Stream:	Internet Engineering Task Force (IETF)
RFC:	8828
Category:	Standards Track
Published:	July 2020
ISSN:	2070-1721
Author:	J. Uberti
	Google

RFC 8828
WebRTC IP Address Handling Requirements

Abstract

This document provides information and requirements for how IP addresses should be handled by Web Real-Time Communication (WebRTC) implementations.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

One of WebRTC's key features is its support of peer-to-peer connections. However, when establishing such a connection, which involves connection attempts from various IP addresses, WebRTC may allow a web application to learn additional information about the user compared to an application that only uses the Hypertext Transfer Protocol (HTTP) [RFC7230]. This may be problematic in certain cases. This document summarizes the concerns and makes recommendations on how WebRTC implementations should best handle the trade-off between privacy and media performance.

2. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Problem Statement

In order to establish a peer-to-peer connection, WebRTC implementations use Interactive Connectivity Establishment (ICE) [RFC8445]. ICE attempts to discover multiple IP addresses using techniques such as Session Traversal Utilities for NAT (STUN) [RFC5389] and Traversal Using Relays around NAT (TURN) [RFC5766] and then checks the connectivity of each local-address-remote-address pair in order to select the best one. The addresses that are collected usually consist of an endpoint's private physical or virtual addresses and its public Internet addresses.

These addresses are provided to the web application so that they can be communicated to the remote endpoint for its checks. This allows the application to learn more about the local network configuration than it would from a typical HTTP scenario, in which the web server would only see a single public Internet address, i.e., the address from which the HTTP request was sent.

The additional information revealed falls into three categories:

1. If the client is multihomed, additional public IP addresses for the client can be learned. In particular, if the client tries to hide its physical location through a Virtual Private Network (VPN), and the VPN and local OS support routing over multiple interfaces (a "split-tunnel" VPN), WebRTC can discover not only the public address for the VPN, but also the ISP public address over which the VPN is running.
2. If the client is behind a Network Address Translator (NAT), the client's private IP addresses, often [RFC1918] addresses, can be learned.
3. If the client is behind a proxy (a client-configured "classical application proxy", as defined in [RFC1919], Section 3), but direct access to the Internet is permitted, WebRTC's STUN checks will bypass the proxy and reveal the public IP address of the client. This concern also applies to the "enterprise TURN server" scenario described in [RFC7478], Section 2.3.5.1 if, as above, direct Internet access is permitted. However, when the term "proxy" is used in this document, it is always in reference to an [RFC1919] proxy server.

Of these three concerns, the first is the most significant, because for some users, the purpose of using a VPN is for anonymity. However, different VPN users will have different needs, and some VPN users (e.g., corporate VPN users) may in fact prefer WebRTC to send media traffic directly -- i.e., not through the VPN.

The second concern is less significant but valid nonetheless. The core issue is that web applications can learn about addresses that are not exposed to the Internet; typically, these address are IPv4, but they can also be IPv6, as in the case of NAT64 [RFC6146]. While disclosure of the [RFC4941] IPv6 addresses recommended by [RFC8835] is fairly benign due to their intentionally short lifetimes, IPv4 addresses present some challenges. Although private IPv4 addresses often contain minimal entropy (e.g., 192.168.0.2, a fairly common address), in the worst case, they can contain 24 bits of entropy with an indefinite lifetime. As such, they can be a fairly significant fingerprinting surface. In addition, intranet web sites can be attacked more easily when their IPv4 address range is externally known.

Private IP addresses can also act as an identifier that allows web applications running in isolated browsing contexts (e.g., normal and private browsing) to learn that they are running on the same device. This could allow the application sessions to be correlated, defeating some of the privacy protections provided by isolation. It should be noted that private addresses are just one potential mechanism for this correlation and this is an area for further study.

The third concern is the least common, as proxy administrators can already control this behavior through organizational firewall policy, and generally, forcing WebRTC traffic through a proxy server will have negative effects on both the proxy and media quality.

Note also that these concerns predate WebRTC; Adobe Flash Player has provided similar functionality since the introduction of Real-Time Media Flow Protocol (RTMFP) support [RFC7016] in 2008.

