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 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.

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.