



SECURITY

DireXions 2024

AGENDA

The background of the slide is a photograph of a multi-story urban building. The building has a mix of grey stone and bright red brick facades. Several windows are visible, some with air conditioning units. A black metal fire escape is attached to the red brick section of the building. The sky is a pale, overcast blue.

SSL/TLS

Two-Factor Authentication

OAuth 2.0 Secure Web Services



SSL/TLS

SSL/TLS - OVERVIEW

- **What does SSL stand for?**
 - Secure Socket Layer
 - A socket is the technical term for a network connection between machines
 - SSL is a layer between the TCP/IP interface and your application
- **TLS is the new terminology**
 - Transport Layer Security
 - Removes the reference to “Socket”
 - Can (in theory) be used on any communication

SSL/TLS - OVERVIEW

PxPlus will connect with SSL 2.0 and above

This can be controlled

Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011
SSL 3.0	1996	Deprecated in 2015
TLS 1.0	1999	Deprecated in 2021
TLS 1.1	2006	Deprecated in 2021
TLS 1.2	2008	In use since 2008
TLS 1.3	2018	In use since 2018

SSL/TLS - OVERVIEW

- **SSL/TLS provides three main services**
 - Data encryption
 - Authentication of the server to the client
 - Authentication of the client to the server (less common)
- **PxPlus uses the industry standard OpenSSL to provide SSL/TLS support**
 - On Windows, PxPlus ships with OpenSSL included
 - Version is updated with major PxPlus version (Windows)
 - On UNIX/Linux, OpenSSL is part of the OS
 - Used by [TCP], Simple Client-Server, Email, EZWeb, ODBC, PxServer
 - Specify which OpenSSL PxPlus should use by setting the environment variables **PXP_CRYPTO_LIB** and **PXP_SSL_LIB**
 - It is also possible to query which version of OpenSSL PxPlus is using by issuing a **TCB("OpenSSL_Version")**

SSL/TLS - DATA ENCRYPTION

Ciphers provide “reversible” encryption

- Data encrypted by “**Encryption key**” can only be decrypted by “**Decryption key**”
 - Key size and the algorithm determines how secure data is
 - Typical key sizes range from 128 to 4096 bits
 - 32 bit is over 4 billion thus 4096 is quite large
 - Algorithms can be found to be faulty and “leak” answers

No cipher is 100% safe - all can be cracked given enough resources and time

SSL/TLS - DATA ENCRYPTION

SSL/TLS encryption algorithms
(ciphers)

Only use modern and unbroken
ciphers

Method	Description
aes	Advanced Encryption Standard (AES) , also known as Rijndael, adopted as an encryption standard by the US government.
aria	ARIA is a block cipher with a block size of 128 bits and key sizes of 128, 192 and 256 bits. It was designed in 2003 by a large group of South Korean researchers. In 2004, the Korean Agency for Technology and Standards selected it as a standard cryptographic technique.
camellia	Camellia is a symmetric key block cipher with a block size of 128 bits and key sizes of 128, 192 and 256 bits. It was jointly developed by Mitsubishi Electric and NTT of Japan. The cipher has been approved for use by the ISO/IEC, the European Union's NESSIE project and the Japanese CRYPTREC project. The cipher has security levels and processing abilities comparable to the Advanced Encryption Standard.
chacha20	ChaCha20 is a stream cipher developed by Daniel J. Bernstein. It was designed in 2005 and then later submitted to the eSTREAM European Union cryptographic validation process by Bernstein.
sm4	SM4 (formerly SMS4) is a block cipher used in the Chinese National Standard for Wireless LAN WAPI (WLAN Authentication and Privacy Infrastructure).

SSL/TLS - DATA ENCRYPTION

How are keys used?

To send data securely to the host

- Encryption key is made **PUBLIC**
 - Key is used to encrypt data
 - Based on the **PRIVATE** key
- Decryption key is kept **PRIVATE** on host
 - Never should be revealed

SSL/TLS - DATA ENCRYPTION

Which cipher is used?

- Server and client negotiate which ciphers they support
 - Client identifies which ciphers it supports
 - Supplied in order of preference
 - Server identifies which cipher it wants
 - Client confirms
- Server will reject any it considers unsafe or unsupported
 - Connection fails if none are acceptable



**Using insecure ciphers
will result in
PCI Compliance failure**

SSL/TLS - AUTHENTICATION

Validation/Authentication of system done using certificates (X509)

- Certificate contains the following:
 - Server Name/Address
 - Start/End Dates for which certificate is valid
 - Issuer identification
 - Name, Country, City, State/Province
 - Public key
- Certificates exchanged during negotiation
- **SHOULD** be validated for secure connection

SSL/TLS - AUTHENTICATION

What is generally validated?

- Current date is within the start and end dates for certificate
- The server address on the certificate matches the server we connected to
- The certificate was issued by a trusted certificate authority
 - The certificate can be found in a list of trusted certificates

Optional test

- Match to previously known Public key

SSL/TLS - AUTHENTICATION

Normally, only Server provides certificate

- Client only provides a Public key

When would Client require certificate?

- Controlled access to specific pre-cleared clients
 - Cannot connect unless you have a known certificate

SSL/TLS - HOW TO ESTABLISH TRUST

SSL/TLS provides a mechanism that establishes “TRUST”

- There are KNOWN “Trusted” companies that provide “certificates”
 - Known as “Certificate Authorities” (**CA**)
 - Most commonly used CA are:
 - Let’s Encrypt
 - GlobalSign
 - IdenTrust
 - Sectigo (Comodo Cybersecurity)
 - DigiCert
 - GoDaddy

SSL/TLS - GETTING A "TRUSTED" CERTIFICATE

You need a certificate from a CA for HTTPS

- If not trusted, browsers will complain
 - Expired certificate is the most common
 - Most will reject connection
 - Certificate **MUST** match site name

How to obtain a certificate?

- 1 year certificate
 - Contact a CA provider
 - Costs around \$100+ per year
 - Requires company background check
- 90-day certificate
 - Let's Encrypt
 - Free
 - Auto renews via domain validation

SSL/TLS - "SELF-SIGNED" CERTIFICATE

You can generate a certificate for yourself

- By default, it will not be trusted
- Can be used by all SSL/TLS software
 - Application can decide if TRUST is required
 - If TRUST required, users can add it to their local store
 - Includes all the same data as a standard certificate

