



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Genesys Mobile Services Deployment Guide

Secure GMS Access Control

4/22/2025

Secure GMS Access Control

This page covers the list of security points, mechanisms, and procedures that you can implement to secure your Genesys Mobile Environment.

Important

If you face any security issues, Genesys recommends that you upgrade to the latest 8.5.3 release.

Use HTTPS to avoid unsecured connections to GMS

HTTPS (Hypertext Transfer Protocol Secure) is an internet communication protocol that protects the integrity and the confidentiality of your users' data between the user's computer and the GMS. Implementing HTTPS connections with GMS prevents the following Transport Confidentiality attacks:

Attack	Description
MITM	Man In The Middle
BEAST	Browser Exploit Against SSL/TLS
CRIME	Compression Ratio Info-leak Made Easy

In 8.5.3, the Jetty server that GMS runs on was upgraded to release 12. As a result, GMS requires JVM 17 or later. In accordance with RFC 7231 recommendations, the Content-Type response header will provide charset values in lowercase: Content-Type → application/json; charset=utf-8

Add a Certificate to JETTY

1. Request a Certificate from a [Certificate Authority \(CA\)](#) to sign your key/certificate.
2. Once the CA has sent you a certificate, load it into a JSSE keystore. Run the JDK's `keytool` to load a certificate in PEM form directly into a keystore. The following command loads a PEM encoded certificate in the `jetty.crt` (JETTY) file into a JSSE keystore:

```
$ keytool -keystore mykeystore -import -alias jetty -file jetty.crt -trustcacerts
```

3. To add the certificate to JETTY, copy the `mykeystore` file to JETTY's `etc` folder, then edit `jetty-ssl-context.xml` to setup the certificate info.

Private key and cert files have to be in the keystore. You may also need to convert PEM-format keys to JKS format and PKCS12 to a JKS file. The PEM format is the most common format that Certificate Authorities issue certificates in.

PEM certificates usually have extensions such as `.pem`, `.crt`, `.cer`, and `.key`. They are Base64-encoded ASCII files and contain `"-----BEGIN CERTIFICATE-----"` and `"-----END CERTIFICATE-----"` statements.

Server certificates, intermediate certificates, and private keys can all be put into the PEM format. Several PEM certificates and even the private key can be included in one file, one below the other.

Here are some command examples to help you to manage .pfx certificate files and the Jetty's keystore:

Get the key only:

```
e:\openssl\openssl pkcs12 -in e:\certtest\upmc.pfx -nocerts -out e:\certtest\upmckey.pem
```

Get the cert only:

```
e:\openssl\openssl pkcs12 -in e:\certtest\upmc.pfx -clcerts -nokeys -out e:\certtest\upmccert.pem
```

Remove the passphrase from the key:

```
e:\openssl\openssl rsa -in e:\certtest\upmckey.pem -out e:\certtest\upmcserver.key
```

Create a combined .p12 file that you can import in the keystore:

```
e:\openssl\openssl pkcs12 -export -in E:\certtest\upmccert.pem -inkey e:\certtest\upmcserver.key -out e:\certtest\combined.p12 -name upmc_cert -CAfile e:\certtest\my_ca_bundle.crt -caname root
```

Add the combined key/cert to the keystore:

```
keytool -importkeystore -deststorepass **** -destkeypass **** -destkeystore mykeystore -srckeystore combined.p12 -srcstoretype PKCS12 -srcstorepass ****
```

Configure JETTY Secured Ports

1. Setup the secured port in the **jetty-ssl.xml** and **jetty.xml** files. The default HTTPS port is 8443.
2. Remove the content of the **jetty-http.xml** file (not relevant). Moreover, you will need to setup (operating system) firewall rules to control access to Jetty HTTP unsecured port (8080 by default).

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "configure_10_0.dtd">

<!-- ===== -->
<!-- Configure the Jetty Server instance with an ID "Server" -->
<!-- by adding an HTTP connector. -->
<!-- This configuration must be used in conjunction with jetty.xml -->
<!-- ===== -->
<Configure id="Server" class="org.eclipse.jetty.server.Server">

  <!-- ===== -->
  <!-- Add an HTTP Connector. -->
  <!-- Configure an o.e.j.server.ServerConnector with a single -->
  <!-- HttpConnectionFactory instance using the common httpConfig -->
  <!-- instance defined in jetty.xml -->
  <!-- -->
  <!-- Consult the javadoc of o.e.j.server.ServerConnector and -->
  <!-- o.e.j.server.HttpConnectionFactory for all configuration -->
  <!-- that may be set here. -->
  <!-- ===== -->
  <!-- -->
```

