Internet Engineering Taskforce
Internet-Draft
Intended status: Informational
Expires: February 14, 2014

N. Wilson
RealVNC Ltd.

August 13, 2013

Use of the WebSocket Protocol as a Transport for the Remote Framebuffer Protocol

draft-realvnc-websocket-00

Abstract

The Remote Framebuffer protocol (RFB) enables clients to connect to and control remote graphical resources. This document describes a transport for RFB using the WebSocket protocol, and defines a corresponding WebSocket subprotocol, enabling an RFB server to offer resources to clients with WebSocket connectivity, such as web-browsers.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 14, 2014.

Copyright Notice

Table of Contents

1. Introduction

1.1. Background

This section is non-normative.

The WebSocket Protocol [RFC6455] provides a reliable, full-duplex, message-oriented transport. The opening handshake is formatted as an HTTP request and response, enabling access to resources through intermediaries obeying HTTP semantics, such as proxies. This enables resources served over a WebSocket-based transport to be accessible to all web user-agents.

In addition, although untrusted websites cannot be given a mechanism to make arbitrary TCP connections, web-browsers are able to offer web

resources such as JavaScript scripts the ability to make arbitrary connections using the WebSocket protocol. This is because the initial HTTP handshake, performed by the user agent rather than the untrusted web resource, conveys origin context, preventing a remote webpage from accessing servers on the local network unless the server is configured to allow such access.

Therefore, offering RFB resources over a WebSocket-based transport opens access to a variety of devices unable to use the TCP transport described in The RFB Protocol [RFC6143].

The purpose of defining a WebSocket subprotocol is firstly to give endpoints a clear way to indicate how the RFB stream is mapped to WebSocket frames, ensuring compatible transport of the stream by using an agreed mapping. Secondly, using a WebSocket subprotocol enables multiple services to run at once on a single server. Services which run over TCP/IP commonly use a port number allocated for each service to enable multiple services, but the behaviour of HTTP proxies makes it likely that WebSocket servers will commonly be run only on ports 80 and 443. The WebSocket subprotocol mechanism is analogous to the port number system of IP addressing, but a short string naturally associated with the service is to identify it, rather than a number.

1.2. Overview of the WebSocket Protocol as a stream transport

This section is non-normative.

The RFB Protocol [RFC6143], section 7 explains that the protocol may operate over any reliable stream- or message-oriented transport, but only describes the RFB as a stream of octets. This gives a clear mapping for the TCP/IP transport, but for message-oriented transport layers, the encapsulation of the RFB octet-stream must be specified.

In this document, the WebSocket subprotocol for RFB is defined to place no importance on the message boundaries of the WebSocket layer. Instead, WebSocket messages are concatenated to form in each direction an octet-stream.

This is firstly because some RFB messages may be large, such as those containing pixel data, and requiring these to be processed as a single message may be large burden for some clients. The WebSocket API [WSAPI] requires an implementation of the API to buffer the fragments of the WebSocket message until the entire message has been received. Although the RFB server and any WebSocket-aware proxy can fragment the message as it chooses, a client application such as a mobile web-browser may have to consume several megabytes of memory to satisfy the requirements of the WebSocket API.

Secondly, it is advantageous to RFB servers to be able to wrap the RFB stream in WebSocket messages flexibly. As well as being a convenience to implementors of RFB servers, it also enables WebSocket connectivity to be added to legacy software using a proxy. Without requiring knowledge the protocol, and generic proxy may be used which concatenates WebSocket messages received from the WebSocket client to send over TCP to the RFB server, and reads bytes from the RFB server and sends them to the client via WebSocket messages.

2. Definitions

RFB client, server, endpoint: As defined in The RFB Protocol [RFC6143], section 1. An RFB endpoint is an RFB client or server.

WebSocket client, server, endpoint: As described in The WebSocket Protocol [RFC6455], section 1.2.

RFB WebSocket subprotocol: The WebSocket subprotocol (described in [RFC6455] section 1.9) which acts as a transport for the RFB Protocol, as described in this document.

RFB WebSocket client, server, endpoint: An RFB client, server, or endpoint respectively which is also a WebSocket client, server, or endpoint and uses the RFB WebSocket subprotocol as the RFB transport.

3. Interaction with the WebSocket Protocol

The WebSocket Protocol contains a number of features not present in TCP. These are discussed here in turn, and their interpretation by RFB entities conforming to the RFB WebSocket subprotocol.

