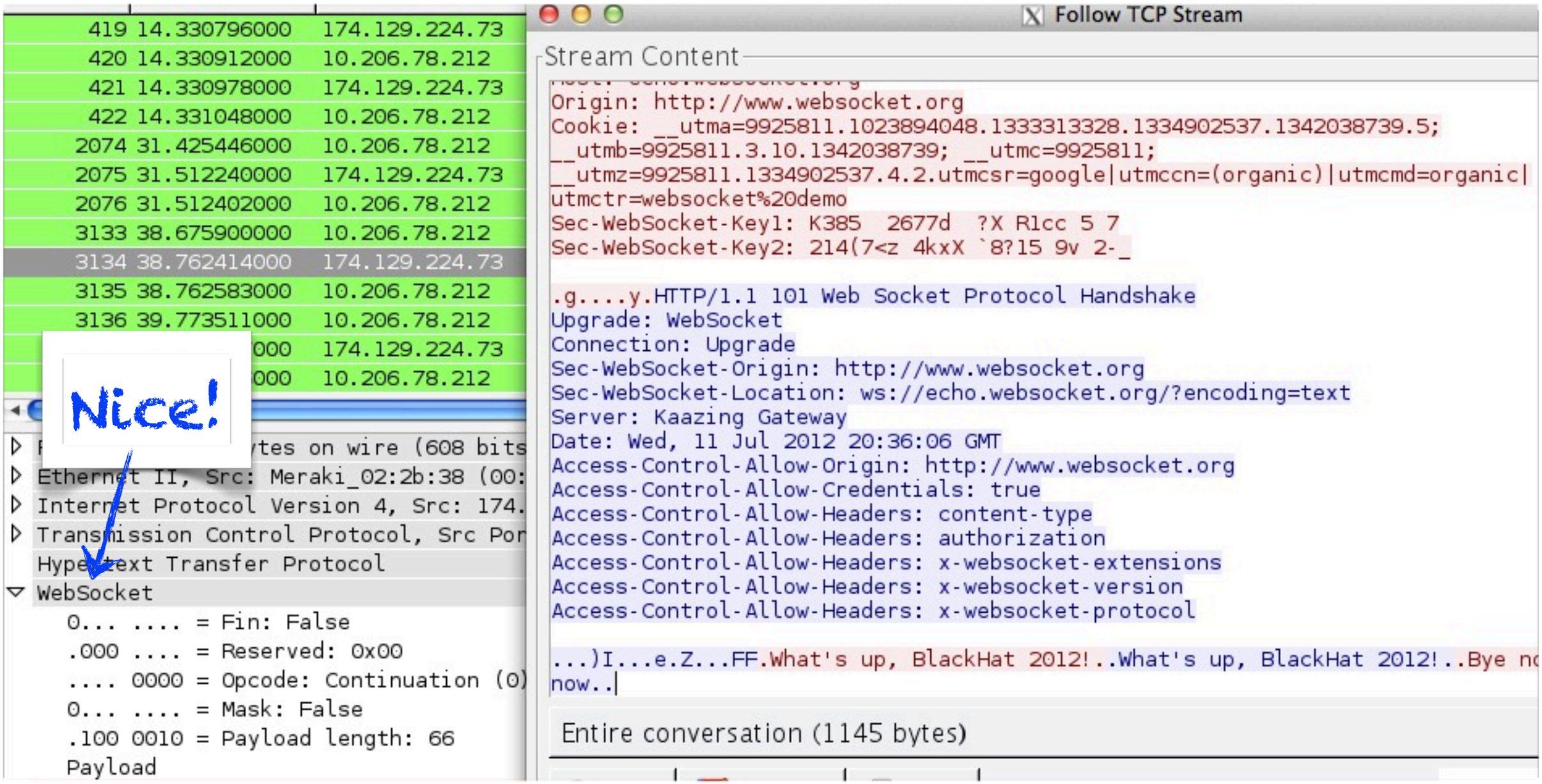
```
GET http://echo.websocket.org/?encoding=text HTTP/1.1
Host: echo.websocket.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:13.0) Gecko/20100101
Firefox/13.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Proxy-Connection: keep-alive
Sec-WebSocket-Version: 13
Origin: http://www.websocket.org
Sec-WebSocket-Key: oGXFLfcPEz1CodvwOldllg==
Cookie: __utma=9925811.273635192.1342038680.1342125045.1342131528.6; __utmc=9925811;
__utmz=9925811.1342064819.2.2.utmcsr=google|utmccn=(organic)|utmcmd=organic|
utmctr=websocket%20onerror; __utmb=9925811.2.10.1342131528
Pragma: no-cache
Cache-Control: no-cache
Connection: Upgrade

HTTP/1.0 400 Bad Request
Server: Kaazing Gateway
Date: Thu, 12 Jul 2012 22:17:26 GMT
Access-Control-Allow-Origin: http://www.websocket.org
Access-Control-Allow-Credentials: true
```

