Paper 273-26

The Web, SAS, and Security

Daryl B. Baird, Trilogy Consulting, Denver, CO

ABSTRACT

With the growing use of web-based applications, the issue of web security has become one of a network administrator establishing firewalls and developing an Intranet. Although the applications may be hidden behind a firewall, they can still be viewed by anyone who has the ability to access the Intranet. Full security control requires that all information be limited to specific users and/or groups from the network through an authentication process. This requirement is often overlooked due to the complexity of initial configuration and the need for ongoing security administration.

This paper is to provide a brief description on how to configure a secure web environment and simplify the security administration tasks. Examples are based using SAS/Intrnet V8 products under Apache 1.3 on a UNIX platform. Included are examples of how to configure the Apache 1.3 Web Server to handle security, server side includes, and common gateway interfaces. Brief descriptions of configuring socket, launch, and pool services under SAS/Intrnet V8 are also provided. Finally, the two topics are combined to explore the use of a web based security interface written in SAS that enables a security administrator the ability to control user and group privileges under Apache 1.3.

APACHE CONFIGURATION

For the purpose of this paper it is assumed that the Apache Web Server has been installed under **/opt/apache**. This directory will be referred to as the *server root directory*. This is different then **/opt/apache/htdocs** directory, which is the *document root directory* and is the location of all web based documents. The apache server uses three configuration files located in the **conf** subdirectory under the server root directory. These files are **httpd.conf**, **srm.conf**, and **access.conf** and are accessed in the respective order by the Apache web server. All configuration changes that are discussed occur in the **httpd.conf** file only. The Apache Web Server is not dynamic. The server must be stopped and started each time the **httpd.conf** is updated for the changes to take effect.

The following four sections discuss the configuration and the contents of the **httpd.conf** and **broker.cfg** files. Excerpts from these files are included following these sections that include all coding examples explained in these sections.

Security

The first step is to define the *AccessFileName* variable to the file that will be containing the security override features from the document root directory and below in the document tree. This is done globally at the beginning of the **httpd.conf** file. In this case, the value is set to **.htaccess**. Then within the *<Directory>* directive that defines the document root directory, the *AllowOverride* flag needs to be set to **All**. This allows for the ability to override default directives and flags as defined in the **httpd.conf** file from a control file such as a **.htaccess** file, which is key in configuring security. At this point, the web server has been configured to allow security to be defined at the directory level within the document tree. If the administrator wishes to add security to all CGI programs the same procedure can be repeated, but within the *<Directory>* directive that define the cgi-bin directory. This will require any execution of a CGI program to be authenticated by the web server.

CGI

Common gateway interface programs (CGI) can be written in any language to perform any task as long as they follow the HyperText Transfer Protocols (HTTP). This allows enormous flexibility to the developer to produce code in the code they are most comfortable with or is the most effective for completing the task. The first step to enable CGI scripts to be executed through the web is to add a *cgi-script* handler by setting the *AddHandler cgi-script* option to *.cgi*. This tells Apache that any script that ends in *.cgi* is to be handled and executed by the web server. This is usually provided in a standard installation of the Apache Web Server. The final step is to define where the CGI scripts are to be located. Including the **ExecCGI** option in the *Options* statement after the *<Directory>* directive that defines the location of the CGI scripts will accomplish this step. In the case of this example the option is added after the *<Directory>* directive used to defined the **/opt/apache/cgi-bin** directory properties.

Server Side Includes

Server Side Includes (SSI) are used to make HTML code somewhat dynamic. SSI includes the use of if/then statements, resolving of server variables and the inclusion of HTML sources from a file or generated from a CGI script. For the Apache Web Server to have this ability, the **mod_include** module must be built in the Apache server. This is a default module and is included in standard distributions of Apache 1.3. With the module installed, a new handler and a new file extension for SSIs must be added to make Apache aware of what files are to be parsed by the web server before they are returned to the browser. The *AddHandler server-parsed* value is set to *.shtml*. This identifies any file that ends with the *.shtml* extension is to be parsed by the web server. The *AddType text/html* value must also be set to *.shtml* to identify what the contents of the file are to the web server. These lines are usually included, but commented out in the **httpd.conf** file. Add or uncomment the following lines from the **httpd.conf** file:

```
AddHandler server-parsed .shtml
AddType text/html .shtml
```

These two lines tell Apache that any file ending in *.shtml* are a text and/or HTML file and should be parsed by the server. These values can be defined globally at the beginning of the **httpd.conf** file or can be defined for a specific directory if they are used between the *<Directory>* directives. Finally, between the *<Directory>* directive that this capability is desired the *Options* statement must include the *+include* value as seen in **Excerpts from httpd.conf** section.

