Deploying the openPDC on POSIX Platforms

Grid Protection Alliance, Inc.
www.gridprotectionalliance.org

Last Updated: March 2015

Contents

Introduction 3
Installation Prerequisite: Mono 4
Installing the openPDC 5
  Troubleshooting 5
Running the openPDC 6
  Execution as Terminal Application 6
  Execution as Daemon 6
    Controlling the Daemon 7
    Running the Remote Console 7
  Testing the Default Configuration 8
    IEEE C37.118 Output Stream 8
    Gateway Exchange Protocol 8
Configuring the openPDC 9
  Manual SQL Configuration 9
  Using the openPDC Manager 9
    Install 9
    Enable Folder Share 9
    Configure 10
  Using Other Databases 10
  Enabling the openHistorian 11
Completing Security Configuration 11
  Managing User Accounts 12
    Adding New Users 12
    Authenticating Cross Platform Users 13
    Disabling Pass-Through Authentication 14
  Updating Transport Layer Security 14
    Manually Creating a Certificate 14
    Using an Existing Certificate 15
    Changing the TLS Version 16
  Running the openPDC with Elevated Rights 17
  Securing the Configuration Database 17
Running on a Raspberry Pi 18

Introduction

With the release of version 2.1 of the openPDC, GPA has implemented features that allow the openPDC to easily be used in deployments running a POSIX[1] compatible operating system such as Linux and Apple OS X. GPA has tested several common distributions and hardware combinations that include:

- Debian on x86 style hardware (64-bit)
- Ubuntu on HyperV (64-bit)
- Ubuntu on SEL-3354 hardware (32-bit)
- Rasbian on Raspberry Pi and Pi 2 hardware (32-bit)
- Apple OS X on MacBook Air hardware (32-bit)

This document describes the process to install, configure and operate the openPDC on a POSIX compatible operating system. The directions that follow focus on Linux deployments of the openPDC; however, where necessary, there are exceptions noted that may be needed for Mac OS X deployments.

[1] POSIX, Portable Operating System Interface, is a family of IEEE standards to assure compatibility among operating systems. Most distributions of Linux, Apple's OS X and iOS as well as Android are POSIX compatible.

Installation Prerequisite: Mono

The openPDC is a .NET application and as such it requires a Common Language Runtime (CLR) engine to execute. For use in POSIX compatible environments, the openPDC uses Mono as its runtime engine. Mono is a cross platform implementation of the .NET framework that is widely available on various hardware and software architectures. For the openPDC version 2.1, Mono version 3.12 is required[2].

To install Mono, follow the installation guide using Xamarin packages for your distribution:

http://www.mono-project.com/download/

Once Mono is installed, validate that the system is running the proper version of Mono using the ***mono*** application with the version parameter, ***-V***, from a terminal session. The reported version should be "version 3.12" (or later) as shown below.

```
$ mono -V

Mono JIT compiler version 3.12.0 (tarball Mon Jan 19 15:40:05 UTC 2015)
Copyright (C) 2002-2014 Novell, Inc, Xamarin Inc and Contributors. www.mono-project.com
        TLS:           __thread
        SIGSEGV:       altstack
        Notifications: epoll
        Architecture:  x86
        Disabled:      none
        Misc:          softdebug
        LLVM:          supported, not enabled.
        GC:            sgen
```

If after installation of the latest version of Mono the ***mono*** application reports an older version[3], modify the system path so that version 3.12 of Mono is found first.

If there are no Mono version 3.12 packages available for your distribution, then Mono must be compiled manually. In this case, follow these instructions:

http://www.mono-project.com/docs/compiling-mono/

[2] It is possible that an older version of Mono may work, but the openPDC version 2.1 incorporates many features of .NET 4.5 that are only available in later versions of Mono. The openPDC has not been fully tested with earlier versions of Mono; therefore, use of Mono version 3.12 (or later) is strongly recommended.
[3] It is possible to have multiple versions of Mono simultaneously installed on a system.

