SYBASE®

PAM USER AUTHENTICATION IN ASE 12.5.2

WHITE PAPER

TABLE OF CONTENTS

FURTHER INFORMATION: WWW.SYBASE.COM/UNWIRE OR WWW.INTEL.COM/UNWIRE

PAM USER AUTHENTICATION

INTRODUCTION

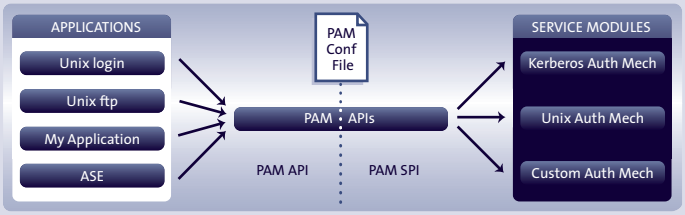
PAM (Pluggable Authentication Module) is an integrated login solution that has been available for almost a decade in the Solaris Operating System and is used to integrate and manage multiple authentication technologies (RSA, Kerberos, DCE...). Initially developed by SunSoft as part of its OS, the PAM framework has been adopted by HP, IBM as well as the Linux community.

This paper introduces the architecture and design of the PAM framework and then discusses how Adaptive Server® Enterprise (ASE) 12.5.2 has leveraged the PAM API to support a new authentication mechanism called PAM User Authentication (PAMUA) and drive down the overall total cost of ownership. This paper concludes with an example of how PAM custom service module can be written and used by ASE for authentication.

PAM ARCHITECTURE AND DESIGN

The PAM architecture can be conceptualized as a puzzle with three pieces. The front-ends or the applications that need some sort of authentication facility, the back-end known as service modules or authentication mechanism modules and finally the middle-tier or PAM configuration file, that tells which front-end application is using what service module for authentication. Both the front-end applications and services modules utilize the PAM APIs available under the SOLARIS system through the libpam library. The libpam library is in turn divided in two API sets, one used to develop applications, generally called PAM API, and the other exclusively used to code service modules, known as PAM SPI (Service Provider Interface). The PAM architecture is illustrated in figure 1.

FIG 1: PAM ARCHITECTURE



This figure illustrates the relationship between applications and service modules. There are four applications; Unix login, Unix ftp, the custom application and the ASE 12.5.2 server, that require PAM authentication services. All are linked with the PAM library. When an application calls the PAM API, it loads at runtime the authentication module indicated in the PAM configuration file. The security request is then forwarded to the service module and the response is returned to the application thru the PAM layer. It should be noted that an application never talks directly to the service module; PAM manages all communications between the two peers. As an example, when a user issues a `login` command, the request for password validity is not checked directly by the `login` executable, it is sent to the Unix authentication mechanism module that in turn accepts or denies the user-password pair.

PAM CONFIGURATION FILE

In order to fully understand the PAM design, the PAM configuration file is a key element. As commented, it maps the relationship between applications and service modules; it is an ASCII file located in Solaris under the `/etc` directory and named `pam.conf`. Table 1 shows its structure and content.

SERVICE	MODULE_TYPE	CONTROL_FLAG	MODULE_PATH	OPTIONS
telnet	Auth	Required	/usr/lib/security/pam_unix.so.1	
Login	Auth	Required	/usr/lib/security/pam_unix.so.1	Debug
Login	Auth	Required	/usr/lib/security/pam_krb5.so.1	
Login	Session	Required	/usr/lib/security/pam_unix.so.1	
Login	Password	Required	/usr/lib/security/pam_unix.so.1	
Login	Account	Required	/usr/lib/security/pam_unix.so.1	
OTHER	Auth	Required	/usr/lib/security/pam_unix.so.1	
OTHER	Session	Required	/usr/lib/security/pam_unix.so.1	Warn

TABLE 1: STRUCTURE AND CONTENT OF THE PAM CONFIGURATION FILE FOR SOLARIS.

The file consists of several fields. The first field, named `Service`, indicates the name of the application requiring an authentication mechanism. The second field, named `Module_type`, denotes the module type for the application. The field named `Control_flag` determines the behavior of stacking multiples modules. The `Module_path` field points to the location of the service module; this is the module doing the real work. Finally, the `Options` field is utilized by the PAM framework to pass module specific options. Options are comma delimited. All the fields except the `Options` field are mandatory and must be space or tab delimited.

The content of Table 1 can be read in the following way:

- The `telnet` application requires the `pam_unix.so.1` module for authentication; it does not have any associated service for password, account and session management
- The `login` application requires `pam_unix.so.1` for account, password and session management, it also requires a double authentication mechanism one using the classic UNIX authentication (`pam_unix.so.1`) and authentication verification based on Kerberos 5 (`pam_krb5.so.1`). This concept is called Multiple Authentication Services Stacking, (this is explained later).
- Please note, a special application name can be used with the OTHER keyword. OTHER means any other application name not listed in this file. So, this means, `rlogin` or `dtlogin` for instance, uses `pam_unix.so.1` for session and authentication services and do not use any mechanism for password and account management.

