WebSocket Security Analysis

Jussi-Pekka Erkkilä
Aalto University School of Science
juerkkil@iki.fi

Abstract

The WebSocket technology provides a bi-directional real-time communications channel for advanced web applications and services. This study overviews the WebSocket protocol and the API, and describes the advantages they provide. The main contribution of this paper is to review and analyze the security concerns related to WebSockets, discuss the possible solutions and provide the best practices for deployment of a WebSocket service. Also, this paper proposes that certain security features should be implemented in web browsers to ensure the security and the privacy of the users. Web browser vendors are in important role when implementing the security features. The WebSocket technology is not yet standardized but it is expected that usage of WebSockets will increase significantly in the near future. Overall, WebSockets solve connectivity problems, not the security problems. Several open issues exist, but with a good design and proper implementation of web browsers and web services, the risk level can be mitigated.

KEYWORDS: websocket, html5, security, browser security, javascript, web

1 Introduction

WebSocket is a technology that provides a bi-directional real-time communications channel over a Transmission Control Protocol (TCP) connection [3]. The WebSocket technology covers JavaScript API (Application Programming Interface) that is specified by World Wide Web Consortium (W3C), and a protocol that is specified by Internet Engineering Task Force (IETF) [5]. The status of the latest specification is a "proposed standard"[1]. WebSocket API is often denoted as HTML5 WebSocket API, although it is actually separated from the HTML5 specification and is currently being developed and specified independently [17]. The Candidate Recommendation of the WebSocket API was released on September 2012, hence the API is not yet standardized. However, the functionality specified in the latest drafts is already implemented in various web browsers.

The WebSocket protocol is a network protocol running on TCP. It is designed to be implemented in web browsers and web servers but it can be used for other purposes as well. The WebSocket protocol is independent of HyperText Transfer Protocol (HTTP), although the protocols have similarities such as using TCP port 80 and the handshake process when initializing a connection [2]. Because the WebSocket technology is expected to influence the communications in the web during the next few years, WebSockets may in turn influence the security of the web. WebSocket is still a relatively new concept and not much public research has been done in the field of WebSocket security. This paper focuses on the possible security concerns in the WebSocket protocol and the API, and also on the security features the technology provides.

This paper is structured as follows. The second chapter reviews the background of WebSocket technology and the usage of the JavaScript API. The third chapter focuses on the technical security issues of WebSockets. The fourth chapter discusses and analyzes the security concerns and potential solutions. The fifth chapter proposes a few solutions on how the security of WebSockets could be improved. The final chapter concludes the whole paper and outlines the future work.

[1]http://www.rfc-editor.org/info/rfc6455

2 Background

This chapter reviews the WebSocket technology in more detail. Section 2.1 discusses the new features and advantages of the technology compared to traditional web communications. Section 2.2 illustrates the usage of the WebSocket JavaScript API on client side.

2.1 Features and advantages of WebSockets

The main difference in WebSockets – compared to the usual network traffic over HTTP – is that the WebSocket protocol does not follow the traditional request-response convention [16]. Once a client and a server have opened a WebSocket connection, both endpoints may asynchronously send data to each other. The connection remains open and active as long as either the client or the server closes the connection. [16]

In contrast, the traditional approach relies on polling. That is, a client opens a new TCP connection and makes an HTTP request to receive data from a server. This requires several round-trips between the client and the server before any actual information is sent. Moreover, a new request is needed by the client for every separate data transmission. Hence, compared to WebSockets, the traditional HTTP request-response convention results in high latency and high amount of network traffic [11]. Figure 1 illustrates how the WebSocket communication differs from traditional HTTP calls.

