WebRTC Enterprise Firewall Traversal

draft-jennings-behave-rtcweb-firewall-01

Cullen Jennings
fluffy@cisco.com

July 2015

v8

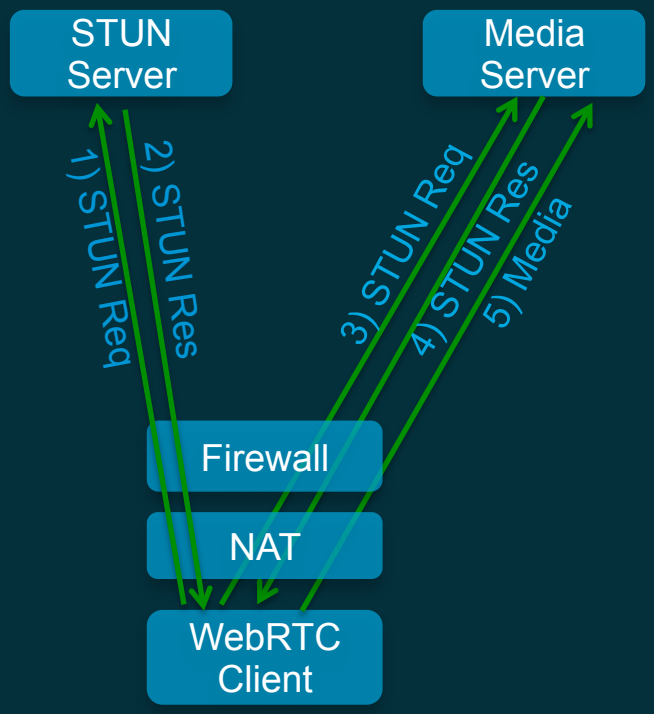
Security Concerns

1. UDP has been used by malware for a control channel
2. UDP has been used for DDoS attacks (DNS, NTP)
3. Encrypted Data Channel can be used to bring malware into the company
4. Encrypted Data Channel can be used to send files out of the company

Proposed Firewall Algorithm for WebRTC

- The firewall looks for any outgoing STUN requests to STUN port (3478). When it finds one, it stores the 3 tuple of the source address port and protocol=UDP and for the next 30 seconds checks any packets from this 3 tuple to see if they are ICE connectivity checks.
- When firewall sees an ICE connectivity check from the inside host (which is a STUN Binding Request containing the ice username), it stores that ice username value, and forwards the packet. Thereafter, STUN Binding Requests and Binding Responses with matching 4-tuple (inside IP address, protocol=UDP, inside port, and matching ice username) are allowed in any direction (inside->outside and outside->inside). For each 5-tuple, if a STUN Binding Request has a Success STUN Binding Response (with same transaction ID), in either direction, that 5-tuple is promoted to allow non-STUN traffic (DTLS, SRTP, and anything else).
- Both promoted and non-promoted pinholes are closed after 30 seconds of not seeing a STUN Binding Request and Response transaction, in either direction.
- (Note to match the ice username external requests will need the halves on either side of ":" swapped)

Cloud Media



1. Outbound STUN request to well known STUN port (3478)

   Firewall creates 4-tuple pinhole for incoming and outgoing STUN message with matching username

2. Inbound STUN response on cone pinhole

   Firewall creates 5-tuple pinhole for flow to Stun server (but this is never used)

3. Outbound STUN request to Media server

   Allowed out due to rule created by 1. Firewall stores 5-tuple with this STUN transaction ID

