

Simplifying SAS® Security

Derek Morgan, Washington University Medical School, St. Louis, MO
Michael Province, Washington University Medical School, St. Louis, MO

Abstract

Our organization has made use of the security built into the SAS® System. Encryption and passwording of the datasets used for this project was relatively easy to accomplish. However, day-to-day uses of the data range from analysis to data entry to ad hoc query. Using macro variables to carry the passwords allows users access without hard coding them or typing in a password each time a data set is needed.

The Method

Creating a SAS system table with the built-in security is easy. There are three levels of password protection: read, write, and alter. You can employ one, two, or all three on any given dataset. In addition, you can encrypt a dataset with the simple dataset option ENCRYPT=YES.

Creating an Encrypted Dataset with Passwords

```
DATA foo.bar (ENCRYPT=YES READ=foo WRITE=bar  
ALTER=foobar);
```

Any time a dataset needs to be opened for access, the password for the desired level of access needs to be specified. Only the maximum level of access necessary needs to be specified for a given dataset request.

Accessing a Dataset with Passwords

```
PROC MEANS DATA = foo.bar (READ=foo); RUN;  
  
PROC FSEDIT DATA=foo.bar (WRITE=bar) SCREEN=proj.foo;  
RUN;  
  
PROC SORT DATA=foo.bar (ALTER=foobar); BY ID; RUN;
```

A protected/encrypted dataset will always need to have a password associated with it in a PROC or DATA step. While you can make a non-encrypted work copy, you will still need to open the original dataset with its read password.

MACROing Your Way to Security

First, each project has its own set of passwords. All the datasets for a given project use the same passwords. This allows us to use macro variables to represent them. Now, instead of being forced to type the passwords in our SAS programs (where anyone who gains access to that ASCII file will have the passwords to the project) we use:

&RPWD – Read-level password.
&WPWD – Write-level password.
&APWD – Alter-level password.

This way, a programmer can write code without knowing the passwords for any dataset, and the passwords are not immediately obvious. The key is in how these macro variables are assigned. The approach we have chosen is to execute an interactive application inside of the AUTOEXEC.SAS file associated with the project. This specific application uses the FSEDIT component of the SAS/FSP® product, but any interactive facility within the SAS System will do. Whenever the SAS System is initialized, this application executes. The user is prompted to enter an access password (which is NOT the same as any of the passwords for the data tables.) The code below details how the password application functions.

SCL for Password verification

```
1  init:  
2  control label;  
3  cursor passwd;  
4  return;  
  
5  main:  
6  return;  
  
7  passwd:  
8  if passwd ne " " then do;  
9      fn = "secure.t4 (read=" || passwd || ")";  
10     pwfil = open(fn);  
11     if pwfil then do;  
12         rc = fetch(pwfil);  
13         call symput('rpwd', getvarc(pwfil,1));  
14         call symput('wpwd', getvarc(pwfil,2));  
15         call symput('apwd', getvarc(pwfil,3));  
16         rc = close(pwfil);  
17         call execcmd('end');  
18         return;  
19     end;  
20     msg = "Password incorrect ";  
21     call execcmd('end');  
22     return;  
23 end;  
24 msg = "Please enter a password";  
25 cursor passwd;  
26 return;  
27  
28 term: return;
```

If the password field is not blank, the SCL program attempts to open the password table, which is encrypted and therefore password-protected. The passwords for this table are not the same as those for the project data. If the correct password is entered, the password record is loaded into the program data vector, CALL SYMPUT is used to fill each macro variable with its corresponding password, and the SAS System resumes. If the password entered is incorrect, the password application terminates without defining the macro variables. To access the project data from code, substitute the macro variable name for the password. Using the original example, we now have:

Accessing a Dataset with Macro Passwords

```
PROC MEANS DATA = foo.bar (READ=&rpwd); RUN;  
  
PROC FSEDIT DATA=foo.bar (WRITE=&wpwd)  
SCREEN=proj.foo;  
RUN;  
  
PROC SORT DATA=foo.bar (ALTER=&apwd); BY ID; RUN;
```

Now there are no exposed passwords in the code. This works well for any SAS System process that is started manually. Since we do not have any automatic batch jobs, this is sufficient for our purposes. The only place we need an automated method for password implementation is in UNIX script jobs that execute several SAS programs. This is handled by invoking the script using the password table's password as a parameter.