Installing the openPDC

Due to the vast number of POSIX operating systems and hardware combinations, there are currently no precompiled deployment packages for the openPDC[4]. To begin the process of installing the openPDC, download the following script file[5]:

http://www.gridprotectionalliance.org/Products/openPDC/Scripts/install-openPDC.sh

Mac OSX Note:

The ***wget*** application must be available before running this script. To install wget, install Brew from http://brew.sh and then run:

```
brew install wget
```

Note that X11/XQuartz is not required for openPDC.

For a default installation and configuration, the script can be executed right away to get things started. However, to customize the installation or complete the security configuration, review the following installation options.

Below are the available script parameters to allow customization of the installation:

-n: Use nightly builds
-p: Preserve source code
-d: Install destination (defaults to /opt/openPDC/)
-u: Uninstall the openPDC

To fully enable security for this installation, as detailed later, preserve the local source code by using the ***-p*** parameter. To install the openPDC using the latest source code instead of the published stable release, use the ***-n*** parameter to install the latest nightly build.

The script should be run using the ***bash***[6] shell with root access, for example, to install the openPDC and preserve the source code after installation run the following:

```
sudo bash install-openPDC.sh -p
```

Troubleshooting

If there are issues with the script during the installation process, validate that the Mono build system, XBuild, is working properly. To do this, execute the ***xbuild*** application with a version parameter, ***/version***, from a terminal session and verify that the application reports version 12.0 of the XBuild Engine, similar to the following:

[4] As time allows, GPA will add deployment packages for common POSIX distributions, e.g., Debian and Ubuntu on Linux and Apple OS X. Third-parties are welcomed to assist with maintaining up-to-date deployment packages for the openPDC on various distributions.
[5] During execution the script will require internet access to download files to be installed. Since other scripts and source code files will be downloaded as part of the installation, it is best to run the script from its own folder.
[6] The ***bash*** shell command is required over the normal ***sh*** command to be able to properly test for Mac OS X platforms. Most all scripts to be executed in this document will require bash for this reason.

```
$ xbuild /version
XBuild Engine Version 12.0
Mono, Version 3.4.1.0
Copyright (C) 2005-2013 Various Mono authors
```

If the version of XBuild is not the same as reported above or the application fails to run, verify the path to the ***xbuild*** command is correct.

If the installation script needs to be run again, clear out the temporary files in the installation folder (i.e., the folder where script was run) and delete the destination folder (e.g., /opt/openPDC) before then next attempt.

Running the openPDC

The installation script will automatically deploy the openPDC with a default configuration, as such the application can be run immediately. The openPDC can be started as a terminal application or registered as a daemon that will run in the background and startup automatically.

Execution as Terminal Application

To test the openPDC or validate its current configuration, it can simply be run as a terminal application. To run the openPDC as a terminal application, execute the ***openPDC.exe*** application using the Mono runtime and the ***RunAsConsole*** parameter:

```
sudo mono /opt/openPDC/openPDC.exe -RunAsConsole
```

This will start the openPDC as a console style application with feedback reported directly to the current session. When running in this mode, input can be accepted just by typing. Note that keyboard input is being logged even if any user input is visually interrupted by a new status update from the openPDC. As an example to start issuing commands, just type ***Help*** and press enter for a list of possible commands.

To exit the application and stop the openPDC, just type the command ***Exit*** and press enter.

Execution as Daemon

To run the openPDC automatically at startup, the application must be deployed to run as daemon in the background. To configure the daemon to run automatically startup, execute the registration script[7]:

```
sudo bash register-openPDC.sh
```

If the openPDC needs to be unregistered later, use the ***-u*** parameter with this script to unregister the service. Note that uninstalling the openPDC will also automatically unregister the service.

When running the openPDC as a background service, input and output from the application must be handled by another application, such as the openPDC Remote Console. See the *Running the Remote Console* section for more details.

[7] Note that needed scripts are downloaded as part of the install process. In order to run properly both the ***openPDC.sh*** and ***register-openPDC.sh*** scripts are required to be in the same folder during execution; for Mac OSX, this also includes the ***openPDC.plist*** file.