4. Goals

WebRTC's support of secure peer-to-peer connections facilitates deployment of decentralized systems, which can have privacy benefits. As a result, blunt solutions that disable WebRTC or make it significantly harder to use are undesirable. This document takes a more nuanced approach, with the following goals:

- Provide a framework for understanding the problem so that controls might be provided to make different trade-offs regarding performance and privacy concerns with WebRTC.
- Using that framework, define settings that enable peer-to-peer communications, each with a different balance between performance and privacy.
- Finally, provide recommendations for default settings that provide reasonable performance without also exposing addressing information in a way that might violate user expectations.

5. Detailed Design

5.1. Principles

The key principles for our framework are stated below:

1. By default, WebRTC traffic should follow typical IP routing (i.e., WebRTC should use the same interface used for HTTP traffic) and only the system's 'typical' public addresses (or those of an enterprise TURN server, if present) should be visible to the application. However, in the interest of optimal media quality, it should be possible to enable WebRTC to make use of all network interfaces to determine the ideal route.

2. By default, WebRTC should be able to negotiate direct peer-to-peer connections between endpoints (i.e., without traversing a NAT or relay server) when such connections are possible. This ensures that applications that need true peer-to-peer routing for bandwidth or latency reasons can operate successfully.
3. It should be possible to configure WebRTC to not disclose private local IP addresses, to avoid the issues associated with web applications learning such addresses. This document does not require this to be the default state, as there is no currently defined mechanism that can satisfy this requirement as well as the aforementioned requirement to allow direct peer-to-peer connections.
4. By default, WebRTC traffic should not be sent through proxy servers, due to the media-quality problems associated with sending WebRTC traffic over TCP, which is almost always used when communicating with such proxies, as well as proxy performance issues that may result from proxying WebRTC's long-lived, high-bandwidth connections. However, it should be possible to force WebRTC to send its traffic through a configured proxy if desired.

5.2. Modes and Recommendations

Based on these ideas, we define four specific modes of WebRTC behavior, reflecting different media quality/privacy trade-offs:

Mode 1 - Enumerate all addresses:
WebRTC **MUST** use all network interfaces to attempt communication with STUN servers, TURN servers, or peers. This will converge on the best media path and is ideal when media performance is the highest priority, but it discloses the most information.

Mode 2 - Default route + associated local addresses:
WebRTC **MUST** follow the kernel routing table rules, which will typically cause media packets to take the same route as the application's HTTP traffic. If an enterprise TURN server is present, the preferred route **MUST** be through this TURN server. Once an interface has been chosen, the private IPv4 and IPv6 addresses associated with this interface **MUST** be discovered and provided to the application as host candidates. This ensures that direct connections can still be established in this mode.

Mode 3 - Default route only:
This is the same as Mode 2, except that the associated private addresses **MUST NOT** be provided; the only IP addresses gathered are those discovered via mechanisms like STUN and TURN (on the default route). This may cause traffic to hairpin through a NAT, fall back to an application TURN server, or fail altogether, with resulting quality implications.

Mode 4 - Force proxy:
This is the same as Mode 3, but when the application's HTTP traffic is sent through a proxy, WebRTC media traffic **MUST** also be proxied. If the proxy does not support UDP (as is the case for all HTTP and most SOCKS [RFC1928] proxies), or the WebRTC implementation does not support UDP proxying, the use of UDP will be disabled, and TCP will be used to send

and receive media through the proxy. Use of TCP will result in reduced media quality, in addition to any performance considerations associated with sending all WebRTC media through the proxy server.

Mode 1 **MUST NOT** be used unless user consent has been provided. The details of this consent are left to the implementation; one potential mechanism is to tie this consent to getUserMedia (device permissions) consent, described in [RFC8827], Section 6.2. Alternatively, implementations can provide a specific mechanism to obtain user consent.

In cases where user consent has not been obtained, Mode 2 **SHOULD** be used.

These defaults provide a reasonable trade-off that permits trusted WebRTC applications to achieve optimal network performance but gives applications without consent (e.g., 1-way streaming or data-channel applications) only the minimum information needed to achieve direct connections, as defined in Mode 2. However, implementations **MAY** choose stricter modes if desired, e.g., if a user indicates they want all WebRTC traffic to follow the default route.

Future documents may define additional modes and/or update the recommended default modes.