SAS Authentication

The last element of Apache configuration is to allow Apache environmental variables to be passed into CGI applications (e.g. The SAS application broker). In order for this to be completed, Apache must be built with the **mod_env** module. This is not a standard module for Apache 1.3, but can be added to the list of modules when Apache is built. With the **mod_env** module added to Apache, the *PassEnv* statement can be used to pass environmental variables into the application broker and other CGI programs.

The **PassEnv** statement must be used between the *<Directory>* directives that define the properties of the CGI directory. An Example of this would be:

```
PassEnv SERVER_NAME SERVER_PORT REMOTE_HOST
REMOTE_ADDR REMOTE_USER
```

These five variables will now be passed to the application broker and handled as defined in the SAS/IntrNet **broker.cfg** file. In this

case, simply uncomment the *EXPORT* statement that passes these Apache environmental variables in to **_SRVNAME**, **_SVRPORT**, **_RMTHOST**, **_RMTADDR**, and **_RMTUSER** respectively. The SAS application broker can now utilize these variables to aid in security refinements. Please remember that different web servers define different environmental variables that can be passed to broker applications and the variables being passed must match those that are being exported in the **broker.cfg** file.

Excerpts From httpd.conf

```
#  Set up for Document Root Directory
#  +Includes for SSI
#  AllowOverride ALL for security
<Directory "/opt/apache/htdocs">
    Options Indexes FollowSymLinks MultiViews
+Includes
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>

#  Define AccessFileName
AccessFileName .htaccess

#  Define cgi-bin directory
ScriptAlias /cgi-bin/ "/opt/apache/cgi-bin/"

#  Define cgi-bin properties
#  AllowOverride ALL for Security
#  PassEnv for broker applications
#  ExecCGI for executables
<Directory "/opt/apache/cgi-bin">
    AllowOverride All
    PassEnv SERVER_NAME SERVER_PORT
REMOTE_HOST REMOTE_ADDR REMOTE_USER
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>

#  CGI scripts must end with .cgi
AddHandler cgi-script .cgi

#  Add handler for SSI
#  SSI must end with .shtml
AddType text/html .shtml
AddHandler server-parsed .shtml
```

Excerpts From broker.cfg

```
# Web server hostname
   Export SERVER_NAME          _SRVNAME
# Web server port number
   Export SERVER_PORT          _SRVPORT
# User's DNS name if known
   Export REMOTE_HOST          _RMTHOST
# User's IP address
   Export REMOTE_ADDR          _RMTADDR
# Username if authenticated
   Export REMOTE_USER          _RMTUSER
```

APACHE SECURITY

With *AccessFileName* value set to **.htaccess** and the *AllowOverride* set to **ALL** security can now be controlled at the directory level by placing an **.htaccess** file within each subdirectory of the document tree. This access file can reference both password files and group identification files that are stored outside the document tree to eliminate the possibility of downloading or tampering with these files over the web. The name and location of these files are identified by the using the *AuthUserFile* and *AuthGroupFile* statements within the **.htaccess** file for password and group information respectively.

For the purpose of this discussion these files have been set to the standard or default values. The location and names of these files can conceivably be anything. Multiple password and group files can even be used on one server depending on the complexity of the security model that is being deployed. The default value for the password and group files are **.htpasswd** and **.htgroup.** These files are stored in */where/ever/it/is*. This is outside the document tree so they can not be accessed through the web.

HTPASSWD and User Security

The **.htpasswd** file is used to establish secure user access to all web pages. The file has a simple text based format of **user name:password.** This allows each user trying to access the web services a unique user name and password. To help insure against security leaks the **.htpasswd** file should be generate by using the program **/opt/apache/bin/htpasswd** provided with the Apache server. This program allows for the encryption of all passwords in the file so that even by viewing them it would be impossible to tell what the password is. The syntax of using this command is as follows:

```
/opt/apache/bin/htpasswd –b –c
passwordfile username password
```

-b: Specifies running in batch mode. Without this option the user would be prompted to enter the passwordfile, username, and password.