Also of interest:

- An application may not use PAM at all,
- An application using PAM may use four different types of modules,
- An application may stack several modules for the same service,
- An application written with the PAM framework and not listed in the configuration file will inherit the behavior of the OTHER application, if mentioned in the file, and
- The service name is typically the familiar name of the corresponding application, here, `telnet` and `login`.

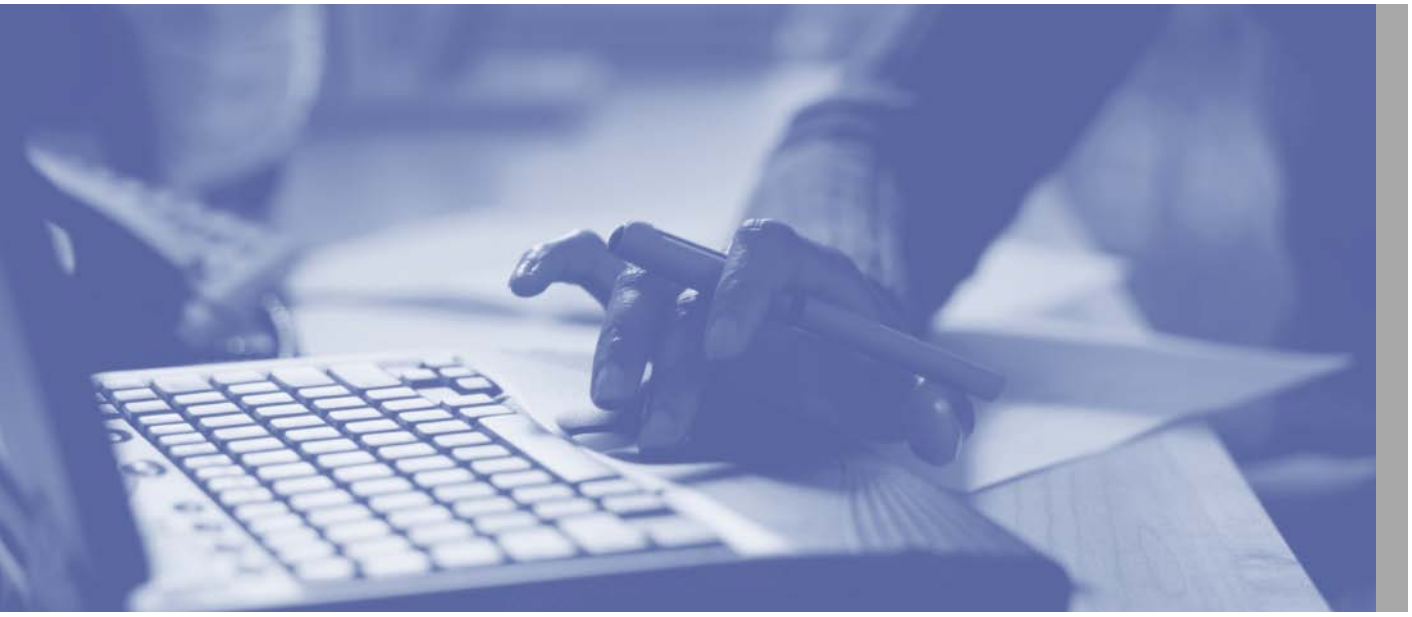
The configuration file for Linux has almost the same structure. While Solaris stores all the application settings in a single file, Linux needs one file per application located under the `/etc/pam.d` directory. The files names are the service name. To follow the example discussed in Table 1, the `/etc/pam.d` directory contains a login, telnet and other files. Due to this design, there is no need to have a `service` field. The structure for the file starts with the `module_type` attribute. The rest is unchanged.

Appendix A and B include real configuration files for SOLARIS and LINUX.

MODULE TYPE VALUES IN PAM CONFIGURATION FILE

Although PAM means Pluggable Authentication Module, its functionality covers four security areas:

- Authentication management, referenced by the `auth` keyword in the configuration file. Authentication allows a user to be identified.
- Password management, referenced as `password` in the file. Password management can be used to setup a password policy.



- Session management, referenced as `session` in the file. Session management handles the opening and closing of an authenticated session. It can be used to log activity during a session.
- Account Management, referenced as `account`. This allows checking account validity, like account expiration.

PAM was broken into four functionalities to make it pluggable. Indeed breaking the behavior into modules, helps the application providers because they can switch plugs as they wish. As a result, an application can use the latest Kerberos version for authentication and password management and use a DCE implementation for account and session management.

STACKING MULTIPLE AUTHENTICATION SERVICES

As described in Table 1, an application (e.g. `login`) may have several service modules associated to a particular functionality. PAM was designed this way to facilitate the integration of multiple authentication mechanisms; this pattern is called stacking and it is specified simply by additional entries in the PAM configuration file. Internally when a PAM API is issued from an application, the modules stacked in the back-end are invoked in the order that they appear in the file.