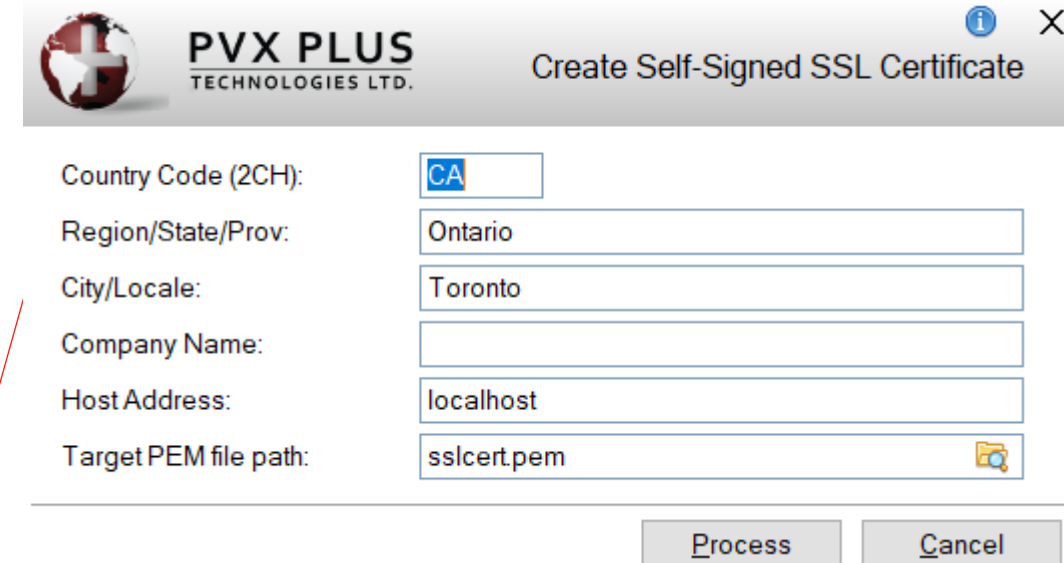
SSL/TLS - "SELF-SIGNED" CERTIFICATE

PxPlus includes [*TOOLS/SSLCERT](#) utility to create self-signed certificate

- To generate a file:

run `"*tools/sslcert"`

- Uses Internet to create certificate on our servers
- Returns single PEM file with certificate and key information
- Generates 2048-bit key
- Text mode version also available



PVX PLUS
TECHNOLOGIES LTD.

Create Self-Signed SSL Certificate

Country Code (2CH):

Region/State/Prov:

City/Locale:

Company Name:

Host Address:

Target PEM file path:

SSL/TLS - PXPLUS SSL OPTIONS

- **To make a client connection using SSL/TLS use the [TCP] option SECURE**

```
OPEN (HFN) "[TCP]https://myserver.com/app;443;SECURE"
```

- **To make a server using SSL/TLS use the [TCP] SECURE=xxxx option**
 - Where xxxx is the path to the certificate
 - Supports X509 certificates created for use with OpenSSL or Apache

SSL/TLS - PXPLUS SSL OPTIONS

- **An X509 certificate can be in the form of a PEM file, which contains both the certificate and Private key**
 - The [TCP] option **SECURE=xxx** can be used to specify the certificate file

```
OPEN (HFN) "[TCP];443;SECURE=/etc/certs/mycert.pem"
```

- **An X509 certificate can be in the form of two PEM files, one containing the certificate and the other containing the Private key**
 - The [TCP] option **SECURE=xxx** can be used to specify the certificate file while **PRIVKEY=xxx** can be used to specify the private key file

```
OPEN (HFN) "[TCP];443;SECURE=/etc/letsencrypt/live/exp.com/fullchain.pem;PRIVKEY=/etc/letsencrypt/live/exp.com/privkey.pem"
```

- **You may get a certificate in a different format, such as the Microsoft PFX file format**
 - Convert to a PEM file using the PxPlus utility, [*TOOLS/PFXCERTCONVERT](#)

```
CALL "*tools/pfxcertconvert", "C:\ProgramData\Certify\certes\assets\pfx\exp.com.pfx", "password", "converted.pem"  
OPEN (HFN) "[TCP];443;SECURE=converted.pem"
```

SSL/TLS - PXPLUS SSL OPTIONS

Defining Supported Protocol

To suppress any of these protocols:

NoSSLv2, NoSSLv3, NoTLSv1, NoTLSv1.1, NoTLSv1.2, NoTLSv1.3

To force one specific protocol:

TLS, TLS1.1, TLS1.2, TLS1.3

- Default will connect using any protocol from SSL v2 through TLS 1.3

```
OPEN (HFN) "[TCP];443;SECURE=/etc/certs/mycert.pem;TLSv1.3"
```

SSL/TLS - PXPLUS SSL OPTIONS

Certificate Validation

Certificates= **I**gnore | **V**alidate | **T**rust

Default set using
PVX_CERTIFICATES
environment variable

- **Ignore** doesn't validate certificate (default)
- **Validate** makes sure certificate:
 - Is not expired
 - Is for the proper server by matching name
- **Trust** extends Validation
 - Certificate must have come from trusted CA
 - PxPlus ships with list of trusted certificates
`<pxplus exe directory>/ca-bundle.crt`
 - This file **MUST** be updated periodically

Can be changed using
PVX_CERTSTORE
environment variable

<https://raw.githubusercontent.com/bagder/ca-bundle/master/ca-bundle.crt>

```
OPEN (HFN) "[TCP];443;SECURE;certificates=Validate"
```

SSL/TLS - PXPLUS SSL OPTIONS

Defining Acceptable/Supported Ciphers

Ciphers= *list of accepted ciphers*

- Contents of list defined at www.openssl.org
- PCI compliance (**currently**)

```
Ciphers=ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-  
GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-  
CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-  
SHA384:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
```

- Includes only known strong ciphers

```
OPEN (HFN) "[TCP];443;SECURE=/etc/certs/mycert.pem;Ciphers=ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-  
SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-  
CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-  
SHA384:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256;NoSSLv2;NoSSLv3;  
NoTLSv1;NoTLSv1.1"
```

SSL/TLS - EZWEB

- **Specify an SSL certificate when launching EZWeb server to enable SSL/TLS encryption**

- EZWeb supports X509 combined and separated PEM files

```
/app/pxplus "*ezweb/server" -arg 443 "/etc/certs/mycert.pem"
```

```
/app/pxplus "*ezweb/server" -arg 443 "/etc/letsencrypt/live/exp.com/fullchain.pem  
privkey=/etc/letsencrypt/live/exp.com/privkey.pem"
```

- EZWeb supports PFX certificates without conversion
 - If the PFX is password protected, use **pfxpswd=** option

