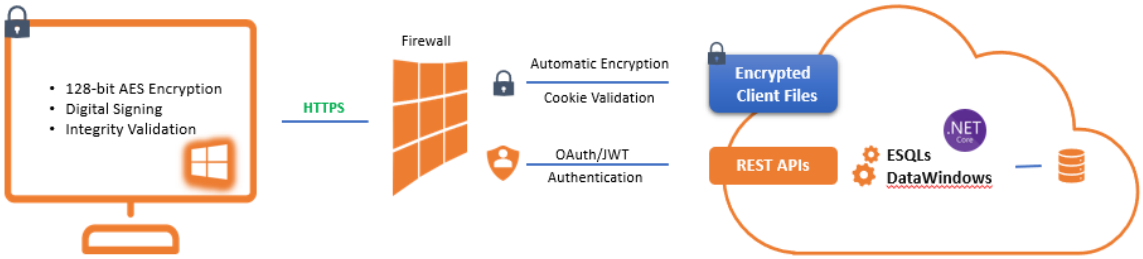
Whitepaper:

POWERSERVER 2021 SECURITY MEASURES

Last Updated: October 2021

INTRODUCTION

Installable cloud apps deployed from PowerServer 2021 adopt standard cloud-native architecture running on the .NET framework and adhere to industry best practices for security. All SQLs and DataWindows execute behind the firewall. Data is accessed through industry-standard REST APIs, secured by OAuth or JWT. The client app is encrypted, digitally signed and checked for integrity.



This document explains the detailed security measures that PowerServer 2021 has inherently implemented, or you can undertake, to protect the privacy, integrity and availability of data and files transmitted for the running of installable cloud apps:

- Automatic Encryption
- Client App Security
- Transport Security
- Cookie Validation By Web Server
- Token-Based Authentication By .NET Server
- Data Security

AUTOMATIC ENCRYPTION

Encryption is an effective way to reduce the probability of a security breach in the apps. PowerServer automatically encrypts the app files and sensitive information. The encryption algorithm is uniformly AES 128-bit.

(1) Encryption of account information in the server configuration

- If you export a PowerServer project to an .srj file, the database connection information in the file is encrypted.
- Because web server profiles contain the FTP username and password, the file that stores the profiles is encrypted.

(2) Encryption of the compiled p-code files

When compiling a PowerServer project, every SRW, SRD, SRU, etc. file from the application PBLs are compiled into its individual corresponding p-code file (that have new file extensions, such as .dwo, .apl, .fun, .win, .udo) instead of a monolithic PBD file. The PowerServer project configuration settings provide the option “Encrypt the compiled p-code files”. If the option is selected, the p-code files generated during app compilation are encrypted.

(3) Encryption of the app manifest file

Each installable cloud app contains an app manifest file that lists every file contained in the installable cloud app, and the file hash code. The manifest is encrypted.

CLIENT APP INTEGRITY

When a user loads and runs an installable cloud app, the files to be downloaded to the client include:

- A supporting program (Cloud App Launcher) and supporting runtime files (relevant PowerBuilder Runtime files). The supporting program and files can be jointly used by multiple installable cloud apps; and
- The app executable file, P-code files compiled and granularized from the source code, resource files, OCX files and other external files used by the app.

It is important to ensure that the client app only uses the files from the original deployment. Therefore, PowerServer is designed with the following mechanisms to enforce the client app integrity.

(1) Integrity assurance of the app files

Each installable cloud app contains a file manifest that lists every file contained in the installable cloud app, and the hash code of each file.

The PowerServer project configuration settings provide the option "Validate the hash of every p-code file before loading it in the app". Checking this option will check that the app files to be executed on the client have not been tampered with outside of the PowerServer compile process.

(2) Digital signing of the app executable

The app executable file (*appname*.exe) can be digitally signed. Having a valid digital signature helps to ensure the authenticity and integrity of the app executable file.

TRANSPORT SECURITY

Installable cloud apps make HTTP requests to the web server to download the app files and necessary runtime files, and make HTTP requests to PowerServer APIs to request services and transfer data. It is strongly recommended that you force TLS 1.2 for the file transfer between the client and the web server, and also force TLS 1.2 for the REST communications between the client and the .NET server.

HTTPS is essential for your app's security. Encrypting data in transit via HTTPS will prevent attackers from being able to view or interact with data. Once the HTTPS is enabled, all information transports are encrypted, which includes the full URL, cookies, data (binary or plain text), session ID, token, and other headers.

HTTPS encryption renders data sent across a network from the client to the server unreadable to sniffers. If a sniffer intercepts the data, it finds the data unusable because the data is encrypted.

COOKIE VALIDATION BY WEB SERVER

It is possible to add a cookie validation script in the app folder of an installable cloud app at the web server. When a client requests to run the app, the web server gets and validates the cookie from the client against the script, and determines whether to allow the client to download the required files (CloudAppLauncer, the app runtime files etc.) for running the app. The cookie validation is performed at the start of every client session.