```

    <Call name="addConnector">
      <Arg>
        <New id="httpConnector" class="org.eclipse.jetty.server.ServerConnector">
          <Arg name="server"><Ref refid="Server" /></Arg>
          <Arg name="acceptors" type="int"><Property name="jetty.http.acceptors"
default="1"/></Arg>
          <Arg name="selectors" type="int"><Property name="jetty.http.selectors"
default="-1"/></Arg>
          <Arg name="factories">
            <Array type="org.eclipse.jetty.server.ConnectionFactory">
              <Item>
                <New class="org.eclipse.jetty.server.HttpConnectionFactory">
                  <Arg name="config"><Ref refid="httpConfig" /></Arg>
                </New>
              </Item>
            </Array>
          </Arg>
          <Set name="host"><Property name="jetty.http.host" deprecated="jetty.host"
/></Set>
          <Set name="port"><Property name="jetty.http.port" deprecated="jetty.port"
default="8080" /></Set>
          <Set name="idleTimeout"><Property name="jetty.http.idleTimeout"
deprecated="http.timeout" default="30000"/></Set>

          <Set name="acceptorPriorityDelta"><Property
name="jetty.http.acceptorPriorityDelta" deprecated="http.acceptorPriorityDelta"
default="0"/></Set>
          <Set name="acceptQueueSize"><Property name="jetty.http.acceptQueueSize"
deprecated="http.acceptQueueSize" default="0"/></Set>
          <Get name="SelectorManager">
            <Set name="connectTimeout"><Property name="jetty.http.connectTimeout"
default="15000"/></Set>
          </Get>
          <Set name="reuseAddress"><Property name="jetty.http.reuseAddress"
default="true"/></Set>
          <Set name="reusePort"><Property name="jetty.http.reusePort"
default="false"/></Set>
          <Set name="acceptedTcpNoDelay"><Property name="jetty.http.acceptedTcpNoDelay"
default="true"/></Set>
          <Set name="acceptedReceiveBufferSize"
property="jetty.http.acceptedReceiveBufferSize" />
          <Set name="acceptedSendBufferSize" property="jetty.http.acceptedSendBufferSize"
/>
        </New>
      </Arg>
    </Call>
  -->
</Configure>

```

3. Update the start.ini file in <GMS_HOME> directory with the following text:

```

--module=server,ee8-jsp,jmx,ee8-websocket-
javax,resources,ext,ee8-plus,ee8-annotations,ee8-deploy,ee8-security,ee8-servlets,rewrite,rewrite-
compactpath,ssl,https
...
...
#=====
jetty.httpConfig.sendServerVersion=false
jetty.sslContext.keyStorePath=etc/keystore
jetty.sslContext.keyStorePassword=<keystorepassword>

```

```
jetty.sslContext.keyManagerPassword=<keystorepassword>
jetty.sslContext.trustStorePath=etc/ANTIMEAPPS01
jetty.sslContext.trustStorePassword=OBF:<encrypted keystore="" password="">
```

In your production environment, you should place your mykeystore file in a private directory with restricted access, not in a directory relative to the JETTY home directory. The mykeystore file in the example above is relative to the JETTY home directory.

The **jetty.sslContext.keyStorePassword** and **jetty.sslContext.keyManagerPassword** properties can also contain obfuscated passwords (OBF). Check the **jetty-ssl-context.xml** file (as an example). You can use the same process that is used for the **jetty.sslContext.trustStorePassword** OBF password generation.

Important

For production, choose a private directory with restricted access to keep your keystore in. Even though it has a password on it, the password may be configured into the runtime environment and is vulnerable to theft. You can now start Jetty the normal way (make sure that in your text cert.jar, jnet.jar, and jsse.jar are on your classpath) and SSL can be used with a URL like: https://localhost:8443/

You should encode the the_password value, for example by using the embedded password tool:

```
C:\8.5.3>java -cp
lib/jetty-http-12.0.8.jar;lib/jetty-util-12.0.8.jar
org.eclipse.jetty.util.security.Password username userpassword
Usage - java org.eclipse.jetty.security.Password [<user>] <password>
```

If the password is ?, the user will be prompted for the password.

4. Force the use of the TLS protocol by adding a new sslConnector connector or by disabling the unsecured port. To modify the secured port, edit the etc/jetty-ssl.xml file:

```
<Call name="addConnector">
  <Arg>
    <New id="sslConnector" class="org.eclipse.jetty.server.ServerConnector">
      <Arg name="server"><Ref refid="Server" /></Arg>
      <Arg name="acceptors" type="int"><Property name="jetty.ssl.acceptors"
default="1"/></Arg>
      <Arg name="selectors" type="int"><Property name="jetty.ssl.selectors"
default="-1"/></Arg>
      <Arg name="factories">
        <Array type="org.eclipse.jetty.server.ConnectionFactory">
          <!-- uncomment to support proxy protocol
          <Item>
            <New class="org.eclipse.jetty.server.ProxyConnectionFactory"/>
          </Item>
          </Array>
        </Arg>
        <Set name="host"><Property name="jetty.ssl.host" deprecated="jetty.host" /></Set>
        <Set name="port"><Property name="jetty.ssl.port" deprecated="ssl.port"
default="8443" /></Set>
        <Set name="idleTimeout"><Property name="jetty.ssl.idleTimeout"
deprecated="ssl.timeout" default="30000"/></Set>
        <Set name="acceptorPriorityDelta"><Property
name="jetty.ssl.acceptorPriorityDelta" deprecated="ssl.acceptorPriorityDelta"
default="0"/></Set>
        <Set name="acceptQueueSize"><Property name="jetty.ssl.acceptQueueSize"
deprecated="ssl.acceptQueueSize" default="0"/></Set>
        <Get name="SelectorManager">
          <Set name="connectTimeout"><Property name="jetty.ssl.connectTimeout"
default="15000"/></Set>
          <!-- Set name="reservedThreads"><Property name="jetty.ssl.reservedThreads"
```

```

default="-2"/></Set -->
  </Get>
  <Set name="reuseAddress"><Property name="jetty.ssl.reuseAddress"
default="true"/></Set>
  <Set name="reusePort"><Property name="jetty.ssl.reusePort"
default="false"/></Set>
  <Set name="acceptedTcpNoDelay"><Property name="jetty.ssl.acceptedTcpNoDelay"
default="true"/></Set>
  <Set name="acceptedReceiveBufferSize"
property="jetty.ssl.acceptedReceiveBufferSize" />
  <Set name="acceptedSendBufferSize" property="jetty.ssl.acceptedSendBufferSize" />
</New>
</Arg>
</Call>

```

5. Edit your **ORS DFM files** to point to the new secured port of the TLS connector (8443 in our example).

```

<dfm name='gsgCallback'
namespace='http://www.genesyslab.com/modules/dfm/gsgCallback/v1'>
  <protocols>
  <protocol type='http'>
    <params>
    <param name='timeout' expr='20' />
    </params>
    <globalevents>
    <event name='done' mapping='200' />
    <event name='error' mapping='0,400-510' />
    </globalevents>
    </protocol>
  <protocol type='https'>
    <params>
    <param name='retries' expr='4' />
    <param name='timeout' expr='20' />
    </params>
    <globalevents>
    <event name='done' mapping='200' />
    <event name='error' mapping='400-510' />
    </globalevents>
    </protocol>
  </protocols>
  <connections>
  <connection name='HTTP_round_robin' type='https'
selectionmode='round-robin'>
    <servers>
    <server name='gsg1' host='1XX.1XX.10.01' port='8443' />
    <server name='gsg2' host='1XX.1XX.10.02' port='8443' />
    <server name='gsg3' host='1XX.1XX.10.03' port='8443' />
    <server name='gsg4' host='1XX.1XX.10.04' port='8443' />
    </servers>
    </connection>
  </connections>
  <transports>
  <transport name='HTTP_GET' conntype='HTTP_round_robin'>
    <param name='method' expr='GET' />
    <param name='enctype' expr='application/x-www-form-urlencoded' />
    </transport>
  <transport name='HTTP_POST' conntype='HTTP_round_robin'>
    <param name='method' expr='POST' />
    <param name='enctype' expr='application/x-www-form-urlencoded' />
    </transport>
  <transport name='HTTP_DELETE' conntype='HTTP_round_robin'>
    <param name='method' expr='DELETE' />
    <param name='enctype' expr='application/x-www-form-urlencoded' />

```