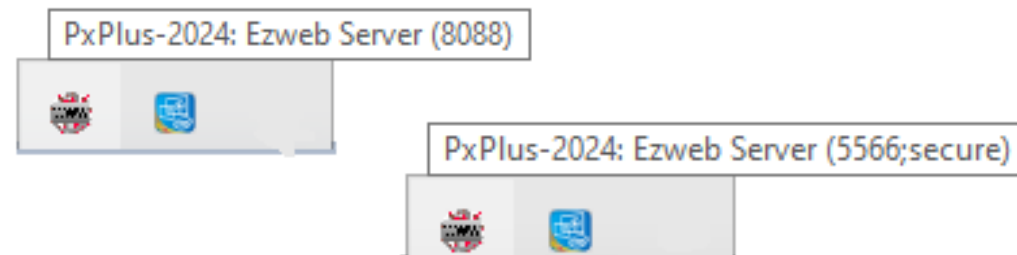
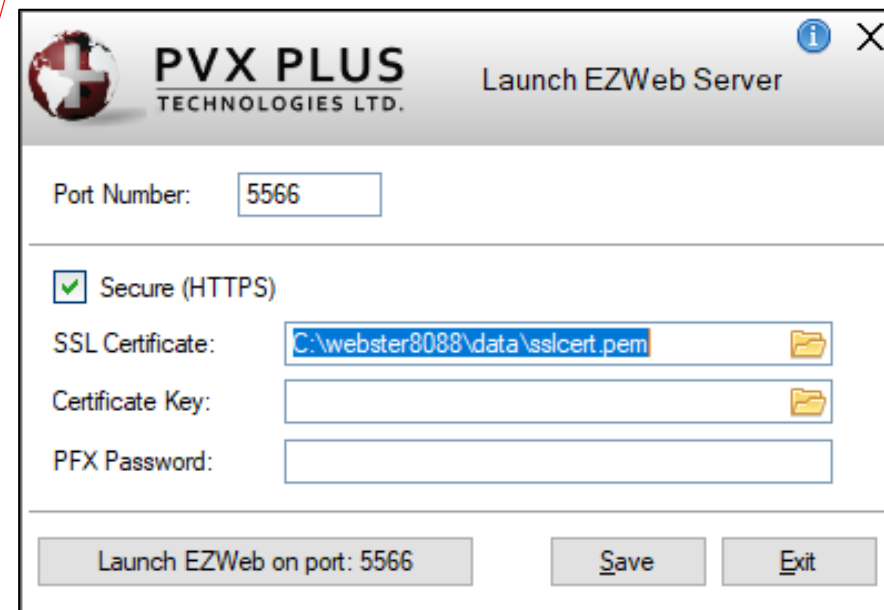
```
"C:\app\pxplus.exe" *ezweb\server -arg 443 "C:\ProgramData\Certify\certes\assets\pfx\exp.com.pfx pfxpswd=password"
```

- Also can be specified via **ezweb.conf** file
 - **SECURE** - to specify combined PEM file, certificate PEM file or PFX file
 - **PRIVKEY** - to specify Private key PEM file
 - **PFXPSWD** - to specify password for the PFX file

```
port 443  
secure "/etc/letsencrypt/live/exp.com/fullchain.pem"  
privkey "/etc/letsencrypt/live/exp.com/privkey.pem"  
nobrowse
```

SSL/TLS - EZWEB

- **Can launch EZWeb Server using graphical utility**
 - IDE > Web Deployment > Launch EZWeb Server
 - Supports same security options as command line
- **System Tray messages updated when secure**



SSL/TLS - PXPLUS CLIENT-SERVER AND SSL

- When the CS host/client is launched, [TCP] options can be used to specify SSL/TLS options

```
pxplus.exe *plus\cs\host -arg 12345;secure= C:\app\certs\app.pem;TLS1.3
```

```
pxplus.exe *plus\cs\client -arg MySrvr;12345;secure;certificates=Validate
```

Host-side CS options
(server)

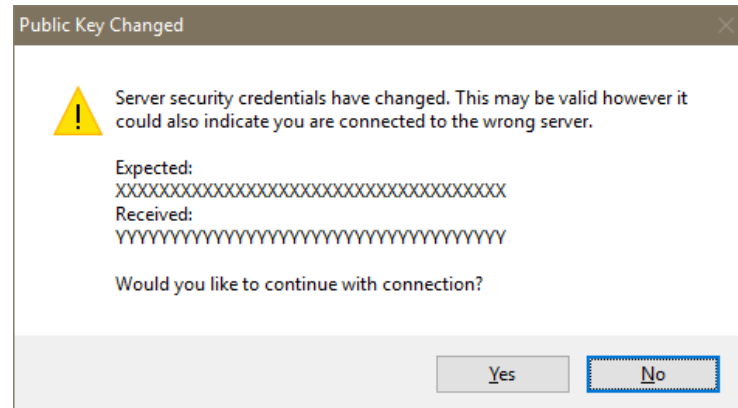
Default options can be set in:
PXP_CS_OPT
Environment variable

Client-side CS options
(workstation)

Default options can be set in:
PXP_CS_OPT_CLIENT
Environment variable

SSL/TLS - PXPLUS CLIENT-SERVER AND SSL

- **PUBKEY=xxxxxxx** can be used on the client to specify which Public key to accept from the server or to ask the user to confirm
- If xxxxxxx contains the word "**check**", then on the first connect to the server, the client process will ask the user to confirm that the Public key signature it received is correct



