APPEON®

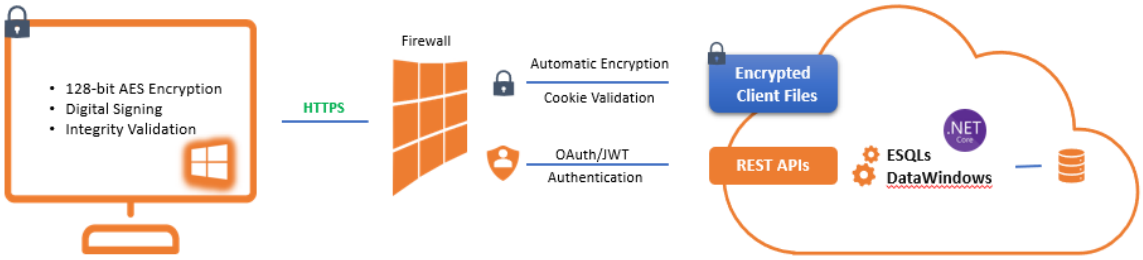
Technical Whitepaper:

POWERSERVER 2025 R2 SECURITY MEASURES

Table of Contents

1 INTRODUCTION

Installable cloud apps deployed from PowerServer 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 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
- Token-Based Authentication
- Data Security

2 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, and the encryption algorithm is uniformly AES 128-bit.

- **Encryption of account information in the server configuration**

  If you export a PowerServer project to a .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 also encrypted.

- **Encryption of the compiled p-code files**

  When compiling a PowerServer project, every SRW, SRD, SRU, etc. file from the application PBLs is 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 all the compiled p-code files”. If the option is selected, the p-code files generated during app compilation are encrypted.

- **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.

3 CLIENT APP INTEGRITY

PowerServer apps may be deployed and executed on client machines outside the direct control of the server environment. To help protect apps against tampering, unauthorized code execution, DLL hijacking, and other client-side attacks, PowerBuilder and PowerServer provide multiple layers of runtime integrity protection.

These protections help ensure that:

- Only trusted app files are executed
- Apps are downloaded from trusted locations
- Runtime components are verified before loading
- Unauthorized or malicious DLLs cannot be injected
- Native executables are hardened against common exploitation techniques

The following sections describe the mechanisms provided by PowerBuilder and PowerServer to strengthen client app integrity and runtime security.

3.1 App file integrity assurance

Each installable cloud app contains a manifest file that records all app files together with their corresponding hash values.

PowerServer can validate app integrity before application startup. When the Validate the app integrity before the app runs option is enabled, the client verifies that app files have not been modified outside the PowerServer build and deployment process.

This mechanism helps detect:

- Unauthorized modification of app files
- Accidental corruption of deployed files
- Tampering of executable or resource components

By validating file integrity before execution, PowerServer helps ensure that only trusted deployment artifacts are used by the client app.

3.2 Digital signing of app executables

PowerServer supports digitally signing app executables, including support for Extended Validation (EV) code-signing certificates.

Digital signing helps verify:

- The authenticity of the app publisher
- The integrity of the executable file
- That the executable has not been modified after signing

Apps signed with trusted certificates provide stronger protection against executable tampering and help improve user trust during installation and execution.

Using EV certificates further strengthens publisher validation through rigorous identity verification performed by a trusted Certificate Authority (CA).

3.3 Verification by the Cloud App Launcher

The Cloud App Launcher incorporates multiple verification mechanisms to ensure a secure and trustworthy client app download, installation, update and startup process:

- **Trusted app validation**

  The Cloud App Launcher only accepts apps that are flagged with a trusted identifier, helping prevent unauthorized or untrusted apps from being launched..

- **URL whitelist enforcement**

  The Cloud App Launcher validates approved download locations using the configuration defined in apprun.json.

  Only files hosted at trusted URLs are allowed for download and execution, helping reduce the risk of malicious redirection or unauthorized package replacement.

- **Digital signing of the Cloud App Launcher executable**
  The Cloud App Launcher itself is digitally signed using an Appeon Extended Validation (EV) code-signing certificate.

  If a custom Cloud App Launcher is deployed, it is strongly recommended to sign the launcher executable using a trusted code-signing certificate issued by a reputable Certificate Authority.

3.4 Secure DLL loading

By default, Windows applications may load DLLs from multiple locations, which can be exploited if malicious DLLs are placed in those paths. PowerServer provides configurable mechanisms to restrict and validate DLL loading behavior, to mitigate the risk of DLL hijacking and unauthorized code execution on the client machine.

3.4.1 Strict DLL loading paths

You can define the allowed paths for third-party DLLs. Only DLLs from these paths will be loaded at runtime. This approach significantly reduces the attack surface by preventing DLL search path manipulation.

