Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 28, 2015

B. Schwartz
Google

August 27, 2014

Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC
draft-schwartz-rtcweb-return-02

Abstract

In the context of WebRTC, the concept of a local TURN proxy has been suggested, but not reviewed in detail. WebRTC applications are already using TURN to enhance connectivity and privacy. This document explains how local TURN proxies and WebRTC applications can work together.

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 28, 2015.

Table of Contents

1. Introduction

TURN [RFC5766] is a protocol for communication between a client and a TURN server, in order to route UDP traffic to and from one or more peers. As noted in [RFC5766], the TURN relay server "typically sits in the public Internet". In a WebRTC context, if a TURN server is to be used, it is typically provided by the application, either to provide connectivity between users whose NATs would otherwise prevent it, or to obscure the identity of the participants by concealing their IP addresses from one another.

In many enterprises, direct UDP transmissions are not permitted between clients on the internal networks and external IP addresses, so media must flow over TCP. To enable WebRTC services in such a situation, clients must use TURN-TCP, or TURN-TLS. These configurations are not ideal: they send all traffic over TCP, which leads to higher latency than would otherwise be necessary, and they force the application provider to operate a TURN server because WebRTC endpoints behind NAT cannot typically act as TCP servers. These configurations may result in especially bad behaviors when operating through TCP or HTTP proxies that were not designed to carry real-time media streams.

To avoid forcing WebRTC media streams through a TCP stage, enterprise network operators may operate a TURN server for their network, which can be discovered by clients using TURN Auto-Discovery [I-D.ietf-tram-turn-server-discovery], or through a proprietary mechanism. Use of the specified TURN server may be the only way for clients on the network to achieve a high quality WebRTC experience. This scenario is required to be supported by the WebRTC requirements document [I-D.ietf-rtcweb-use-cases-and-requirements] Section 3.3.5.1.

When the application intends to use a TURN server for identity cloaking, and the enterprise network administrator intends to use a TURN server for connectivity, there is a conflict. In current WebRTC implementations, TURN can only be used on a single-hop basis in each candidate, but using only the enterprise's TURN server reveals information about the user (e.g. organizational affiliation), and using only the application's TURN server may be blocked by the network administrator, or may require using TURN-TCP or TURN-TLS, resulting in a significant sacrifice in latency.

To resolve this conflict, we introduce Recursively Encapsulated TURN, a procedure that allows a WebRTC endpoint to route traffic through multiple TURN servers, and get improved connectivity and privacy in return.

2. Goals

These goals are requirements on this document (not on implementations of the specification).

2.1. Connectivity

As noted in [I-D.ietf-rtcweb-use-cases-and-requirements] Section 3.3.5.1, a WebRTC browser endpoint MUST be able to direct UDP connections through a designated TURN server configured by enterprise policy (a "proxy").

It MUST be possible to configure a WebRTC endpoint that supports proxies to achieve connectivity no worse than if the endpoint were operating at the proxy's address.

For efficiency, network administrators SHOULD be able to prevent browsers from attempting to send traffic through routes that are already known to be blocked.

2.2. Privacy

To prevent WebRTC peers from determining each others' IP addresses, applications MUST have the ability to direct all traffic through an application-specified TURN server.

A compatible WebRTC browser MAY attempt to prevent a hostile web page from determining the endpoint's public IP address. (The measures proposed here are not sufficient by themselves to achieve this goal. Implementing this specification in current browsers would still leave many other ways for a malicious website to determine the endpoint's IP address. Operating-system-wide VPN configurations are therefore currently preferred for this purpose.)

A compatible WebRTC browser MAY allow the user to prevent non-malicious web pages from accidentally revealing the IP address of remote peers to a local passive network adversary. This ability SHOULD NOT reduce performance when it is not in use. (Due to the difficulty of distinguishing between stupidity and malice, this goal is principally aspirational.)

3. Concepts

To achieve our goals, we introduce the following new concepts:

3.1. Proxy

In this document a "proxy" is any TURN server that was provided by any mechanism other than through the standard WebRTC-application ICE candidate provisioning API [I-D.ietf-rtcweb-jsep]. If a proxy is to be used, it will be the destination of traffic generated by the client. There is no analogue to the transparent/intercepting HTTP proxy configuration, which modifies traffic at the network layer. Mechanisms to configure a proxy include Auto-Discovery [I-D.ietf-tram-turn-server-discovery] and local policy ([I-D.ietf-rtcweb-jsep], "ICE candidate policy").

In an application context, a proxy may be "active" (producing candidates) or "inactive" (not in use, having no effect on the context).

