Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 18, 2017

J. Uberti
G. Shieh
Google

January 14, 2017

WebRTC IP Address Handling Requirements
draft-ietf-rtcweb-ip-handling-03

Abstract

This document provides information and requirements for how IP addresses should be handled by WebRTC applications.

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 July 18, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
2. Problem Statement 2
3. Goals 3
4. Detailed Design 4
5. Application Guidance 6
6. Security Considerations 6
7. IANA Considerations 6
8. Acknowledgements 6
9. Informative References 6
Appendix A. Change log 8
Authors' Addresses 8

1. Introduction

As a technology that supports peer-to-peer connections, WebRTC may send data over different network paths than the path used for HTTP traffic. This may allow a web application to learn additional information about the user, which may be problematic in certain cases. This document summarizes the concerns, and makes recommendations on how best to handle the tradeoff between privacy and media performance.

2. Problem Statement

WebRTC enables real-time peer-to-peer communications by enumerating network interfaces and discovering the best route through the ICE [RFC5245]protocol. During the ICE process, the peers involved in a session gather and exchange all the IP addresses they can discover, so that the connectivity of each IP pair can be checked, and the best path chosen. The addresses that are gathered 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. 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 behind a NAT, the client's private IP addresses, typically [RFC1918]addresses, can be learned.

2. If the client tries to hide its physical location through a VPN, and the VPN and local OS support routing over multiple interfaces (i.e., a "split-tunnel" VPN), WebRTC will discover the public address for the VPN as well as the ISP public address that the VPN runs over.

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 [RFC5389]checks will bypass the proxy and reveal the public address of the client.

Of these three concerns, #2 is the most significant concern, since 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.

#3 is a less common concern, as proxy administrators can control this behavior through organization firewall policy if desired, coupled with the fact that forcing WebRTC traffic through a proxy will have negative effects on both the proxy and on media quality. For situations where this is an important consideration, use of a RETURN proxy, as described below, can be an effective solution.

#1 is considered to be the least 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. [RFC4941]IPv6 addresses).

Note also that these concerns predate WebRTC; Adobe Flash Player has provided similar functionality since the introduction of RTMFP [RFC7016]in 2008.

3. Goals

Being peer-to-peer, WebRTC represents a privacy-enabling technology, and therefore we want to avoid solutions that disable WebRTC or make it harder to use. This means that WebRTC should be configured by default to only reveal the minimum amount of information needed to establish a performant WebRTC session, while providing options to reveal additional information upon user consent, or further limit this information if the user has specifically requested this. Specifically, WebRTC should:

- Provide a privacy-friendly default behavior which strikes the right balance between privacy and media performance for most users and use cases.

- For users who care more about one versus the other, provide a means to customize the experience.

4. Detailed Design

The key principles for the design are listed below:

1. By default, WebRTC should follow normal IP routing rules, to the extent that this is easy to determine (i.e., not considering proxies). This can be accomplished by binding local sockets 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, and allows only the 'typical' public addresses to be discovered.

2. By default, support for direct connections between hosts (i.e., without traversing a NAT or relay server) should be maintained. To accomplish this, the local IPv4 and IPv6 addresses of the interface used for outgoing STUN traffic should still be surfaced as candidates, even when binding to the wildcard addresses as mentioned above. The appropriate addresses here can be discovered by the common trick of binding sockets to the wildcard addresses, connect()ing those sockets to some well-known public IP address (one particular example being "8.8.8.8"), and then reading the bound local addresses via getsockname(). This approach requires no data exchange; it simply provides a mechanism for applications to retrieve the desired information from the kernel routing table.

3. Determining whether a web proxy is in use is a complex process, as the answer can depend on the exact site or address being contacted. Furthermore, web proxies that support UDP are not widely deployed today. As a result, when WebRTC is made to go through a proxy, it typically needs to use TCP, either ICE-TCP [RFC6544]or TURN-over-TCP [RFC5766]. Naturally, this has attendant costs on media quality as well as proxy performance, and should be avoided where possible.

4. RETURN [I-D.ietf-rtcweb-return]is a proposal for explicit proxying of WebRTC media traffic. When RETURN proxies are deployed, media and STUN checks will go through the proxy, but without the performance issues associated with sending through a typical web proxy.

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 bind to all interfaces individually and use them all 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 (e.g., by binding solely to the wildcard address), which will typically cause media packets to take the same route as the application's HTTP traffic. In addition, any private IPv4 and IPv6 addresses associated with the kernel-chosen interface MUST be discovered through getsockname, as mentioned above, 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 address MUST NOT be provided. This may cause traffic to hairpin through a NAT, fall back to the application TURN server, or fail altogether, with resulting quality implications.

Mode 4: Force proxy: This forces all WebRTC media traffic 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 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; this thwarts the typical drive-by enumeration attacks. 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. This allows applications to still achieve direct connections in many cases, even without consent (e.g., data channel applications). However, user agents MAY choose a stricter default policy in certain circumstances.

Note that when a RETURN proxy is configured for the interface associated with the default route, Mode 2 and 3 will cause any external media traffic to go through the RETURN proxy. While the RETURN approach gives the best performance, a similar result can be achieved for non-RETURN proxies via an organization firewall policy

that only allows external WebRTC traffic to leave through the proxy (typically, over TCP). This provides a way to ensure the proxy is used for any external traffic, but avoids the performance issues of Mode 4, where all media is forced through said proxy, for intra-organization traffic.

5. 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.

- Future versions of browsers may present an indicator to signify that the page is using WebRTC to set up a peer-to-peer connection. Applications MUST only use WebRTC in a fashion that is consistent with user expectations.

6. Security Considerations

This document is entirely devoted to security considerations.

7. IANA Considerations

This document requires no actions from IANA.

8. Acknowledgements

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

9. Informative References

[I-D.ietf-rtcweb-return]
Schwartz, B. and J. Uberti, "Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC", draft-ietf-rtcweb-return-01 (work in progress), January 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, <http://www.rfc-editor.org/info/rfc1918>.

[RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, DOI 10.17487/RFC1919, March 1996, <http://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, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5766>.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <http://www.rfc-editor.org/info/rfc6544>.

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

Appendix A. Change log

Changes in draft -03:

- Clarified when to use which modes.
- Use 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
Google
747 6th St S
Kirkland, WA 98033
USA

Email: guoweis@google.com