```
</transport>
<transport name='HTTP_PUT' conntype='HTTP_round_robin'>
<param name='method' expr='PUT' />
<param name='enctype' expr='application/json' />
</transport>
</transports>
<actions>
<action name='start-callback' transport='HTTP_POST' >
<overrides>
<param name='url' expr=
'/genesys/${apiVersion}/service/callback/${callbackexecutionname}' />
</overrides>
<attributes>
<attribute name='apiVersion' type='String' optional='false' />
<attribute name='callbackexecutionname' type='String'
optional='false' />
<attribute name='content' type='Expression' optional='false'
mapping='content' />
</attributes>
</action>
<action name='cancel-callback' transport='HTTP_DELETE' >
<overrides>
<param name='url'
expr='/genesys/${apiVersion}/service/callback/${callbackexecutionname}/${service_id}' />
</overrides>
<attributes>
<attribute name='apiVersion' type='String' optional='false'
/>
<attribute name='callbackexecutionname' type='String' optional='false'
/>
<attribute name='service_id' type='String' optional='false' />
</attributes>
</action>
<action name='reschedule-callback' transport='HTTP_PUT' >
<overrides>
<param name='url'
expr=
'/genesys/${apiVersion}/service/callback/${callbackexecutionname}/${service_id}'
/>
</overrides>
<attributes>
<attribute name='apiVersion' type='String' optional='false' />
<attribute name='callbackexecutionname' type='String' optional='false' />
<attribute name='service_id' type='String' optional='false' />
<attribute name='content' type='Expression' optional='false'
mapping='content' />
</attributes>
</action>
<action name='query-callback' transport='HTTP_GET' >
<overrides>
<param name='url'
expr='/genesys/${apiVersion}/service/callback/${callbackexecutionname}
?customer_number=${customer_number_value}' />
</overrides>
<attributes>
<attribute name='apiVersion' type='String' optional='false' />
<attribute name='callbackexecutionname' type='String' optional='false'
/>
<attribute name='customer_number_value' type='String' optional='false'
/>
</attributes>
</action>
<action name='query-availability-timestamp' transport='HTTP_GET' >
```

```
<overrides>
<param name='url'
expr='/genesys/${apiVersion}/service/callback/${callbackexecutionname}
/availability?timestamp=${timestamp_value}'/>
</overrides>
<attributes>
<attribute name='apiVersion' type='String' optional='false' />
<attribute name='callbackexecutionname' type='String' optional='false'
/>
<attribute name='timestamp_value' type='String' optional='false' />
</attributes>
</action>
<action name='query-availability' transport='HTTP_GET' >
<overrides>
<param name='url'
expr='/genesys/${apiVersion}/service/callback/${callbackexecutionname}
/availability' />
</overrides>
<attributes>
<attribute name='apiVersion' type='String' optional='false' />
<attribute name='callbackexecutionname' type='String' optional='false'
/>
</attributes>
</action>
</actions>
</dfm>
```

6. Make the following changes to your Jetty server:

a. Download and copy the following files to <GMS_HOME>/etc directory:

- [keystore](#)
- [jetty-ssl-context.xml](#)

b. Update the start.ini file by adding "etc/jetty-ssl-context.xml" to the following list:

```
etc/jetty.xml
etc/jetty-ssl.xml
etc/jetty-ssl-context.xml
etc/jetty-deploy.xml
etc/jetty-http.xml
etc/jetty-https.xml
etc/jetty-rewrite.xml
#etc/jetty-overlay.xml
#etc/jetty-webapps.xml
#etc/jetty-contexts.xml
```

Redirect HTTP requests to HTTPS

If you want to keep an unsecured port (for instance port 8080) and redirect HTTP to HTTPS, the webapp should indicate that it needs CONFIDENTIAL or INTEGRAL connections from users. This is done in <GMS_HOME>/etc/webdefault-ee8.xml

```
<web-app>
<security-constraint>
<web-resource-collection>
<web-resource-name>Everything</web-resource-name>
<url-pattern>/*</url-pattern>
</web-resource-collection>
<user-data-constraint>
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

```
</user-data-constraint>
</security-constraint>
```

GMS Options to Setup

Set the following options set in your GMS application at the server level, not the cluster level:

```
[server]
external_url_base=https://gmshost:8443/
web_port=8443
web_scheme=https
```

Configure HTTP Strict Transport Security

By default, HTTP Strict Transport Security (HSTS) is configured in GMS as follows:

Strict-Transport-Security max age: 31536000

Include subdomain: true

To update this configuration, edit the `etc/jetty-ssl.xml` file in the GMS installation directory as follows:

```
<New id="sslHttpConfig" class="org.eclipse.jetty.server.HttpConfiguration">
  <Arg><Ref refid="httpConfig"/></Arg>
  <Call name="addCustomizer">
    <Arg>
      <New class="org.eclipse.jetty.server.SecureRequestCustomizer">
        <Arg name="sniRequired" type="boolean"><Property name="jetty.ssl.sniRequired"
default="false"/></Arg>
        <Arg name="sniHostCheck" type="boolean"><Property name="jetty.ssl.sniHostCheck"
default="true"/></Arg>
        <Arg name="stsMaxAgeSeconds" type="int"><Property name="jetty.ssl.stsMaxAgeSeconds"
default="31536000"/></Arg>
        <Arg name="stsIncludeSubdomains" type="boolean"><Property
name="jetty.ssl.stsIncludeSubdomains" default="true"/></Arg>
      </New>
    </Arg>
  </Call>
</New>
```