TOKEN-BASED AUTHENTICATION BY .NET SERVER

Using the traditional authentication method for development only

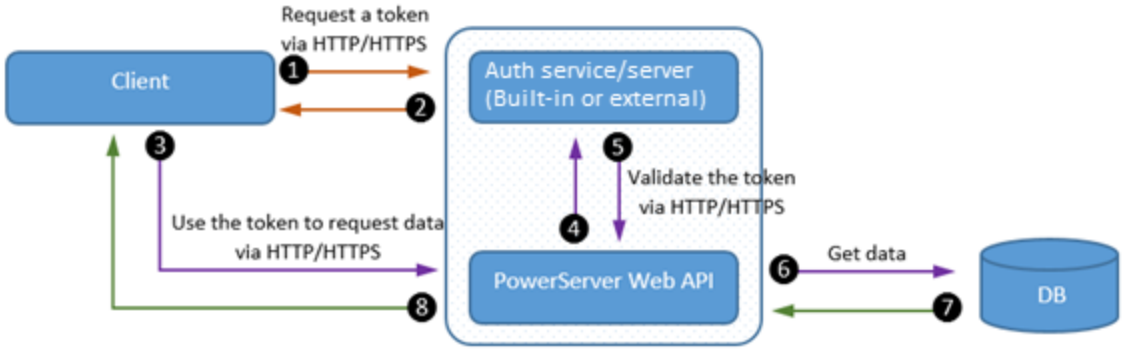
Communication between the client app and the .NET server is through standard REST APIs. If you select the "Do not use auth service" option in the PowerServer project settings, the generated REST APIs do not perform any authentication and can be readily accessed. During development it should not be a big concern, but when going into production, especially if the REST APIs are exposed over the Internet, then it is highly recommended to implement some token-based authentication for the REST APIs.

Implementing token-based authentication for production

Token-based authentication helps to mitigate REST API security risks as follows:

- An app user's access to any resource will be denied unless he/she is granted a token.
- An app user should only have the required set of permissions to perform actions on PowerServer's REST APIs for which they are authorized, and no more.

Below is a sample workflow of token-based authentication. As illustrated in the workflow, the client sends the user credentials to the authentication service/server (built-in or external). The authentication service/server validates the user, and if validation is successful, it authorizes and returns a token to the client. Then, the client will send all the subsequent requests with the token, and the PowerServer's REST APIs will validate token in the request with the authentication service/server before handing the request.



There are various approaches to configuring an authentication service/server to work with PowerServer's REST APIs. Which one to select would depend on your security requirements and the existing security infrastructure in your company.

Approach 1: Use a built-in auth template to set up built-in authentication

The steps are simple and quick if you decide to implement token-based authentication by utilizing one of the three built-in auth template options in the PowerServer project settings:

- Use built-in JWT auth
- Use built-in OAuth server
- Use built-in AWS Cognito auth

With this built-in approach, the auth service/server is hosted in the PowerServer solution, and the users and their credentials can be either stored in a .cs file in the solution, or the designated authentication database. Follow the instructions in the respective document to implement the built-in authentication for the .NET server: Using JWT, Using OAuth 2.0, Using Amazon Cognito.

Approach 2: Use an authentication server that already exists

PowerServer 2021 provides templates that can be easily extended to support an existing authentication server that works with the OAuth 2.0 flows, such as Azure AD or Azure AD B2C. If you have already implemented such an authentication server, you can select the "Use external auth service" option in the PowerServer project settings, and follow the instructions in this document to implement it for the .NET server: Using other authentication servers.

Approach 3: Set up a new authentication server

Sooner or later you may want to set up your own authentication server so that you can share it among multiple projects that may be developed with other tools and languages. The requirement for the new authentication server is that it must comply with the OAuth 2.0 flows. There is a long list of notable OAuth providers (see https://en.wikipedia.org/wiki/List_of_OAuth_providers). Select whichever one you like, and then follow the relevant documentation to set up the server.

To have the new authentication server working with a PowerServer project, please select the "Use external auth service" option in the PowerServer project settings, and follow the instructions in this document: Using other authentication servers.

DATA SECURITY

Data security is a core focus of cloud app security. PowerServer has taken a number of measures to enhance the data security in installable cloud apps, assuming that you follow the best practices recommended in this section.

Hosting the .NET server and database in the same LAN

It is important to publish the PowerServer's REST APIs to a server that resides in the same LAN as the database server. This way, the data operations of the app will be executed in the LAN behind the firewall, which reduces the vulnerability to cyber attacks.

Specifically speaking, in installable cloud apps,

- Static DataWindows/DataStores are all .NET models and hosted on the .NET server;
- Embedded SQLs are all SQL strings hosted on the .NET server.
- Dynamic DataWindows/DataStores will be created at the client and have its SQL sent as strings, encrypted, to the .NET server during runtime.
- Dynamic SQLs will be created at the client and sent as strings, encrypted, to the .NET server during runtime.

When the client app performs a data request, it calls the corresponding REST API, and then the REST API is executed by the .NET server using the corresponding .NET model or SQL strings. After that, the .NET server returns the result-set in JSON format to the client. During the process, the data accessing and handling functions are all restricted, and under control, at the server side (.NET server and database server).

Using a firewall

Firewalls have long been the first line of defense in network security. In installable cloud apps, data is exchanged between the client, and the servers (the .NET server and the database server), so it is important to implement a firewall solution for the apps that adhere to best practices, such as configuring a whitelist for web intrusion prevention.

Setting up database connections using caches

An inherent security feature of installable cloud apps is that you no longer need to keep the database connection information (including user ID, password, and DB connection string, etc.) at the client side. To

implement a database connection, you configure the connection information in the PowerServer REST API project (that resides on the server), and then map each transaction object from the client app to a server cache. This way, the cache settings are deployed and stored in the server APIs, and the connection information will be only used by the .NET server for the database connection.

Session and transaction management

Session and transaction timeouts can be easily applied to installable cloud apps by simply configuring a setting in the Applications.json file of the PowerServer REST API project. This feature helps safeguard the application from unauthorized access when authorized users have stepped away momentarily or forgotten to log out from the system.