The `control_flag` field in the PAM configuration file governs PAM behavior. It may have the values, `required`, `requisite`, `optional` and `sufficient`.

- `Required` means the module must return success in order to have the overall result be successful
- `Requisite` means a failure in the module results in the immediate termination of the authentication process
- `Sufficient` means the success of the module is enough to satisfy the authentication requirements of the stack of modules (if a prior required module has failed the success of this one is ignored)
- `Optional` means if this module fails the overall result can be successful if another module in this stack returns success

SERVICE	MODULE_TYPE	CONTROL_FLAG	MODULE_PATH	OPTIONS
login	Auth	required	/usr/lib/security/pam_unix.so.1	
login	Auth	required	/usr/lib/security/pam_dial_auth.so.1	
login	Auth	optional	/usr/lib/security/pam_krb5.so.1	

TABLE 2: STACKING EXAMPLE WITH REQUIRED CONTROL FLAG

Table 2, shows how the login command is stacked. The `required` key word for `control_flag` enforces users to log in only if they are authenticated by both UNIX authentication and Unix authentication for dialups. The Kerberos authentication here is optional as indicated by the `optional` key word. So this means that the user can still log in, even if the `Kerberos` authentication fails.

PAM FRAMEWORK

This section briefly describes the main API used by both the applications (PAM API) and services modules (PAM SPI).

PAM Application APIs: The PAM library interface consists of functions, which can be grouped into five categories. The names for all the authentication library functions start with `pam_`.

- The first category contains functions for establishing and terminating an authentication activity (`pam_start` and `pam_end`), and functions to maintain state information (`pam_get_item` and `pam_set_item`).
- Authentication management contains functions to authenticate an individual user; this is done with `pam_authenticate`. It also contains the function `pam_setcred` to set, refresh or destroys the user's credentials.
- Account management contains a function `pam_acct_mgmt`. This function is used to check whether authenticated users can access their accounts. This function can also implement account expiration and access hour restrictions.
- Session management contains functions to perform log activity after access to the system has been granted. The functions are `pam_open_session` and `pam_close_session`.
- Password management consists of functions to change authentication tokens. An authentication token is the object used to verify the identity of the user. In UNIX, an authentication token is a user's password. The function to achieve this goal is `pam_chauthtok`.

All the `pam_*()` functions are implemented through the library `libpam`.

PAM Service Provider (SPI) APIs: SPI categorizes functions in four categories, which are the same as PAM API, authentication, account, session and password management. The names for all the authentication library functions start with `pam_sm` prefix. The only difference between the `pam_*()` interfaces and their corresponding `pam_sm_*()` interfaces is that all the `pam_sm_*()` interfaces require extra parameters to pass service specific options to the shared modules. They are otherwise identical.

PAM USER AUTHENTICATION IN ASE

With this latest release, ASE provides another user authentication mechanism. As described in the first part of this article and, thanks to the PAM framework, ASE is now much more integrated with the operating system authentication procedures; user and account management have been greatly simplified.

This section explains how ASE can be configured for PAM, it discusses the modified system store procedures and it finally describes the liaison with CIS and RPC.

ASE AND PAM CONFIGURATION FILE

Although the OTHER key word can be used as a generic application name in the PAM configuration file, when an application is not explicitly listed, Sybase recommends the use of the `ase` key word to refer to the database server. As a result, a configuration file under Solaris should contain the following lines for the authentication and account services. Please note that a Solaris administrator should modify the pam.conf file since it is ownd by the root user.

```
$pwd
/etc
$cat pam.conf
#
#non ase applications have been cut from this trace.
#
ase  auth    required    usr/lib/security/$ISA/pam_unix.so.1
ase  account required    usr/lib/security/$ISA/pam_unix.so.1
```

Under the Linux operating system, a file called 'ase' must be created with the content.

```
$pwd
/etc/pam.d
$cat ase
auth        required    pam_unix.so
account     required    pam_unix.so
```

As it can be deduced from the content of those PAM configuration file, ASE 12.5.2 only supports two of the four PAM services, user authentication and account management. The account management functionality is governed by trace flag **3636**, and is disabled by default.

NEW SP_CONFIGURE SERVER CONFIGURATION

To enable the use of PAM within ASE 12.5.2 a new configuration parameter has been added, its name is "`enable pam user auth`". This parameter is dynamic and a user must have the sso role to run it. To benefit from this server configuration, customers need to purchase the ASE Security & Directory Services Package, referenced as ASE_DIRS in the sysam `license.dat` file. The possible values for this new configuration are described in Table 3.

VALUE	DESCRIPTION
0	Enables only syslogins authentication. ASE manages access control using the syslogins system table. This is the default value.
1	Enables PAM and syslogins authentication. If the PAM authentication fails, ASE searches the syslogins to verify users' authentication.
2	PAM is the only authentication mechanism[1]

TABLE 3: "ENABLE PAM USER AUTH" POSSIBLE VALUES

1 The authentication mechanism is defined at login level. When 'enable pam user auth' is 2, it means that any user configured to use PAM, will use PAM and only PAM. Other users may however be using different authentication methods.