Controlling the Daemon

To manually control the openPDC service running as a daemon, use the following commands:

Start Service

```
sudo /opt/openPDC/openPDC start
```

Restart Service

```
sudo /opt/openPDC/openPDC restart
```

Pause Service

```
sudo /opt/openPDC/openPDC pause
```

Resume Service

```
sudo /opt/openPDC/openPDC resume
```

Stop Service

```
sudo /opt/openPDC/openPDC stop
```

Running the Remote Console

To view the current activity and issue commands to the openPDC while it is running as a daemon, run the openPDC Remote Console.

Run Locally

The remote console is already preconfigured to properly connect to the locally running openPDC when executed from the same machine. To start the remote console, execute the following command from a terminal session[8]:

```
mono /opt/openPDC/openPDCConsole.exe
```

Run Remotely

The openPDC can be accessed from a remote machine by running the remote console application on another machine, including from a Windows installation[9]. In order for the remote console application to properly connect to the machine running the openPDC, the machine's IP address or DNS name will be required. To make these changes on the remote machine, make sure the remote console application is not running then edit the ***openPDCConsole.exe.config*** file and change the following settings:

[8] The remote console application does not require being run as the root user, i.e., no ***sudo*** required.
[9] Run the Windows installer for the openPDC to get the openPDC Remote Console running in a Windows environment; download from here: https://openpdc.codeplex.com/releases/. The installed components can be reduced to only tools during the installation process by deselecting the openPDC Service option.

```
<?xml version="1.0"?>
<configuration>
  <categorizedSettings>
    <remotingClient>
      <add name="ConnectionString" value="server=openPDC:8500; interface=0.0.0.0 />
      <add name="IntegratedSecurity" value="False />
    </remotingClient>
</configuration>
```

Replace the ***openPDC*** text, as shown in the ***ConnectionString*** above, with the machine name or IP that is running the openPDC. Do not forget to include "`; interface=0.0.0.0`" in the changes as this forces the connection to be IPv4[10]. If the openPDC Remote Console application is being used on a Windows machine to connect to the openPDC service running in a POSIX environment, also set ***IntegratedSecurity*** to ***False***[11]. Integrated security allows an application to login as the currently authenticated user without reentering credentials, but this is only available on Windows.

Exit Console

The remote console application can be closed by issuing an ***Exit*** command and pressing enter. Note that unlike when running the openPDC as a terminal application, issuing the ***Exit*** command from a remote console session will not terminate the remotely running openPDC application, it will only close the remote console session.

Testing the Default Configuration

When the openPDC is installed and running with its default configuration it includes a sample PMU (repeating file based input), an IEEE C37.118 output stream and a Gateway Exchange Protocol data publisher. The openPDC can be verified to be operating as expected by exercising its outputs.

IEEE C37.118 Output Stream

The openPDC's IEEE C7.118 output stream can be easily tested by running the PMU Connection Tester[12]. Also, if there is another PDC or application available that can interpret IEEE C37.118, then that system can be connected to the openPDC output stream as well.

The default configuration defines a test output stream that will be listening on TCP port 8900. The output stream is configured with a TCP channel only, so both commands and data will be handled through the same channel.

Gateway Exchange Protocol

The Gateway Exchange Protocol, or GEP, is an open source measurement-based publish/subscribe transport protocol used for exchanging time-series data and automatically synchronizing meta-data

---

[10] The default configuration is setup as IPv4 to support as many possible distributions as possible. If the system running the openPDC supports IPv6, the server and client connections can be configured to use IPv6 by specifying "`; interface=::0`" in the relevant configuration settings and connection strings. The ***interface*** setting is used to specify the IP of the network interface controller (NIC) to use for the connection – an IP of zero means that the default NIC should be used for the connection; the format of the interface IP setting determines the IP stack version, i.e., IPv4 or IPv6, to use for the connection.

[11] The ***IntegratedSecurity*** setting is ignored in POSIX environments.