Proxy might remove this!

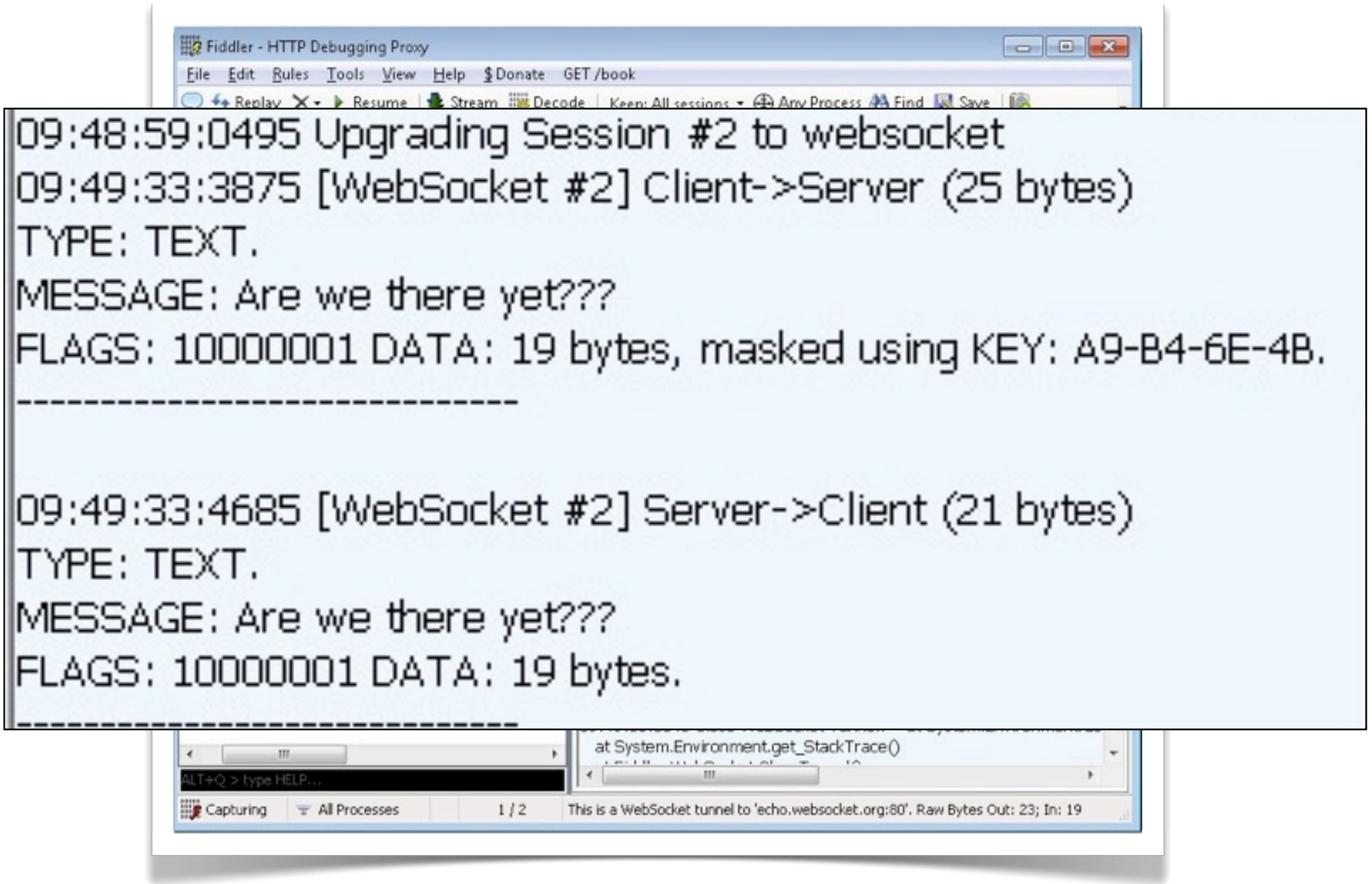
Looking for WS Security Issues

- How to inspect WS traffic
- How to manipulate WS traffic?
- Are there browser plugins to help?
- Are there proxies that support WebSockets?