This option is initially available on the following operating systems:

- Solaris 32 bits, SPARC processor.
- Solaris 64 bits, SPARC processor.
- Linux 32 bits, Intel processor.
- Linux 64 bits, Intel processor.

MODIFIED STORED PROCEDURES FOR PAM UA

Before discussing the syntax changes in existing stored procedures, some ASE internal values need to be explained. The main change to be observed is the definition of new constants for user authentication to support PAM UA. Running the following SQL statement outputs these values:

```
1> select name,number,low from master.dbo.spt_values
   where type = 'ua'
2> go
 name                          number      low
 ----------------------------- ----------- -----------
 ASE                                     0          32
 LDAP                                  418          64
 PAM                                   432         128
 ANY                                     0         224
 (4 rows affected)
1>
```

The result set shows four different types of user authentication mechanisms. `ASE` is the regular ASE syslogins authentication. LDAP is the feature that was introduced in ASE 12.5.1 to authenticate users stored in an LDAP repository; PAM is the new PAM authentication mechanism object of this paper and finally ANY is the result of bit "OR-ing" all the values. The four authentication values are used by the stored procedures described in the next sections.

sp_addlogin: A new parameter has been added to the `sp_addlogin` stored procedure, identifying the authentication mechanism to be used at login time. The full syntax is:

```
sp_addlogin loginame, passwd [, defdb] [, deflanguage] [,
fullname] [, passwdexp] [, minpwdlen] [, maxfailedlogins]
[, auth_mech]
```

`auth_mech` is the new parameter and can be assigned any of the values seen in the previous section. If the parameter is not supplied, `sp_addlogin` defaults to ANY, meaning the login can be identified by any authentication mechanism, if configured.

sp_displaylogin: The sp_displaylogin stored procedure now displays the authentication mechanism associated with the login. Its syntax has not changed.

```
sp_displaylogin [loginame [, expand_up | expand_down]]
```

The examples below show the new generated output.

This example displays the information of a login whose authentication mechanism is ASE syslogins.

```
1> sp_displaylogin login_ase
2> go
Suid: 6
Loginame: login_ase
Fullname:
Default Database: master
Default Language:
Auto Login Script:
Configured Authorization:
Locked: NO
Date of Last Password Change: Mar 26 2004  7:20AM
Password expiration interval: 0
Password expired: NO
Minimum password length: 6
Maximum failed logins: 0
Current failed login attempts:
Authenticate with: ASE
(return status = 0)
1>
```

This other example shows the authentication mechanism for a user logging in through PAM.

```
1> sp_displaylogin login_pam
2> go
Suid: 7
Loginame: login_pam
Fullname:
Default Database: master
Default Language:
Auto Login Script:
Configured Authorization:
Locked: NO
Date of Last Password Change: Mar 29 2004  1:36AM
Password expiration interval: 0
Password expired: NO
Minimum password length: 6
Maximum failed logins: 0
Current failed login attempts:
Authenticate with: PAM
(return status = 0)
1>
```

Finally this shows a value NONE for the SA user.

```
1> sp_displaylogin sa
2> go
Suid: 1
Loginame: sa
Fullname:
Default Database: master
Default Language:
Auto Login Script:
Configured Authorization:
        sa_role (default ON)
        sso_role (default ON)
        oper_role (default ON)
        sybase_ts_role (default ON)
Locked: NO
Date of Last Password Change: Mar  2 2004  3:06AM
Password expiration interval: 0
Password expired: NO
Minimum password length: 6
Maximum failed logins: 0
Current failed login attempts:
Authenticate with: NONE
(return status = 0)
1>
```

The `NONE` value is displayed for logins that do not have any authentication mechanism associated to it, this is typically the case for logins created before ASE 12.5.2. NONE means the authentication mechanism is based on the `syslogings` table; this is the same as the `ANY` value.

Another feature to be stressed is the ASE behavior. Say a login has been created and its authentication mechanism is set to PAM, and the server has the "`enable pam user auth`" configuration set. Even this login has its authentication mechanism set to PAM, this does not mean that PAM is the only mechanism to be used. The first check will be done by PAM, and after a failure, ASE will check if the login/password pair is valid in `syslogins`. If this is the case, the login will be allowed to log and right after `syslogins` password will be updated with the password coming from PAM.

sp_modifylogin: Although the syntax has not changed for this procedure, a new option can be passed, this value is "`authenticate with`" and allows changing the authentication mechanism to an existing login. This example shows how to change the authentication mechanism from ASE to PAM.

```
1> sp_modifylogin login_ase , "authenticate with", PAM
2> go
Option changed.
(return status = 0)
1> sp_displaylogin login_ase
2> go
Suid: 6
Loginame: login_ase
Fullname:
Default Database: master
Default Language:
Auto Login Script:
Configured Authorization:
Locked: NO
Date of Last Password Change: Mar 26 2004  7:20AM
Password expiration interval: 0
Password expired: NO
Minimum password length: 6
Maximum failed logins: 0
Current failed login attempts:
Authenticate with: PAM
(return status = 0)
1>
```