- If xxxxxxx contains a Public key signature (Base 64 of the SHA-256 of the X509 Public key), then it **MUST** match the server value

```
pxplus.exe *plus\cs\client -arg MySrvr;12345;secure;pubkey=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

SSL/TLS - FUTURE CONSIDERATIONS

SSL is constantly changing to address new vulnerabilities

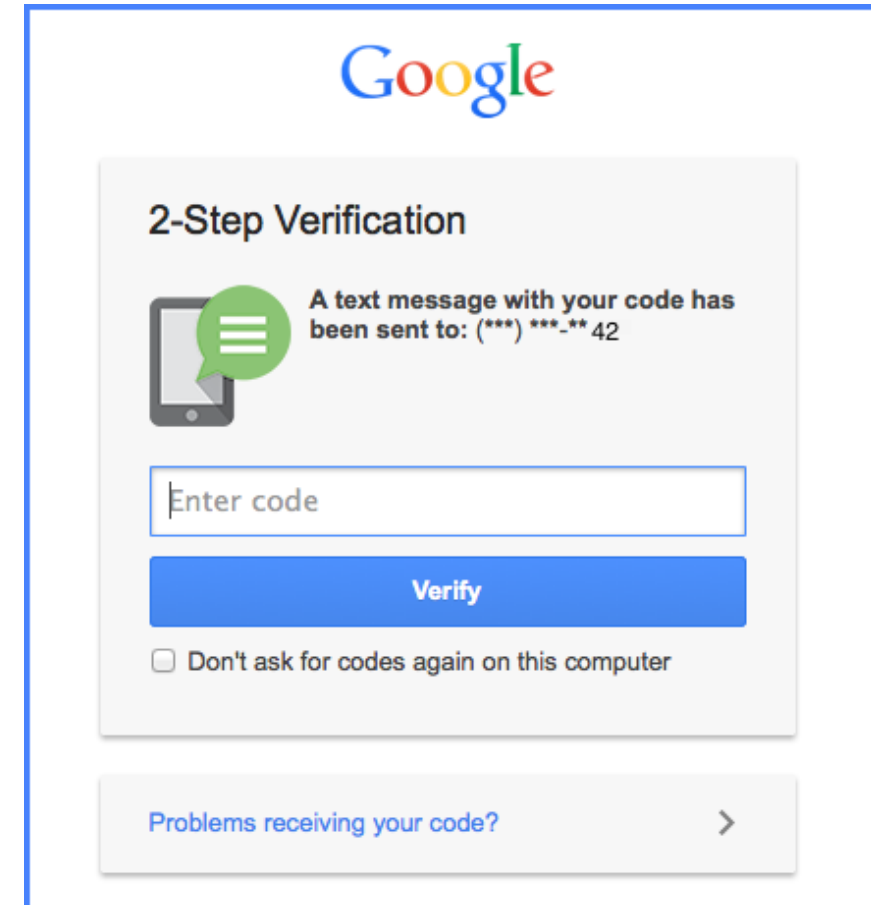
- Maintain your PxPlus version current
 - We update SSL to latest options with each release
- On Linux, keep your OpenSSL current
- For Windows, we ship current OpenSSL libraries
- If using **trust** relationships, update **ca-bundle.crt**

A photograph of the Tower Bridge in London, showing its two stone towers and blue suspension cables. The bridge is partially open, with the walkways raised. The River Thames is visible in the foreground with a small boat. The sky is overcast with grey clouds. A large red banner with white text is superimposed over the center of the image.

TWO-FACTOR AUTHENTICATION

TWO-FACTOR AUTHENTICATION

- **Two-Factor Authentication (TFA) increases system security by requiring users to validate their identity beyond entering their user name and password before they are allowed to log on**
- **The method of validation varies but common ones are**
 - E-Mail
 - SMS
 - Authenticator app
 - Hardware token
- **Likely you have used this with many different web services**

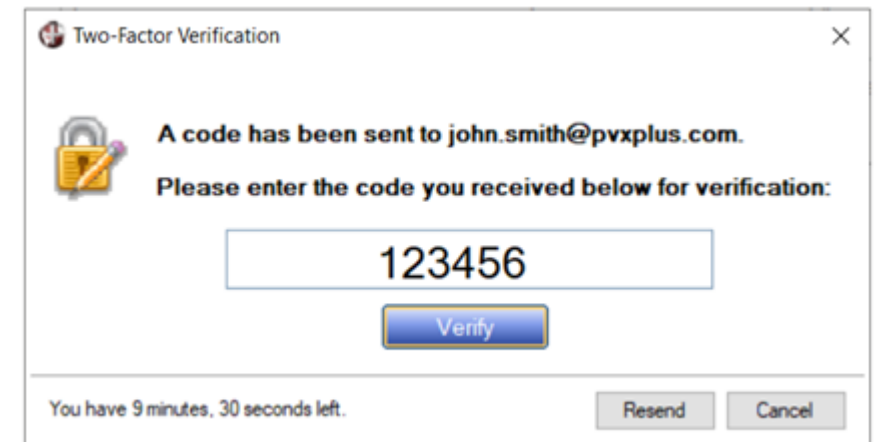
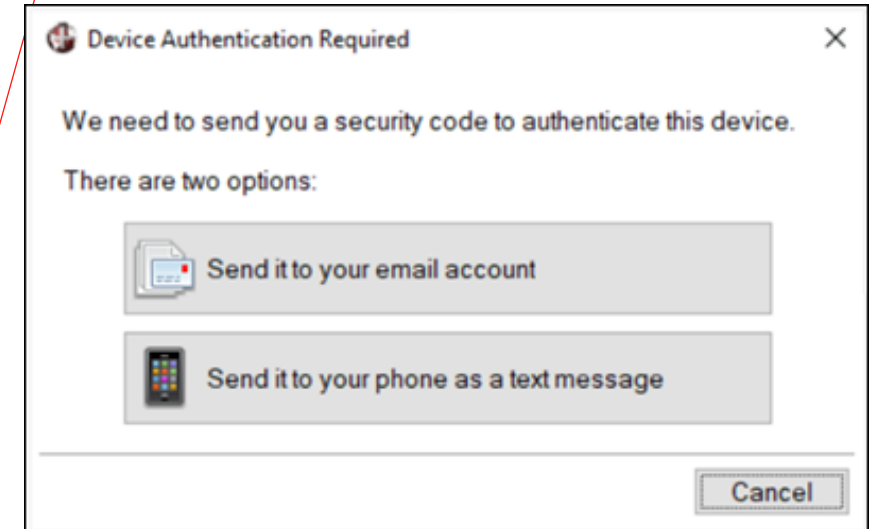


The screenshot shows the Google 2-Step Verification interface. At the top is the Google logo. Below it, the text "2-Step Verification" is displayed. A notification icon of a smartphone with a green speech bubble indicates that a text message with a code has been sent to the phone number "(***). (***)-**-** 42". Below the notification is a text input field labeled "Enter code". A blue "Verify" button is positioned below the input field. At the bottom of the main form area, there is a checkbox labeled "Don't ask for codes again on this computer". Below the main form area, there is a link "Problems receiving your code?" with a right-pointing arrow.

TWO-FACTOR AUTHENTICATION


- **Nomads/iNomads support Two-Factor Authentication via the built in Nomads security/logon system**
 - Nomads security must be setup
 - Users created and security classes defined
 - E-Mail and SMS verification is supported
 - If both setup, user can choose method to use when logging in
 - The **Two-Factor Verification** window displays when a user is required to provide identity verification before being allowed to log on
 - This window instructs the user to enter the security code sent to his/her email address or SMS phone number
- **Webster also supports Two-Factor Authentication**

Learn more at the 'PxPlus on the Web' session



TWO-FACTOR AUTHENTICATION

- To set up TFA, click the “Two-Factor Authentication Setup” button in Nomads security [User Maintenance](#)
 - This button is available only to users with the ADMIN classification
- TFA can be disabled, optional by user, or mandatory
- Specify an email server with account and/or a text message (SMS) provider and account (Both can be set up)
- TFA authentication can be saved per device, thereby avoiding having to authenticate every time
 - Period of time before you need to reauthenticate is configurable



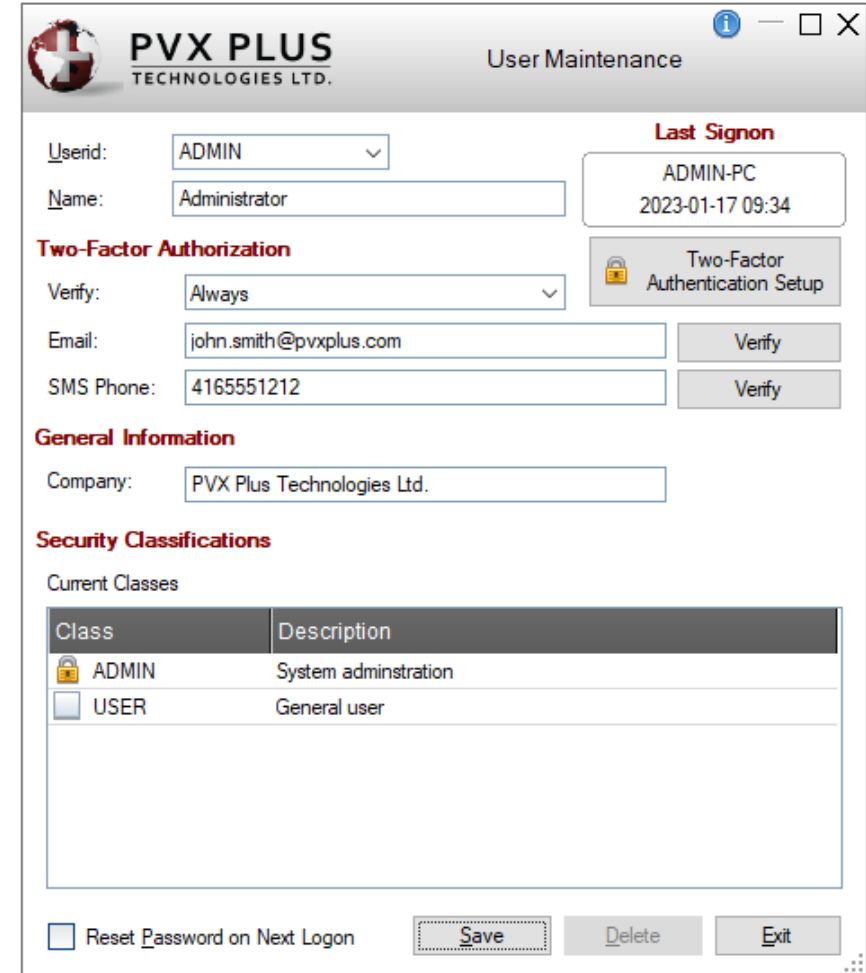
The screenshot shows the 'Setup Two-Factor Authentication' window in the PVX PLUS interface. The window title is 'Setup Two-Factor Authentication' and it includes the PVX PLUS TECHNOLOGIES LTD. logo. The configuration is as follows:

- Authentication Required:** Optional by user (dropdown menu)
- Application Name:** Sample Application (text field)
- Email Server:**
 - SMTP Server: mail.example.com (text field)
 - Port Number: 465 (text field) with a checked 'Use SSL/TLS' checkbox
 - Send From: your.name@example.com (text field)
 - Userid: your.name@example.com (text field)
 - Password: \$\$\$\$\$\$\$\$\$\$ (password field with eye icon) and a 'Test Email' button
- SMS Text Message Server:**
 - SMS Provider: smsmatrix (dropdown menu)
 - Account Information: \$ (password field with eye icon) and a 'Test SMS' button
- Authentication Duration New/Expired Devices:**
 - Windows Workstation: 1 Days (text field and dropdown menu)
 - iNomads: 30 Minutes (text field and dropdown menu)

At the bottom right, there are 'Save' and 'Cancel' buttons.

TWO-FACTOR AUTHENTICATION

- To define a users TFA settings use [User Maintenance](#)
- TFA can be disabled, required every time, or saved per device for a time
- Each user then needs to provide a verifiable email address and/or an SMS compatible phone number
- If only one of these is provided, then that option will be used
- If both an email address and SMS phone number are provided, the user will be allowed to select which one he/she wishes to use for verification



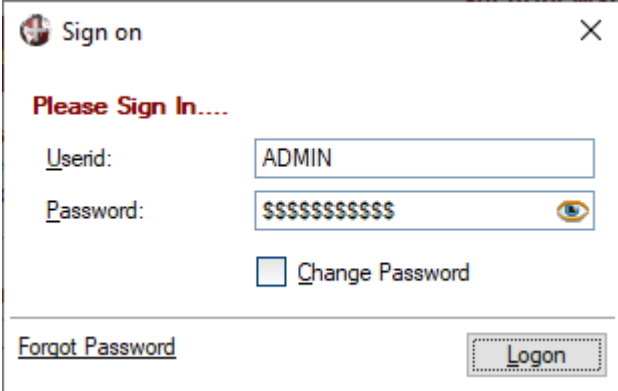