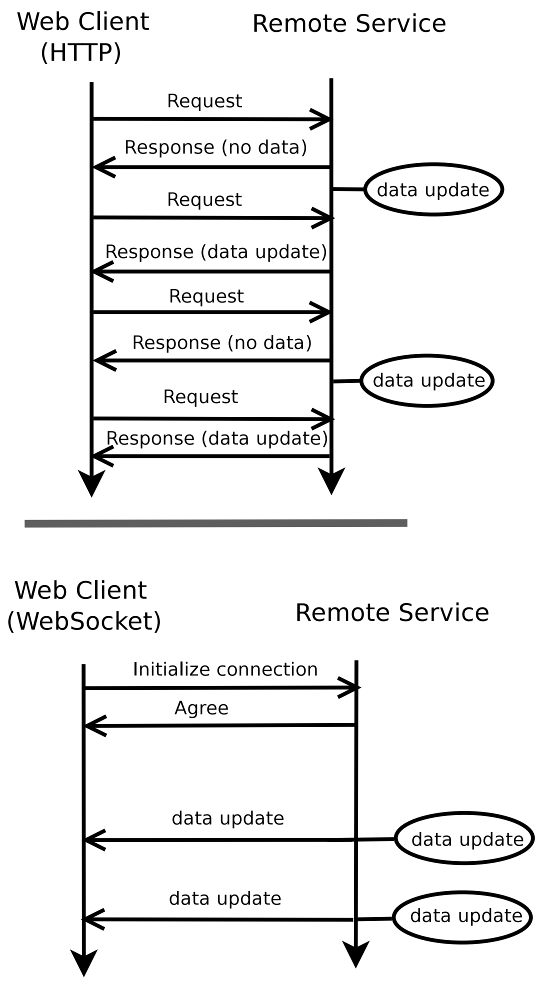

Figure 1: Retrieving real-time data with HTTP and WebSocket.

The lower redundancy of the WebSocket protocol implies that fewer computing resources are required for handling and parsing the data. Moreover, lower latency results in a better user experience in the client application [11]. However, from the web client's point of view, the advantages of the WebSocket technology depends on the use case. For delivering a small number of large files the advantages of WebSockets are not very significant, because the amount of overhead traffic is minimal compared to the actual payload data.

On the other hand, while delivering small amounts of data periodically, the WebSocket protocol may reduce the total amount of network traffic by several magnitudes. Overall, compared to traditional HTTP request-response convention, the WebSocket protocol may provide up to 500:1 reduction in unnecessary network traffic and a 3:1 reduction in latency [9]. An experimental study by Pimentel et. al (2012) shows that the latency of HTTP polling is from 2.3 to 4.5 times higher than the latency of the WebSocket protocol [12].

2.2 Functionality of WebSocket

A client establishes a WebSocket channel by sending a usual HTTP request to a server and asking for an upgrade of an existing connection [3]. Once the server has accepted the request, the subsequent messages are sent using the WebSocket protocol.

The WebSocket API can be used in web applications with JavaScript in a quite similar manner to XmlHttpRequests [7], as listing 1 illustrates.

```
var socket = new WebSocket
  ("ws://<address>:<port>");
socket.onopen = function(e) {
  // socket opened
};
socket.onmessage = function(e) {
  alert("data sent by server: " + e.data);
};
socket.onclose = function(e) {
  // socket closed
};
```

Listing 1: Initializing a WebSocket connection using JavaScript API

In the listing above we define the remote service and TCP port to which the connection is made. The API will take care of the handshake and protocol upgrade. After this, we assign the callback functions that are invoked in the event of any action. Once the connection is active, we can send data to the server or close the connection. This is illustrated in listing 2.

```
socket.send("hello world");
socket.close();
```

Listing 2: Sending data and closing the WebSocket connection using JavaScript API

3 Security concerns of the WebSockets

In web services, the security depends a lot on Transport Layer Security (TLS) encryption and the same-origin policy implemented in web browsers. Generally, TLS encryption provides a secure communication channel, whereas the same-origin policy shields the user against malicious cross-site references.

However, because the WebSocket protocol differs from HTTP in various ways, the differences may also affect to the security of the web, even though that is not always obvious. For instance, WebSocket messages do not include the HTTP headers. This may affect the behaviour of web proxies and firewalls. Moreover, the origin policy of WebSocket protocol is different compared to HTTP. Also, several other potential threats exist that are involed in the WebSocket technology. These are described and discussed in more detail in the following sections.

