Security design for H54S + Postgres based SAS Solutions

This document describes the security principles that H54S-enabled solutions rely on, particularly those that require secured, programmatic access to an embedded database (such as the SharedServices db) as part of their operation.

Contents:

1. How a H54S based solution works
2. Metadata and Front-end security design
3. OS and Database security design

How a H54S (AppFactory) based solution works

Applications built using the HTML5 Data Adapter for SAS are thin-client Javascript-based Web Applications which utilise a SAS-based backend as the data and security provider.

The data flow in a typical application looks like this:

- User loads the App in their browser (or containerised browser instance when Progressive Web App), which is loaded from SASWebServer htdocs, a .war, or the secured AppFactory CMS DB (Jackrabbit, uses SAS as security provider)
  - Browser / WebView loads HTML, JS, CSS, images, etc. that the App is composed of
  - Upon page initialisation the App uses h54s.js to make a XHR request to the SPWA or Execution Service in order to execute the app's boostrap program, typically called `startupService`
    - If the user is not authenticated and the request is redirected to SASLogon, h54s.js will issue a callback requesting that the user provides credentials, pushing the `startupService` request to a post-authentication queue for execution
  - The `startupService` provides initial information on user, the roles that user has access to in SAS (Metadata or DB) , and the initial values for any data-driven menus, dropdowns or items (secured against their identity)
  - Subsequent user actions trigger interacts with SAS programs in a similar way, sending datasets that describe user selection, data, or behaviour to the server. The SAS code interprets those input datasets and returns results
  - If a user session expires after a period of inactivity, the SASLogon redirect from the next user action is queued by the adapter, allowing for the client session and session data to (optionally) persist without interruption in UX

Security design for applications deployed on SASv9 Metadata

H54S based Apps can utilise SAS Metadata as the authentication + authorisation provider (all known production apps at time of writing do). Apps require an A+A backend provider and are unable to load any data or meaningful info without the user being able to execute the back-end data services that return relevant content.

The apps work on the following principles:

- SAS Back-end Services (SAS Stored Processes in v9) are grouped into SAS Folders (Metadata Folders in v9).
  - Each app has a dedicated Metadata root folder. ReadMetadata (RM) permissions restrict users' access to an app's data services. If a user does not have the RM access neccessary to execute at least the `startupService` program, the App halts and returns a message explaining that access is denied.
  - The services that power the app are segregated into logical *roles* within the App. The services that each role requires are themselves separated into Folders, which are subfolders within the Metadata Root Folder of the app. RM permissions for each role-folder are then assigned to the relevant Groups or Roles within SAS Metadata, and users are added to those groups in order to grant them access to the Application.
- The `startupService` executing under the context of the authenticated Metadata User, attempts to retrieve the metadata for each of the role Subfolders, building a `permissionsTable` of the subfolders that the currently logged on user has read access to. This table is returned to the frontend as part of the execution of the `startupService` and logic within the front end hides

the relevant App functionality from the user, as it is benign without the relevant RM permissions. The code to evaluate those permissions can look like this:

```
data permissionsTable(drop=id type);
  length APPPERMITTED 8. id $20 type $256 METADATAROOT $256;
    id=""; type="";
    set roleList;
    appPermitted=max(0,metadata_pathobj("",roleFolderName,"",type,id));
run;
```

- As the SPWA checks the metadata authentication for each request, every subsequent call to a back-end service depends on the user continuing to have the relevant authorisation permissions to access that data service. If a user's authorisation to access a particular role within metadata is denied, the section of the app that depends on that user's access to that role/service can be restricted with immediate effect without revoking the session and requiring re-authentication.

OS and Database security design for applications deployed on SASv9 Metadata

The back-end SAS code that powers the app executes within a SAS Multibridge Session, which itself executes under the Operating System identity of a dedicated SAS Server user, as configured within a dedicated Logical SAS Application Server context.

A dedicated Application Server context and a dedicated OS user for that context are typically required when the SAS code that drives the app needs to rely on embedded program logic for its operation - i.e. when permissions or logic are driven by data which can not be secured by SAS Metadata, or when an app requires an operational transactional database for managing concurrency or locking at scale. In these cases, a connection is issued at the start of each Multibridge session for that dedicated context, via a LIBNAME statement sourced from `StoredProcessServer/autoexec_usermods.sas`. Connection credentials specified here can safely rely on Operating System level security (i.e. with -r-x------ permissions sourced from a `.dbCreds.sas` file stored in the Server User's home directory), as SAS Metadata Security dictates that only administrative users with the required permissions will be able to update the deployed SAS service code, which is the only way of gaining meaningful access to that database connection.

For versions prior to SAS 9.3, this design relied on the capability of Metadata Security to restrict users' ability to execute proprietary user-written code on a dedicated server's Multibridge sessions, by requiring the user to have WM (WriteMetadata) permissions on the Application Server before they could register and run their own STP code on that server, effectively restricting their access to the App's dedicated SAS Server User and the Database Connection created in the autoexec. However, with the introduction of the capability to store STP code in Metadata TextStores (with SAS 9.3), that design no longer holds, as users are now able to register a Stored Process (& code TextStore) in their Home directory, associate it with any Application Server Context they have ReadMetadata permissions for (required for users of any App running on that context), and execute their own code as that SAS Server User with full access to that pre-assigned Database connection.

In order to fully secure access to the App's OS user and pre-assigned database connection in later versions of SAS, an `Initialization program` Property is added to the `LogicalServer` definition of the dedicated context's Logical Stored Process Server in Metadata, with a value of `/home/sassef/permTest.sas`. This instructs the STP server that it must execute the code stored within permTest.sas prior to executing any user-written STP code. That program, `permTest.sas`, looks something like this:

```
%macro stpSecurityTest;
  %if %qsubstr(&_PROGRAM, 1, 32) ^= %nrbquote(/Products/SAS Energy Forecasting) %then %do;
    data _null_;
      file _webout;
      put "You are not permitted to execute code on this Server Context. This incident may be reported.";
    run;
    %abort cancel;
  %end;
%mend;
%stpSecurityTest;
```

This ensures that any code that is about to execute on the dedicated STP server is running from a permitted Metadata Folder root, which continues to be enforced with standard SAS Metadata permissions in SAS 9.4. Any programs that are executed on that STP

server from other locations, such as those from `/User Folders`, are subject to the `%abort cancel;` before a single line of user-written code is executed. This closes the security workaround enabled by Metadata-stored STP code in SAS 9.3 and 9.4, preventing users from any unauthorised access to the OS Server User or the App's associated pre-assigned database connection, allowing the application logic within the SAS code to use that connection safely.