3.1. The "Sec-WebSocket-Protocol" header

The WebSocket Protocol [RFC6455] section 4, "Opening Handshake", describes the use of the "Sec-WebSocket-Protocol" header to indicate negotiation of a WebSocket subprotocol. The requirements of this section as described by the key words "MUST", "SHOULD", and so on, are not superseded by use of the RFB WebSocket subprotocol. A WebSocket client aware of the RFB WebSocket subprotocol may choose to request the subprotocol by including the token "rfb" in the "Sec-WebSocket-Protocol" header in its request. A WebSocket server aware of the RFB WebSocket subprotocol may choose to respond to such a request by including a "Sec-WebSocket-Protocol" header in its response containing the token "rfb".

The interpretation of any data following the opening WebSocket handshake is determined by any subprotocols in effect. If the RFB

WebSocket subprotocol was not requested by the client or was not selected by the server, then this document does not place any interpretation on the subsequent data. In particular, if a client requests any subprotocol but the server not include it in its response, the client need not assume any particular meaning for the data that follows. This is because WebSocket servers are likely to ignore requsts for any unknown subprotocols and proceed. If the WebSocket client requires use of a particular subprotocol, it is its responsibility to close the connection if use of the subprotocol was not successfully negotiated.

The RFB WebSocket subprotocol does not place any restrictions on use of the subprotocol alongside WebSocket extensions. (Note that only one subprotocol may be used by a WebSocket connection.) The effect of any such extensions is outside the scope of this document.

3.2. Close Frames

When the RFB WebSocket subprotocol is in use, the status code and reason of any WebSocket Close frames relate only to the WebSocket transport, not the RFB stream using the transport. The WebSocket connection will normally be closed by a status code 1000 ("Normal Closure") or 1001 ("Going Away"). Any status code or reason sent by the WebSocket client or server SHOULD NOT convey RFB-specific information. No status codes in the private use range 4000-4999 are defined by this subprotocol. No mapping is provided between WebSocket Close frame status codes and strings using RFB messages.

Any RFB-specific close data MAY be conveyed using an appropriate RFB message. For example, in the case of an RFB authentication failure, the close condition may be conveyed using an RFB SecurityResult message as appropriate, after which the WebSocket connection may be closed using a Close frame status code indicating success. As long as there were no errors in the transport, the WebSocket Close frame does not use a status code indicating failure, even though the RFB connection failed to be established, because the RFB error was conveyed as application data over the WebSocket transport.

The meaning of any status codes used in Close frames MUST refer to the state of the WebSocket protocol, for status codes defined in the WebSocket Protocol and any subsequent versions or other specifications registered by the IANA in the Close Code Number Registry. For example, the status code 1002 ("Protocol Error") describes errors in the WebSocket protocol and not an error in the RFB stream carried by the transport.

3.3. Data Frames

The RFB octet-stream is transported using Data frames with opcode 0x2 (Binary). When the RFB WebSocket subprotocol is in use and no WebSocket extensions are in use, WebSocket clients MUST send RFB data using Binary messages.

RFB WebSocket subprotocol does not specify any multiplexing of connections or interleaving of data with other streams. Where no WebSocket extensions are in use, RFB WebSocket clients MUST use Binary messages exclusively for RFB data, such that the octets from the ordered stream of Binary WebSocket messages when truncated conform with the description given in the RFB Protocol [RFC6143].

The frame boundaries do not have to be aligned in any way with the RFB stream. RFB WebSocket endpoints, when receiving messages, MUST NOT vary their behaviour based on the framing of the RFB stream using WebSocket messages. It is suggested that RFB WebSocket endpoints avoid sending empty messages, and that endpoints impose a suitable limit on the size of the messages they send to avoid placing unnecessary load on clients.

The interpration of Text messages (with opcode 0x1) is unspecified. RFB WebSocket endpoints SHOULD NOT send Text messages, but if a WebSocket extension is in use which uses these messages they may be sent. An RFB WebSocket client receiving such a message SHOULD fail the WebSocket connection (as defined in section 7.1.7 of [RFC6455]) except where any mechanism has been used to negotiate a meaning for these messages. In general, WebSocket extensions may modify the interpretation of data, and as appropriate each the definition of each extension must specify how it interacts with application data using Binary messages in order to be compatible with the RFB WebSocket subprotocol, which is beyond the scope of this document.