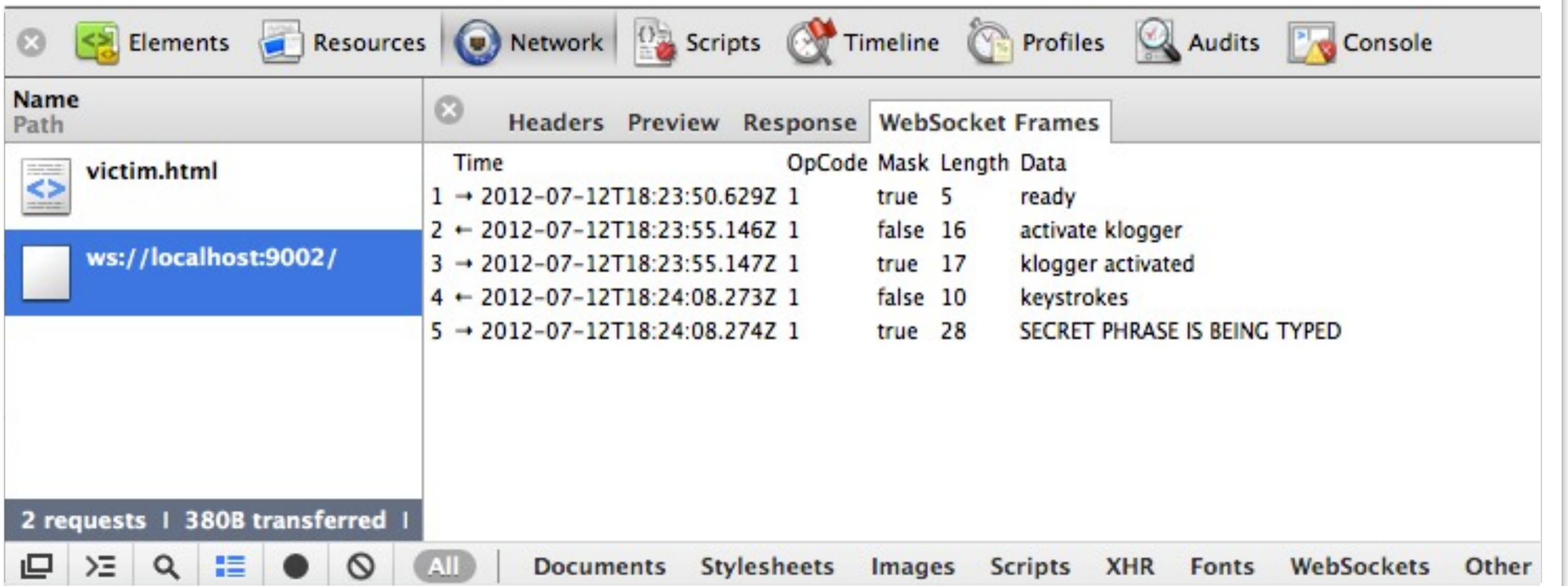
WireShark


419 14.330796000 174.129.224.73
420 14.330912000 10.206.78.212
421 14.330978000 174.129.224.73
422 14.331048000 10.206.78.212
2074 31.425446000 10.206.78.212
2075 31.512240000 174.129.224.73
2076 31.512402000 10.206.78.212
3133 38.675900000 10.206.78.212
3134 38.762414000 174.129.224.73
3135 38.762583000 10.206.78.212
3136 39.773511000 10.206.78.212
Nice!
Ethernet II, Src: Meraki_02:2b:38 (00:
Internet Protocol Version 4, Src: 174.
Transmission Control Protocol, Src Por
Hypertext Transfer Protocol
WebSocket
0... .... = Fin: False
.000 .... = Reserved: 0x00
.... 0000 = Opcode: Continuation (0)
0... .... = Mask: False
.100 0010 = Payload length: 66
Payload
Follow TCP Stream
Stream Content
Origin: http://www.websocket.org
Cookie: __utma=9925811.1023894048.1333313328.1334902537.1342038739.5;
__utmb=9925811.3.10.1342038739; __utmc=9925811;
__utmz=9925811.1334902537.4.2.utmcsr=google|utmccn=(organic)|utmcmd=organic|
utmctr=websocket%20demo
Sec-WebSocket-Key1: K385 2677d ?X R1cc 5 7
Sec-WebSocket-Key2: 214(7<z 4kxX `8?15 9v 2-_
.g....y.HTTP/1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://www.websocket.org
Sec-WebSocket-Location: ws://echo.websocket.org/?encoding=text
Server: Kaazing Gateway
Date: Wed, 11 Jul 2012 20:36:06 GMT
Access-Control-Allow-Origin: http://www.websocket.org
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: content-type
Access-Control-Allow-Headers: authorization
Access-Control-Allow-Headers: x-websocket-extensions
Access-Control-Allow-Headers: x-websocket-version
Access-Control-Allow-Headers: x-websocket-protocol
...)I...e.Z...FF.What's up, BlackHat 2012!..What's up, BlackHat 2012!..Bye no
now..
Entire conversation (1145 bytes)

Fiddler Web Debugger



Chrome Developer Tools



JavaScript overload

```
WebSocket.prototype._send = WebSocket.prototype.send;
WebSocket.prototype.send = function (data) {
  console.log("\u2192 " + data);
  this._send(data);
  this.addEventListener('message', function (msg) {
    console.log('\u2190 ' + msg.data);
  }, false);
  this.send = function (data) {
    this._send(data);
    console.log("\u2192 " + data);
  };
}
```

Security of the tunneled protocol

- Outside of HTTP cookies, form-based auth, etc.
- Possible that devs create a protocol with basic security problems (e.g. “chat” with spoofable user ids, information leakage, crypto mistakes)