Secure Access to Proxy Configuration

GMS can send outbound HTTP requests to Push Notification servers through the following notification services:

- APNS (Apple Push Notification Service)

- FCM (Firebase Cloud Messaging Service)
- Third Party Servers for custom notification

To secure access, GMS provides a Proxy Auto-Configuration file (PAC) mechanism. You can configure a PAC file in your GMS Application's settings, section gms:

Option	Type	Default value	Restriction on value	Description
http.proxy-auto-config-file	String		Optional Valid URL	Location of the proxy auto-config (PAC) file. For example: • file:///C:/GMS/proxy.pac : for a local file • http://127.0.0.1:8082/deploy/proxy.pac
http.proxy-cache-size	Integer	32	Valid integer (optional)	Size of the cache that stores URLs that were already processed. If the requested URL is in the cache, GMS will not process the PAC file.
http.proxy-ttl	Integer	5	Valid integer (Optional)	Interval in minutes before refreshing the PAC content.

You can use the following PAC file to secure GMS access for Apple and Google Servers:

```
function FindProxyForURL(url, host) {  
  if (dnsDomainIs(host, ".googleapis.com"))  
    return "PROXY 127.0.0.1:808";  
  
  if (dnsDomainIs(host, ".wns.windows.com"))  
    return "PROXY 127.0.0.1:808";  
  
  if (dnsDomainIs(host, ".apple.com"))  
    return "SOCKS 127.0.0.1:1080";  
  
  return "DIRECT";  
}
```

Possible Values for Access-Control-Allow-Origin Header (CORS)

By default, the value for all the properties mentioned below is set to *, meaning *all origins*.

If you need to configure CORS for a specific domain, method, or header, then edit the **launcher.xml** file as described below; the following are the three types of properties used to configure CORS.

- com.genesyslab.common.cors.allowed-origins
- com.genesyslab.common.cors.allowed-methods
- com.genesyslab.common.cors.allowed-headers

Important

The CORS configuration options above apply for REST API /genesys URL. For the /cometd URL (cometd servlet), the CORS configuration options are the same and set up in the etc/webdefault-ee8.xml file.

Example:

```
<filter>
  <filter-name>cross-origin</filter-name>
  <filter-class>org.eclipse.jetty.servlets.CrossOriginFilter</filter-class>
  <async-supported>true</async-supported>
  <!-- default init-param !-->
  <init-param>
    <param-name>allowedOrigins</param-name>
    <param-value>*</param-value>
  </init-param>
  <init-param>
    <param-name>allowedMethods</param-name>
    <param-value>GET,POST,OPTIONS,DELETE,PUT,HEAD</param-value>
  </init-param>
  <init-param>
    <param-name>allowedHeaders</param-name>
    <param-value>origin, content-type, accept, authorization,Access-
Control-Request-Method, gms_user</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>cross-origin</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```

Allowed Origins

This is the list of allowed origins that can be specific origins. For example, <https://domain1.com>. A matched origin is listed in the 'Access-Control-Allow-Origin' response header of preflight actual CORS requests.

If the below property is not available, then, by default, all origins are allowed.

```
<parameter name="cors-allowed-origins" displayName="cors-allowed-origins" mandatory="false">
```

```
<description><![CDATA[This should have the origins which we need to allow in CORS
policy, multiple origins can be
separated by coma]]></description>
<valid-description><![CDATA[]]></valid-description>
<effective-description/>
  <format type="string" default="-Dcom.genesyslab.common.cors.allowed-
origins=http://sample1.com,https://sample1
.com,http://sample2:8080,https://sample2:8080" />
  <validation/>
</parameter>
```

Important

- The value for this property should be a literal match for the string, that is, <http://sample1.com> is not equal to <https://sample1.com>.
- There are no line breaks in between domains or domain names.
- Blank spaces are not permitted in between domains or domain names.

Allowed Methods

Set the HTTP methods to allow specific methods, for example, **GET**, **POST**, and so on. By default all methods are allowed.