3.4.2 DLL signature verification

PowerServer supports verifying the digital signatures of DLLs during application startup. When enabled, only trusted and properly signed DLLs are permitted to load into the app process. This mechanism helps prevent the loading of unauthorized or tampered DLLs and strengthens runtime component integrity.

Note: Currently, only Appeon-signed DLLs are verified.

3.5 Executable hardening and exploit mitigation

PowerServere 2025 R2 introduces support for advanced executable security flags that enhance runtime protection against memory corruption and code injection attacks.

These protections are applied at the operating system level and help mitigate common exploitation techniques targeting native Windows applications.

The following security features for application executables:

- DEP (Data Execution Prevention)
  Prevents execution of code in non-executable memory regions, mitigating buffer overflow and memory corruption attacks.
- ASLR (Address Space Layout Randomization)
  Randomizes memory address layout, making it significantly harder for attackers to predict target addresses during exploitation.
- CFG (Control Flow Guard)
  Validates indirect calls and jumps to ensure they target legitimate locations, reducing the risk of control-flow hijacking and code injection.
- SafeSEH (Safe Structured Exception Handling)
  Protects the exception handling chain from tampering (applicable to 32-bit executables).

These protections help:

- Reduce the risk of buffer overflow exploitation
- Mitigate code injection attacks
- Prevent control-flow hijacking

- Increase resistance against return-oriented programming (ROP) techniques
- Improve overall runtime resilience of native applications

Notes:

- This feature is currently in beta.
- Currently, these protections apply only to application executables.
- Support for PowerBuilder runtime components will be expanded in future releases.

4 TRANSPORT SECURITY

4.1 Securing communication with TLS and HTTPS

Installable cloud apps communicate with two main components:

- The web server, to download application files and runtime resources via HTTP requests
- The PowerServer Web APIs, to access services and exchange data

To protect these communications, it is strongly recommended to enable TLS 1.2 or TLS 1.3 for both channels:

- File downloads between the client and the web server
- REST API calls between the client and the .NET server

Enabling HTTPS is essential for your app's security. It ensures that all data in transit is encrypted, which protects against interception, tampering, or eavesdropping. Once HTTPS is configured:

- All data—including full URLs, cookies, request/response bodies (binary or text), session IDs, tokens, and headers—is encrypted during transmission.
- Sensitive information cannot be viewed or manipulated by third parties during transfer.

Once your server is configured to use HTTPS, both the Cloud App Launcher's download process and the client app's runtime calls to the PowerServer Web API perform strict server certificate validation by default. This ensures that every TLS handshake verifies the server's identity against a trusted root and confirms that the certificate chain is unbroken and unrevoked. By enforcing robust TLS 1.2/1.3 settings and rejecting any

certificate errors, you effectively eliminate the risk of man-in-the-middle attacks, data interception, or unauthorized content injection.

4.2 Bypassing certificate validation (for testing only)

In certain controlled, non-production scenarios—such as development environment or testing environment that uses self-signed server certificate—you may temporarily enable the “Ignore Server Certificate” option to bypass validation errors. However, this exception must be scoped strictly to your test environment. Always verify that production builds continue to enforce full certificate validation to maintain end-user trust and compliance with industry best practices.

5 COOKIE VALIDATION

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 (CloudAppLauncher, the app runtime files etc.) for running the app. The cookie validation is performed at the start of every client session. In addition, with the GetHttpRequestHeader and GetHttpRequestHeaders functions of the Application object, the HTTPClient object in the app can get the cookie from the HTTP request header, and then validate the cookie if necessary.

6 AUTHENTICATION AND AUTHORIZATION

This section describes how to secure access to PowerServer applications and APIs by using authentication and authorization mechanisms.

6.1 Development mode (no authentication — not for production)

Communication between the client app and the .NET server is performed 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 accessed directly.

This approach may be acceptable during development, but it is not secure and must not be used in production, especially when APIs are exposed over the Internet.

6.2 Windows authentication

PowerServer 2025 R2 provides built-in support for Windows authentication in PowerServer and PowerClient. Applications can use the Windows credentials that users have already provided when signing in to their computers.

With this approach, users do not need to enter or manage a separate username and password for the application. This simplifies authentication and reduces the risk associated with credential management.

This method is particularly suitable for enterprise environments where Windows domain authentication is already in use.