The screenshot displays the 'User Maintenance' window for PVX PLUS TECHNOLOGIES LTD. The user 'ADMIN' is selected, with the name 'Administrator' and last signon '2023-01-17 09:34'. Under 'Two-Factor Authorization', the 'Verify' setting is 'Always'. The email 'john.smith@pvxplus.com' and SMS phone '4165551212' are provided. A 'Two-Factor Authentication Setup' button is visible. The 'General Information' section shows the company as 'PVX Plus Technologies Ltd.'. The 'Security Classifications' section lists 'ADMIN' (System administration) and 'USER' (General user). At the bottom, there are checkboxes for 'Reset Password on Next Logon' and buttons for 'Save', 'Delete', and 'Exit'.

Class	Description
ADMIN	System administration
USER	General user

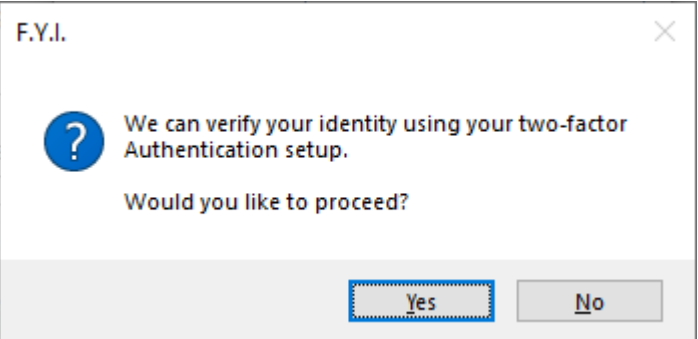
TWO-FACTOR AUTHENTICATION

Forgot Password

- **When Two-Factor Authentication is set up, if a user enters an incorrect password during system logon and the user's email address and/or SMS phone number are available, the system will provide a “Forgot Password” option in the Sign on window**
- **Selecting this option allows the system to re-authenticate the user for the purpose of resetting his/her password by displaying the following message**
- **If the user responds Yes, the system proceeds to verify the user's identity by sending a security code to the user's email address or SMS phone number. If the verification is successful, the Password Change window displays to allow the user to create a new password.**
- **If the user responds No, no identity verification is done, and the user is returned to the Sign on window**



The screenshot shows a 'Sign on' window with a close button (X) in the top right corner. Below the title bar, the text 'Please Sign In....' is displayed. There are two input fields: 'Userid:' containing 'ADMIN' and 'Password:' containing '\$\$\$\$\$\$\$\$\$\$'. To the right of the password field is an eye icon. Below the password field is a checkbox labeled 'Change Password'. At the bottom left, there is a link 'Forgot Password' and at the bottom right, a 'Logon' button.



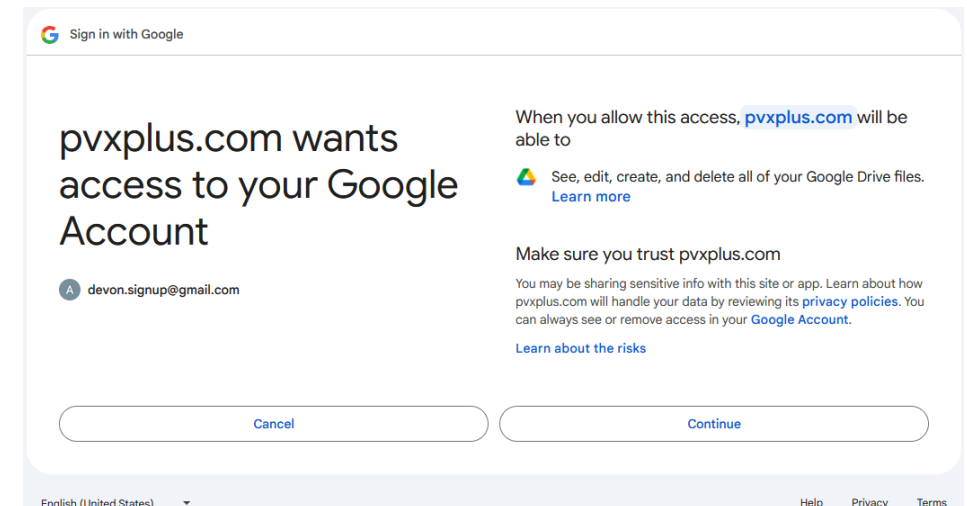
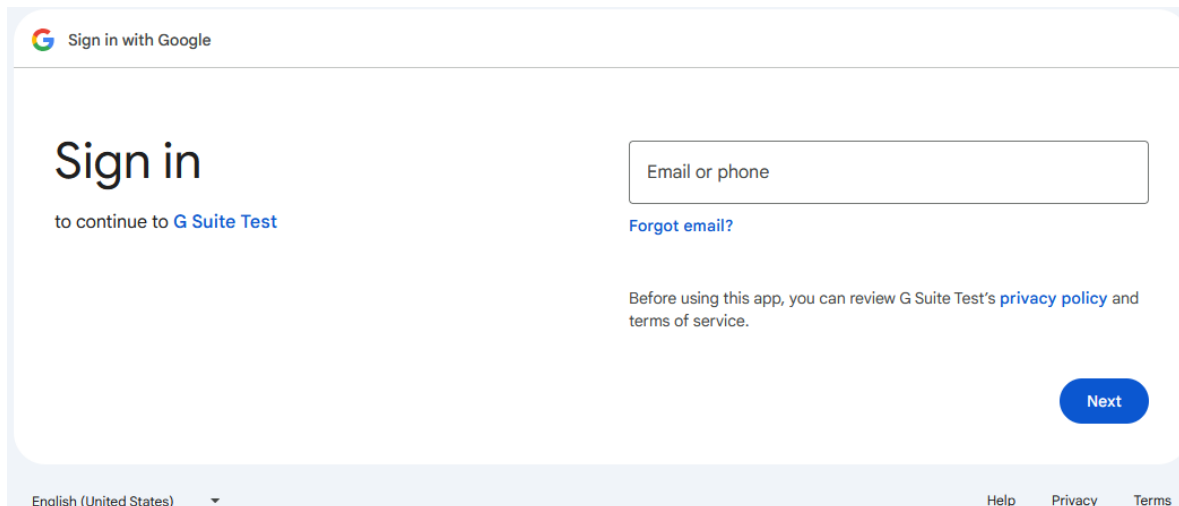
The screenshot shows an 'F.Y.I.' dialog box with a close button (X) in the top right corner. On the left is a blue circle with a white question mark. To the right of the icon, the text reads: 'We can verify your identity using your two-factor Authentication setup. Would you like to proceed?'. At the bottom, there are two buttons: 'Yes' and 'No'.



OAUTH 2.0 SECURE WEB SERVICES

OAUTH 2.0 SECURE WEB SERVICES

- **OAuth 2.0 is the modern standard for securing access to Web services**
- **Allows you to get authorized with Web services, such as Google, Salesforce or any Web service that uses OAuth 2.0**
- **Example: An application wants to be able to upload and download a file to a Google Drive**
 - OAuth 2.0 is used by the application to get the user to sign in to their Google account and allow their application access to their Google drive



OAuth 2.0 SECURE WEB SERVICES

- **Two types of OAuth 2.0 to consider**
 - Grant type
 - **client_credentials** - simpler and can be handled with a simple Web request to the token endpoint URL
 - Adds extra layer on top of username and password that is needed for web service access
 - This layer can be modified/revoked at any time separate to username and password
 - **authorization_code** - requires user to allow access via Web browser
 - Adds same extra layer as above with same benefits
 - Adds another extra layer where a user has to manually allow the application access
 - The application can ask for specific access and the user can pick and choose which they grant
 - Supports refresh tokens to avoid asking the user every time

OAuth 2.0 SECURE WEB SERVICES

- **You set up a User ID/Client with the Web service provider and they will provide you with a Client ID and a secret code, as well as one or two URLs (these may be the same)**
 - Authorization endpoint URL (request URL for users to allow access)
 - Token endpoint URL (get access/refresh token)
- **To access an OAuth 2.0 restricted Web service, an access token must be acquired and then passed in with the header of the Web service request**
 - Access tokens expire, and once expired, a new access token must be requested to make a new Web service request
 - This token must be included in the HTML header for any subsequent requests

OAUTH 2.0 SECURE WEB SERVICES

How to get access token for grant type client_credentials

- **Make an HTTP POST request to the token endpoint of the OAuth 2.0 Web service you want access to**
 - The header of the request must include "Authorization: Basic " followed by the BASE64 encoded Client ID and Client secret separated by a : (colon)
 - The body of the request must be "grant_type=client_credentials"
 - The response from a successful request is a 200 status in the response header and the access token via a JSON response