```
<parameter name="allowed-method" displayName="com.genesyslab.common.cors.allowed-methods"
mandatory="false">
  <description><![CDATA[This should have the methods which we need to allow in cors
policy, multiple methods can be separated by coma]]></description>
  <valid-description><![CDATA[]]></valid-description>
  <effective-description/>
  <format type="string" default="-Dcom.genesyslab.common.cors.allowed-
methods=GET,POST,PUT" />
  <validation/></validation>
</parameter>
```

Important

- There are no line breaks in between method names.
- Blank spaces are not permitted in between method names.

Allowed Headers

Set the list of headers that a pre-flight request can list as allowed for use during an actual request. As per the CORS spec, the header name is not required to be listed if it is one of the below:

- Cache-Control

- Content-Language
- Expires
- Last-Modified
- Pragma

```
<parameter name="allowed-header" displayName="allowed-header" mandatory="false">
  <description><![CDATA[This should have the headers which we need to allow in cors
policy, multiple headers can be separated by coma]]></description>
  <valid-description><![CDATA[]]></valid-description>
  <effective-description/>
  <format type="string" default="-Dcom.genesyslab.common.cors.allowed-headers=X-
Requested-With,Origin" />
  <validation></validation>
</parameter>
```

Important

- There are no line breaks in between header names.
- Blank spaces are not permitted in between header names.

Add Custom Headers

To add custom headers to the GMS response (X-Frame-Options, X-Content-Type-Options, XSS-Protection, Content-Security-Policy, Access-Control-Allow-Headers, and so on), use Jetty to rewrite handles. This procedure is documented in the [Rewrite Handler](#) chapter of the official Jetty documentation. For instance, the following snippets added to the **jetty-rewrite.xml** file create the appropriate custom headers for the Storage API, based on the `/genesys/1/storage/*` pattern attribute:

```
<Call name="addRule">
  <Arg>
    <New class="org.eclipse.jetty.rewrite.handler.HeaderPatternRule">
      <Set name="pattern">/genesys/1/storage/*</Set>
      <Set name="headerName">Access-Control-Allow-Credentials</Set>
      <Set name="headerValue">>false</Set>
    </New>
  </Arg>
</Call>
<Call name="addRule">
  <Arg>
    <New class="org.eclipse.jetty.rewrite.handler.HeaderPatternRule">
      <Set name="pattern">/genesys/1/storage/*</Set>
      <Set name="headerName">X-Frame-Options</Set>
      <Set name="headerValue">SAMEORIGIN</Set>
    </New>
  </Arg>
</Call>
<Call name="addRule">
  <Arg>
```

```
<New class="org.eclipse.jetty.rewrite.handler.HeaderPatternRule">
  <Set name="pattern">/genesys/1/storage/*</Set>
  <Set name="headerName">X-XSS-Protection</Set>
  <Set name="headerValue">1; mode=block</Set>
</New>
</Arg>
</Call>
<Call name="addRule">
  <Arg>
    <New class="org.eclipse.jetty.rewrite.handler.HeaderPatternRule">
      <Set name="pattern">/genesys/1/storage/*</Set>
      <Set name="headerName">Content-Security-Policy</Set>
      <Set name="headerValue">default-src 'self'</Set>
      <Set name="terminating">true</Set>
    </New>
  </Arg>
</Call>
<Call name="addRule">
  <Arg>
    <New class="org.eclipse.jetty.rewrite.handler.HeaderPatternRule">
      <Set name="pattern">/genesys/1/storage/*</Set>
      <Set name="headerName">Access-Control-Allow-Headers</Set>
      <Set name="headerValue">customerHeader1,customerHeader2</Set>
    </New>
  </Arg>
</Call>
```

As a result, you will get the following storage response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: false
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Content-Security-Policy: default-src 'self'
Content-Type: application/json;charset=utf-8
Transfer-Encoding: chunked
```

```
{"GMS_KEY": "GMS_VALUE"}
```

Set up DoS Filters to Limit Requests

An attacker can consume all of some required resources by generating enough traffic from your host to prevent legitimate users from using the system. See [Application Denial of Service](#)

The Denial of Service (DoS) filter limits exposure to request flooding, whether malicious or as a result of a misconfigured client. The DoS filter keeps track of the number of requests from a connection per second. If the number of requests exceeds the limit, JETTY rejects, delays, or throttles additional requests, and sends a warning message. The filter works on the assumption that the attacker might be written in a simple blocking style, so by suspending requests you are hopefully consuming the attacker's resources.

To configure the DoS filter in GMS, edit the `<GMS_HOME>/etc/webdefault-ee8.xml` file. Configure the options list below, save, and restart GMS to take the new settings into account.

Important

If you are using a load balancer in front of the GMS nodes, the load balancer must set the X-Forwarded-For HTTP header to allow the DoS filter to identify the client IP and control the request rates.