Changing the authentication mechanism to a non-configured value at the server level is authorized; however a warning message is printed.

```
1> sp_modifylogin login_ase , "authenticate with", LDAP
2> go
Warning. Authentication mechanism 'LDAP' is not enabled.
Option changed.
(return status = 0)
1> sp_displaylogin login_ase
2> go
Suid: 6
Loginame: login_ase
Fullname:
Default Database: master
Default Language:
Auto Login Script:
Configured Authorization:
Locked: NO
Date of Last Password Change: Mar 26 2004  7:20AM
Password expiration interval: 0
Password expired: NO
Minimum password length: 6
Maximum failed logins: 0
Current failed login attempts:
Authenticate with: LDAP
(return status = 0)
1>
```

NEW STORED PROCEDURES

ASE now provides two new stored procedures, `sp_maplogin` and `sp_helpmaplogin`.

sp_maplogin: This stored procedure allows the creation of mappings between external client names and ASE logins. The sso_role is required to run this command.

The syntax is:

```
sp_maplogin(authentication_mech|NULL),(client_username
|NULL),(action|login_name|NULL)
```

Where `authentication_mech` is the authentication mode to use for the mapping. The possible values are:

- PAM
- ASE
- ANY
- LDAP

`Client_username` is the external login to ASE, and this value is something meaningful for the PAM library. The third parameter value can be either an action, or the `ASE` login. The action value can be "`create login`" or "`drop`" value.

The call to this procedure is really optional, only if the login name exists in syslogins and the login name is meaningful to the PAM library. For example, say the PAM module is based on Unix authentication.

A login is added to syslogins, requiring pam as the authentication mechanism.

```
sp_addlogin cuerva, cuerva, @auth_mech=pam
```

cuerva is a valid UNIX login

```
$>isql -Ucuerva -Pcuerva -Sase_1252
1>quit
$>isql -Ucuerva  -Sase_1252
Password: <type in the UNIX password>
1>
```

The login is authenticated and the password field is updated with the UNIX password.

```
$>isql -Ucuerva -Pcuerva -Sase_1252
Msg 4002, Level 14, State 1:
Server 's1':
Login failed.
CT-LIBRARY error:
      ct_connect(): protocol specific layer: external error:
The attempt to connect to the server failed.
$>
```

Using the "`create login`" option and a meaningful PAM library client name, ASE can dynamically populate the syslogins table. This is done the first time this external user logs into ASE. As long as the mapping is still in effect, even if the login is dropped explicitly using the `sp_droplogin` command, ASE recreates the record in `syslogins` next time the external user connects to the server.

```
1> sp_displaylogin cuerva
2> go
No login with the specified name exists.
(return status = 1)
```

cuerva is a valid UNIX login.

```
1> sp_maplogin pam, "cuerva", "create login"
2> go
Client authentication mapping updated.
(return status = 0)
1> sp_displaylogin cuerva
2> go
No login with the specified name exists.
(return status = 1)
1>
1> sp_helpmaplogin
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            cruiz                sa
 PAM            cuerva               CREATE LOGIN
(2 rows affected)
(return status = 0)
```

The external name is added to the syslogins table, the first time it connects to the server.

```
$>isql -Ucuerva -P<Pam Password> -Sase_1252
1> quit
```

Connecting with sa.

```
1> sp_displaylogin cuerva
2> go
Suid: 9
Loginame: cuerva
Fullname:
Default Database: master
Default Language:
Auto Login Script:
Configured Authorization:
Locked: NO
Date of Last Password Change: Mar 29 2004  8:01AM
Password expiration interval: 0
Password expired: NO
Minimum password length: 6
Maximum failed logins: 0
Current failed login attempts:
Authenticate with: PAM
(return status = 0)
```

Dropping the login.

```
1> sp_droplogin cuerva
2> go
Account locked.
Login dropped.
(return status = 0)
1>quit
```

Reconnecting with the PAM is possible due to the fact the mapping is still valid. This creates a new record in the syslogins table.

```
$>isql -Ucuerva –P<Pam Password> -Sase_1252
1> quit
```

Deleting a mapping can be done using the drop command; this will not delete the entry in syslogins, if it exists.

```
1> sp_maplogin pam, "cuerva", "drop"
2> go
Client authentication mapping updated.
(return status = 0)
1> sp_helpmaplogin
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            cruiz                sa

(1 row affected)
(return status = 0)
```

The procedure sp_maplogin can be used to map external login names with existing ASE logins, as shown below.

```
1> sp_maplogin pam, "cuerva", "sa"
2> go
Client authentication mapping updated.
(return status = 0)
1> sp_helpmaplogin
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            cruiz                sa
 PAM            cuerva               sa

(2 rows affected)
(return status = 0)
1>
```