```
{  
  "access_token": "Access-Token",  
  "token_type": "Bearer",  
  "expires_in": 3600  
}
```

OAUTH 2.0 SECURE WEB SERVICES

Example

```
clientId$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"  
clientSecret$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"  
call "*WEB/BASE64;ENCODE_STR",clientId$+": "+clientSecret$,usrpsw$  
extrahdr$="Authorization: Basic "+usrpsw$  
reqData$="grant_type=client_credentials"  
call "*plus/web/request","https://www.exsrvr.com/oauth2/token",reqData$,resp$,resphdr$,"","",extrahdr$  
dim load jsonAuth${all}=resp$  
accessToken$=jsonAuth$["access_token"]
```

OAUTH 2.0 SECURE WEB SERVICES

How to get access token for grant type `authorization_code`

- **The First stage is to have the user grant your application access to an account of the provider (e.g. grant your application access to a Salesforce account)**
- **If you have done this step before and saved the Refresh token, you can skip this step**
- **First stage steps are:**
 1. The application identifies itself to the provider, giving it the Client ID and secret code
 2. The service provider returns PVX Plus a URL that the user must go to in order to authorize access
 3. The user authorizes access to the provider via a Web browser

OAUTH 2.0 SECURE WEB SERVICES

How to get access token for grant type `authorization_code`

- **The Second stage is where you request an access and refresh using the token endpoint of the OAuth 2.0 Web service you want access to**
- **Second stage steps are:**
 1. Request an Access token and a Refresh token from the web service provider using the `Token_URL$`
 2. If this is the first time, save the Refresh token to avoid logging in the next time

OAUTH 2.0 SECURE WEB SERVICES

- The **"*obj/oauth2"** object handles OAuth 2.0 grant type `authorization_code` for you
- The object has a few predefined services so you don't have to specify the authorization and token URLs
 - Google
 - Salesforce
 - If accessing a non-predefined service, just specify URLs via the **Authorization_URL\$** and **Token_URL\$** properties
- Properties used to set **ClientID\$** and **client_secret\$**
- Methods used to perform first and second stage of authorization
 - First stage:
 - **Enable_Certification(msg\$)**
 - `msg$` will be the message that appears on the authorization accepted screen
 - **Get_Authorization_URL\$(scope\$, prompt\$)**
 - `scope$` is used by some OAuth 2.0 servers when they define multiple scopes of access to specify what access they require
 - `prompt$` specifies whether the user is prompted and for what when they request authorization
<https://developers.google.com/identity/openid-connect/openid-connect#prompt>
 - Second stage: **Get_Access_token()**

OAUTH 2.0 SECURE WEB SERVICES

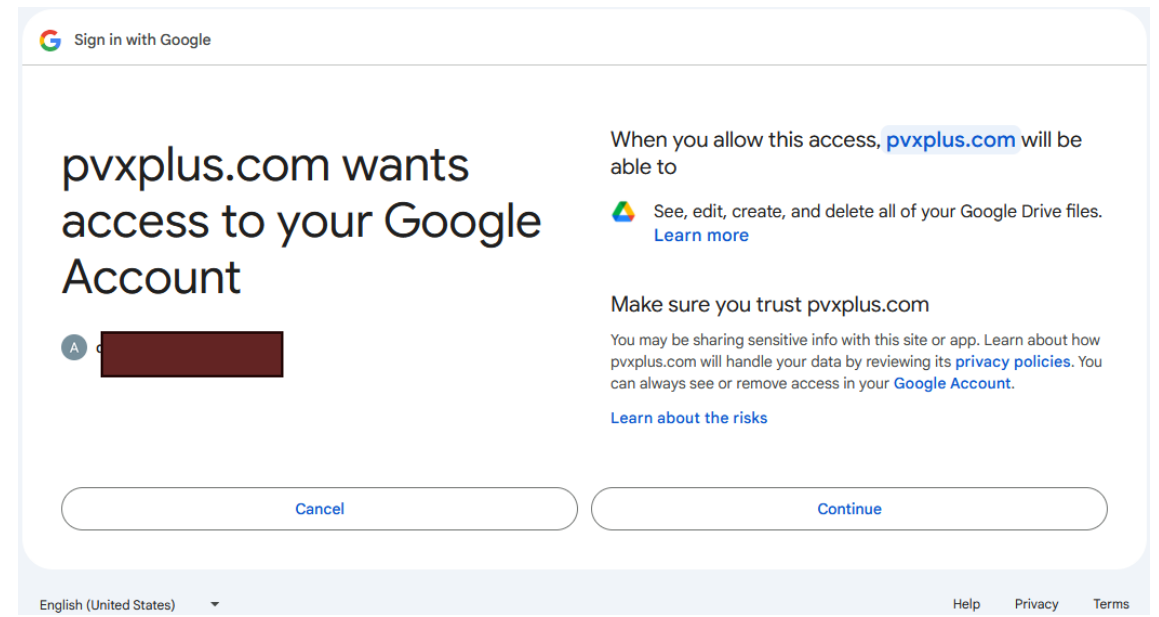
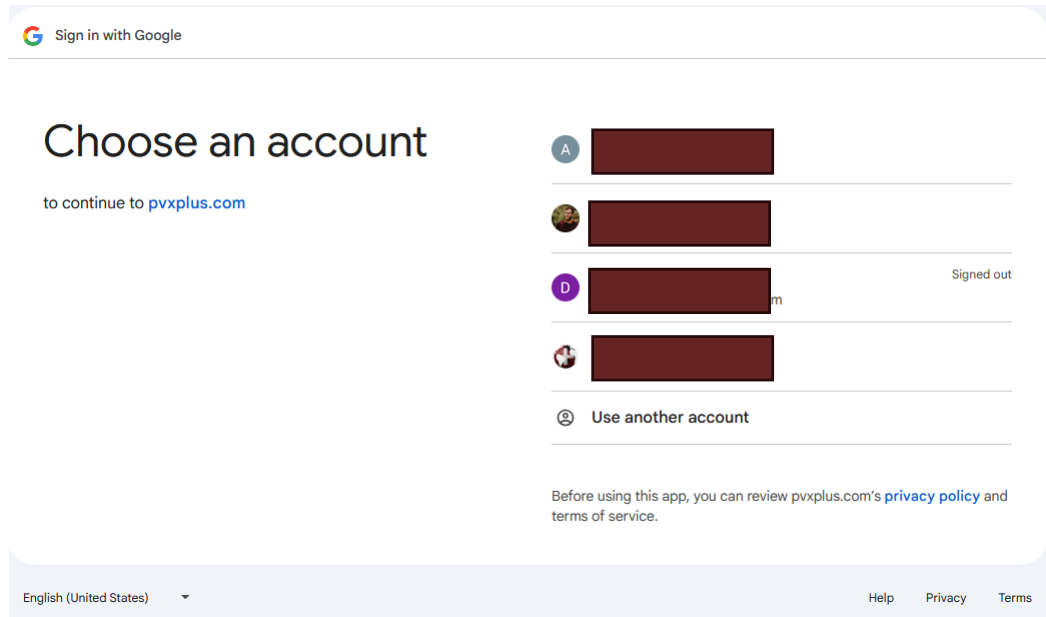
- **The First stage requires an OAuth 2.0 agent to process the user authorization**
 - This *obj/oauth2 object, by default, points to a PVX Plus Technologies hosted OAuth 2.0 agent
 - <https://www.pvxplus.com/oauth.pvp>
 - You may need to register the agent URL used with the Web service so that it knows it is safe to redirect to that URL
 - This is usually done from a Web browser via a site provided by the Web service
 - It is also possible to self-host the OAuth 2.0 agent
 - Self-hosting may be desirable if you want to avoid relying on the PVX Plus servers being up or if you want to keep it in house for security
 - To Self-host
 - Have a web server setup that can run PxPlus programs
 - Copy the files from the *web/services/oauth2/agent directory to the Web server docroot directory
 - Set the **Agent_URL\$** property to my_server_url/oauthagent.pvp

OAUTH 2.0 SECURE WEB SERVICES

Example