3.2. Virtual interface

A typical WebRTC browser endpoint may have multiple network interfaces available, such as wired ethernet, wireless ethernet, and WAN. In this document, a "virtual interface" is a procedure for generating ICE candidates that are not simply generated by a particular physical interface. A virtual interface can produce "host", "server-reflexive", and "relay" candidates, but may be restricted to only some type of candidate (e.g. UDP-only).

3.3. Proxy configuration leakiness

"Leakiness" is an attribute of a proxy configuration. This document defines two values for the "leakiness" of a proxy configuration: "leaky" and "sealed". Proxy configuration, including leakiness, may be set by local policy ([I-D.ietf-rtcweb-jsep], "ICE candidate policy") or other mechanisms.

A leaky configuration adds a proxy and also allows the browser to use routes that transit directly via the endpoint's physical interfaces (not through the proxy). In a leaky configuration, setting a proxy augments the available set of ICE candidates. Multiple leaky-configuration proxies may therefore be active simultaneously.

A sealed proxy configuration requires the browser to route all WebRTC traffic through the proxy, eliminating all ICE candidates that do not go through the proxy. Only one sealed proxy may be active at a time.

3.4. Sealed proxy rank

In some configurations, an endpoint may be subject to multiple sealed proxy settings at the same time. In that case, one of those settings will have highest rank, and it will be the active proxy. In a given application context (e.g. a webpage), there is at most one active sealed proxy. This document does not specify a representation for rank.

4. Requirements

4.1. ICE candidates produced in the presence of a proxy

When a proxy is configured, by Auto-Discovery or a proprietary means, the browser MUST NOT report a "relay" candidate representing the proxy. Instead, for each active proxy, the browser MUST connect to the proxy and then, if the connection is successful, treat the TURN tunnel as a UDP-only virtual interface.

For a virtual interface representing a TURN proxy, this means that the browser MUST report the public-facing IP address and port acquired through TURN as a "host" candidate, the browser MUST perform STUN through the TURN proxy (if STUN is configured), and it MUST perform TURN by recursive encapsulation through the TURN proxy, resulting in TURN candidates whose "raddr" and "rport" attributes match the acquired public-facing IP address and port on the proxy.

Because the virtual interface has some additional overhead due to indirection, it SHOULD have lower priority than the physical interfaces if physical interfaces are also active. Specifically, even host candidates generated by a virtual interface SHOULD have priority 0 when physical interfaces are active (similar to [RFC5245] Section 4.1.2.2, "the local preference for host candidates from a VPN interface SHOULD have a priority of 0").

4.2. Leaky proxy configuration

If the active proxy for an application is leaky, the browser should undertake the standard ICE candidate discovery mechanism [RFC5245] on the available physical and virtual interfaces.

4.3. Sealed proxy configuration

If the active proxy for an application is sealed, the browser MUST NOT gather or produce any candidates on physical interfaces. The WebRTC implementation MUST direct its traffic from those interfaces only to the proxy, and perform ICE candidate discovery only on the single virtual interface representing the active proxy.

4.4. Proxy rank

Any browser mechanism for specifying a proxy SHOULD allow the caller to indicate a higher rank than the proxy provided by Auto-Discovery [I-D.ietf-tram-turn-server-discovery].

4.5. Multiple physical interfaces

Some operating systems allow the browser to use multiple interfaces to contact a single remote IP address. To avoid producing an excessive number of candidates, WebRTC endpoints MUST NOT use multiple physical interfaces to connect to a single proxy simultaneously. (If this were violated, it could produce a number of virtual interfaces equal to the product of the number of physical interfaces and the number of active proxies.)

For strategies to choose the best interface for communication with a proxy, see [I-D.reddy-mmusic-ice-best-interface-pcp]. Similar

considerations apply when connecting to an application-specified TURN server in the presence of physical and virtual interfaces.

4.6. Unspecified leakiness

If a proxy configuration mechanism does not specify leakiness, browsers SHOULD treat the proxy as leaky. This is similar to current WebRTC implementations' behavior in the presence of SOCKS and HTTP proxies: the candidate allocation code continues to generate UDP candidates that do not transit through the proxy.

4.7. Interaction with SOCKS5-UDP

The SOCKS5 proxy standard [RFC1928] permits compliant SOCKS proxies to support UDP traffic. However, most implementations of SOCKS5 today do not support UDP. Accordingly, WebRTC browsers MUST by default (i.e. unless deliberately configured otherwise) treat SOCKS5 proxies as leaky and having lower rank than any configured TURN proxies.