3.1 Origin Policy

The WebSocket API lets the web application open a connection and send arbitrary data to any server [4]. For the web application developers, this is useful and flexible when sharing the resources across the different services. In contrast to HTTP, the WebSocket protocol uses a verified-origin mechanism. This mechanism lets the target server decide, from which origins it allows connections.

WebSocket frames do not include HTTP headers, thus the origin host is sent to the server in an upgrade-request, which is a normal HTTP request. It is the responsibility of the service provider to check the origin host and either allow or block the connection. The verified-origin policy does not actually prevent anyone from connecting to the service, since the origin field can be easily spoofed. However, the verified-origin policy shields the client against cross-site request forgery (CSFR) attacks[2].

3.2 Proxies and Firewalls

Proxy traversal and firewalls have been considered in the design principles of the WebSocket protocol. The handshake process is compatible with HTTP and the standard HTTP port 80 is commonly used with the WebSocket protocol. However, known issues exist, especially in proxy traversal through transparent web proxies [1, 8]. Moreover, a vulnerability involved to the WebSocket protocol was reported[3] in November 2010. Malicious usage of a WebSocket channel allowed cache poisoning of some transparent web proxies. The web browser vendors disabled[4] WebSockets until the WebSocket protocol working group introduced frame-masking to avoid the vulnerability [4, 5]. Frame-masking prevents WebSocket clients and servers from injecting malicious content in network intermediaries, since the contents of WebSocket frames are not delivered in plain text. In practice, frame-masking is implemented with a 32-bit random nonce that is delivered at the beginning of each frame. It is done by masking byte n in the payload data with the byte $n \ mod \ 4$ of the nonce using XOR operation [5].

The lack of HTTP headers in WebSocket frames may potentially become an issue also for firewalls and for malware scanners. Advanced firewalls and malware detection tools that classify and process the data according to the protocol and the content type, may not be aware of the WebSocket protocol. Since the WebSocket protocol implements frame-masking to prevent proxy cache poisoning, this also inhibits firewalls and anti-virus tools from analyzing the data patterns and detecting the malicious content [13]. The lack of metadata, such as HTTP headers, complicates the malware analysis on WebSocket frames even more [10, 13].

The WebSocket protocol does not specify the format of payload data. Hence, a developer can specify a custom protocol or use an existing application-level protocol when communicating through a WebSocket. However, the WebSocket protocol specifies a header "Sec-WebSocket-Protocol", which can be used during the handshake to advertise the application-level protocol used on WebSockets. This allows firewalls and routers to set the security policies accordingly[5] – provided that those are aware of the WebSocket protocol. However, the threat is that the malicious content sent over a WebSocket may bypass the firewalls and anti-virus tools, and harm the end-user.

[2]http://learnitcorrect.com/blog/websocket-is-great-but-not-the-origin-policy.html

[3]http://www.ietf.org/mail-archive/web/hybi/current/msg04744.html

[4]https://hacks.mozilla.org/2010/12/websockets-disabled-in-firefox-4/

[5]http://blog.kaazing.com/2012/02/28/html5-websocket-security-is-strong/

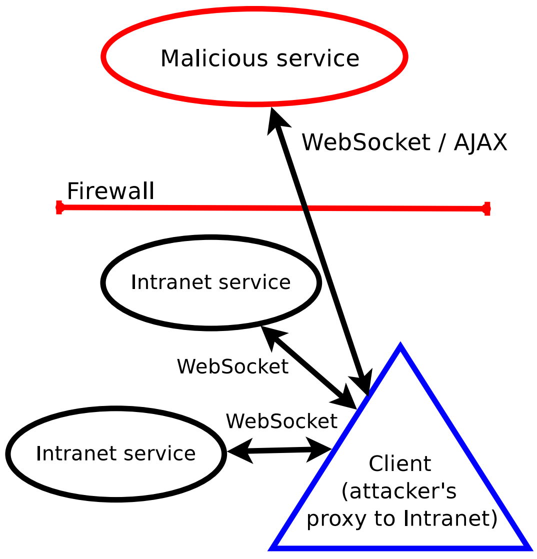

Figure 2: A malicious server may be able to access behind a firewall through a WebSocket client.