```
clientId$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
clientSecret$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
oAuth2=new("*obj/oauth2")
oAuth2'Service$="google"
oAuth2'client_id$=clientId$; oAuth2'client_secret$=clientSecret$
if refreshToken$="" then {
oAuth2'Enable_Certification("Do you consent to allow access of your Google account to example app?")
wait 1
url$=oAuth2'Get_Authorization_URL$("https://www.googleapis.com/auth/drive","consent select_account")
system_help url$
input "Press any key to continue after logging into account and allowing PxPlus access:","*;print ""
} else { oAuth2'Refresh_token$=refreshToken$ }
oAuth2'Get_Access_token()
refreshToken$=oAuth2'Refresh_token$
accessToken$=oAuth2'Access_token$
drop object oAuth2
```

OAUTH 2.0 SECURE WEB SERVICES



Authorization accepted

Do you consent to allow access of your Google drive account to example app?

Authorization services provided by PVX Plus Technologies Ltd.
<http://www.pvxplus.com>

OAUTH 2.0 SECURE WEB SERVICES

Make Request with Access Token

- **Either way you acquire an Access token making the Web request is the same**
- **Request the Web service with an "Authorization: Bearer " followed by the Access token in the header**
 - The Web service request is otherwise the same as a request to a Web service with no OAuth2 security

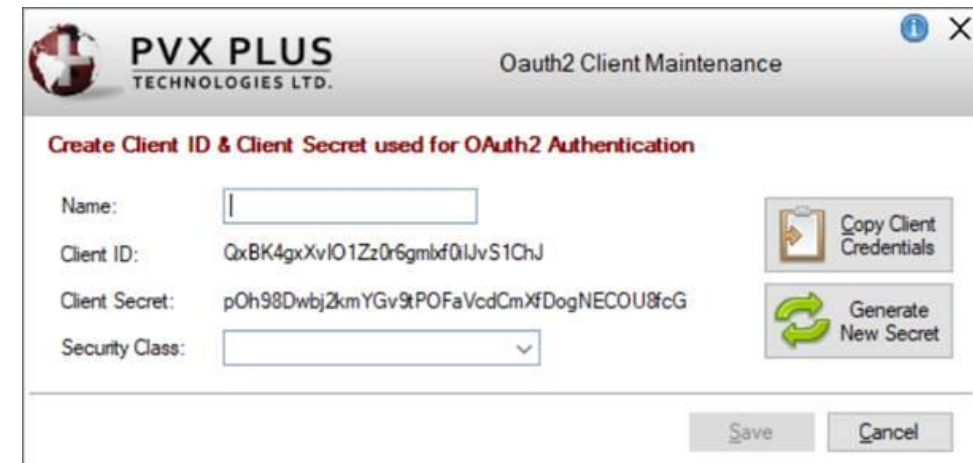
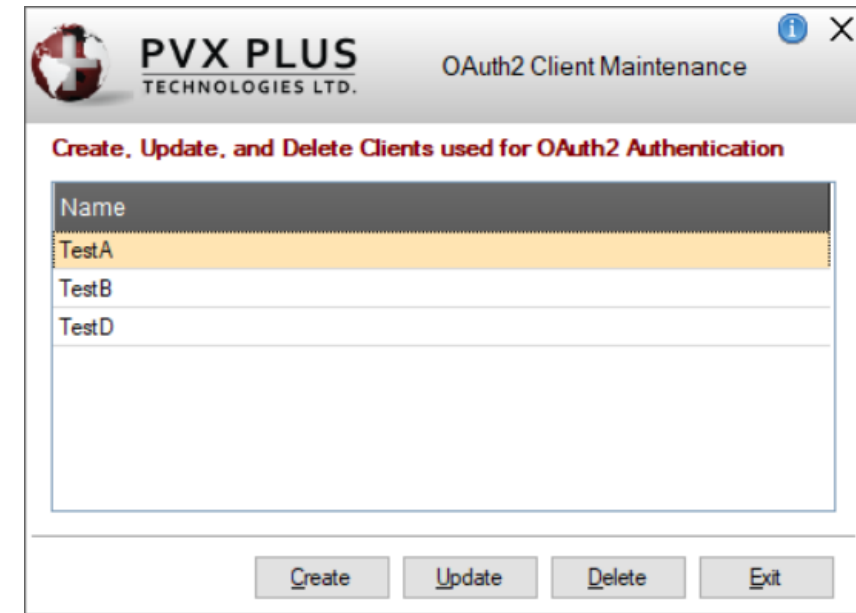
```
authHdr$="Authorization: Bearer "+accessToken$  
call "*plus/web/request","https://www.exsrvr.com/exService","",resp$,resphdr$,"","",authHdr$
```

OAUTH 2.0 SECURE WEB SERVICES

- **PxPlus provides some built-in Web Services**
 - Query
 - Chart
 - Report
 - File Maintenance
 - File Access
- **OAuth2 security can be added to restrict access to [PxPlus Web Services](#)**
 - First, OAuth2 clients must be defined using either [OAuth2 Client Maintenance](#) or the [OAuth2 Clients Object](#)
 - Next, access is restricted either via NOMADS security on a query or report or by security enabled in [Web Services Maintenance](#)
- **The grant type is `client_credentials` and the token URL is `pxplusServer/services/oauth2/token.pxp`**

OAuth 2.0 SECURE WEB SERVICES

- You must first set up [Security Classifications](#) and at least an **ADMIN User** in [User Maintenance](#) prior to setting up OAuth2 clients
- [OAuth2 Client Maintenance](#) is used for adding and maintaining OAuth2 clients
 - OAuth2 clients are required to access PxPlus Web Services that have access restricted either via NOMADS security on the query or report or by security enabled in Web Services Maintenance
- **OAuth2 allows for strong security by properly managing clients**
 - If a user's system has been compromised, you can change the Client secret, thus revoking the compromised credentials access
 - If a user no longer needs access or access needs to be revoked, the client can be deleted, thus revoking the user access
 - OAuth2 clients can be managed programmatically and/or without a graphical user interface using the [OAuth2 Clients Object](#)



OAUTH 2.0 SECURE WEB SERVICES

- [OAuth 2.0 Clients Object](#) (*web/services/oauth2/clients)
 - Maintain OAuth 2.0 clients programmatically and without the need for a user interface
 - Generate and validate Access tokens

! Create a new Oauth 2.0 client