-c: First time only options that creates the initial HTPASSWD file.

passwordfile: Name of the file to contain the user name and password. If -c is given, this file is created if it does not already exist, or rewritten and truncated if it does exist.

username: The username to be created or update in passwordfile. If username does not exist in this file, an entry is added. If it does exist, the password is changed.

password: The plain text password to be encrypted and stored in the file. Only used with the -b flag.

The following is an example of using the **htpasswd** command:

```
/opt/apache/bin/htpasswd –b –c .htpasswd
sasuser sasuser
```

This command creates a password file called .**htpasswd** in the current working directory. The "." is used to hide the file from the web server so the file can not accidentally downloaded via the browser. The user name of **sasuser** with the password of **sasuser** is added to the file. It should be noted that having the same user name and password is a major security risk and should not be allowed. By examining the contents of **.htpasswd** is easy to see that the password has been encrypted. If this command were executed again a different encryption algorithm would be used resulting in different output.

CONTENTS of **.htpasswd** file

```
sasuser:su8JHWnsAz1c2
```

User level security consist of placing a **.htaccess** file in the web-server's document directory. This can be the document root directory, document project directory, or CGI directory. This file will cause the web server to prompt a user for a user name and password when entering the system and authenticate the user name and password against the **.htpasswd** file. Only a valid user will be allowed to view the information in the directory. The following is an example of the **.htaccess** file that allows all authenticated users access to the directory.

CONTENTS of **.htaccess** file

```
AuthName "Top Level Security"
AuthType  Basic
AuthUserFile /where/ever/it/is/.htpasswd
require valid-user
```

The two key variables that are being set are *AuthUserFile*, which identifies the location of the encrypted **.htpasswd** file and the *require*, being set to **valid-user**. This variable could be set to a list of users as well. This example allows all valid users who are in **.htpasswd** file to be granted permissions to view information in the directory. If a list of users is defined by the *required* variable, the server would still require an user name and password and authenticate this against the **.htpasswd** file as defined by the *AuthUserFile* variable as well as being in the user list.

HTGROUP and Group Security

The second element to web security is that of the **.htgroup** file. This file is also a simple text file, which can be manipulated by any editor or by any batch process. The purpose of this file is to define what group(s) the users from the **.htpasswd** file are in. Contents of this file are very similar to **.htpasswd** file. The format is **group name: user name(s)**. The following is an example of the contents of the **.htgroup** file, assuming that the users listed exist in the **.htpasswd** file.

CONTENTS of **.htgroup** file

```
stooges:  moe larry curly
trilogy:  milee htavel dbaird
```

This **.htgroup** file defines two groups. The first group, stooges, consist of the user names moe, larry, and curly. The second group, trilogy, consist of the user names milee, htavel, and dbaird. Although these two groups have much in common they are still two distinct groups which could have completely different privileges on the system.

Group Level Security builds upon the Top Level Security protocol by using authentication not only against the **.htpasswd** file, but also includes authentication against the **.htgroup** file. The following is an example of the **.htaccess** file that utilizes both layers of security.

CONTENTS of **.htaccess** file

```
AuthName "Project Level Security"
AuthType  Basic
AuthUserFile /where/ever/it/is/.htpasswd
AuthGroupFile /where/ever/it/is/.htgroup
require group stooges
```

In this case when the user tries to access information within the directory they will be prompted for a user name and password. The web server will then authenticate the user name in the **.htgroup** file as defined by the *AuthGroupFile* variable. If the user is found to be within the group the server will then authenticate the user name and password against the **.htpasswd** file as defined by the *AuthUserFile*. If both security conditions are satisfied, the information will be displayed on the browser.

SAS SERVICE CONFIGURATION

SAS/IntrNet configuration is controlled through the **broker.cfg** file. As mentioned in previous sections the *EXPORT* statements must be set and correspond to the *PassEnv* statement in the **httpd.conf** file. The rest of the configuration is the defining of services that will be available. Included are a brief description of each type of service and an example of how the service is configured in the **broker.cfg** file, with the exception of Spawner services.

For each service that is utilized a PROC APPSRV must exist. This procedure is used to define the characteristics of the services and the location of data and program libraries. The PROC APPSRV may be executed as a background process or called directly for the **broker.cfg** file depending on the type of service. Following the discussion on the type of services is an example of a PROC APPSRV that could be used for any of these services.