4.8. Encapsulation overhead, fragmentation, and Path MTU

Encapsulating a link in TURN adds overhead on the path between the client and the TURN server, because each packet must be wrapped in a TURN message. This overhead is sometimes doubled in RETURN proxying. To avoid excessive overhead, client implementations SHOULD use ChannelBind and ChannelData messages to connect and send data through proxies and application TURN servers when possible. Clients MAY buffer messages to be sent until the ChannelBind command completes (requiring one round trip to the proxy), or they MAY use CreatePermission and Send messages for the first few packets to reduce startup latency at the cost of higher overhead.

Adding overhead to packets on a link decreases the effective Maximum Transmissible Unit on that link. Accordingly, clients that support proxying MUST NOT rely on the effective MTU complying with the Internet Protocol's minimum MTU requirement.

ChannelData messages have constant overheard, enabling consistent effective PMTU, but Send messages do not necessarily have constant overhead. TURN messages may be fragmented and reassembled if they are not marked with the Don't Fragment (DF) IP bit or the DONT-FRAGMENT TURN attribute. Client implementors should keep this in mind, especially if they choose to implement PMTU discovery through the proxy.

4.9. Interaction with alternate TURN server fallback

As per [RFC5766], a TURN server MAY respond to an Allocate request with an error code of 300 and an ALTERNATE-SERVER indication. When connecting to proxies or application TURN servers, clients SHOULD attempt to connect to the specified alternate server in accordance with [RFC5766]. The client MUST route a connection to the alternate server through the proxy if and only if the original connection attempt was routed through the proxy.

5. Examples

5.1. Firewalled enterprise network with a basic application

In this example, an enterprise network is configured with a firewall that blocks all UDP traffic, and a TURN server is advertised for Auto-Discovery in accordance with [I-D.ietf-tram-turn-server-discovery]. The proxy leakiness of the TURN server is unspecified, so the browser treats it as leaky.

The application specifies a STUN and TURN server on the public net. In accordance with the ICE candidate gathering algorithm RFC 5245 [RFC5245], it receives a set of candidates like:

1. A host candidate acquired from one interface.
    * e.g. candidate:1610808681 1 udp 2122194687 [internal ip addr for interface 0] 63555 typ host generation 0
2. A host candidate acquired from a different interface.
    * e.g. candidate:1610808681 1 udp 2122194687 [internal ip addr for interface 1] 54253 typ host generation 0
3. The proxy, as a host candidate.
    * e.g. candidate:3458234523 1 udp 24584191 [public ip addr for the proxy] 54606 typ host generation 0
4. The virtual interface also generates a STUN candidate, but it is eliminated because it is redundant with the host candidate, as noted in [RFC5245] Sec 4.1.2..
5. The application-provided TURN server as seen through the virtual interface. (Traffic through this candidate is recursively encapsulated.)

* e.g. candidate:702786350 1 udp 24583935 [public ip addr of the application TURN server] 52631 typ relay raddr [public ip addr for the proxy] rport 54606 generation 0

There are no STUN or TURN candidates on the physical interfaces, because the application-specified STUN and TURN servers are not reachable through the firewall.

If the remote peer is within the same network, it may be possible to establish a direct connection using both peers' host candidates. If the network prevents this kind of direct connection, the path will instead take a "hairpin" route through the enterprise's proxy, using one peer's physical "host" candidate and the other's virtual "host" candidate, or (if that is also disallowed by the network configuration) a "double hairpin" using both endpoints' virtual "host" candidates.

5.2. Conflicting proxies configured by Auto-Discovery and local policy

Consider an enterprise network with TURN and HTTP proxies advertised for Auto-Discovery with unspecified leakiness (thus defaulting to leaky). The browser endpoint configures an additional TURN proxy by a proprietary local mechanism.

If the locally configured proxy is leaky, then the browser MUST produce candidates representing any physical interfaces (including SSLTCP routes through the HTTP proxy), plus candidates for both UDP-only virtual interfaces created by the two TURN servers.

There MUST NOT be any candidate that uses both proxies. Multiple configured proxies are not chained recursively.

If the locally configured proxy is "sealed", then the browser MUST produce only candidates from the virtual interface associated with that proxy.

If both proxies are configured for "sealed" use, then the browser MUST produce only candidates from the virtual interface associated with the proxy with higher rank.

6. Diagrams

This figure shows the connections that provide the ICE candidates for WebRTC in the basic configuration (no proxy). This figure is provided in order to provide a baseline against which to compare the candidate routes that make use of a proxy.

