USING HTML5 WEBSOCKETS SECURELY

Mike Shema
Qualys, Inc.
WebSockets

Protocol & API
Development
Deployment

Security
Forcing Persistence onto HTML4

...often at the server’s expense of managing one thread/request

...and restricted by the browser’s per-domain connection limit

...while trying to balance an efficient polling frequency

...just to know when the server has some data ready.
Partial Solutions

Pseudo-bidirectional communication

**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.

**HTML5 Server-Sent Events**

Properly-implemented long-polling

Content only flows from server → client
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.
Speak to Me

Protocol
Low overhead
Simple format
Content agnostic
HTTP compatible*

JavaScript API
Simple interface
Text or binary data
Origin security
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==

HTTP/1.1 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: c4RVZSknSoEHizZu6BKl3v+xUuI=

SHA1(challenge + GUID)
Handshake
Establishes mutual agreement to speak WebSockets
Not intended to prove trust or identity
Prevents cross-protocol attacks
One handshake per Origin
With (mostly) unlimited concurrent connections
Data Frame Format

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>opcode</th>
<th></th>
<th>Payload len</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>(4)</td>
</tr>
<tr>
<td>I</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td>K</td>
</tr>
</tbody>
</table>

Extended payload length (16/64)
(if payload len == 126/127)

Extended payload length continued, if payload len == 127

Masking-key, if MASK set to 1

Masking-key (continued)

Payload Data

Payload Data continued ...

Payload Data continued ...
## Variable Length Payloads

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Length (7 bits)</th>
<th>Extended Length (16- or 64-bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000000</td>
<td>n/a</td>
</tr>
<tr>
<td>128</td>
<td>01111111</td>
<td>000000001000000000</td>
</tr>
<tr>
<td>65535</td>
<td>01111111</td>
<td>111111111111111111</td>
</tr>
<tr>
<td>65536</td>
<td>11111111</td>
<td>0000000000000000100...</td>
</tr>
<tr>
<td>2^64 - 1</td>
<td>11111111</td>
<td>1111111111...1111111111111111111</td>
</tr>
<tr>
<td>19</td>
<td>1100100</td>
<td>n/a</td>
</tr>
<tr>
<td>19</td>
<td>01111111</td>
<td>110010000000000000000000000...</td>
</tr>
<tr>
<td>19</td>
<td>11111111</td>
<td>110010000000000000000000000...</td>
</tr>
</tbody>
</table>
XOR Masks Data from Browser ➔ Server

WebSocket
flags
opcode
data
from
Browser
➔
Server
32 random bits

<table>
<thead>
<tr>
<th>bd</th>
<th>cc</th>
<th>ef</th>
<th>e0</th>
</tr>
</thead>
<tbody>
<tr>
<td>bd</td>
<td>cc</td>
<td>ef</td>
<td>e0</td>
</tr>
<tr>
<td>e9</td>
<td>a4</td>
<td>8a</td>
<td>99</td>
</tr>
<tr>
<td>bd</td>
<td>cc</td>
<td>ef</td>
<td>e0</td>
</tr>
<tr>
<td>9a</td>
<td>be</td>
<td>8a</td>
<td>c0</td>
</tr>
<tr>
<td>bd</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e9</td>
<td>a4</td>
<td>8a</td>
<td>99</td>
</tr>
<tr>
<td>de</td>
<td>e3</td>
<td>2b</td>
<td>...</td>
</tr>
<tr>
<td>T h e y ' r e c</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Frame Security

Handshake headers provide security context

Don’t lose this

[ insert your protocol here ]

It is pitch dark.

You are likely to be eaten by a grue.
Sending & Receiving Data

[ Demo ]
Basic WebSockets Security
Wherever WebSockets May Roam

Transport messages for chatty apps
Load HTML, JavaScript, CSS resources
Transport media (images, audio, video)
Tunnel a non-HTTP protocol
Browser Hacking
Mixed Content

If you can sniff `http` you can sniff `ws`.
If you can intercept or inject `http`, you can take over `ws`.
WebSocket API forbids using `ws` from resource obtained with `https`.
**Denial of Service vs. ...**

**Client**

One handshake at a time per Origin

Unlimited* WS connections to Origin vs. single-digit limits for HTTP connections

**Server**

Overwhelmed by poor connection/thread management

Attacks like Slowloris drain resources with persistent connections -- exactly what WebSockets rely on.
HTTP Headers

Handshake request includes Origin

Should match the origin you expect, verify this on the server.

Incorrect value may indicate forged requests from a victim’s browser (e.g. CSRF).

Spoofable -- not a reliable primary access control measure.
HTTP Headers

Content Security Policy

Restrict to JavaScript resources loaded from approved sources with `script-src`.

Restrict WebSocket (and XHR) connections to approved servers with `connect-src`.
X-WebKit-CSP:

script-src ‘self’
script-src https://static.cdn

column-src ‘self’ (use ‘self’ for XHR)

column-src wss://web.site

report-uri https://log.site
Hostile Clients

Overloaded JavaScript

```javascript
var fuzz_replace = '';
var fuzz_append = '';

WebSocket.prototype._send = WebSocket.prototype.send;
WebSocket.prototype.send = function(data) {
    this._send(data);
    this.addEventListener('message', function(msg) {}, false);
    this.send = function(data) {
        if(fuzz_replace !== '') {
            data = fuzz_replace;
        } else if(fuzz_append !== '') {
            data = data + fuzz_append;
        }
        this._send(data);
    };
};
```
Browser Hacking

[ Demo ]
WebSocket Data Frame Security
Areas of Concern

Client

Executing in a hostile environment

Server

Implement the handshake, not a new web server

Protocol

Kerckhoff’s Principle & Shannon’s Maxim
WebSocket Server
Fingerprinting
Stability
Protecting Resources
Protocol Hardening Against...

Spoofing

Replay

Repudiation

Length manipulation
Defensive Handling of Payloads

Verify payload lengths

- Avoid buffer over-/under-runs

Avoid resource exhaustion

- Client declares 2GB length, only contains a few bytes
JavaScript Programming

“use strict”

Typed Arrays

Strings are not binary containers

Server-side considerations

!?
Examples
Epilogue
Wish You Were Here

Security devices unaware of the protocol

- WAF won’t filter
- IDS won’t detect

Data masking complicates protocol inspection

(and mix in compression, too!)
Points to Remember

Connection limits, data size limits

Security of tunneled protocol cannot rely on confidentiality within browser

Maintain session context

RSACONFERENCE2013
Client & Server

Restrict the browser’s resources with Content Security Policy.

Connect with \texttt{wss:} scheme.

Security of data can’t rely on secrecy of JavaScript.

Sec-WebSocket-Protocol: 13

The client is not trustworthy.

Verify the Origin header matches the value you expect.

Maintain coupling between the HTTP and WebSocket security contexts.
Apply

WebSockets solve connection problems, not security problems.

Remember basic security principles, especially for data frames’ content.

“The new port 80” -- security devices have nonexistent(!?) awareness of the protocol.
Thank You!

Questions?

Mike @CodexWebSecurum
http://deadliestwebattacks.com
References

http://dev.w3.org/html5/websockets/
http://dev.w3.org/html5/eventsource/
Tools

Browsers’ Developer Tools

JavaScript

Zed Attack Proxy