Socket Services

The first or simplest of services is the socket service. This is a continually running service. The advantage to this is that each time the service is requested the service is already running. There is no launch time. The resources that a socket service uses while waiting for a request are minimal. The only real disadvantage is that one socket is continuously utilized and multiple jobs will be queued for this service. The following is an example of a socket service, *Develop,* that is continuously running on *web.example.com* through port 5001.

```
SocketService Develop "Develop Session"
   ServiceDescription "DEVELOPMENT
ENVIRONMENT ONLY"
   ServiceAdmin "SAS Web Administrator"
   ServiceAdminMail "sas@web.example.com"
   Server web.example.com
   Port 5001
   ServiceTimeout 300
```

Running a PROC APPSRV in which the port, data libraries and program libraries are all defined starts the socket service. This procedure is executed as a continuously running background process. The service is only available as long as the service process is executing. It is also important to consider the ownership of this background process. If the initial process is started by *root* then any command executed through this service are owned by *root* with *root* privileges. To help maintain any form of security this service needs to be executed by a user of restricted system access. This user will require read access to data and program libraries, but write and execute privileges should only be granted only as needed.

Launch Services

The application dispatcher initiates the launch service when a request is made through the URL for the service. The service is only running while the request from the application dispatcher is using the service and terminates execution immediately upon completion. The service is also owned by the user identification that is executing the web server. In the default case, the web server that is running is owned by *nobody* which would result in all launch services being owned by *nobody* as well. The advantage to launch services is no single port is being continuously used. The disadvantage is the time required for the application broker to launch the service. By only defining the required data libraries and program libraries that are to be used for a service can help minimize this disadvantage. Running multiple launch services for multiple projects or multiple users can enhance performance. The following is an example of a launch service, *Apollo*, which is being executed on the same application server as the application broker.

```
LaunchService Apollo "Development Services"
   ServiceDescription "DEVELOPMENT
ENVIRONMENT ONLY"
   ServiceAdmin "SAS Web Administrator"
   ServiceAdminMail "sas@web.example.com"
   SasCommand "/usr/local/sas8/sas +
/opt/apache/htdocs/sasweb/IntrNet8/ +
services/apollo/appstart.sas +
-work /tmp/ -sasuser /tmp/ -rsasuser +
-noterminal -noprint -nolog  -SYSPARM "
   ServiceTimeout 300
```

In this example the PROC APPSRV is in the *appstart.sas* script that is defined by the *SasCommand* statement. When the application broker requests the service this script is then executed, defining the service. Notice that no port is specified. This allows the application server to define the port as needed

Pool Services
With two different types of services being available it is only understandable that one would want the best of both worlds. The combination of socket and launch services can be found in pool services. The pool service not only combines the functionality of socket and launch services, but also allows for multiple services to be executed simultaneously.

The heart of using pool services is the load manager. The load manager acts as a traffic officer balancing the load over multiple ports. Although no discussion is provided about the load manager it should be noted that all pool services are started by the load manager and are executed under the same user identification as the load manager. Each pool service has the same user privileges as the user running the load manager. The following excerpt from the **broker.cfg** file defines the pool service *pond*.

```
PoolService pond "Development Pool Services"
   ServiceDescription "Pool SAS servers for
development"
   ServiceAdmin "SAS Web Administrator"
   ServiceAdminMail "sas@web.example.com"
   ServiceLoadManager web:5000
   SasCommand "/usr/local/sas8/sas +
/opt/apache/htdocs/sasweb/IntrNet8/ +
services/pond/appstart.sas +
-work /tmp/ -sasuser /tmp/ -nolog +
-rsasuser -noterminal -noprint -SYSPARM "
   IdleTimeout 30
   Server web
   Port 5
```

At first glance this service looks much like a launch service with a few additions. The first of these additions is the *ServiceLoadManager* command. This command specifies the server and port the load manager is running on. The last line of the examples states the number of ports that will be made available for the service, 5 in this example. When a request for the pool service occurs, the load manager verifies if the service is available. If the service is currently available and running then service is used much like a socket service. If the service is busy or not running then the load manager will launch the service using the script defined by the *SasCommand* statement. The load manager will launch up to 5 services as specified by the *Port* command. Unlike the launch service, which will terminate when finished, each pool service will remain active for the time specified by the *IdleTimeout* command, 30 minutes in this example. These features allow the application dispatcher service to be completely customized and optimized to meet any requirement.