Parameter name	Default Value	Description
maxRequestsPerSec	25	Maximum number of requests per second from a connection. Additional requests are first delayed, then throttled.
delayMs	100	Delay in milliseconds that all requests over the rate limit must wait before they get processed: • -1 = Reject request • 0 = No delay
maxWaitMs	50	Maximum time to wait, in milliseconds, blocking for the throttle semaphore.
throttledRequests	5	Number of requests over the rate limit able to be considered at once.
throttleMs	30000L	Time, in milliseconds, to asynchronously wait for the throttle semaphore.
maxRequestMs	30000	Time, in milliseconds, to allow the request to run.
maxIdleTrackerMs	30000	Time, in milliseconds, to keep track of request rates for a connection, before deciding that the user has gone away, and discarding it.
insertHeaders	true	Keep true to insert the DoSFilter headers into the response; otherwise, false.
remotePort	false	Set to true to not use session tracking, then track by IP+port (effectively connection).
ipWhitelist		A comma-separated list of IP addresses that will not be rate-limited.
managedAttr		Set to true to set this servlet as a ServletContext attribute with the filter name as the attribute name. This allows a context external mechanism such as for

Parameter name	Default Value	Description
		example JMX via ContextHandler.MANAGED_ATTRIBUTES) to manage the configuration of the filter.

The following code is an example of DoS Filter file. In the **webdefault-ee8.xml** file the below snippet should be placed before the `<servlet>` tag.

```
<filter>
  <filter-name>DoSFilter</filter-name>
  <filter-class>org.eclipse.jetty.ee8.servlets.DoSFilter</filter-class>
  <async-supported>true</async-supported>
  <init-param>
    <param-name>maxRequestsPerSec</param-name>
    <param-value>200</param-value>
  </init-param>
  <init-param>
    <param-name>ipWhitelist</param-name>
    <param-value>127.0.0.1</param-value>
  </init-param>
</filter>
```

Secure your GMS Sessions

Attack	Description
Session Fixation	Session fixation attacks represent a potential risk where a malicious attacker tries to create a session by accessing a site, then persuades other users to log in with the same session by sending them a link containing the session identifier as a parameter, for example.

To prevent attacks on your sessions, you should configure the following mechanisms.

Mechanism	Description
HttpOnly flag is true by default	HttpOnly is an additional flag included in a Set-Cookie HTTP response header. Using the HttpOnly flag when generating a cookie helps mitigate the risk of client side script accessing the protected cookie (if the browser supports it).
Enable the secure flag in HTTPS environment	The secure flag is an option that you can set in the application server when sending a new cookie to the user within an HTTP Response. The purpose of the secure flag is to prevent cookies from being observed by unauthorized parties due to the transmission of a cookie in clear text. To enable it, navigate to the <code><GMS_HOME>/etc</code> folder and edit the <code>webdefault-ee8.xml</code> file. Secure flag setting

Mechanism	Description
	<pre> <session-config> <session-timeout>20</session- timeout> <cookie-config> <secure>false</secure> <!-- true if HTTPS is used--> <path>/genesys</path> <http-only>true</http-only> <max-age>86400</max-age> </cookie-config> </session-config> </pre>
Session timeout (default is 20 minutes)	To change this timeout, navigate to the <GMS_HOME>/etc/ folder, and the edit webdefault-ee8.xml file. Session timeout setting <pre> <session-config> <session-timeout>20</session- timeout> <!-- Value is in minute--> ... </session-config> </pre> The session timeout defines the action window time for a user, and thus, the period while an attacker can try to steal and use an existing user session. • Set session-timeout to a minimal value, according to the application context. • Avoid "infinite" session timeout. • Prefer a declarative definition of the session timeout to apply a global timeout for all application sessions. • Trace session creation/destruction to analyze the creation trend and detect an abnormal number of session creations, that would indicate some application profiling phase during an attack.

Configure Access Rules for Security Gateway in Front of Genesys Mobile Services

The following access rules should be used as a model for your environment, allowing a list of services provided by your Genesys Mobile Services deployment to be accessed while restricting the use of internal-only services.

Important

Only allow access to the limited set of services listed by name.