The external client name `cuerva` can now log into ASE using its credentials. However in the server, it is seen as the `sa` user. This functionality does not create any new record in syslogins.

```
$>isql -Ucuerva -P<Pam Password> -Sase_1252
1> sp_displaylogin cuerva
2> go
No login with the specified name exists.
(return status = 1)
1> sp_displaylogin
2> go
Suid: 1
Loginame: sa
Fullname:
Default Database: master
Default Language:
Auto Login Script:
Configured Authorization:
        sa_role (default ON)
        sso_role (default ON)
        oper_role (default ON)
        sybase_ts_role (default ON)
Locked: NO
Date of Last Password Change: Mar  2 2004  3:06AM
Password expiration interval: 0
Password expired: NO
Minimum password length: 6
Maximum failed logins: 0
Current failed login attempts:
Authenticate with: NONE
(return status = 0)
```

During the testing of the sp_maplogin procedure some curiosities were found. The first is referenced below.

```
1> sp_maplogin pam
2> go
Client authentication mapping updated.
(return status = 0)
1> sp_helpmaplogin
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            ANY                  ANY
 PAM            cruiz                sa
 PAM            cuerva               sa
(3 rows affected)
(return status = 0)
```

The ANY-ANY map has no sense. It maps any external user name with the same name in ASE. The same behavior is achieved when no PAM mappings are defined.

The second curiosity found in the syntax is the possibility to use wildcard characters for the external client name, like in the following example.

```
1> sp_maplogin pam, "*", sa
2> go
Client authentication mapping updated.
(return status = 0)
1> sp_helpmaplogin
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            ANY                  sa
 PAM            cruiz                sa

(2 rows affected)
(return status = 0)
1>
```

This command is obviously not recommended.

The `sp_maplogin` stored procedure allows only a mapping between an external entity and a `syslogins` login. It is not possible to define different mappings for the same user for different authentication mechanisms. Mappings to special external names, such as sa, guest, probe, sybmail or jstask will simply be ignored. Those special logins use only ASE authentication mechanism. Internally these mappings are stored in the sysattributes system table.

sp_helpmaplogin: This stored procedure lists all the available mappings created in the server. The sso_role is required to run this command. The syntax is:

```
sp_helpmaplogin [(authentication_mech|NULL),(client_user-
name| NULL)]
```

Where authentication_mech is the authentication mechanism used to filter the result set.
The possible values are:

- `PAM` means all the mappings using PAM.
- `LDAP` means all the mappings using LDAP.
- `ASE` means the ASE.
- `ANY` this means any authentication mechanism.
- A `NULL` value for this parameters is internally equivalent to ANY.

`client_username` is used to filter the result set on the client name, NULL means do not filter the result set on client user name.

This example shows `sp_helpmaplogin` in action with no parameters

```
1> sp_helpmaplogin
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            cruiz                sa
 PAM            cuerva               CREATE LOGIN

(2 rows affected)
(return status = 0)
1>
```

This second example allows filtering the output based upon the client user name.

```
1> sp_helpmaplogin null, cuerva
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            cuerva               CREATE LOGIN

(1 row affected)
(return status = 0)
1>
```

This example shows all the mappings for the PAM authentication mechanism

```
1> sp_helpmaplogin 'PAM'
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            cruiz                sa
 PAM            cuerva               CREATE LOGIN

(3 rows affected)
(return status = 0)
1>
```

NEW GLOBAL VARIABLE @@authmech: The authentication mechanism used by any connection is stored in a field in the pss called '*pauth*'. The command dbcc pss(0, spid) can be used to display it.

Any user can verify the authentication mechanism used reading the global variable @@authmech. There are two reasons why it exists:

- because we have a failover mechanism, and this is the way to know whether the external authentication defined in syslogins was used, or it failed and syslogins authentication was done
- because a login uses ANY authentication

The syntax is:

```
select @@authmech
```

PAM UA, CIS AND RPC

With PAM, a new mapping mechanism has been introduced through the sp_maplogin stored procedure. Up to the 12.5.1 release, ASE featured several mapping solutions. For instance the server allows the mapping between local server logins and remote server logins to enable RPCs invocation, using `sp_addremotelogin`. On the other hand, CIS provides the `sp_addexternlogin` stored procedure to define alternate logins account when communicating with remote servers.

It is interesting to describe the liaison between the PAM mapping and the existing one. To avoid a common pitfall, it must be stressed sp_maplogin using the login name option, allows the mapping of an external client name to a server **login** only. Another rule of thumb of PAM User Authentication is to keep the external client name when involved in external server communications.

To explain the PAM UA / CIS relationship, let's define the scenario where logins from server A need to connect using CIS to server B. Server A has a PAM mapping to the sa login, using the sentence sp_maplogin. Server A does not have initially any external login defined. Server A and Server B have been altered by the `sp_addserver` to reference each other. Finally Server B and Server A have the same password for the sa login.