[12] Currently the PMU Connection Tester application is still only available from a Windows machine.

between two applications. The protocol supports sending real-time and historical data at full or down-sampled resolutions. When sending historical data the replay speed can be controlled dynamically to provide data as quickly as possible, e.g., a data download, or slowed for visualization streaming.

Included with the openPDC POSIX installation is a program that can verify that the GEP publication server is working properly called the GEP Subscription Tester. This application will trend received data from the openPDC in real-time and replay historical data if the local openPDC archive is enabled (see section *Enabling the openHistorian*). The GEP Subscription Tester can be run on the same machine as the openPDC as long as the POSIX environment has a UI[13].

The GEP Subscription Tester application can be found in the ***/opt/openPDC/GEPTester*** folder.

The GEP Subscription Tester can also be run from another machine and connected to the openPDC remotely. The GEP Subscription Tester can run on Linux, Windows, Apple OS X and Android devices. See the following link to download the tool for use on other platforms:

http://openpdc.codeplex.com/wikipage?title=GEP%20Subscription%20Tester

Configuring the openPDC

ccc

Manual SQL Configuration

Use SQL statements to configure openPDC…

See the following link for a complete reference on how to manually configure the openPDC by updating the configuration database:

https://openpdc.codeplex.com/wikipage?title=Manual%20Configuration

Using the openPDC Manager

The openPDC Manager application can be used to simplify configuration of the openPDC. Currently the openPDC Manager is written as a WPF application and not available to run on Mono. However, an instance of the openPDC Manager running on a Windows machine can be used to remotely configure and monitor a Linux or Apple OS X based openPDC service.

Install

Install from here…

Enable Folder Share

Since the openPDC will need access to the default SQLite database to make configuration changes, a file share on the machine running the openPDC will need to be established that is accessible from the Windows machine running the openPDC Manager.

To enable a share on a POSIX system, the **net usershare** command can be used, for example:

[13] The GEP Subscription Tester is a visualization application built using the Unity 3D gaming platform. This application will only run properly from within an actual UI environment; attempting to use the application from VNC or other virtualized UI environments may not work.

```
sudo net usershare add openPDC /opt/openPDC "openPDC" everyone:F guest_ok=y
```

Note that the ***net*** command requires that Samba already be installed, for example:

```
sudo apt-get install samba samba-common-bin
```

For more information on installing Samba, see the following:

https://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/install.html#id2551914

Once the share exists, a Windows network drive must be mapped to the folder so that the openPDC Manager will be able to successfully access the SQLite configuration database.

Configure

Run configuration setup utility and follow these steps…

Using Other Databases

The default openPDC installation configuration comes with a preconfigured SQLite database. SQLite is convenient because it is self-contained and requires no additional installation components. However, several other database types are natively supported by the openPDC, including: SQL Server, MySQL and Oracle.

Database configurations can be installed remotely or locally (as supported) with the openPDC service configured to connect to the database for its primary configuration. This can often optimize and simplify interaction with the openPDC Manager since a file share will not need to be established for the SQLite database.

The simplest way to setup a new database configuration for an openPDC instance is to run the Configuration Setup Utility that is installed along with the openPDC Manager. This utility is currently only available on a Windows environment, however, it can create a new configuration for a database system running in a POSIX environment. If the Configuration Setup Utility is being used to convert from one database to another, the ***DatabaseMigrationUtility*** will be run at the end of the process to migrate the data in the old database to the new one[14].

Without access to the Configuration Setup Utility, a new configuration database can still be manually created. Schemas for all supported database types are installed with the openPDC and can be found in the ***/opt/openPDC/Database Scripts*** folder.

Once a new database has been created, the ***openPDC.exe.config*** file ***systemSettings*** category properties called ***ConnectionString*** and ***DataProviderString*** have to be updated[15] with connection information for the new database. If the Configuration Setup Utility was used to create the new database, the ***systemSettings*** category property values applied to the ***openPDCManager.exe.config*** file by the can often be copied and used as-is[16] in the ***openPDC.exe.config*** file.

---

[14] This UI based utility is also installed with the openPDC in POSIX environments to allow for data migration.

