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
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.

- 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

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

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

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

```javascript
function(evt) {
  ...
}
```
Data Frame Details

<table>
<thead>
<tr>
<th>F</th>
<th>R</th>
<th>R</th>
<th>R</th>
<th>opcode</th>
<th>M</th>
<th>Payload len</th>
<th>Extended payload length</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>(4)</td>
<td>A</td>
<td>(7)</td>
<td>(16/64)</td>
</tr>
<tr>
<td>N</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td></td>
<td>S</td>
<td></td>
<td>(if payload len==126/127)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td>K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 ...
Masking Data

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

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Length (7 bits)</th>
<th>Variable Length (16- or 64-bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 0 0 0 0</td>
<td>n/a</td>
</tr>
<tr>
<td>128</td>
<td>0 1 1 1 1 1 1</td>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>65535</td>
<td>0 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>65536</td>
<td>1 1 1 1 1 1 1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 . . .</td>
</tr>
<tr>
<td>$2^{64} - 1$</td>
<td>1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>19</td>
<td>1 1 0 0 1 0 0</td>
<td>n/a</td>
</tr>
<tr>
<td>19</td>
<td>0 1 1 1 1 1 1</td>
<td>1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
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))
    ]
[insert your protocol here]

*crickets*

It is pitch dark.

You are likely to be eaten by a grue.
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

Others

libwebsockets

apache-websocket

pywebsocket

WebSocket-Node

JavaScript

jQuery

Socket IO
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
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:

```javascript
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);
}
```
Distribution of Alexa Top 600K websites that use WebSockets

Not really...
- **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...

Distribution of Alexa Top 600K websites that use WebSockets

Ranges of the samples

- 100K
- 200K
- 300K
- 400K
- 500K
- 600K

WebSocket instances found, excluding CS chat system
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!
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

1 Give me the page!

2 <script src="chat.js">

2.1 ..new WebSocket("ws://darksite");..

FAIL!

3 Cool, I am chatting with CS!
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

<table>
<thead>
<tr>
<th>Chromium</th>
<th>Chrome</th>
<th>Safari</th>
<th>Firefox</th>
<th>Opera</th>
</tr>
</thead>
<tbody>
<tr>
<td>924</td>
<td>3237</td>
<td>2970</td>
<td>200</td>
<td>900</td>
</tr>
</tbody>
</table>
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?
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:

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

The image shows a screenshot of WireShark, a network protocol analyzer. The screenshot includes a packet capture showing a WebSocket handshake and some stream content. The UI shows the protocol stack, including TCP, HTTP/1.1, and WebSocket, along with a message indicating a successful handshake and the start of communication.

The output includes details such as:
- IP addresses
- Port numbers
- HTTP headers
- Protocol versions
- WebSocket details

The text highlights a successful WebSocket handshake, with messages indicating a WebSocket Key and an upgrade sequence. The conversation is marked as "Nice!" and includes a payload length of 66 bytes.

The overall content suggests a demonstration of network traffic analysis using WireShark for understanding WebSocket communications.
Fiddler Web Debugger

09:48:59:0495 Upgrading Session #2 to websocket

TYPE: TEXT.
MESSAGE: Are we there yet???
FLAGS: 10000001 DATA: 19 bytes, masked using KEY: A9-B4-6E-4B.

TYPE: TEXT.
MESSAGE: Are we there yet???
FLAGS: 10000001 DATA: 19 bytes.
Chrome Developer Tools

---

[Image of Chrome Developer Tools interface showing network requests and WebSocket frames.

- `victim.html` page:
  - Requested at `ws://localhost:9002/`.

- WebSocket frames:
  - Frame 4: `OpCode: false`, `Data: 10`, `keystrokes`.

- Summary:
  - 2 requests, 380B transferred.

---

QUALYS®

BLACK HAT USA 2012
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('← ' + 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
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
References

Thank You!