>monthly.report rosebud

AUTOEXEC.SAS checks for the presence of a UNIX environment variable using %SYSGET. If it exists, then this code executes instead of the interactive application so that each time a new SAS System job starts inside the script, the user is not queried for the password.

```
%LET batchvar=%SYSGET(parm1);  
...  
DATA _NULL_;  
SET secure.t4b5d9l3 (READ=&batchvar);  
CALL SYMPUT('rpwd',ereaaede);  
CALL SYMPUT('wpwd',iwiritei);  
CALL SYMPUT('apwd',alataear);  
RUN;
```

Security Within A SAS System Application

Another situation where we use macro variables to carry the passwords is within our data entry and management applications. The application has its own user identification and authorization facility, which is the entry point into the application. After a userid/password combination is entered, the module opens the security dataset where the userids and passwords are stored. This dataset is encrypted, and the passwords are not the same as the passwords for the other datasets. If the userid and password combination matches the values in the security dataset, the macro variables are loaded, and the main menu for the system is displayed. When the user closes the main menu to exit the application, all of the security macro variables are set to blank, and the application terminates.

If the userid/password combination does not match, the system clears the userid and password fields, prompts the user to try again, and counts an incorrect try. If the maximum number of tries has been reached, then the application is closed. Otherwise, the user is allowed to try again.

This module is the only place in the application where the data passwords are hard-coded. The security dataset passwords are also hard-coded in any module that alters the security dataset, such as changing a user password, or adding/removing users. These modules are distributed without source code, so there's no way to get the passwords from the code. You can remove source code from program modules using the following:

```
PROC BUILD CAT=libname.nocodecatalog BATCH;  
MERGE CAT=libname.codecatalog NOSOURCE;  
RUN;
```

This copies the contents of the modules in CODECATALOG to NOCODECATALOG without the source code.

Accessing Encrypted Datasets Within the Application

When a dataset needs to be opened inside the application, any passwords are attached by using the SYMGET() function. This example stores a full SAS dataset name with options in the variable DSN, which is then used in the OPEN() function.

```
dsn = "gcx.reg (read=" || symget('rpwd') || ")";  
register = open(dsn,'i');
```

The SYMGET() function is used because a simple macro reference will be resolved at the time the program module is compiled, not at program execution. If you have all the macro variables correctly assigned in the SAS session while you are compiling these modules, then the code will work, otherwise, your users will be prompted for passwords. Using macro passwords inside of a SUBMIT block is a little more complicated, requiring a separate macro variable to hold the syntactically correct password option:

```
call symput('k'," (read=" || symget('rpwd') ||  
")");  
submit continue sql;  
create table address as  
select a.id, a.name, b.*  
from tempy as A, gc1.hs2 &k as B  
where a.id=b.id;  
endsubmit;
```

The macro variable &K is resolved when it is passed to the SAS System during the execution of the SUBMIT block. Of course, for full security, &K is blanked after it has been used.

What about %PUT _ALL_?

Yes, this will dump all the macro variables, along with all of their values to the SAS System log. As long as the application is not active, none of the password macro variables are defined, and while their names may be revealed, their values will not. You still need to log into the application to gain access to these passwords via this method. The security dataset passwords are hard-coded in source code that is missing, so it becomes a matter of hacking the userid/password combination, not cracking the macro variable password scheme.

Summary

Leaving passwords to SAS System data tables in open code negates any security enhancement that passwords and/or encryption provides. Macro variables can be substituted in the code for the passwords, and assigned by using an interactive SCL-based application, or by passing an environment variable to the program.

The same macro method of providing passwords can be used for SAS System-based applications. Once access to the application is granted, the passwords are assigned to macro variables that are used in the application code whenever dataset access is required. Use the SYMGET() function to resolve macro references during the execution of the application, instead of during compilation of the program modules.

While this method can be easily defeated with "%PUT _ALL_";, it does make it more difficult for an unauthorized person to gain access to data.

This work was partially supported by NHLBI grants HL 54473 and HL 47317.

Further inquiries are welcome to:

Derek Morgan
Division of Biostatistics
Washington University Medical School
Box 8067, 660 S. Euclid Ave.
St. Louis, MO 63110
Phone: (314) 362-3685 FAX: (314) 362-2693
E-mail: derek@wubios.wustl.edu

This and other SAS System examples and papers can be found on the World Wide Web at:

<http://www.biostat.wustl.edu/~derek/sasindex.html>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.