Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 2, 2018

J. Uberti
Google
G. Shieh
Facebook

March 1, 2018

WebRTC IP Address Handling Requirements
draft-ietf-rtcweb-ip-handling-06

Abstract

This document provides information and requirements for how IP addresses should be handled by 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 tradeoff between privacy and media performance.

2. Terminology

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 [RFC2119].

3. Problem Statement

In order to establish a peer-to-peer connection, WebRTC implementations use Interactive Connectivity Establishment (ICE) [RFC5245], which 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/virtual addresses and its public Internet addresses.

These addresses are exposed upwards 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 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 will 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 also supported, WebRTC's STUN checks will bypass the proxy and reveal the public IP address of the client.

Of these three concerns, #1 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.

#2 is considered to be a less significant concern, given that the local address values often contain minimal information (e.g., 192.168.0.2), or have built-in privacy protection (e.g., the [RFC4941] IPv6 addresses recommended by [I-D.ietf-rtcweb-transports]).

#3 is the least common concern, 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 on media quality.

Note also that these concerns predate WebRTC; Adobe Flash Player has provided similar functionality since the introduction of RTMFP [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, we want to avoid blunt solutions that disable WebRTC or make it significantly harder to use. 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 tradeoffs 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 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), by providing a minimal set of local IP addresses to the application for use in the ICE process. This ensures that applications that need true peer-to-peer routing for bandwidth or latency reasons can operate successfully. However, it should be possible to suppress these addresses (with the resultant impact on direct connections) if desired.

3. 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 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 tradeoffs:

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. In addition, the private IPv4 and IPv6 addresses associated with the kernel-chosen interface MUST be discovered and provided to the application. This ensures that direct connections can still be established in this mode.

Mode 3: Default route only: This is the 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 all WebRTC media traffic is forced through a proxy, if one is configured. 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 only be used when 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 consent.

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

These defaults provide a reasonable tradeoff 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.

Note that the suggested defaults can still be used even for organizations that want all external WebRTC traffic to traverse a proxy, simply by setting an organizational firewall policy that allows WebRTC traffic to only leave through the proxy. This provides a way to ensure the proxy is used for any external traffic, but avoids the performance issues associated with Mode 4 (where all media is forced 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, 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 Host Candidates

When binding to a wildcard address, some extra work is needed to determine a suitable host candidate, 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). 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 tell it to 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 host candidate.

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 are 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 above is in use; this knowledge can be useful for diagnostic purposes.

8. Security Considerations

This document is entirely devoted to security considerations.

9. IANA Considerations

This document requires no actions from IANA.

10. Acknowledgements

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

11. References

11.1. 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>.

11.2. Informative References

[I-D.ietf-rtcweb-transports]
Alvestrand, H., "Transports for WebRTC", draft-ietf-rtcweb-transports-17 (work in progress), October 2016.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., 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>.

[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>.

[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>.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <https://www.rfc-editor.org/info/rfc5245>.

[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>.

[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>.

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

Appendix A. Change log

Changes in draft -06:

o Clarify recommendations.

o Split implementation guidance into two sections.

Changes in draft -05:

o Separated framework definition from implementation techniques.

o Removed RETURN references.

o Use origin when determining local IPs, rather than a well-known IP.

Changes in draft -04:

o Rewording and cleanup in abstract, intro, and problem statement.

o Added 2119 boilerplate.

o Fixed weird reference spacing.

o Expanded acronyms on first use.

o Removed 8.8.8.8 mention.

- Removed mention of future browser considerations.

Changes in draft -03:

- Clarified when to use which modes.

- Added 2119 qualifiers to make normative statements.

- Defined 'proxy'.

- Mentioned split tunnels in problem statement.

Changes in draft -02:

- Recommendations -> Requirements

- Updated text regarding consent.

Changes in draft -01:

- Incorporated feedback from Adam Roach; changes to discussion of cam/mic permission, as well as use of proxies, and various editorial changes.

- Added several more references.

Changes in draft -00:

- Published as WG draft.

Authors' Addresses

Justin Uberti
Google
747 6th St S
Kirkland, WA 98033
USA

Email: justin@uberti.name

Guo-wei Shieh
Facebook
1101 Dexter Ave
Seattle, WA 98109
USA

Email: guoweis@facebook.com