Genesys Pulse Deployment Guide

Configuring System Security

4/5/2026

Contents

Configuring System Security

Genesys Pulse has features that enhance your system security. This section discusses Genesys Pulse security features and describes how to configure them.

TLS: Configuring the Genesys Pulse Database

You must configure your Oracle, Microsoft SQL, or PostgreSQL server to use TLS. In addition to the appropriate procedure below, refer to the documentation that came with your database for information on how to use TLS security.

Oracle

1. Set up the Genesys Pulse database (for Oracle).
2. Configure Oracle as described in the related database guides, and configure a TCPS listener. See Management Framework documentation for more information.
3. Configure the **jdbc_url** option in the **[pulse]** section of your Genesys Pulse DAP application object:
   `jdbc_url=jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=<Database host>)(PORT=<Database port>))(CONNECT_DATA=(SERVICE_NAME=<Database Service name>)))`

SSL connection using TLS v1.2
JDK 7 and JDK 8 releases support TLS v1.2 protocol. Other protocols, such as TLS v1.1, TLS v1, SSL v3, and SSL v2 have security vulnerabilities. Genesys recommends to use the latest standard TLS v1.2 version and use more secure SSL cipher suites.

The correct JDBC Thin driver is required in order to use TLS v1.2.

> **Important**
>
> If you are using the `ojdbc8.jar` from 12.2.0.1 version then you are all set.
> If you are using the 12.1.0.2 JDBC driver, you need to either download the 12.1.0.2 patched driver or apply the patch (that allows TLS v1.2) for the bug 19030178. The patch allows TLS v1.2 but does not enable it by default. So, you must set the `oracle.net.ssl_version=1.2` property. This property can be set either as the system property (using -D) or through the datasource properties.

MS SQL

1. Set up the Genesys Pulse database (for MS SQL).

2. Configure Microsoft SQL Server as described in the related database guides. See Management Framework documentation for more information.
3. Configure the **jdbc_url** option in the **[pulse]** section of your Genesys Pulse DAP application object:
   `jdbc_url=jdbc:sqlserver://<Database host>:<Database port>;databaseName=<Database name>;encrypt=true;trustServerCertificate=false`

PostgreSQL

1. Set up the Genesys Pulse database (for PostgreSQL).
2. Configure PostgreSQL as described in the related database guides. See Management Framework documentation for more information.
3. Configure the **jdbc_url** option in the **[pulse]** section of your Genesys Pulse DAP application object:
   `jdbc_url=jdbc:postgresql://<Database host>:<Database port>/<Database name>?ssl=true&sslcert=<path to certificate>&sslkey=<path to key>&sslrootcert=<path to root cerificate>&sslmode=verify-full`

> **Important**
>
> The certificate key must be in the `pkcs8` format. You can use the `openssl` utility to convert the key:
> `openssl pkcs8 -topk8 -nocrypt -inform PEM -outform DER -in <server.key> -out <server-key.pk8>`

TLS: Configuring Genesys Pulse

Genesys Pulse supports Transport Layer Security (TLS) communications between Genesys Pulse server and client-side connections using the web browser interface.

Genesys Pulse can support connections through HTTP or HTTPS simultaneously. This is controlled by the **supported_protocol** parameter (valid values are `http`, `https`, or `both`) in the `pulse.properties` file, located in the `conf` directory of your Genesys Pulse installation.

> **Important**
>
> Starting with release 9.0.006, Genesys Pulse does not support hard-coded encryption keys for passwords:
>
> - Genesys Pulse no longer supports the encrypted form of the `keystore_password` property for the unofficial HTTPS activation workaround. The `keystore_password` property is not encrypted and can be passed as an environment variable or command line argument.

- Genesys Pulse no longer sends user passwords in encoded form. Genesys Pulse must be used with HTTPS enabled or behind HTTPS-enabled proxy or load balancer to protect users credentials.

Starting with Genesys Pulse release 9.0.006, use the following steps to set up HTTPS:

1. Create a keystore file (PKCS12 and JKS keystore types are supported) with the private key and certificate for the Genesys Pulse server using one of the following ways:
   - Create a keystore with a self-signed certificate by executing the following command:

     ```
     keytool -keystore <full keystore path> -alias pulse -genkey -keyalg RSA
     ```

     and enter the required information as prompted.
   - Export the existing certificate to the PKCS12 format and import into an existing keystore:

     ```
     openssl pkcs12 -export -in <src certificate> -inkey <src key> -out <pkcs12 keystore
     path> -name pulse
     keytool -importkeystore -srckeystore <pkcs12 keystore path> -srcstoretype pkcs12
     -destkeystore <full keystore path>
     ```
2. Define the **https_port**, **supported_protocol**, and **keystore_path** parameters in the `pulse.properties` file. The default **https_port** is 443.
   - `https_port=8443`
   - `supported_protocol=https` or `both`
   - `keystore_path=`full path to the location of the keystore
3. Choose how the keystore password is provided to Genesys Pulse. Genesys Pulse application supports the following options:
   - Add the command line argument to java command (by changing the `pulse_startup.sh` (on Linux) or `pulse_startup.bat` (on Windows) file or by adding this option to the `JAVA_OPTS` environment variable):
     `-Dcom.genesys.pulse.keystore.password=<password>`
   - Set the password as a value of the `KEYSTORE_PASSWORD` environment variable.
   - Add the `keystore_password` property with your password to the `pulse.properties` file.

