Control access to WebRTC publishing and playback

Originally Published on 10/31/2018 | Updated on 03/26/2021 1:23 pm PDT

Wowza Streaming Engine™ media server software version 4.7.7 and later supports WebRTC streaming, however, we recommend that you update to version 4.8.5 and later to capitalize on expanded functionality and enhancements to publisher reliability. WebRTC is a collection of APIs that run on the Real-time Transport Protocol (RTP). This article provides commands and code that can help manage RTP network sessions and the exchange of data between a WebRTC application and Wowza Streaming Engine.

Listen for and reject WebRTC session create and destroy commands

The following code is a module that listens for and rejects WebRTC session create and destroy commands. You can use this module to control access over WebRTC streams and to track how long WebRTC streams run. To implement it, you need to use the Wowza IDE. For more information, see Extend Wowza Streaming Engine using the Wowza IDE.

```
import com.wowza.wms.module.*;
import com.wowza.wms.rtp.model.*;

public class ModuleListenWebRTCSession extends ModuleBase
{
  public void onRTPSessionCreate(RTPSession rtpSession)
  {
    if (rtpSession.isWebRTC())
    {
      getLogger().info("ModuleListenWebRTCSession.onRTPSessionCreate["+rtpSession.getSessionId()+"]")
;

// The userData element is where customized players are allowed to send their own data.
Map userData = (Map)rtpSession.getWebRTCSession().getCommandRequest().getJSONEntries().get("userData"
);

      // Call rejectSession to stop the session immediately
      //rtpSession.rejectSession();
    }
  }

  public void onRTPSessionDestroy(RTPSession rtpSession)
  {
    if (rtpSession.isWebRTC())
    {
      getLogger().info("ModuleListenWebRTCSession.onRTPSessionDestroy["+rtpSession.getSessionId()+"]"
);
    }
  }
},>,>
```

Determine if an RTP session is a WebRTC session

Use the following command to see if an RTP session was created by WebRTC.

```
boolean RTPSession.isWebRTC();
```

Get a WebRTCSession interface to/from RTPSession

Retrieve information tracked by the WebRTC session, but not by the RTP session.

```
WebRTCSession rtpSession.getWebRTCSession();
RTPSession WebrtcSession.getRTPSession();
```

Get WebRTC session information

Retrieve information, such as session counts and stream names, about a WebRTC session.

```
List getWebRTCSessions()
List getWebRTCSessions(String streamName)
int getWebRTCSessionCount()
int getWebRTCSessionCount(String streamName)
Map getWebRTCSessionCountsByName()
```

Forcefully disconnect a WebRTC session

Control access to live streams after they've started by forcefully disconnecting a WebRTC session on command.

```
vhost.getWebRTCContext().shutdownSession(webRTCSession.getSessionId());
```

Control maximum playback connections

Use the **ModuleCoreSecurity** built-in Java module to restrict the number of concurrent connections accepted by the Wowza Streaming Engine live application that's configured for WebRTC. See Configure security using Wowza Streaming Engine Manager for information on limiting maximum playback connections.

Protect WebRTC playback using SecureToken

Wowza Streaming Engine 4.8.11 and later supports SecureToken playback protection for WebRTC streams. The **SecureToken** module uses a challenge/response system to protect against spoofing threats. To enable playback protection for WebRTC using SecureToken, see Protect streaming using SecureToken in Wowza Streaming Engine.

Enhance WebRTC publishing and playback security with a custom HTTP provider

For additional security for your WebRTC application, you can override the default HTTP provider with a custom provider that offers additional methods for controlling publishing, playback, and querying the WebRTC application.

The following code demonstrates how to extend the default WebRTC HTTP provider

HTTPWebRTCExchangeSessionInfo, which is configured in Use WebRTC with Wowza Streaming Engine, with **HTTPWebRTCExchangeSessionInfoCustom**. The custom subclass overrides the default SDP exchange and allows you to pass security information from HTML or JavaScript to Wowza Streaming Engine either through query parameters or through the JSON data that's passed over the WebSocket. A **userData** object can hold the data, and you can control publishing, playback, and query through the commented-out **commandControl.canPlay**, **commandControl.canPublish**, and **commandControl.canQuery** variables.

```
import java.util.*;

import com.wowza.util.*;
import com.wowza.wms.application.*;
import com.wowza.wms.logging.*;
import com.wowza.wms.webrtc.http.*;
import com.wowza.wms.websocket.model.*;

public class HTTPWebRTCExchangeSessionInfoCustom extends HTTPWebRTCExchangeSessionInfo
{
    private static final Class CLASS = HTTPWebRTCExchangeSessionInfoCustom.class;
  private static final String CLASSNAME = "HTTPWebRTCExchangeSessionInfoCustom";

  @Override
  protected void websocketSessionCreate(IWebSocketSession webSocketSession)
  {
    super.websocketSessionCreate(webSocketSession);
    WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME+".websocketSessionCreate: "+webSocketSession.get
SessionId());
  }

  @Override
  protected void websocketSessionDestroy(IWebSocketSession webSocketSession)
  {
    super.websocketSessionDestroy(webSocketSession);
    WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME+".websocketSessionDestroy: "+webSocketSession.ge
tSessionId());
  }

  @Override
  protected void authenticateRequest(CommandContext commandContext, CommandControl commandControl)
  {
    super.authenticateRequest(commandContext, commandControl);

    WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME+".authenticateRequest: reqURI:"+commandContext.r
eqURI);

    int qloc = commandContext.reqURI.indexOf("?");
    if (qloc >= 0)
    {
      String queryStr = commandContext.reqURI.substring(qloc+1).trim();
      if (queryStr.length() > 0)
      {
        Map queryMap = HTTPUtils.splitQueryStr(queryStr);
        for(Map.Entry entry : queryMap.entrySet())
        {
          WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME+".authenticateRequest: queryMap["+entry.ge
tKey()+"]: "+entry.getValue());
        }
      }
    }

    IApplicationInstance appInstance = commandContext.commandRequest.getApplicationInstance(commandCo
ntext.vhost);
```

```
    if (appInstance != null)
    {
      WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME+".authenticateRequest: application: "+appInsta
nce.getContextStr());

      // application level properties
      WMSProperties prop = appInstance.getProperties();
    }

    Map jsonEntries = commandContext.commandRequest.getJSONEntries();
    if (jsonEntries != null)
    {
      Map userData = (Map)jsonEntries.get("userData");
      if (userData != null)
      {
        for(Map.Entry entry : userData.entrySet())
        {
          WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME+".authenticateRequest: userData["+entry.ge
tKey()+"]: "+entry.getValue());
        }
      }
    }

    // Perform authentication here and set

    // commandControl.canPlay
    // commandControl.canPublish
    // commandControl.canQuery
  }
}
```

More resources

- WebRTC workflows in Wowza Streaming Engine
- Use WebRTC with Wowza Streaming Engine
- Record WebRTC streams with Wowza Streaming Engine