[15] Changes to the ***openPDC.exe.config*** file should only be made while the openPDC service is not running.

[16] One exception is the ***DataProviderString*** for SQLite. Under POSIX systems, the SQLite data provider string is:
`AssemblyName={Mono.Data.Sqlite, Version=4.0.0.0, Culture=neutral, PublicKeyToken=0738eb9f132ed756}; ConnectionType=Mono.Data.Sqlite.SqliteConnection; AdapterType=Mono.Data.Sqlite.SqliteDataAdapter`

See the following links for more information:

http://openpdc.codeplex.com/wikipage?title=Getting%20Started#set_up_database

http://www.connectionstrings.com/

Enabling the openHistorian

The openHistorian 1.0 built into the openPDC also works on POSIX platforms. From a fresh install of the openPDC[17], download an updated ***openPDC.exe.config*** and ***openPDC.db*** configuration database that have the historian enabled by default:

http://www.gridprotectionalliance.org/NightlyBuilds/openPDC/Scripts/POSIXConfigHistorian.zip

To deploy, unzip the files from the download then, with the openPDC is stopped, copy the ***openPDC.exe.config*** into the ***/opt/openPDC*** folder and copy the ***openPDC.db*** SQLite configuration database into the ***/opt/openPDC/ConfigurationCache*** folder – overwriting the existing files. Restart the openPDC and the local historian will now be archiving data. If the POSIX distribution running the openPDC service also has a UI, the ***HistorianPlaybackUtility.exe*** found in the openPDC installation folder can be executed to extract data from the archive.

Completing Security Configuration

Although security is fully enabled by default when installing the openPDC on Windows platforms, completely enabling security on Linux and Apple OS X platforms is a multi-step process that includes building the shared Grid Solutions Framework (GSF) POSIX library that requires distribution specific libraries[18]. As a result, the default configuration that gets installed on a POSIX platform has some security features initially turned off to simplify deployment. The following sections describe how to fully enable all the security features of the openPDC when running on a POSIX environment. The example steps shown below are for Ubuntu, but this may be different for any given distribution:

1) Get the GNU Compiler Collection (gcc) for your distribution, e.g., for Ubuntu:
```
sudo apt-get install build-essential
```

2) Get the Pluggable Authentication Module (PAM) libraries for your distribution, e.g., for Ubuntu:
```
sudo apt-get install libpam0g-dev
```

3) With the openPDC is stopped, execute the security setup script[19]:
```
sudo bash enable-security.sh
```

[17] If the openPDC system is already installed and configured and the current configuration needs to be preserved, the historian must be enabled manually in the configuration or by using the openPDC Manager.
[18] In order to enable user authentication and validation on a POSIX platform, the GSF POSIX functions library must be compiled locally using libraries, such as PAM, specific to the distribution running the openPDC.
[19] This script is downloaded as part of the installation process and can be found in the original installation folder along with ***install-openPDC.sh***. The script depends on the source code being preserved during the installation, i.e., make sure to use the ***-p*** parameter when installing the openPDC.

The ***enable-security.sh*** script will compile and deploy[20] the shared GSF POSIX library and update the openPDC configuration to require secured remote interactions, i.e., commands can only be issued by authenticated users meeting a minimum role-based permission (e.g., the ***Administrator*** role).

