INTRODUCTION

Pharos controllers with embedded web servers support the use of HTTPS with certificates. HTTPS is the secure version of HTTP, used for securing online transactions such as booking with a credit card, or entering sensitive information such as passwords or bank details.

HTTPS can be a complex system to set up. This document will provide a gentle introduction to HTTPS, why you might or might not need it, and how to configure it in Pharos products.

DO I NEED HTTPS?

If you are operating your controllers without network connected, or in a small isolated network without security threats, it is quite likely you don't need to configure HTTPS.

If you operate in a larger, IT-managed network with other systems, or with easy public access such as WiFi access points, it may be a good idea to enable HTTPS. Enabling HTTPS comes with some complexities as outlined below.

HOW DOES HTTPS WORK?

HTTPS performs two major functions:

Encryption – encryption is the act of making data readable only to the devices exchanging it, but not to any third parties who are able to intercept the data as it is being sent. This is important, for example, in the case of a credit card number being sent across the internet – the internet is not secure, and you can assume many people in the middle can intercept traffic. Encryption means that even if the data is intercepted, your credit card details cannot be stolen.

Authentication – authentication ensures that the site you are communicating with is who it says it is. It means that, for example, a malicious party cannot set up a site that looks like ebay.com and use it to make you think you are completing a purchase while really stealing your card details.

When a website's connection is secured using HTTPS, the browser typically shows you via a padlock icon near the URL:

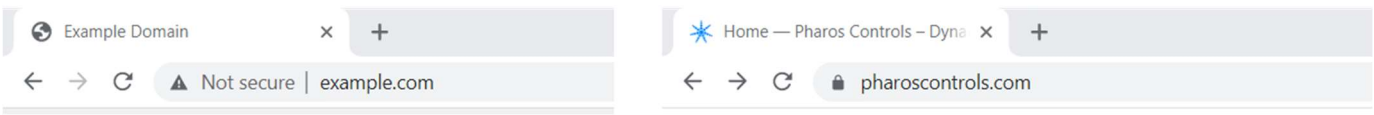

Figure 1 - Example showing Insecure (HTTP) and Secure (HTTPS) web pages in a browser

The padlock icon indicates you can be certain the website is not impersonating another site, and your communication with it is secured.

CONSIDERATIONS FOR USING HTTPS IN A PRIVATE NETWORK

The infrastructure for Authentication in HTTPS is based around sites on the internet – that is, sites which have a domain name (e.g. pharoscontrols.com) which resolve to an IP address for communication. HTTPS does not easily work with IP addresses directly. So if you access your controller by typing an IP address such as http://192.168.1.42, you will not be able to set up HTTPS. You will need to work with your IT administrator to set up a DNS system whereby you can resolve the controller by name, for example http://pharoscontroller.mybusiness.net.

ENABLING HTTPS IN DESIGNER

For any controller you can enable HTTP, HTTPS, or both. By default, both are enabled. In a secured system you most likely would want to disable HTTP, leaving just HTTPS enabled. To do this:

1. In the *Network* tab of designer, right click the controller and select *Configure*.
2. Under Network, enable HTTPS and disable HTTP

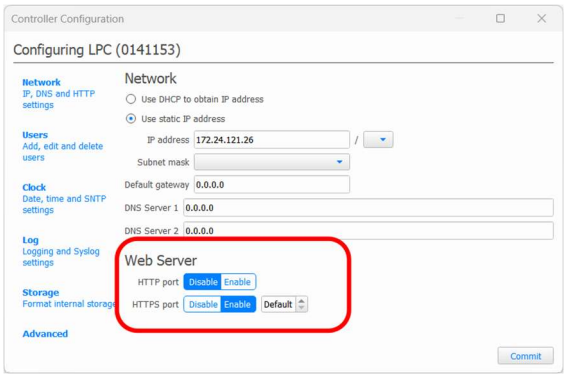

Figure 2 - Disabling HTTP and Enabling HTTPS in Designer

CONFIGURING AN HTTPS CERTIFICATE

Initially, the web server will use an automatically generated certificate for encryption of the HTTPS connection. That will allow HTTPS connections to work, but most browsers will show an error when connecting to a server with an automatically generated certificate like this:

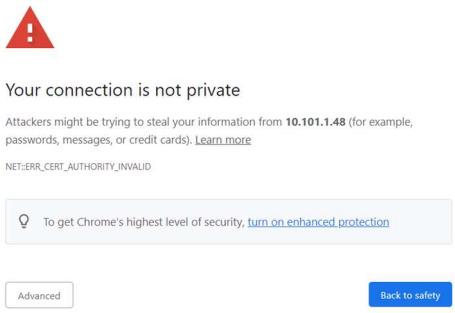

Figure 3 - Typical Warning using a self-signed certificate

The reason for this warning is the Authentication aspect of HTTPS. The HTTPS certificate is part of a chain of trust – trusted organizations can provide certificates that guarantee a site is what it says it is. The automatically generated certificate on the device's web server cannot provide that, in order to do that we need to upload an appropriate, signed certificate.

CERTIFICATE FORMATS

There are a variety of formats used for keys and certificates. The web server in Designer products uses a PEM file, which combines the private key and certificate(s).

A .PEM file can be opened as a text file (e.g., with Notepad) and the contents should look something like this:

```
-----BEGIN CERTIFICATE-----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
-----END RSA PRIVATE KEY-----
```

The sections filled with xxx will be encoded data, appearing as a mixture of numbers and letters. It is important that the file contains at least the RSA Private Key section and at least one Certificate section – there may be multiple certificate parts.

Your IT department may be able to simply provide you with an appropriate PEM file, in which case you can skip to the Uploading a Certificate section below. Or you may need to generate your own key and use a Certificate Signing Request (CSR) to obtain a signature - if so, follow the steps below to generate a Key and CSR.

GENERATING YOUR OWN KEY & CSR

It is possible to generate your own key and submit a certificate signing request (CSR) to your IT department or a certificate registrar to get a signed SSL certificate. To do that will require tools – in this example we will look at OpenSSL on Windows. There are several other ways to do this on different platforms, but they are outside the scope of this article.

First, download and install the latest version of OpenSSL for Windows from https://slproweb.com/products/Win32OpenSSL.html

Open a Windows Command Prompt

Run the command:

```
"C:\Program Files\OpenSSL-Win64\bin\openssl.exe" req -newkey rsa:1024 -
keyout PRIVATEKEY.key -out MYCSR.csr
```

This will create a 1024 bit RSA key called PRIVATEKEY.key in the directory you are working in, and also create a certificate signing request called MYCSR.csr in the same directory.

You will be prompted for a passphrase as you do this. Make sure to note down the passphrase and keep it secure – you will need it later.

Then you will be asked some questions about identifying information which can be added to the CSR. Once complete you can submit that CSR file to your IT department or signing authority to obtain a certificate.

The certificate should come back to you as a .CRT certificate file. We will call it CERTIFICATE.crt for this example.

You now need to combine that certificate file with a decrypted version of your key to make a PEM file suitable for the server to use. To do that, run the command:

```
"C:\Program Files\OpenSSL-Win64\bin\openssl.exe" rsa -in PRIVATEKEY.key -
out PRIVATEKEY.key.unenc
```

This makes a new file, PRIVATEKEY.key.unenc which contains the key. Now we will join the certificate file you received with the private key to make the PEM file

```
copy CERTIFICATE.crt SERVER.pem
type PRIVATEKEY.key.unenc >> SERVER.pem
```

This will result in a PEM file that looks like the example above. That can now be uploaded in Designer for use by the server.

UPLOADING A CERTIFICATE

The certificate is uploaded to the device via Designer software, in the format of a .PEM file.

To configure your controller to use the certificate:

1. In Designer, under Project -> Web Interface, click the blue folder under Custom Certificate.
2. Select your .PEM file.
3. Now upload your project, the .PEM file will be used for the web server's HTTPS connections.