- Just waiting for mistakes to happen
  - using session cookies as chat IDs (visible to the recipient)
  - replay
  - spoofing
  - fragmentation, overlapping fragments
  - server-side buffer overflows, underflows

Wish You Were Here

- Unawareness of WebSocket protocol by security devices (firewalls, IDS, IPS) makes them ineffective against malicious traffic
  - Masking inhibits identifying patterns in traffic
  - Missing auxiliary data type information makes it even harder
- Covert channels, command & control
  - Resurrect Loki (Phrack 49)
  - Sources of entropy: reserved flags, length representations, mask

Fingerprinting & Fuzzing

- Hard-coded HTTP handshake on top of WebSocket server
  - not a “real” HTTP server
  - order/case/presence/absence of headers

- Reaction to reserved flags

- Reaction to reserved opcodes

Recommendations

- What it's good for
  - Time critical data delivery
  - Apps that require true bidirectional flow
  - Interactivity
  - Higher throughput

- What it doesn’t do
  - It doesn’t fix existing vulnerabilities

What makes them better

Hacking with WebSockets

Deploy WebSockets Securely

- uh...?
- Capacity planning & measurement
- Assume the client isn't a browser -- in other words, don't trust it.
- Be careful when implementing the HTTP handshake.
- Watch out for **`Access-Control-Allow-Origin: *`**

Secure protocol for WebSockets

- **`wss:`** means secure transport, not secure app

- Remember security basics
  - Authn/Authz
  - Session identifiers
  - Server-side input validation
  - Resource exhaustion
  - Failure states

Summary

- WebSockets solve connection problems, not security problems.

- Basic security principles still apply, especially for data frames' content.

- "The new port 80" -- security devices have poor (nonexistent!?) awareness of the protocol.

Still evolving

- Draft updated as recently as July 2012, browser support still in flux.
- Contribute, adopt
- Update tools
- Create more JavaScript libraries
- Need more good protocol/libs/docs/debugging tools

Q&A

References

- http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2008-June/015108.html
- http://www.ietf.org/mail-archive/web/tls/current/msg05593.html
- http://en.wikipedia.org/wiki/Comparison_of_WebSocket_implementations
- http://webtide.intalio.com/2011/09/cometd-2-4-0-websocket-benchmarks/

Thank You!