   [+] Example. How to obscure password with base64 encoding

   Linux:
   I. Obscure the password:

   ```
   echo mypassword | base64 > <path_to_key>/key.txt
   ```

   II. Use obscured password in the startup script:

   ```
   java -Dcom.genesys.pulse.keystore.password=$(base64 -d <path_to_key>/key.txt) -jar
   ./pulse.war
   ```

   Or set the environment variable that Genesys Pulse can read:

```
export KEYSTORE_PASSWORD=$(base64 -d <path_to_key>/key.txt)
```

Windows:
I. Obscure the password:

```
echo mypassword > key.txt
certutil -encode key.txt tmp.b64 && findstr /v /c:- tmp.b64 > key.txt
del tmp.b64
```

II. Use obscured password in the startup script:

```
certutil -decode key.txt tmp.b64
set /P password=<tmp.b64
del tmp.b64
java -Dcom.genesys.pulse.keystore.password=%password% -jar ./pulse.war
```

Or set the environment variable that Genesys Pulse can read:

```
certutil -decode key.txt tmp.b64
set /P password=<tmp.b64
del tmp.b64
setx KEYSTORE_PASSWORD %password%
```

4. (optional) Choose how the individual key password is provided to Genesys Pulse. If not provided, keystore password will be used.
   Genesys Pulse application supports the following options:
   - Add the command line argument to java command (by changing the `pulse_startup.sh` (on Linux) or `pulse_startup.bat` (on Windows) file or by adding this option to the `JAVA_OPTS` environment variable): `-Dcom.genesys.pulse.key.password=<password>`
   - Set the password as a value of the `KEY_PASSWORD` environment variable.
   - Add the `key_password` property with your password to the `pulse.properties` file.
5. Start Genesys Pulse.

Cipher Lists

> **Important**
> Genesys does not support the tools mentioned in this section. They are used here only to illustrate.

The Java Runtime Environment (JRE) provides the cipher suites that Pulse uses. Before enabling TLS, please ensure JRE supports actual cipher suites that your browser can accept. Otherwise, refer to your JRE distribution documentation to enable ciphers accepted by your browser.

- You can access a list of ciphers accepted by your browser here.
- For a list ciphers provided by JRE:
  1. Download and extract Ciphers.zip to a temporary location.
  2. Execute the following command from that location:

```
java Ciphers
```

- If a cipher suite or protocol that you require is provided by JRE but not enabled by default in Pulse, there is a mechanism that lets you enable the cipher suite or protocol during startup. Be aware that you must specify in preference order.
- If a vulnerability is discovered in a cipher or protocol, or if it is considered too weak to use, you can exclude it during Pulse startup.
- If a cipher suite or protocol is both included and excluded as part of the same configuration, it is disabled (excludes will always win).

> **Important**
>
> By default, the following weak cipher suites are disabled in Pulse: SSL_*, TLS_RSA_*, MD5, SHA, and SHA1-based. You can still accept the security risk and enable them by removing them from the excluded cipher suites. Once a weak cipher is enabled, you will need to restart Pulse and you will see a warning in the Pulse log:
> `Weak cipher suite <suite name> enabled for Server.`

The table below lists the available parameters in pulse.properties (regex values supported).

Parameter	Description	Default (if not specified)
setIncludeProtocols	Specifies one or more protocols to be supported.	NA
setIncludeCipherSuites	Specifies one or more cipher suites to be used for encoding data.	NA
setExcludeProtocols	Specifies one or more protocols that are not to be supported.	`"SSL", "SSLv2", "SSLv2Hello", "SSLv3"`
setExcludeCipherSuites	Specifies one or more cipher suites that are not to be used for encoding data.	`"^.*_(MD5|SHA|SHA1)$","^TLS_RSA_.*$","^SSL_.*$","^.*_NULL_`
addExcludeProtocols	Specifies additional protocols that are not to be supported.	NA
addExcludeCipherSuites	Specifies additional cipher suites that are not to be used for encoding data.	NA

Example:

```
host=pulse-host
port=8080
app=pulse
...
setExcludeCipherSuites=^.*_(MD5|SHA|SHA1)$
setIncludeCipherSuites=TLS_ECDHE_.*,TLS_RSA_.*
```

HTTP Strict Transport Security (HSTS)

HSTS is disabled by default and you can enable it by setting the **enable_hsts** option in the `pulse.properties` file to `true`. Once HSTS is enabled, Genesys Pulse prevents downgrading of encrypted HTTPS connection to unencrypted HTTP. It is implemented by sending a response header record from the server indicating that compliant Web browsers or other HTTP client programs must use HTTPS and they must display the appropriate confirmation message or an error message in the browser console.

Securely embed Pulse into iframe

Starting with 9.0.008.03 Pulse allows embedding into iframe. Use the following step to allow embedding:

- Make sure Pulse is configured over HTTPS
- Set appropriate options in `pulse.properties` file:
  - `disable_xframe_options=true`
  - `session_samesite=none` (or `lax` if the iframe source is on the same domain as Pulse)
  - `supported_protocol=https`, or `supported_protocol=both` and `enable_hsts=true`
  - edit `content_security_policy` property to contain `frame-ancestors` directive with valid parents that may embed, for example: `content_security_policy=frame-ancestors 'self' https://example.com;`