4. Versioning Considerations

The RFB WebSocket subprotocol is identified by the token "rfb". This token contains no version component, since the RFB protocol is already versioned in its initial handshake. The definition of this subprotocol makes no reference to the specific format of messages in RFB 3.8, so is applicable to subsequent versions of the RFB protocol.

5. IANA Considerations

RFC Editor Note: Please set the RFC number assigned for this document in the sub-sections below and remove this note.

5.1. Registration of the RFB WebSocket Subprotocol

This specification describes a WebSocket subprotocol registered in the WebSocket Subprotocol Name Registry defined in [RFC6455], section 11.5.

Subprotocol Identifier: "rfb"

Subprotocol Common Name: RFB

Subprotocol Definition: RFC??? (this document)

6. Security Considerations

6.1. Origin checking

Using the WebSocket protocol as a transport presents fresh challenges, since the connections can be created by untrusted resources which originate outside the local subnetwork and have traversed any firewalls in place. This differs from TCP connections. For example, an RFB server accessible over TCP on the local subnetwork may be configured on the assumption that connections originate inside the trusted subnet, and this assumption may be enforced using a firewall. To make a connection, any client has to have already gained access to the subnet.

This is not the case for a RFB server which accepts connections over the WebSocket protocol, which is specifically designed so that it is safe to allow untrusted resources to make WebSocket connections, on the assumption that WebSocket servers carefully enforce any applicable restrictions on the origin of content. In the TCP example, the RFB server does not need to enforce the restriction that connections originate inside the subnet, as this is implemented using the firewall. A web-browser running on a machine in the subnet may open up WebSocket connections though based on scripts loaded from any source at all on a webpage, originating outside the subnet. The web-browser is only able to allow the script to do this on the basis that the Origin header it sends conveys enough information for the WebSocket server to apply any policies and decide if the connection is to be accepted.

Therefore, any WebSocket server implementers must carefully consider the implications of opening up access to resources via the WebSocket Protocol. In the case of an RFB server which is accessible over TCP as well as the RFB WebSocket subprotocol, the TCP connection may be hidden behind a firewall or NAT or for any other reason may be not publicly accessible on the internet. In this case, the origin restrictions in place for the TCP connections should be enforced for the WebSocket server also, or else clearly documented in such a way that administrators of the software do not misunderstand the scope of who can connect in to the server.

Unless all WebSocket software that runs in a LAN environment is implemented to enforce these restrictions, web-browsers vendors may not be able to justify continuing to permit untrusted web resources (JavaScript) to make WebSocket connections.

6.2. Encryption

Where applicable, the Secure WebSocket Protocol (using the WebSocket Protocol over TLS [RFC5246]) may be used. However, it is not practicable in all circumstances to provision many dynamic RFB servers on a LAN with a certificate which browsers can verify, so implementors may choose to perform encryption at the application level using an encrypting RFB Security Type, and verify the peer using identities which can be verified by the RFB implementation rather than the browser.

6.3. Creating a Safe JavaScript Environment

Many of the RFB clients using WebSockets are likely to be implemented in JavaScript and executed by web-browsers. In this case, implementors must be aware of the difficulties of executing JavaScript in a safe context. Banners and other resources loaded alongside the page may substitute functions into top-level objects and subvert the security of the connection or skim passwords. When implementing any application which prompts for a user's password or sends and receives data which may be sensitive, the application must be loaded from a safe context, such as a web page served over HTTPS, and which loads no untrusted external resources. Certain operations required for encryption, such as secure random number generation, may require browser support such as the Web Cryptography API [WCAPI].

7. Acknowledgements

Thanks to Pierre Garnero of Visteon for feedback during drafting.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6143] Richardson, T. and J. Levine, "The Remote Framebuffer Protocol", RFC 6143, March 2011.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

8.2. Informative References

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[WCAPI] Dahl, D., Ed. and R. Sleevi, Ed., "Web Cryptography API, W3C Working Draft", June 2013.

[WSAPI] Hickson, I., Ed., "The WebSocket API", April 2013.

Author's Address

Nicholas Wilson
RealVNC Ltd.
Betjeman House, 104 Hills Road
Cambridge CB2 1LQ
UK

Phone: +44 1223 310411
Email: ncw@realvnc.com