Allow access from the end-user application or proxy - mobile, web, etc:

```
-- async notifications over HTTP API:
If client is using cometd transport (typically for chat):
{base url:port}/genesys/{version}/cometd
-- service API:
{base url:port}/genesys/{version}/service/{service name}
{base url:port}/genesys/{version}/service/{id}/storage
{base url:port}/genesys/{version}/service/{id}
{base url:port}/genesys/{version}/service/{id}/{request name}
-- chat media API:
{base url:port}/genesys/{version}/service/{id}/ixn/chat
{base url:port}/genesys/{version}/service/{id}/ixn/chat/refresh
{base url:port}/genesys/{version}/service/{id}/ixn/chat/disconnect
{base url:port}/genesys/{version}/service/{id}/ixn/chat/startTyping
{base url:port}/genesys/{version}/service/{id}/ixn/chat/stopTyping
{base url:port}/genesys/{version}/service/{id}/ixn/chat/*
```

Only when client need direct access allow (in most cases only

ORS/scxml need it):

```
-- storage API
{base url:port}/genesys/{version}/storage/{ttl}
{base url:port}/genesys/{version}/storage/{data id}/{ttl}
{base url:port}/genesys/{version}/storage/{data id}
{base url:port}/genesys/{version}/storage/{data id}/{key}
```

```
-- callback (management) API
{base url:port}/genesys/{version}/service/callback
```

`/callback-execution-name/{service_id}`

```
{base url:port}/genesys/{version}/service/callback
```

`/callback-execution-name/{service_id}`

```
{base url:port}/genesys/{version}/service/callback
```

`/callback-execution-name/availability?timestamp=value}`

```
{base url:port}/genesys/{version}/admin/callback
```

`/queues?target={target_name}&end_time={iso_end_time}`

Allow access from load balancer for node health check:

```
{base url:port}/genesys/{version}/node
```

Allow access from the intranet:

```
-- admin UI
{base url:port}/genesys/admin/*
{base url:port}/genesys/{version}/admin/*
{base url:port}/genesys/{version}/reports/*
{base url:port}/genesys/{version}/statistic/*
-- datadepot (typically accessed from ORS/scxml):
{base url:port}/genesys/{version}/datadepot/{tenantId}
```

/ {activityName} / dates

```
{base url:port}/genesys/{version}/datadepot/{tenantId}/activities
```

```
{base url:port}/genesys/{version}/datadepot/{tenantId}
```

/ {activityName} / aggregates

```
{base url:port}/genesys/{version}/datadepot/{tenantId}
```

/ {activityName} / aggregates / unique

```
-- all built-in services (except chat)
{base url:port}/genesys/{version}/service/request-access
{base url:port}/genesys/{version}/service/match-interaction
-- notification API (typically accessed from ORS/scxml):
{base url:port}/genesys/{version}/notification/subscription
{base url:port}/genesys/{version}/notification/subscription/{id}
{base url:port}/genesys/{version}/notification/publish
```

Mobile Native Push Notification Configuration Details

Three major mobile platforms are currently supported: Android, Apple, and Windows. Details about Genesys Mobile Services configuration could be found on the [Push Notification Service](#) page of the API Reference.

You can set sensitive options in different sections of your application configuration. In the following example, the configuration structure allows you to set the password in another section.

```
[push]
apple=apple-desc;
debug-apple=debug-apple-desc
...
[apple-desc]
password=*****
apple.keystore=xxxxxxxxxxx
[debug-apple-desc]
password=*****
apple.keystore=xxxxxxxxxxx
```

In case of:

- IOS (APNS): The password option is used instead of "apple.keystorePassword".

- Android (FCM): The password option is used instead of "fcm.apiKey".
- Microsoft (WNS): The password option is used instead of "wns.clientSecret".

Set all the mandatory options and keep optional options in the push section. See the [options' detail](#) for more information.

If GMS is deployed behind a firewall, edit your rules to allow the server fcm.googleapis.com and range 5228-5230 for ports, as detailed in the [FCM ports and your firewall](#) section of the Firebase Cloud Messaging documentation.

Hide Sensitive Data in Logs

Genesys recommends using log data filtering to hide sensitive configuration data in log files. The given product will then use that filter to determine what content should be filtered or masked when creating log files. The only real interfaces to Genesys Mobile Services are APIs and having to configure filter criteria every time you add a new parameter or API is cumbersome and complicated. Therefore, Genesys Mobile Services supports filtering and masking data in log files for any API parameter that matches the following naming convention:

- Filter: Any parameter name that is prefixed with XXX will be filtered out of the log files entirely. Example: *XXX_FilterParam*
- Mask: Any parameter name that is prefixed with MMM will be masked in the log files, showing in the log with the letters MMM. Example: *MMM_MaskParam*

Use of the Storage API

For security reasons, if you plan using the [GMS Storage API](#), you can define a list of supported mime types in the allowed-mime-types option and configure allow-file-control = true to enable file control.