3.3 Malicious services

A malicious website can easily set up a remote shell access to the victim's web browser. Once the attacker is able to run arbitrary JavaScript code inside the web browser, she is also able to initiate a WebSocket connection to an arbitrary service. After this, the attacker can utilize the existing WebSocket channel to control the web browser in real-time within the limits of JavaScript [14]. Listing 3 illustrates an example of this type of attack.

```
var socket = new WebSocket
  ("ws://malicious-service.invalid:80");
socket.onmessage = function(e) {
  eval(e.data);
};
```

Listing 3: Example of the browser shell access using WebSocket

The WebSocket API does not only allow requests to an arbitrary host, but also to an arbitrary TCP port (except the common ports blocked by the browsers[6]). Thus, the technology can be utilized for port scanning and network mapping in the internal network to which the victim is connected [15]. This can be executed by timing analysis, that is, observing the response times from the server when initiating the WebSocket handshakes[7]. For instance, if the remote server is listening a certain TCP port, a WebSocket connection attempt to that port initiates a TCP handshake. Since TCP handshake requires couple of round-trips, web application may be able to distinguish open and blocked ports on the remote server according to the response time. [6].

Hence, the attacker may be able to bypass the firewalls by setting the victim's web browser to work as a WebSocket proxy between the attacker and the internal network [15]. The concept is illustrated in Figure 2. A live example of a network and port scanning tool using the WebSocket API is

[6]http://www-archive.mozilla.org/projects/netlib/PortBanning.html

[7]http://www.andlabs.org/tools/jsrecon/jsrecon.html

available in the web[8].

Another common type of attack is denial of service (DoS). Here, a client or a server is flooded with data or connection requests to such a degree that the endpoint is unable to handle all the requests. This may cause a temporary unavailability of the service or a crashing of the attacked application. In addition, by hijacking a large number of web clients, an attacker may be able to implement a distributed denial of service attack against third parties by using WebSockets. Since the limit of the simultaneous WebSocket connections is relatively high – between 900 and 3000 in the most browsers[9] – WebSocket may be a powerful technology for such activity.

All of these types of attacks naturally require either tricking the victim to visit a malicious website, or injecting malicious code to some trusted website. This can be achieved, for instance, by using XSS vulnerabilities which are discussed next.

3.4 XSS vulnerabilties

Cross-site scripting (XSS) is a common type of security vulnerability in web applications. The XSS vulnerabilities allow the attacker to inject malicious code in websites that display parameters or variables that can be modified by user. For instance, the following script includes a so called reflected XSS vulnerability, which can be exploited by injecting JavaScript or HTML code in the GET parameter *name*.

```php
<?php
print "Hello, ".$_GET['name'];
?>
```

Listing 4: An XSS vulnerability in a dynamic web page

The XSS vulnerabilities are common in the web and their existence does not depend on whether the WebSockets are used or not. Thus, many kinds of web services may include XSS vulnerabilities and there are several ways to attack users through an XSS vulnerability.

However, on the websites that utilize WebSockets, the XSS vulnerabilities open up several new threats. For instance, with an XSS vulnerability the attacker may be able to override the callback functions of a WebSocket connection with custom ones. This approach allows the attacker to sniff the traffic, manipulate the data, or implement a man-in-the-middle attack against WebSocket connections. In addition, by utilizing an XSS vulnerability, the attacker is able to implement practically any attack described in the section 3.3 against an unsuspecting web client.

3.5 Encryption and data validation

The WebSocket protocol does not specify mechanisms for authentication or encryption of connections [5]. The protocol specification suggests to use TLS for providing confidential communication channels. Since the the TLS provides transport layer encryption, it also provides confidentiality and integrity for the WebSocket frames. Authentication can be also implemented using TLS or other generic methods for web authentication, such as cookies or HTTP Authentication [5].