For reference, the script is modifying the ***openPDC.exe.config*** file and changing the following setting:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <categorizedSettings>
    <serviceHelper>
      <add name="SecureRemoteInteractions" value="True" />
    </serviceHelper>
  </categorizedSettings>
</configuration>
```

Once the GSF POSIX library has been compiled and the security updates have been applied, restart the openPDC. At this point the openPDC will be essentially locked-out; i.e., until a user with a valid role assignment is added to the system, no remote command access to the openPDC will be allowed without user verification[21]. Once a user is authenticated by the openPDC with a valid role assignment, all commands issued to through remote sessions require a minimum role-based permission for the issued command. For more information on role-based security in the openPDC, see the following documentation:

http://openpdc.codeplex.com/wikipage?title=Remote%20Console%20Security

Managing User Accounts

The openPDC supports two kinds of users: (1) system authenticated users and, (2) database authenticated users. As their name suggests, system authenticated users will be authenticated by the local operating system according to configured authentication policies, e.g., the PAM configuration. This can be local accounts or domain accounts, if enabled. Database authenticated users will be authenticated by matching a user name and a hash of the password as defined in the openPDC configuration database. Generally, system authenticated users are the ideal choice since the user credentials and account information is maintained by the operating system.

Groups, both for system and database, are also supported. Groups allow easy management of role assignments to any set of users that exist within a group. Note that explicitly defined role assignments for a user will always take precedence over implicitly acquired role assignments that are derived from a group that contains the user as a member.

Adding New Users

xxx

<Need to mention installing sqlite3>

[20] Once compiled, the shared object library, GSF.POSIX.so, will be deployed into the openPDC installation folder.
[21] Note that the openPDC primary functions, e.g., PMU data acquisition, IEEE C37.118 output streams and GEP data publishers, will continue to function as normal as configured, but no runtime commands or configuration changes will be accepted without proper user authentication and role validation.

Authenticating Cross Platform Users

The openPDC can be configured to allow authentication from both POSIX and Windows environments simultaneously. Cross platform authentication comes into play when using the openPDC Manager running on Windows to manage the configuration of an openPDC service running in a POSIX environment.

Since the openPDC Manager application can directly manage a configuration database even without access to a running openPDC service, it has to independently authenticate a user. Also, since the openPDC Manager currently only runs on Windows, its user authentication will always be Windows based. However, when connected to a remote openPDC, the openPDC Manager also has to authenticate the user with the remote openPDC service, which can be running in a POSIX environment.

To allow cross platform authentication, a user entry must exist in the openPDC configuration database for each platform for which a user needs to be authenticated. To prevent user configuration duplication, system users and groups added to the openPDC configuration are converted to a corresponding system identifier (SID) as provided by the operating system. Duplication is prevented because different operating systems provide different system identifiers[22], allowing the same user to be added the security configuration for different platforms.

Cross Platform Domain Users

When both the POSIX and Windows environments are configured to authenticate against the same domain, a domain user or group can be added for both environments effectively.

When adding a domain user to the database configuration the domain name prefix must be specified. For example, if the domain name is “GPA” and the user name is “openPDCUser”, the username to be added to the database configuration should look like “GPA\openPDCUser”[23].

When a system user is added to the database as a domain prefixed username and is subsequently recognized as a system user during the next openPDC service configuration load, e.g., issuing the ***reloadconfig*** command, the system user will be changed to an operating system specific SID in the database. When adding a recognized domain user from the openPDC Manager, it automatically writes the Windows SID to the database. Accordingly, the domain user will need to be added from the openPDC Manager for proper authentication on Windows and added to the configuration database from within the POSIX environment for proper authentication by the openPDC service running there.

Cross Platform Local Users

The only current way to manage a cross platform local user is to create a user with the same name and password on both the POSIX and Windows environments. However, this does not have to be a local

[22] Helping with uniqueness, POSIX platform user and group SIDs for the openPDC will also be prefixed with “user:” and “group:” respectively. This is needed to create a unique SID that spans both user and group namespaces because in POSIX environments users and groups are managed separately and their IDs often overlap.
[23] Use of the backslash delimiter for the openPDC configuration usernames is required even if the PAM configuration on the POSIX configuration commonly uses another symbol to delimit the domain and user name. Consequently, if the openPDC configuration database being used is sensitive to characters escaped with a backslash, it may be necessary to format the username like “GPA\\openPDCUser”.

system account on both systems – one option is use a database user in one environment with the same credentials as the local system user in the other environment[24].

Disabling Pass-Through Authentication

There is no functionality in a POSIX environment to impersonate a user's security context, so openPDC pass through authentication and integrated security need to be disabled. In practice this means credentials will be required to login to the openPDC Manager on Windows when connecting to an openPDC running on Linux or Apple OS X. As a result, if the Windows based openPDC Manager is being used to maintain a POSIX openPDC configuration, the openPDC Manager configuration must be modified to require login credentials[25]. To make this change, stop the openPDC Manager, edit the ***openPDCManager.exe.config*** file and change the following setting:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <userSettings>
    <openPDCManager.Properties.Settings>
      <setting name="ForceLoginDisplay" serializeAs="String">
        <value>True</value>
      </setting>
    </openPDCManager.Properties.Settings>
  </userSettings>
</configuration>
```