Excerpts From PROC APPSRV

```
/****************************************
 * This file starts an Application Server
 ****************************************/

proc appsrv port=0 unsafe='&";%''' launch
&sysparm ;

*  Program Libs and logfile;
allocate file ProjLib '/project/programs';
allocate file ProgLib
  '/useful/stuff/programs';
allocate file logfile
  '/service/logs/%a_%p.log';

*  Data Libs
allocate library DataMart '/big/drive/data'
  access=readonly;
allocate library summary  '/summary/data' ;
allocate library tmp      '/tmp';

proglibs ProjLib ProgLib;
datalibs DataMart summary tmp;
log file=logfile;
request timeout=600;

run;
```

SECURITY MODEL
The security model that has been developed uses the basic concepts of web security, as previously outlined, combined with the functionality of SAS/Intrnet products. Within the document root directory, no security has been implemented. The only document that was placed at this level was a site home page. The page is an introduction to the purpose of the site and contains static links to all pages and reports available to the site. Each set of reports are divided by topic and stored within various project subdirectories by topic beneath the document root directory.

Within each project subdirectory is a **.htaccess** file that limits access to a specified group of users as defined in the global **.htgroup** file. Once a user penetrates the project subdirectory, they are prompted for an username and password. The Apache web servers then verifies if the username is in the group(s) specified in the **.htaccess** file by comparing the username to the group definitions in the **.htgroup** file. If the username is contained within the group, the web server then verifies the correct password by authenticating against **.htpasswd** file. If both conditions are met, then the project home page is displayed. This home page is usually a static HTML page that provides the first level of selection for the report. To make the page dynamic, server-side includes and htmSQL can be utilized.

From the project home page, some type of application dispatcher program is usually utilized. This requires the use of a **broker** program that is located in the cgi-bin directory. An additional **.htaccess** file is contained within the cgi-bin directory that allows only valid-user access. At this point the web server re-authenticates the username and password against the **.htpasswd** file. The user is not prompted at this point to enter their username and password a second time. This limits the use of bookmarking secure application dispatcher programs. The variables that are listed under the **PassEnv** directive are also passed into the **broker** for further levels of authentication by the application dispatcher programs.

As projects are brought online it is the responsibility of the project developer(s) to place an **.htaccess** file in the project document directory that specifies the appropriate user(s) and/or group(s) that have access to the project information. This must be done before the directory is web-enabled, or viewable from the web. The critical elements of the security system now lies with the creation and updating of the **.htpasswd** and **.htgroup** files.

THE WEB TOOL
The heart of the Web Administration tool is the ability to maintain and update **.htpasswd** and .**htgroup** files. The first step is to store the data in SAS data sets and create a web utility that allows a select group of users to access and update the information. The second step is to write a program that is executable over the web, which uses the *htpasswd* program provided by Apache to create these access files.

Under the document tree a project subdirectory is created for administration to house all administrative programs. Within this

subdirectory a **.htaccess** file is created that only allows one group to access the subdirectory. For this example the name of the group is *admin*. This group is allowed to utilize all administration programs and HTML files under this directory.

Within the *admin* project subdirectory numerous files can be stored to perform basic administration tasks. An example of this could be a simple HTML program to edit user's password. One possible illustration of this is as follows:

```
<HTML>
<HEAD>
<TITLE>User Edit</TITLE>
</HEAD>

<BODY><div align=center>
<form action='/cgi-bin/broker' target=_self>
<input type=hidden name=_service value=admin>
<input type=hidden name=_program
value=adm.edit_user.sas>
<input type=hidden name=_debug value=0>

<font class=orchid>User Name
</font>
<input type=text name=user maxlength=10>

<font class=orchid>New Password
</font>
<input type=password name=passwd1
maxlength=10>

<font class=orchid>Validate New Password
</font>
<input type=password name=passwd2
maxlength=10>

<input type=submit value='Change Password'>
</form></div></BODY></HTML>
```

Because this program is within the *admin* project subdirectory it is only accessible by members of the admin group. This program calls a SAS program that is used to perform the update to the SAS data set(s) that contain the user information. Additional capabilities can be added to change groups a user is associated with, add new users and delete existing users. Any basic administrative or security function could be added to the web tool. Even system administrative tasks such as viewing/killing web processes, viewing disk space, viewing user processes, …etc. This flexibility allows the quick development of a security/administrative web tool that can greatly simplify administrative tasks.