6.3 Token-based authentication

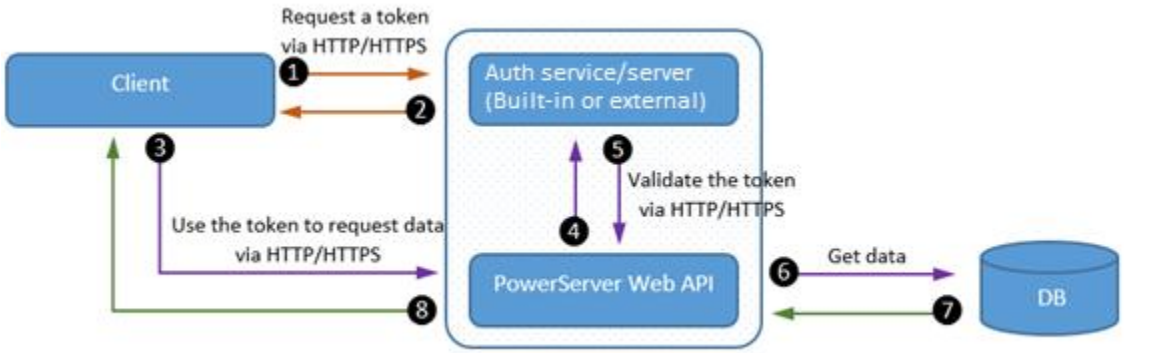
For production environments, it is strongly recommended to secure REST APIs using token-based authentication.

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 (Figure 2), 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 the token in the request with the authentication service/server before handing the request.

Figure 2



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.

6.3.1 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 server
- Use built-in AWS Cognito server

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, and Using Amazon Cognito.

6.3.2 Approach 2: Use an authentication server that already exists

PowerServer provides templates that can be easily extended to support an existing authentication server that works with the OAuth 2.0 flows or JWT, such as Azure AD, Azure AD B2C, Visual Guard, and Okta OIDC (OpenID Connect). If you have already implemented such an authentication server, you can select the "Use external Azure Active Directory service" (for Azure AD and Azure AD B2C) or "Use external auth service" (for Visual Guard and Okta OIDC) option in the PowerServer project settings, and follow

the instructions in this document to implement it for the .NET server: Using Microsoft Entra ID; and Using other authentication servers.

6.3.3 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 or JWT. There is a long list of notable 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.

6.4 Authorization for management APIs and custom endpoints

PowerServer Management APIs are disabled by default for security reasons and must be explicitly enabled via app.UsePowerServerManagementAPI().

Once Management APIs are enabled—or if you expose custom Web API endpoints—you must enforce strict authorization policies.

We recommend defining role-based or claim-based requirements in AuthenticationExtensions.cs (for example, requiring an Admin role for management API access) and applying equivalent policies to all custom APIs.

By combining token-based authentication with proper authorization checks, you ensure that only properly privileged users can access or modify sensitive resources, significantly strengthening your security posture.

7 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.

7.1 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, on the server side (.NET server and database server).

7.2 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.

7.3 Securing database connection information

PowerServer applications should avoid storing or exposing database connection information on the client side, including user IDs, passwords, and database connection strings.
To protect database connection information, PowerServer provides two security measures:

- Server-side database connection caches

- Secure connection strings for dynamically configured connection properties

7.3.1 Using server-side database connection caches

For installable cloud apps, database connection information is configured in the PowerServer REST API project on the server. Each transaction object in the client application is mapped to a server-side cache.

With this approach, the connection information is deployed and stored in the server APIs, and is used only by the .NET server to establish database connections. The client application does not need to store database user IDs, passwords, or connection strings.

7.3.2 Securing database connection strings

If your application dynamically sets connection properties, such as the user ID or password of a transaction object, these values may be stored in plain text and may remain in memory. As a result, they could potentially be accessed from the client process.

Starting from PowerServer 2025 R2, you can use the secure connection encryptor provided in the IDE to generate encrypted database connection strings. You can then pass the encrypted string when setting up the database connection.

Example:

```
sqlca.EnableSecureConnection (true)

sqlca.SetSecureConnectionString("xqRAXnVg6rXmXbnDh0KDONUBU1rnnrjoD91RyP
Elnxrf0l9SnSlfpVvBhjVILLH9+mvvHhYexk0niM+N17rdZQLFCEW0x7z89qnNpVpIDgbhK
HB7tvPyM2CaoFc6gsdvifEmUh0ICExJAyfzAA+GcA==")

sqlca.SetSecureConnectionProperty("logid", "appeon")

sqlca.AutoCommit = false

connect;
```

7.3.3 Configuring database connections using environment variables

PowerServer 2025 R2 supports configuring and overriding database connection settings through environment variables, aligning with ASP.NET Core configuration best practices. This approach improves both security and deployment flexibility, especially in containerized and cloud environments such as Docker and Kubernetes.

By using environment variables, you can avoid storing sensitive database connection information in configuration files. At runtime, environment variables can override the values defined in the Applications.json file.
In previous versions, database connection settings had to be manually modified in the Applications.json file under the AppConfig folder. With environment variable support, these settings can now be managed more securely and dynamically during deployment.