Updating Transport Layer Security

The follow documentation describes how to manually update Transport Layer Security (TLS) settings. The openPDC TLS configuration is automatically enabled in the installation script along with a newly created self-signed certificate, as a result, steps to update TLS are generally not required. However, this section is provided in case the TLS configuration needs to be updated or customized.

For high level information related to using TLS/SSL with Mono, be sure to review the following:

http://www.mono-project.com/docs/faq/security/

Manually Creating a Certificate

Using the ***MonoGenCert*** tool that gets deployed with the openPDC will be the simplest way to generate a new certificate, for example, change directory into the openPDC installation folder (typically ***/opt/openPDC***) and execute the following:

```
sudo mono MonoGenCert.exe openPDC
```

[24] When adding a database user to the openPDC configuration database, only the password's hash will be stored, not the actual password.

[25] Technically with a properly defined Windows account in the configuration database, the openPDC Manager can still login with pass-through authentication. However, in order to also successfully connect to the openPDC service running in a POSIX environment and issue commands, valid user credentials must be collected.

Internally this command calls ***makecert*** with the proper parameters to create all the needed keys and certificate files. The tool also automatically generates a list of all known common names for the machine.

For more control of the certificate creation process, the ***makecert*** command can be used directly, for example:

```
makecert -r -n "CN=openPDC,CN=linux.domain.com,CN=192.168.1.149,CN=..." -p12 openPDC.p12
"" -sv openPDC.pvk openPDC.cer
```

When creating certificates using the ***makecert*** command, note carefully the list of common names after the ***-n*** flag. If the list does not contain an entry matching that which was used to connect to this server, then the client configurations, e.g., console and manager applications, will have to add ***RemoteCertificateNameMismatch*** to the list of ***ValidPolicyErrors***. Note that the only exception is localhost, which the local client applications will trust regardless.

If the ***makecert*** command completes successfully, the result will be two new files. The first file is ***openPDC.p12*** which contains the public key, certificate information and the private key – this file is required by Mono in order to properly secure network traffic. The second file is ***openPDC.pvk*** which is the standalone private key file[26]. If needed, call the ***makecert*** command again without the ***-p12*** parameter to create a public key certificate file, i.e., ***openPDC.cer***[27].

To protect generated private keys, make sure to restrict access to the files[28]:

```
chmod 600 openPDC.p12 openPDC.pvk
```

Using an Existing Certificate

Existing certificates, such as one obtained from a public certificate authority, can be used by updating the needed ***openPDC.exe.config*** file settings:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <categorizedSettings>
    <remotingServer>
      <add name="LocalCertificate" value="openPDC.p12" />
    </remotingServer>
</configuration>
```

[26] If the private key file ***openPDC.pvk*** already exists before the call to ***makecert***, a new file will not be created – the tool will instead use the existing private key.
[27] Using the ***MonoGenCert*** tool automatically creates all three files.
[28] This step happens automatically when using the installation script.

As always, when making modifications to the openPDC configuration, stop the openPDC, make the needed changes, and then restart the openPDC. Additionally, as mentioned in the *Manually Creating a Certificate* section, always make sure to protect the private key files.

Changing the TLS Version

As of version 3.12 of Mono, only TLS 1.0 and older protocols, e.g., SSL versions 1.0 to 3.0, are currently supported[29]. The default configuration of the openPDC installation enables the highest level security available for the environment. To change the supported transport protocols, stop the openPDC, edit the ***openPDC.exe.config*** file and update the following setting:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <categorizedSettings>
    <remotingServer>
      <add name="EnabledSslProtocols" value="Tls" />
    </remotingServer>
</configuration>
```

As newer versions of Mono are released that enable updated versions of the TLS protocols, just update the setting with the latest protocol version, e.g., ***Tls12*** – see table below. Note that client application configuration settings, e.g., the console and manager, should be updated to match the minimum specified server protocol in order to properly connect.