The second task is the creation of the **.htpasswd** and **.htgroup** file. The group file can be produce by using SAS/Intrnet products and simple put statements within a data step. The only concern is the limitation of Apache to read long record lengths. For this reason the group name may need to be repeated several times with different group members following each group statement if a very large group is involved.

The creation of the **.htpasswd** file requires use of the *htpasswd* program provided with apache to enable encryption of the password file. Since this program can be utilized in batch mode a SAS program can be used to execute the *htpasswd* command for each user. The following program uses this technique in creating the **.htpasswd** file.

```
/* ---------------------------------- */
/* This program will create the       */
/* Encrypted .htpasswd file           */
/* ---------------------------------- */

* Define Passwd file location;
%global PassFile;
%let PassFile=/where/ever/it/is/.htpasswd;

* Eliminate Multiple Users;
proc sort data=admdat.users out=users
nodupkey;
  by UserID;
run;

* Write out passwd file using the apache;
* command /opt/apache/bin/htpasswd;
* Options -b  Batch Mode;
*         -c  Create File;
data _null_;
  set users;
  retain count 1;

* Check for missing values;
if UserID ne "" and Passwd ne "" then do;
  * First time only run with -c;
  * to create file;
  if count eq 1 then do;
        com = "/opt/apache/bin/htpasswd
               -b -c &PassFile "
               ||UserID||" "||Passwd;
        count = count +1;
        end;
   else do;
        com = "/opt/apache/bin/htpasswd
               -b &PassFile "
               ||UserID||" "||Passwd;
        count = count +1;
        end;

call system(com);
end;

run;

* HTML output for display;
data _null_;
   set users end=eof;
   file _webout;
   by UserID;

* HTML and table headers
if _n_ eq 1 then do;
   put 'Content-type: text/html';
   put ;
   put '<HTML>';
   put '<HEAD>';
   put '<TITLE>Passwd</TITLE>';
   put '</HEAD>';
   put '<BODY>';
   put '<TABLE ALIGN=CENTER BORDER=3>';
   put '<TR>';
   put '<TH><font class=green> User ID
</TH>';
   put '<TH><font class=green> User Name
</TH>';
   put '</TR>';
end;

  * Fill the table;
  put '<TR>';
  put '<TD>' userid %nrstr(' ') '</TD>';
  put '<TD>' user %nrstr(' ') '</TD>';
  put '</TR>';

* Close table and HTML;
if eof then do;
   put '</TABLE>';
```

```
      put '</BODY>';
      put '</HTML>';
   end;

   run;
```

This program is divided into two data steps. The first data steps executes the *htpasswd* command for each user, while the second builds a table displaying user identification and user name. The second data step is tell the user that the program has completed. If the program is halted during execution only a partial password file will be produced.

Before the password file can be written, the location and name of the file must be assigned. This assignment is done using a *%let* command. The location of the file should be outside the document root directory and correspond to the *AuthUserFile* variable from the **.htaccess** file. Placing the file outside the document tree and hiding the file by using the "." are added security precautions so the file can not be downloaded over the Internet.

The PROC SORT is then used to remove multiple users of the same user ID. The step is just a precaution in case the same user was added two or more times. If a user has been added more than once and has been assigned a different password each time, Apache will only recognize the first password appearing in the password file for a given user.

For the first record that has a user name and password populated the *htpasswd* command is then executed using the *–c* and *–b* options. The *–c* option is used only for the first user as it will create the password file. It the file already exists, the entire contents of the file will be replaced with the new single user. For any addition users to be appended to the file only *–b* options is required signifying execution in batch mode.

The command to be executed is then stored as a character variable. This character string is then passed to the SAS CALL routine *system*. This routine then executes the character string as an operating system command. This routine is executed once for each user. If the program is halted for some reason during execution then the password file is only partially completed.

It should also be noted that since this program is executed via the Internet, the user identification operating the web process owns the file created. In the default scenario, the user *nobody* owns all web processes. *Nobody* will own any file created or program executed through a web interface. Since one user controls everything it becomes very simple to restrict access by controlling the permissions of a single user.

CONTACT INFORMATION

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

Daryl B. Baird
WORK: Trilogy Consulting
9800 MT. Pyramid Court, Suit 400
Englewood CO 80112
720.895.1980
HOME: daryl_baird@yahoo.com
303.663.4164