

Keep your database passwords out of the clear: Quick and easy tips to protect yourself

David C. Steven, Hybrid Data Systems, Seattle, WA

ABSTRACT

Using SAS® to access remote database servers, such as Oracle®, DB2®, Teradata®, etc., usually requires your database userid and password. Many folks simply add this information into their SAS program, in clear text. Obviously, this is a bit of a security concern. The two primary reasons why people use this method are (1) That's what they know will work, and (2) Having SAS prompt for the password each time it's required often is not an acceptable alternative. This paper addresses a "quick and easy" way to implement a better means of keeping your password secure from someone else casually seeing it in your SAS code.

A FEW WORDS ABOUT SECURITY REQUIREMENTS

Enterprise Security Offices often require strong encryption methods for data and data streams. For example, one will usually find a requirement that passwords sent over the network must be encrypted using the equivalent of at least a 128-bit encryption key. This sort of a requirement is managed by the IT department in the way that your organization's network is set-up. While SAS programmers generally aren't expected to ensure the security of the network infrastructure, we're usually required to be vigilant in adhering to policies requiring that we do not share our passwords with anyone else. We'll focus on this aspect of password security for passwords used to access remote database systems from within SAS programs.

Most of the time, the situation is one in which our work is being done behind a firewall and the system security is designed to keep unauthorized individuals from gaining access the network; access to the remote database system requires an initial secure account login on a desktop, laptop, or other gateway computer; and the security requirement on the remote database password is for audit purposes and to ensure that other users on the system are not allowed to access sensitive data for which they do not have authorization.

And a lot of us (you know who you are!) enter database passwords right into our SAS code. On first glance, that might not seem "so bad", given all that security keeping unauthorized evildoers from gaining access to the network to begin with. Well, that may be true, but if that were enough, you'd think we wouldn't need the passwords at all. When working with sensitive data, many organizations need to ensure that access to certain data is audited as well as controlled to specific named individuals. And unless your organization has an enterprise "single sign on" system, if you're going to access a remote database server from within a SAS program, you're going to need to send an account name and password in some manner.

While your login password may appear to keep others from seeing any database passwords you may have in your SAS files, don't be fooled. It's only too easy for the file permissions on SAS files created in the hectic process of ad hoc analytics to be improperly set such that everyone else on the system can view your files. And it's all too often that we might send a friend or coworker some "problem" SAS program for help with troubleshooting. Oops! Did that program include your database password? Most of the time, I'm sure, it's all benign. However, as one who has provided troubleshooting support, I can tell you that in addition to adhering to security policies, it's simply considerate of others not to show them your password. Once you show someone else your password, you've put them into the position of having to share the blame if security on your account should become compromised later. It won't get you off the hook—trust me on that!—but it could potentially pull your friends and coworkers into a professionally embarrassing situation. And that's not all. Suppose the person who helped with the troubleshooting forwards your program on to someone else to help? And then one of them prints the program and fails to shred it when they're done. The entire staff of the building now potentially has access to your database account. This is why folks who work in the Security Office always seem so high-strung. Little slips can become nightmares.

Of course, work needs to get done and the best security is no access at all. We'll need to find a middle ground, take the human element into consideration, and see if we can find an easy-to-adopt technique to keep clear-text

passwords out of our SAS programs. The aim here is not to provide iron-clad “uncrackable” security for your database password used in SAS code. However, we can come-up with a method that would require deliberate and relatively sophisticated methods to get at what your password is. Given that many folks (shockingly) hard-code clear-text passwords into SAS code at present, if we can improve on that, we’ll all be better off. How about an easy method to encrypt passwords, store the encrypted password in a file that only the owner can read, and access it via a SAS macro variable? If we can do that instead of hard-coding it in the clear, I think we’ll eliminate nearly the entire problem.

PASSWORDS IN THE CLEAR

The issue we’re addressing here is the use of remote database passwords as “clear text” within a SAS program. By “clear text”, we mean that the password is unencrypted and appears as you would type it when prompted for it. In short, it’s the password, plain and clear. For example:

```
LIBNAME persinfo ORACLE USER=dave PASS="St3alMyPasswd" PATH=TAXINFO.IRS.GOV;
```

This is a common method used to define a SAS/ACCESS connection to an oracle database server. In this example, my Oracle userid is “dave”, and my Oracle password is “St3alMyPassed”. Anyone who can see my SAS program will know everything they need to know to access my Oracle database account. And sometimes all it takes is a quick glance, and a person can’t help but notice what the password is.

A BETTER ALTERNATIVE

We certainly should be able to improve upon hard-coding the clear-text password in our SAS code files. The good news is that it’s fairly painless, and once you set it up, there’s literally no extra work involved until the next time your database password expires and you need to update it. In short, here’s what we will do:

- Create a SAS-encrypted password string using PROC PWENCODE
- Save the encrypted password string in a file as a macro variable declaration using %LET
- Set permissions on the file so that only the owner of the file can read it
- Use %INCLUDE (or autoexec.sas) in our SAS programs to pull-in the macro variable
- Presto! Clear-text passwords disappear

PROC PWENCODE can be used to generate an encrypted password string. If you take a look at the *Base SAS Procedures Guide* in the SAS OnlineDoc for the Web (<http://support.sas.com/onlinedoc/913/docMainpage.jsp>), as of this writing it states that “Currently, sas001 is the only supported encoding method and is the default if the METHOD= option is omitted.” The “sas001” method generates an encoded—not encrypted—password string. The difference is that encoding is weaker than encrypting. Encoded data can be decoded by simply reversing the function used to encode it whereas encrypted data requires not only the encryption function but a key (encryption data) as well. Thus, while an *encoded* password certainly would be better than clear text, an *encrypted* password would be preferable.