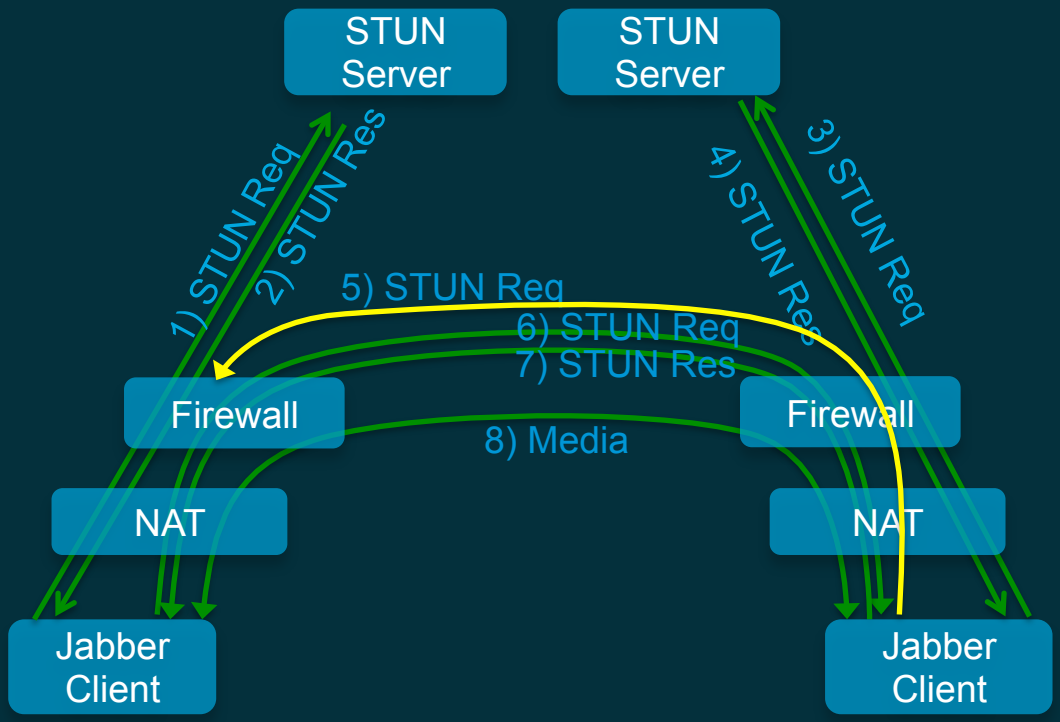
4. Inbound STUN response with matching username

   Allowed in due to rule created by 1. This triggers creation of 5-tuple pinhole for flow to media server

5. Media flowing on 5-tuple to media server

Note that is the Stun Server and Media Server are the same address, this still works the same way

P2P Media



Right firewall sees:

1. outbound STUN request and response to well known STUN port (msg 3, 4)
2. outbound STUN request to IP of far NAT on random port (msg 5). Blocked by far firewall
3. inbound STUN request on same 4-tuple from step 2 followed by response from inside (msg 6,7). Note msg 5 and 6 are different
4. media on same 5-tuple as step 3

Advantages

- STUN packets are only allowed “in” if they know the crypto random username generated by a client inside the firewall
- Non STUN packets are only allowed “in” if they match a 5 tuple that a client inside the firewall sent a packet too
- Non STUN packets are only allowed “out” if the destination they are sending to did a stun consent handshake

- Issue : Once a valid 5 tuple is set up, an attacker outside the firewall could spoof the source address and send in a packet that would be forwarded (basically we are not checking sequence numbers of DTLS & SRTP in 5 tuple). However, if this is only used for DTLS / SRTP, the client is going to discard the packet as invalid as long as we don’t have problems in tls or libsrtp

What to do about media hiding in HTTPS

- In draft-ietf-rtcweb-transports-09, webrtc clients uses draft-ietf-httpbis-tunnel-protocol which uses HTTP connect to go thought proxy to TURN server. This defines the header that indicates support for doing this as optional so there is no way for the client to know that proxy is available to e used this way
- Video sent to a proxy not scaled for video makes a huge impact on performance
- Firewall vendors are asking what algorithm they can use to protect HTTPY proxies form unintended used for RTP relay
- Better to have consistent behavior than some of the things proposed.
- Current proposal :

  Any HTTP or HTTPS connection that sends more than 10 requests per second for longer than 10 seconds should be paused for 1 second