Another relevant point is data validation. The WebSocket protocol specifies that both a client and a server must validate the data received from each other. If invalid data is received, the WebSocket connection should be closed [5]. Commonly, WebSocket servers should always assume that clients may not comply with the protocol, and the origin host may be spoofed.

3.6 Lack of official standards

Other issues with WebSockets result from the unfinished standards and the early implementations of the WebSocket protocol and the API. For instance, the WebSocket protocol defines that only encrypted WebSocket connections should be initiated from a TLS-secured website. However, currently only Mozilla Firefox complies with this policy [13]. Essentially, this is same issue as loading plain HTTP content in TLS-secured website, which is commonly either disabled or reported to the user.

Also, since the specifications are not standardized yet, both the API and the protocol might change in future. This may also slow down the adaption of the WebSocket protocol in firewalls, network analysis tools and proxies. Hence, when deploying a service using WebSockets it is important to comply with the latest specifications and to keep the implementations up to date.

4 Discussion and analysis

WebSockets provide many advantages but also open up potential security issues. Some of them are rather theoretical and require further studying so that we could understand them and their countermeasures better. On the other hand, other security issues are more visible and proof-of-concepts have been implemented. Many of the attacks discussed in this paper are not specific to the WebSocket API or the protocol. For instance, denial of service attacks can be implemented on many protocols and practically on any network layer. The WebSocket protocol is simply a one more protocol that can be used for denial of service attacks.

Not all of the security issues are caused by the WebSocket protocol or API specification. The issues are also related to other common network components, such as proxies and firewalls, which do not understand the WebSocket protocol, as well as the web browsers that do not comply with the protocol specification or security policies. However, this is expected to change in the long-term, once the WebSocket protocol has been standardized and adopted widely

Overall, WebSockets provide many advantages for real-time communications. Trustworthy web services must be able to take advantage of them, however, in such a way that the security of the service is not compromised. On the other hand, we can not prevent malicious websites intentionally attacking their visitors. Hence, when analyzing possible solutions for the security issues involving the WebSocket technology, we can divide the problem into two separate parts.

1. Deploying the WebSocket technology in trustworthy

[8] http://andlabs.org/tools/jsrecon.html

[9] https://community.qualys.com/blogs/securitylabs/2012/08/15/would-you-let-your-grandma-use-websockets

web services, while preserving the security of the service and privacy of users.

2. Shielding web browsers and end-users against malicious usage of WebSockets.

The solution proposals for the first problem are covered in section 4.1. In general, TLS encryption and careful implementation provide a security level that meets the requirements of the most services. However, the second issue is more complicated. The problem is to determine, whether a website is using WebSockets against the client or not. If the website or the service is suspicious, the browser may either block the WebSocket connections or warn the user, depending on policy. We analyze these issues more closely in section 4.2

4.1 Deployment recommendations

Deploying WebSockets in a public web service is the responsibility of service developers and administrators. The developers need to consider both security of service back end, such as the servers and databases, as well as privacy of the users. Below are the general guidelines for the safe deployment of a web application utilizing WebSockets. The list is based on the findings of this paper and the “HTML5 Security Cheat Sheet”[10]:

- Both a service and a client should be implemented carefully, strictly following the latest specifications.
- Developers should be aware that XSS vulnerabilities exist, and they can be used to compromise WebSocket based communication.
- To achieve confidentiality and integrity, TLS encryption should be used. If the website initiating the WebSocket connection is delivered over HTTPS, TLS should be used anyway.
- Attention should be paid to implementing the authentication and the session management.
- Endpoints should not trust each other. Thus, both a client and a server should validate all the input received through WebSocket connections.

When possible, it may be a good choice to utilize existing and well-known solutions to implement the critical parts of the WebSocket service. For instance, using a common application level protocol as a subprotocol for the WebSocket communications is reasonable, since implementing a new protocol securely is demanding and requires wide knowledge.

4.2 Analysis of possible solutions

To protect end-users against malicious websites and services, a natural approach would be to always request a permission from the user for opening a WebSocket connection, and letting the user decide whether the connection is trustworthy. However, the vast majority of the users do not understand such technical matters. They would only probably accept the requests to disable the irritating confirmation popups. Moreover, once the WebSocket technology is widely adopted, it is not practical to request permissions all the time.