As it turns-out, there is an undocumented second method available for PROC PWENCODE, “sasenc”, which generates an encrypted password string. The “sasenc” method employs SAS’s proprietary 32-bit key algorithm to generate encrypted password strings. Here’s how it works, from the SAS OnlineDoc:

```
PROC PWENCODE IN='password' <OUT=fileref> <METHOD=encoding-method>;
```

If you don’t specify an output file reference (OUT=), the encrypted password is written to your SAS log. When using METHOD=sas001, the first part of the encoded password string is {sas001} whereas when using METHOD=sasenc, the first part of the encrypted password string is {sasenc}. Don’t forget to include these prefaces as part of the encoded/encrypted password string!

To generate an encrypted password string for our example, this is what the SAS code would look like:

```
PROC PWENCODE IN='St3alMyPasswd' METHOD=sasenc;  
RUN;
```

I didn't specify an output file reference, so the encrypted password string appears in the SAS log, looking something like this:

```
1  proc pwencode in='StealMyPasswd' method=sasenc;  
2  run;  
  
{sasenc}DBCC5712385EBCEF38F9F09A10BD08D43BEE17EE  
  
NOTE: PROCEDURE PWENCODE used (Total process time):  
      real time           0.00 seconds  
      user cpu time       0.00 seconds  
      system cpu time     0.00 seconds
```

Be sure that you DO NOT save the PROC PWENCODE program containing your clear-text password!

The next step is to copy the encrypted password string, assign the string to a SAS macro variable using the %LET statement, and save this to a file in a convenient place. After doing that, be sure to set permissions on that file so that nobody else has read access to it. Here's what the contents of that file may look like; just a one-line SAS program:

```
%LET mydbpasswd=%STR({sasenc}2232812D2803115544DBF4E0451B3D01) ;
```

This SAS command creates the SAS macro variable, "mydbpasswd" (You can call this anything you want—so long as it follows SAS macro variable naming conventions) and assigns to this macro variable the result we obtained from running PROC PWENCODE on the password. The encrypted password string is put inside a %STR() function so that it's taken as a string literal.

Save this one-line SAS program somewhere convenient, name it anything you like. For this example, I'll choose to name it "dbpwd.sas" because that's short, descriptive, and easy to type.

That's all we need to do to set it up! All that's left now is to make use of it. By using a %INCLUDE statement and the "mydbpasswd" macro variable, we can keep clear-text passwords out of our SAS programs. Here's an example:

```
OPTIONS symbolgen;  
%INCLUDE '/home/dave/dbpwd.sas' ;  
LIBNAME persinfo ORACLE USER=dave PASS="&mydbpasswd" PATH=TAXINFO.IRS.GOV;
```

The OPTIONS statement is used here to show the macro variable resolution in the SAS LOG; it's not needed to use the encrypted password. Next, the %INCLUDE statement brings-in the %LET statement from the "dbpwd.sas" file, thereby creating and assigning the "&mydbpasswd" macro variable. Finally, use the "&mydbpasswd" macro variable in the LIBNAME statement instead of using the clear-text password.

This is what the SAS LOG looks like:

```
1  OPTIONS symbolgen;
2  %INCLUDE '/home/dave/dbpwd.sas';
6  LIBNAME persinfo ORACLE
SYMBOLGEN: Macro variable MYDBPASSWD resolves to
{sasenc}6103E01B1D50D45639CDB6EE245D05D0
SYMBOLGEN: Some characters in the above value which were subject to macro
quoting have been unquoted for printing.
6      !                      USER=dave PASS=&mydbpasswd" PATH=TAXINFO.IRS.GOV;
NOTE: Libref PERSINFO was successfully assigned as follows:
      Engine:          ORACLE
      Physical Name:  TAXINFO.IRS.GOV
```

We can see that a usable password doesn't even appear in the SAS LOG.

CONCLUSION

Password security is often taken for granted, but on those occasions when it's compromised, the consequences can be overwhelming, not to mention possibly newsworthy. Using PROC PWENCODE along with the techniques described here yields a quick and easy way to virtually eliminate any possibility of someone casually observing your database password(s). There's no excuse not to use it.

ACKNOWLEDGMENTS

I owe a debt of gratitude to Joel Danke of Washington Mutual for a great year of trying to "out-geek" each other with SAS techniques, and for showing me this method of using encoded database passwords. I just had to out-geek him, though, by coming-up with the undocumented METHOD=SASENC... which he consequently topped with a Unix ACL suggestion. I believe I first learned of the undocumented METHOD=SASENC either from Jawna Gardner or from that font of SAS information, SAS-L. As all good SAS folks know, there's strength in numbers (data too!).

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David C. Steven
dcsteven@dcsteven.com
<http://www.dcsteven.com/sas/presentations/>

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 trademarks of their respective companies.