Commands executed in Server A:

```
1> sp_helpmaplogin
2> go
 authentication client name          login name
 -------------- -------------------- --------------------
 PAM            cruiz                sa
 PAM            cuerva               sa

(2 rows affected)
(return status = 0)
1>
```

The ASE behavior, when connecting to the Server B using CIS command "`connect to`" statement (for instance), is two-fold. First Server A sends user cuerva, and not the sa login although cuerva is seen as the sa login after the PAM mapping. This is due to the PAM design; this can be however work-around thanks to the new trace flag **8017**. The second important thing to see is Server A always sends cuerva's password, and never sa's password. So for this command to succeed a pair (user cuerva, cuerva's password) must exist in Server B.

After issuing `dbcc traceon (8017)` on server A, the login record sent to Server B is changed from cuerva to the mapped login, here the sa login. However, the connection to Server B also fails, because cuerva's password is sent along with the sa identity. The thing to keep in mind is PAM always uses the external client's password. Can `sp_password` help? Using the sp_maplogin to map an external client user name to a server login does not trigger the creation of a record in the syslogins table. So once an external user has opened a session with its credential, the password is only kept in memory, and there is no way today to alter it. So to work, sa's password in Server B needs to match cuerva's in server A.

A functionality worth to be noticed is the possibility to create a CIS mapping with the `sp_addexternlogin` procedure. In this case the priority is always given to this mapping, and in case of a failure, PAM authentication kicks in as a fallback solution.

TROUBLESHOOTING PAM UA

In order to debug ASE / PAM interaction, the system trace flag **3635** can be used. To turn it on, the sso role is required. Once activated extra output is written to the error log, the password is however not displayed.

```
1>dbcc traceon (3635)
2>go
DBCC execution completed. If DBCC printed error messages,
contact a user with System Administrator (SA) role.
```

Another tip to troubleshoot PAM is to print the authentication mechanism used by a particular login. A new global variable @@authmech is defined.

```
$>isql -Upamuser -Ppampwd -Ss
1> select @@authmech
2> go

 ------------------------------
 pam

(1 row affected)
1>
```

ASE TRACE FLAGS LIST

The trace flags listed in Table 4 control the PAM UA implementation in ASE.

TRACE FLAGS	DESCRIPTION
3635	Allows PAM UA debugging.
3636	Enables PAM account management.
8017	Avoids Client External Name switching in a RPC call.

TABLE 4: PAM UA TRACE FLAG LIST.

PAM EXAMPLE

So far, the examples in this paper have mainly used the pam_unix authentication service module. This section explains how a custom authentication module can be written, and how ASE can utilize it to externalize login access.

The purpose of this example is to authenticate users referenced by a simple text file living in the file system, hard coded to be "/tmp/pam_userauth.map". The fields in this file are semi-colon delimited, and follow this structure:

```
login_name ; login_password ; Valid Days to Connect ;
Expiration Date, Minimum Password Length
```

An instance of this file could be:

```
cruiz ; mypasswd ; mon-fri ; E4-25-2004 12:00:00, L6
cuerva ; cuerva_pwd; mon-sat ; E12-12-2004 12:00:00, L6
```

Where the login cruiz is allowed to connect from Monday to Friday. His password expires April 25th 2004 and the password has a minimum of 6 characters. The login cuerva can connect from Monday to Saturday, the password expires in December 12th, and the password has also at least 6 characters.

This sample being used has service module, includes the `pam_appl.h` and `pam_modules.h` header files as seen in the first section of the article. It only implements the authentication and the account management services, so the interesting code is located in the `pam_sm_authenticate` and `pam_sm_acct_mgmt` functions.

Once the dynamic library is generated, ensure the "`so`" file is owned by root and has the 755 permission; update the pam configuration with the following entries:

```
$>cat /etc/pam.conf
#top of file deleted.
ase   auth  required    /any_path/pam_authtest.so
ase   acct  required    /any_path/pam_authtest.so
```

The Solaris 32 bits as well as the full code for this PAM module are given in Appendix C.

CONCLUSION

With support for Pluggable Authentication Module, Sybase brings to its customer base an extendible, open and powerful authentication mechanism. Using this technique, any kind of security check can be easily plugged into the ASE runtime engine. The PAM design permits those extensions to be done in a flexible and elegant way, reducing the overall costs of managing the ASE server.

REFERENCE DOCUMENTS

- [PAMWP] Making Login Services Independent of Authentication Technologies, a whitepaper by Vipin Samar and Charlie Lai, SunSoft Inc.
- [ASEFS] PAM User Authentication Functional Specification, by Carlos Ruiz, Sybase Inc.
- [ASEDS] PAM User Authentication Design Specification, byb Carlos Ruiz, Sybase Inc.
- [PAMADM] PAM Administration, Sun Microsystems Computer Company.
- [PAMMAN] Online Solaris 2.8 PAM manuals.
- [PAMSAMP] Solaris PAM sample.