7.4 Session and transaction management

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

7.5 Message-level integrity (HMAC)

PowerServer offers an application-level client message authentication feature based on HMAC (Hash-based Message Authentication Code). When enabled, the client generates a unique authentication code for each request using the request data and a shared secret key. This code helps ensure the integrity of the message — if the data is tampered with during transmission, the code will no longer match.

On the server side, PowerServer verifies the HMAC to ensure that the request has not been altered during transmission. While HTTPS encrypts data in transit, it does not guarantee that the client itself is trustworthy — for example, when running on unmanaged or potentially compromised devices. In such cases, HMAC adds a critical layer of protection by ensuring that only properly signed requests are accepted, effectively detecting and rejecting any unauthorized modifications — even from compromised clients.

8 THIRD-PARTY DEPENDENCIES AND PLATFORM COMPONENTS SECURITY

Third party components — including operating systems, the .NET runtime/SDK, direct NuGet packages, and their transitive dependencies — constitute a material software supply chain risk that can affect runtime and data security. PowerServer supports production deployments only on .NET Long Term Support (LTS) releases. Earlier or non LTS versions may continue to operate but are not recommended for production due to reduced security maintenance and shorter support windows. PowerServer performs compatibility validation and will announce support for new .NET LTS releases in advance to facilitate customer upgrades.

PowerServer supports running with up to date .NET SDK security patches. Deployment owners should adopt an internal policy to apply vendor security updates promptly and to include SDK patch checks in CI/CD pipelines.

Operating systems hosting PowerServer components must run supported releases and receive timely security updates. Production environments should implement controlled patch windows, change approval and rollback procedures, and baseline host hardening (disable unnecessary services, enable host firewalls, and follow vendor or CIS hardening guidance).

The PowerServer team conducts regular security scans and maintainability assessments of third party NuGet packages and their transitive dependencies used in the C# solution template. Dependencies identified as high risk or associated with publicly disclosed CVEs are prioritized for compatibility testing and remediation. Fixes are delivered to customers via merge requests (MRs) or, for urgent cases, emergency security patches.

9 INFRASTRUCTURE AND PERIMETER SECURITY

PowerServer provides built-in security mechanisms for authentication, authorization, transport security, and data protection. Organizations may also deploy additional infrastructure-level security controls as part of a defense-in-depth strategy.

These complementary controls help mitigate environment-specific risks without requiring changes to the application code.

PowerServer security mechanisms are designed to work together with standard enterprise security infrastructure rather than replace it.

9.1 Web Application Firewall (WAF) integration

PowerServer apps are deployed as standard ASP.NET Core Web APIs and can be integrated with industry-standard Web Application Firewalls (WAFs) and reverse proxy solutions.

A WAF inspects incoming HTTP/HTTPS traffic and helps block malicious requests before they reach the PowerServer server components.

Common protections provided by a WAF include:

- Filtering common web application attacks, such as SQL injection (SQLi) and cross-site scripting (XSS)
- Rate limiting and bot mitigation
- Traffic inspection and anomaly detection
- Virtual patching for newly discovered vulnerabilities

Deploying a WAF can help reduce exposure to application-layer attacks and strengthen perimeter security.

9.2 Recommended deployment scenarios

The use of a WAF or reverse proxy security layer is particularly recommended for:

- Public-facing applications
- SaaS or multi-tenant deployments
- Internet-accessible APIs

- Environments handling regulated or sensitive data

In these scenarios, the WAF serves as an additional security layer at the network boundary, allowing PowerServer to focus on application and business logic execution.

9.3 Compatibility with enterprise security infrastructure

PowerServer is designed to integrate seamlessly with standard enterprise and cloud security infrastructure.

Organizations can deploy PowerServer behind cloud-native security services such as:

- AWS WAF
- Azure Application Gateway
- Cloudflare

PowerServer can also be integrated with on-premises security appliances and reverse proxy solutions, including:

- F5
- Fortinet

These integrations can typically be implemented without requiring modifications to PowerServer app code.

This separation of responsibilities allows infrastructure security policies to be managed independently from application development and deployment.

About Appeon

Appeon helps software developers build faster, better, business apps. PowerBuilder, the flagship product of the company, has been used by over 18,000 organizations worldwide, and has a long history of being relied on for mission-critical systems. Appeon's products simplify and accelerate the development of data-rich cloud apps for Windows.

For more information visit www.appeon.com, or follow Appeon on social media: Facebook, YouTube, Twitter, LinkedIn.

For customers interested in learning more about Appeon products:
Global Customer Service Center: +1-877-3APPEON (+1-877-327-7366), sales@appeon.com