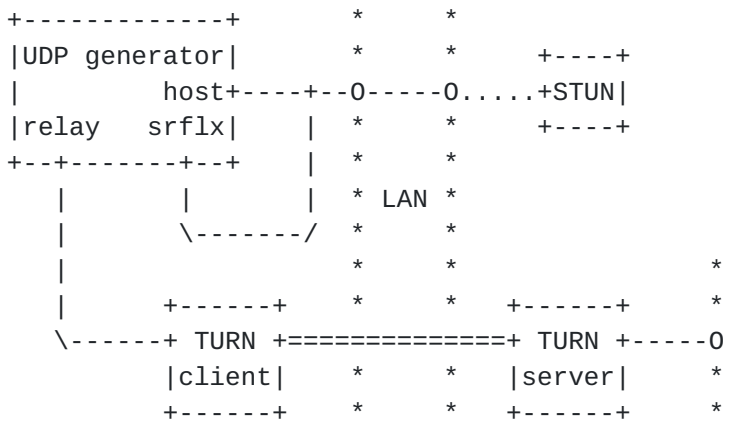

Figure 1: Basic WebRTC ICE candidates (no proxy)

This figure shows the connections that provide the ICE candidates for WebRTC when making use of a proxy.

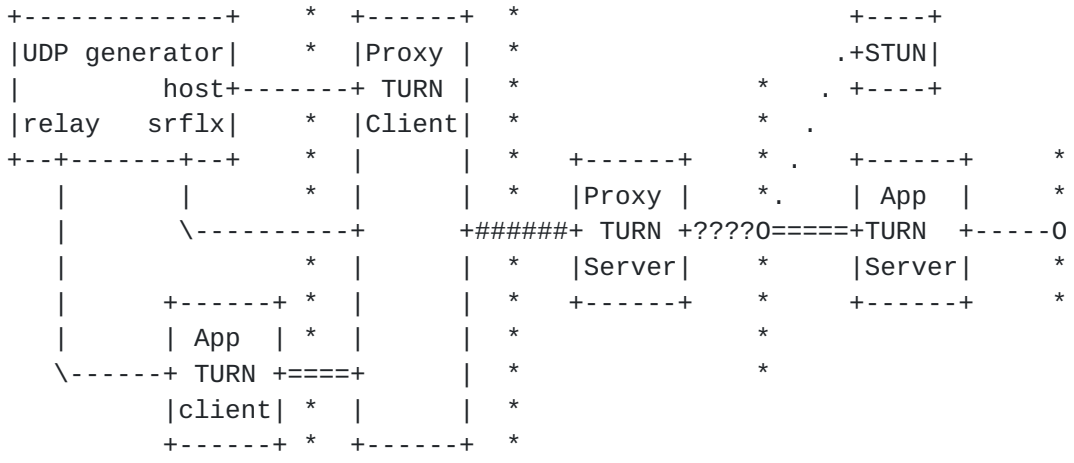

Figure 2: WebRTC ICE candidates using a proxy

7. Security Considerations

This document describes web browser behaviors that, if implemented correctly, allow users to achieve greater identity-confidentiality during WebRTC calls under certain configurations.

If a site administrator offers the site's users a TURN proxy, websites running in the users' browsers will be able to initiate a UDP-based WebRTC connection to any UDP transport address via the proxy. Websites' connections will quickly terminate if the remote endpoint does not reply with a positive indication of ICE consent, but no such restriction applies to other applications that access the TURN server. Administrators should take care to provide TURN access credentials only to the users who are authorized to have global UDP network access.

8. IANA Considerations

This document requires no actions from IANA.

9. Acknowledgements

Significant review, including the virtual-interface formulation, was provided by Justin Uberti. Thanks to Harald Alvestrand, Phillip Hancke, and Tirumaleswar Reddy for suggestions to improve the content and presentation.

10. References

10.1. Normative References

[I-D.ietf-rtcweb-jsep] Uberti, J. and C. Jennings, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-06 (work in progress), February 2014.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 5766, March 1996.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[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, April 2010.

10.2. Informative References

[I-D.ietf-rtcweb-use-cases-and-requirements]
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", ietf-rtcweb-use-cases-and-requirements-14 (work in progress), February 2014.

[I-D.ietf-tram-turn-server-discovery]
Patil, P., Reddy, T., and D. Wing, "TURN Server Auto Discovery", draft-ietf-tram-turn-server-discovery-00 (work in progress), July 2014.

[I-D.reddy-mmusic-ice-best-interface-pcp]
Reddy, T., Wing, D., VerSteeg, B., Penno, R., and V. Singh, "Improving ICE Interface Selection Using Port Control Protocol (PCP) Flow Extension", draft-ietf-tram-turn-server-discovery-00 (work in progress), October 2013.

Author's Address

Benjamin M. Schwartz
Google
747 6th Ave S
Kirkland, WA 98033
USA

Email: bemasc@webrtc.org