Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 21, 2016

J. Uberti
G. Shieh
Google

March 20, 2016

WebRTC IP Address Handling Recommendations
draft-ietf-rtcweb-ip-handling-01

Abstract

This document provides best practices for how IP addresses should be handled by WebRTC applications.

Status of This Memo

Copyright Notice

Table of Contents

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, 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, 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 local 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 main ideas for the design are the following:

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. When used with audio and video devices, WebRTC requires explicit user permission to access those devices. We propose that this permission grant be expanded to include consent to allow WebRTC to access all IP addresses associated with the user agent, for the purpose of finding the absolute best route for media traffic. Combining these permission grants, rather than having the user grant permission individually, is a considered balance; this balance takes into account that the user has placed enough trust into the application to allow it to access their devices, that when doing so the user typically wants to engage in a conversational session, which benefits most from an optimal network path, and lastly, the fact that the underlying issue is complex, and difficult to explain meaningfully to the user.

4. 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 must use TCP, either ICE-TCP [RFC6544] or TURN-over-TCP [RFC5766]. Naturally, this has attendant costs on media quality and also proxy performance.

5. RETURN [I-D.ietf-rtcweb-return] is a new 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 modes of WebRTC behavior, reflecting different privacy/media tradeoffs:

Mode 1: Enumerate all addresses: WebRTC will 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. As such, this should only be performed when the user has explicitly given consent for local media access, as indicated in design idea #3 above.

Mode 2: Default route + the single associated local address: By binding solely to the wildcard address, media packets will follow the kernel routing table rules, which will typically result in the same route as the application's HTTP traffic. In addition, the associated private address will be discovered through getsockname, as mentioned above. This ensures that direct connections can still be established even when local media access is not granted, e.g., for data channel applications.

Mode 3: Default route only: This is the the same as Mode 2, except that the associated private address is not provided, which 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.

We recommend Mode 1 as the default behavior only if cam/mic permission has been granted, or Mode 2 if this is not the case.

Users who prefer Mode 3 or 4 should be able to select a preference or install an extension to force their browser to operate in the specified mode.

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. This provides a way to ensure the proxy is used for external traffic, but without the performance issues of forcing all media through said proxy.

5. Application Guidance

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

- 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 can 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.

- 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 should be careful to 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, and Adam Roach.

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>.

[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 -01:

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

- o Added several more references.

Changes in draft -00:

- o 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