Note that the suggested defaults can still be used even for organizations that want all external WebRTC traffic to traverse a proxy or enterprise TURN server, simply by setting an organizational firewall policy that allows WebRTC traffic to only leave through the proxy or TURN server. This provides a way to ensure the proxy or TURN server is used for any external traffic but still allows direct connections (and, in the proxy case, avoids the performance issues associated with forcing media through said proxy) for intra-organization traffic.

6. Implementation Guidance

This section provides guidance to WebRTC implementations on how to implement the policies described above.

6.1. Ensuring Normal Routing

When trying to follow typical IP routing, as required by Modes 2 and 3, the simplest approach is to bind() the sockets used for peer-to-peer connections to the wildcard addresses (0.0.0.0 for IPv4, :: for IPv6), which allows the OS to route WebRTC traffic the same way as it would HTTP traffic. STUN and TURN will work as usual, and host candidates can still be determined as mentioned below.

6.2. Determining Associated Local Addresses

When binding to a wildcard address, some extra work is needed to determine the associated local address required by Mode 2, which we define as the source address that would be used for any packets sent to the web application host (assuming that UDP and TCP get the same routing treatment). Use of the web-application host as a destination ensures the right source address is selected, regardless of where the application resides (e.g., on an intranet).

First, the appropriate remote IPv4/IPv6 address is obtained by resolving the host component of the web application URI [RFC3986]. If the client is behind a proxy and cannot resolve these IPs via DNS, the address of the proxy can be used instead. Or, if the web application was loaded from a file:// URI [RFC8089] rather than over the network, the implementation can fall back to a well-known DNS name or IP address.

Once a suitable remote IP has been determined, the implementation can create a UDP socket, bind() it to the appropriate wildcard address, and then connect() to the remote IP. Generally, this results in the socket being assigned a local address based on the kernel routing table, without sending any packets over the network.

Finally, the socket can be queried using getsockname() or the equivalent to determine the appropriate local address.

7. Application Guidance

The recommendations mentioned in this document may cause certain WebRTC applications to malfunction. In order to be robust in all scenarios, the following guidelines are provided for applications:

- Applications **SHOULD** deploy a TURN server with support for both UDP and TCP connections to the server. This ensures that connectivity can still be established, even when Mode 3 or 4 is in use, assuming the TURN server can be reached.
- Applications **SHOULD** detect when they don't have access to the full set of ICE candidates by checking for the presence of host candidates. If no host candidates are present, Mode 3 or 4 is in use; this knowledge can be useful for diagnostic purposes.

8. Security Considerations

This document describes several potential privacy and security concerns associated with WebRTC peer-to-peer connections and provides mechanisms and recommendations for WebRTC implementations to address these concerns.

9. IANA Considerations

This document has no IANA actions.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <https://www.rfc-editor.org/info/rfc5389>.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, DOI 10.17487/RFC5766, April 2010, <https://www.rfc-editor.org/info/rfc5766>.

[RFC8089] Kerwin, M., "The "file" URI Scheme", RFC 8089, DOI 10.17487/RFC8089, February 2017, <https://www.rfc-editor.org/info/rfc8089>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

10.2. Informative References

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, DOI 10.17487/RFC1919, March 1996, <https://www.rfc-editor.org/info/rfc1919>.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <https://www.rfc-editor.org/info/rfc1928>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <https://www.rfc-editor.org/info/rfc6146>.

[RFC7016] Thornburgh, M., "Adobe's Secure Real-Time Media Flow Protocol", RFC 7016, DOI 10.17487/RFC7016, November 2013, <https://www.rfc-editor.org/info/rfc7016>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7478] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <https://www.rfc-editor.org/info/rfc7478>.

[RFC8827] Rescorla, E., "WebRTC Security Architecture", RFC 8827, DOI 10.17487/RFC8827, July 2020, <https://www.rfc-editor.org/info/rfc8827>.

[RFC8835] Alvestrand, H., "Transports for WebRTC", RFC 8835, DOI 10.17487/RFC8835, July 2020, <https://www.rfc-editor.org/info/rfc8835>.

Acknowledgements

Several people provided input into this document, including Bernard Aboba, Harald Alvestrand, Youenn Fablet, Ted Hardie, Matthew Kaufmann, Eric Rescorla, Adam Roach, and Martin Thomson.

Author's Address

Justin Uberti
Google
747 6th St S
Kirkland, WA 98033
United States of America
Email: justin@uberti.name