Another possible solution would be to implement an intrusion detection system for web browsers. Here, a web browser would maintain a list of requested WebSocket connections and according to destination hosts and ports, try to detect malicious operations such as port scanning or network mapping. For example, web browsers could detect and block WebSocket connection attempts to subsequent IP addresses or network ports. This should be possible, since many firewalls already include such features – albeit implemented on lower level. However, firewalls usually protect networks against inbound connection attempts, whereas WebSockets may be utilized for scanning internal network from inside, as we illustrated in figure 2.

As explained in the section 3.3, port scanning and network mapping with WebSockets is based in response times of WebSocket connection attempts. Hence, port scanning could be hindered by forcing a minimum delay for the responses of WebSocket handshake requests. This inhibits the timing analysis which in turn inhibits distinguishing open and closed ports on the target server. However, it is also a tradeoff between security and performance.

To shield the user from denial of service attacks, we need to limit both the number of simultaneous connections and maximum size of messages. The number of simultaneous WebSocket connections is an implementation-specific issue — and the browser vendors have already set the limits [13]. The protocol specifies that a single WebSocket message may consist of an infinite number of frames, where the frame size is practically unlimited [5]. In addition, the underlying implementation should concatenate the frames of a fragmented message and provide the complete message to the application level. Thus, both servers and clients can easily run out of memory, if this issue is not handled correctly.

Preventing a remote shell access to the web browser is problematic. Because of dynamic nature of JavaScript, automatic analysis and detection of code injection flaws is practically not possible. Another approach could be detecting possible program code received through a WebSocket. However, the data can be masked using a custom algorithm. Also, preventing the usage of JavaScript $eval()$ function with the WebSocket API is inadequate, since the malicious code can be pre-written on the website and invoked depending on the WebSocket communications.

4.3 Solution proposals

Since usually the WebSocket API is used by the web browsers, the browser vendors play an important role. Hence, we propose that web browsers should implement certain security features to guarantee secure browsing for users. In practice, this could be implemented by analyzing the metadata of website and the WebSocket connections it initiates. The decision about the legitimacy of the website could be made according to, for example, following metrics:

- Number of WebSocket connection attempts.

[10]https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

- WebSocket connection attempts to subsequent IP addresses or port numbers.
- WebSocket connection attempts to suspicious destinations, such as localhot or internal network.
- Existence of the origin host in public blacklist or whitelist services.
- Other anomalies in WebSocket communication, such as exceptional size of WebSocket messages or frames.

Monitoring these indicators does not require analysis of the website source code. It would be sufficient to monitor the network connections and to call the whitelist services occasionally. The metrics could be optimized in order to maximize the hit rate and to minimize false positives.

For instance, if a public website, that does not appear in any whitelist service, attempts to open WebSocket connections to IP addresses in internal network, it should be classified as suspicious. On the other hand, a website that exists in a public whitelist service is probably legitimate and can be allowed to use WebSockets. In many cases it may be difficult to determine, whether the connection is legitimate or not. Hence, in case of doubt the web browser could either warn the user about suspicious network connection, or require a permission for the connection by the user. However, this should be rather an exception than a rule.

In addition, as mentioned in previous sections, network scanning may be hindered by forcing a minimum delay by forcing a minimum response time for WebSocket connections. This inhibits detection of open and blocked ports by preventing timing analysis. Also, denial of service attacks could be restricted by lowering the number of allowed WebSocket connections and restricting size of WebSocket messages and frames. Preferably the values should not be hardcoded, since they can be found too insecure or strict.

Overall, WebSockets can be deployed safely, but malicious websites may also intentionally utilize WebSockets against end-users. Open issues, such as JavaScript shell prevention, exist. However, we expect that together with a proper implementation and optimization, these kind of features would enhance the security of web browsers.