[29] With the recent open sourcing of Microsoft's .NET framework, Mono is making improvements on many fronts, including security. As of the writing of this document, TLS1.2 has been enabled in the latest Mono source code but has not yet been made available in a release build. When Mono gets a new release that contains updates that the openPDC can use for improvements related to security, e.g., TLS 1.2, these updates will be tested with the openPDC so that improved security features can be enabled as soon as they are available.

Multiple transport protocols can be specified separated by a comma. The possible options are as follows:

Protocol	Description
None	No SSL protocol is specified. Cannot be combined with other protocols.
Ssl2	Specifies the SSL 2.0 protocol. SSL 2.0 has been superseded by the TLS protocol and is provided for backward compatibility only.
Ssl3	Specifies the SSL 3.0 protocol. SSL 3.0 has been superseded by the TLS protocol and is provided for backward compatibility only.
Tls	Specifies the TLS 1.0 security protocol. The TLS protocol is defined in IETF RFC 2246.
Tls11	Specifies the TLS 1.1 security protocol. The TLS protocol is defined in IETF RFC 4346.
Tls12	Specifies the TLS 1.2 security protocol. The TLS protocol is defined in IETF RFC 5246.

Running the openPDC with Elevated Rights

In POSIX compatible applications, security for an application starts high then is reduced to a user's permission level upon access. The openPDC is no exception. To make sure the openPDC has access to needed security information for user authentication and validation, it must be started with elevated privileges[30].

When running the openPDC as a daemon, the service application registration script already makes sure the openPDC is running with needed elevated privileges such that security functions can be accessed as needed.

Securing the Configuration Database

The default configuration deployed during installation includes a simple SQLite database for its configuration. To continue using the SQLite database, the Samba share (if enabled in prior steps for use with the openPDC Manager) should be reconfigured to require security and the access rights of the ***/opt/openPDC/ConfigurationCache*** folder should be reduced to a specific set of users.

For optimal configuration security, migrating to another type of database and using its security mechanisms is highly recommended. SQL Server[31], MySQL and Oracle can all be used with the openPDC. See the *Using Other Databases* section for more information.

[30] If the openPDC was installed using ***sudo***, folder rights will already require the openPDC to run with elevated privileges; however, elevated privileges will still be required by the openPDC when security is fully enabled even if folder rights are changed not to require root access. Without the needed run-time privileges with security fully enabled, the openPDC will not be able to validate user credentials and remote command access into the openPDC will be locked-out.

[31] An openPDC running in a POSIX environment can connect to SQL Server running on Windows.

Running on a Raspberry Pi

At the time of writing there were no common distributions for the Raspberry Pi that contained Mono at version 3.12; therefore, Mono must be compiled before openPDC installation. In our tests, it took about 8 hours to completely build the needed Mono components on an original Raspberry Pi system and less than 3 hours on a Raspberry Pi 2 system.

To avoid needing to compile Mono and speed up the installation process, GPA has posted an image for download with the needed version of Mono and the openPDC preinstalled for running on a Raspberry Pi and Pi 2 with the Raspbian OS. See the following link for instructions:

https://openpdc.codeplex.com/wikipage?title=Running%20openPDC%20on%20a%20Raspberry%20Pi

For best openPDC performance, the Raspberry Pi 2 is recommended. The new Raspberry Pi 2 Model B has 4 cores, 1 GB of RAM and better CPU performance all of which provide a very practical and performant micro-environment for running the openPDC.

The openPDC also runs on the original Raspberry Pi. For optimal performance on this single core system it is recommended that the configuration of the openPDC on the Raspberry Pi be reduced its primary tasks.