```
oauth2_clients=new("*web/services/oauth2/clients",adminUsername$,adminPassword$)  
read data from oauth2_clients'SaveNewClient$("ABC Shipping", "USER") to client_Id$,client_Secret$,access_Token_Key$
```

! Revoke Access to Compromised Client by Changing Client Secret

```
oauth2_clients=new("*web/services/oauth2/clients",adminUsername$,adminPassword$)  
read data from oauth2_clients'GetClient$("ABC Shipping") to client_Id$,client_Secret$,access_Token_Key$,security_Class$  
oauth2_clients'SaveClient("ABC Shipping", client_Id$, oauth2_clients'NewClientSecret$(), access_Token_Key$,security_Class$)
```

OAuth 2.0 Secure Web Services

- **Add OAuth2 Security to PxPlus-built Web Service**
- **The grant type is client_credentials and the token URL is pxplusServer/services/oauth2/token.pxp**
 - Provide this to the consumers of your Web service
- **It is possible to implement your own OAuth2 Access token server using the OAuth2 Clients Object if the one provided with PxPlus does not meet an application's requirements**
- **Use the OAuth 2.0 Clients Object in the code for your Web service to validate Access token**

```
if len(%http_authorization$)>=7 and lcs(mid(%http_authorization$,1,7))="bearer " {  
  base64AccessToken$=stp(mid(%http_authorization$,8,err=Return_auth_err),"B")  
  accessToken$=cvs(base64AccessToken$,"BASE64URL:ASCII",0)  
  oauth2clients=new("*web/services/oauth2/clients",err=return_auth_err)  
  if oauth2clients'ValidateAccessToken$(accessToken$)="" then goto return_auth_err  
  drop object oauth2clients,err=*next  
}
```