Web Real-Time Communications Deployment Guide

Security considerations

12/18/2025

Contents

Security considerations

Secure transports

With WebRTC, the browser is initiating both an HTTP connection and a media session that connects it with another WebRTC peer. Since the description of the media session is signaled over HTTP, HTTP can be protected with the use of HTTPS. The media session is secured through the use of Secure Real-time Transport (SRTP), meaning all media are encrypted before being sent over the wire. The security keys for encrypting the media are exchanged in a secure manner through either the signal path (SDP security descriptions) or the media path itself (DTLS).

By default, the Genesys WebRTC Service enables SRTP, to ensure that all communication paths are secure by default.

Enabling DTLS-SRTP

DTLS-SRTP (RFC 5763) is enabled on the web side by default. However, if you need to re-enable it for some reason, simply set the `web-enable-dtls` configuration option to `true`.

When `web-enable-dtls` is enabled, it will be signalled in an SDP offer sent by the gateway using the fingerprint attributes, though there will also be crypto attributes in SDP for SDES-SRTP (RFC 4568) support. When an offer or answer comes in with only crypto attributes, then SDES-SRTP will still be supported. When `web-enable-dtls` is set to `false`, only SDES-SRTP will be supported.

Securing the host

The best practice for securing a host that is deployed on the web is to place the host behind a firewall. The Genesys WebRTC Service requires the firewall to open up the HTTPS port for inbound traffic and a range of SRTP ports for inbound and outbound traffic.

The WebRTC Service also needs to establish SIP calls to Genesys SIP Server. Genesys SIP Server and agent endpoints are typically deployed within the enterprise firewall, so in this case, the WebRTC Service only needs to access the SIP port and a range of RTP ports on the enterprise network.

The WebRTC Service also connects to the Genesys Management Framework, which is deployed within the enterprise network. This means that the Service requires access to Local Control Agent and Configuration Server.

Other ports may be allowed by the firewall for other administration purposes. It is recommended that they only be accessible either from within the enterprise network or through a secure communication channel from the Internet, such as ssh.

Security for the web application

The user does not directly access the WebRTC Service to place calls, as it is the responsibility of the web application to deliver a user interface to the browser for the user. The WebRTC Service includes a JavaScript API that provides a set of HTTP API methods by means of which the web application can access the WebRTC Service separately from the web server that hosts the web application.

Cross-Origin Resource Sharing (CORS)

Since the browser Same Origin Policy prevents a web page from making an `XMLHttpRequest` to another domain, the WebRTC Service supports Cross Origin Resource Sharing (CORS) to allow the web application to interact with the WebRTC Service across domains.

For a simple request—one that uses either `GET` or `POST` and whose body is `text/plain`—the request is sent with an extra header called `Origin`. The `Origin` header contains the origin URI (scheme, domain name or address, and port, as per RFC 6454) of the requesting page so that the server can easily determine whether or not it should serve a response. An example `Origin` header might look like this:

```
Origin: http://www.genesys.com:8080
```

The WebRTC Service provides a configuration option (`domain-whitelist`) to specify the allowed list of domains, and any CORS request specifying an `Origin` that is not allowed will result in an error. However, if the list of allowed domains is set to empty, the WebRTC Service then allows interactions with all sites with CORS by setting the `Access-Control-Allow-Origin` HTTP header in the response to the value specified in the `Origin` header value from the original request.

If the server decides that the request should be allowed, it either sends an `Access-Control-Allow-Origin` header echoing back the same origin that was sent or `'*'` if it is a public resource.

For example:

```
Access-Control-Allow-Origin: http://www.genesys.com:8080
```

If this header is missing, or the value of this header does not match the value of `Origin` header, then the browser disallows the request. If all is well, then the browser processes the response.