5 Conclusion

As we have seen in this paper, the WebSocket technology increases the opportunities of real-time web applications. Nevertheless, the technology is not yet standardized or widely deployed. Although the expectations of HTML5 and the related web technologies have been high, we are yet to see the final breakthrough. However, we expect that in the near future the WebSocket API and the protocol will be more widely adopted, especially in time-critical web applications and services.

Once the WebSocket technology is widely adopted, the security threats will become more concrete. Whereas the WebSocket services can be deployed safely, they also can be utilized for malicious purposes. All the parties, including the WebSocket protocol working group, web browser vendors and service developers, should pay attention to the known security issues. It would seem reasonable to restrict the usage of the WebSocket API in web browsers, in a way or another, in order to prevent attacks on users via WebSocket connections. Overall, the WebSocket protocol solves connectivity problems, not security problems. To address the security issues, a comprehensive security model for the WebSocket protocol and the API is needed to define the responsibilities of the protocol specification, web browsers and service providers.

References

[1] O. Cassetti. Technical Report on Websockets and their interaction with firewall. `http://www.scss.tcd.ie/~casseto/websockets-firewall-proxies.pdf`, March 2011. [Online; received 22 Sep 2012].

[2] O. Cassetti and S. Luz. The WebSocket API as supporting technology for distributed and agent-driven data mining. `http://www.scss.tcd.ie/~casseto/NGDM11-websockets.pdf`, 2011. [Online; received 22 Sep 2012].

[3] R. R. Ganji, M. Mitrea, B. Joveski, and F. Preteux. HTML5 as an application virtualization tool. In *Consumer Electronics (ISCE), 2012 IEEE 16th International Symposium on*, pages 1 –4, June 2012.

[4] L. Huang, E. Chen, A. Barth, E. Rescorla, and C. Jackson. Talking to yourself for fun and profit. *Proceedings of Web 2.0 Security and Privacy 2011*, 2011.

[5] Internet Engineering Task Force (IETF). The WebSocket Protocol, RFC 6455.

[6] L. Kuppan. Attacking with HTML5. *BlackHat WebCast*, December 2010.

[7] R. M. Lerner. At the forge: communication in HTML5. *Linux Journal*, 2011(202), Feb. 2011.

[8] P. Lubbers. How HTML5 Web Sockets Interact With Proxy Servers. `http://www.infoq.com/articles/Web-Sockets-Proxy-Servers`, 2010. [Online; received 16 Oct 2012].

[9] P. Lubbers and F. Greco. HTML5 Web Sockets: A Quantum Leap in Scalability for the Web. `http://www.websocket.org/quantum.html`. [Online; received 25 Sep 2012].

[10] L. McVittie. Oops! HTML5 Does it Again. `https://devcentral.f5.com/weblogs/macvittie/archive/2012/02/15/oops-html5-does-it-again.aspx`, February 2012. [Online; received 17 Oct 2012].

[11] G. Nicolas, K. Sbata, and E. Najm. Websocket enabler: achieving IMS and web services end-to-end convergence. In *Proceedings of the 5th International Conference on Principles, Systems and Applications of*

IP Telecommunications, IPTcomm '11, pages 3:1–3:3, New York, NY, USA, 2011. ACM.

[12] V. Pimentel and B. Nickerson. Communicating and Displaying Real-Time Data with WebSocket. *Internet Computing, IEEE*, 16(4):45 –53, July-Aug. 2012.

[13] M. Schema, S. Shekyan, and V. Toukharian. Hacking with WebSockets. *BlackHat USA 2012*.

[14] M. Schmidt. HTML5 web security V1.0, December 6th, 2011. *Compass Security AG*, pages 23–26, 2011.

[15] S. Shah. HTML5 Top 10 Threats Stealth Attacks and Silent Exploits. *BlackHat Europe 2012*.

[16] A. Wessels, M. Purvis, J. Jackson, and S. Rahman. Remote Data Visualization through WebSockets. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 1050 –1051, April 2011.

[17] World Wide Web Consortium W3C. The WebSocket API: W3C Candidate Recommendation. `http://www.w3.org/TR/websockets/`. [Online; received 25 Sep 2012].