APPENDIX A

Generic PAM Configuration File for SOLARIS

This is a working example of a pam.conf file in a Solaris machine:

```
$ pwd
/etc
$ cat pam.conf
#
# PAM configuration
#
# Authentication management
#
login auth required
/usr/lib/security/$ISA/pam_unix.so.1
login auth required
/usr/lib/security/$ISA/pam_dial_auth.so.1
#
rlogin auth sufficient
/usr/lib/security/$ISA/pam_rhosts_auth.so.1
rlogin auth required
/usr/lib/security/$ISA/pam_unix.so.1
#
dtlogin auth required
/usr/lib/security/$ISA/pam_unix.so.1
#
rsh   auth required
/usr/lib/security/$ISA/pam_rhosts_auth.so.1
other auth required
/usr/lib/security/$ISA/pam_unix.so.1
#
```

(continued on next page)


```
# Account management
#
login account requisite
/usr/lib/security/$ISA/pam_roles.so.1
login account required  /usr/lib/security/$ISA/pam_proj-
ects.so.1
login account required
/usr/lib/security/$ISA/pam_unix.so.1
#
dtlogin account requisite
/usr/lib/security/$ISA/pam_roles.so.1
dtlogin account required/usr/lib/security/$ISA/pam_proj-
ects.so.1
dtlogin account required
/usr/lib/security/$ISA/pam_unix.so.1
#
other account requisite
/usr/lib/security/$ISA/pam_roles.so.1
other account required  /usr/lib/security/$ISA/pam_proj-
ects.so.1
other account required
/usr/lib/security/$ISA/pam_unix.so.1
#
# Session management
#
other session required
/usr/lib/security/$ISA/pam_unix.so.1
#
# Password management
#
other password required
/usr/lib/security/$ISA/pam_unix.so.1
dtsession auth required
/usr/lib/security/$ISA/pam_unix.so.1
$
```

APPENDIX B

Generic PAM Configuration File for LINUX

```
$ pwd
/etc/pam.d
$ cat login
#%PAM-1.0
#auth       required    /lib/security/pam_securetty.so
auth        required    /lib/security/pam_stack.so serv-
ice=system-auth
auth        required    /lib/security/pam_nologin.so
account     required    /lib/security/pam_stack.so serv-
ice=system-auth
password    required    /lib/security/pam_stack.so serv-
ice=system-auth
session     required    /lib/security/pam_stack.so serv-
ice=system-auth
session     optional    /lib/security/pam_console.so
$
```

APPENDIX C

Makefile for Solaris 32 Bits to Build Sample Module

```
srcdir          = .

### Compiler and link options
CC              = /opt/SUNWspro6.2/bin/cc
CPPFLAGS        = -I${srcdir}

CFLAGS          = -O -mt
LDFLAGS         = -Bdynamic
SHLIB_CFLAGS    = -Kpic
SHLIB_LDFLAGS   = -dy -G
LIBS            = -lpam -ldl

### Makefile rules - no user-servicable parts below
```

(continued on next page)

```
HDRS    = config.h

ALL     = pam_authtest.so.1

all: ${ALL}
```

```
pam_authtest.so.1: pam_authtest.o
        ${CC} ${SHLIB_CFLAGS} ${SHLIB_LDFLAGS} ${LDFLAGS} -
o $@ pam_authtest.o ${LIBS}

.c.o: ${HDRS}
        ${CC} ${CPPFLAGS} ${SHLIB_CFLAGS} ${CFLAGS} -c -o
$@ $<

clean:
        rm -f *.o core
```

Code for a Sample Authentication Module

```
//code has been deleted … see full code for running exam-
ple.

int pam_sm_authenticate(pam_handle_t *pamh,int flags,int
argc,const char *argv[])

{
      char *user, *pass, *mapfile = NULL;
      struct pam_message prompt = { PAM_PROMPT_ECHO_OFF,
NULL };
      struct pam_message *prompt1 = &prompt;
      struct pam_message **msg = &prompt1;
      struct pam_conv *conv;
      char buf[PAM_USER_BUFSIZE];
      struct pam_response *response = NULL;
      int i, map_flags = 0;
      int ret;

#if DEBUG
      printf("==> pam_sm_authenticate(pamh=0x%lx,
flags=%d)\n",
             pamh, flags);
#endif

      for (i = 0; i < argc; i++)
      {
#if DEBUG
            printf("    pam_sm_authenticate(): argv[%d] =
\"%s\"\n",
                   i, argv[i]);
#endif

            if (strncmp(argv[i], "mapfile=", 8) == 0)
            {
                  mapfile = (char *)argv[i] + 8;
                  continue;
            }

            printf("pam_sm_authenticate: unknown argument
'%s'",argv[i]);
            syslog(LOG_AUTHPRIV | LOG_ERR,
                   "pam_sm_authenticate: unknown "
```

```
                    "argument \"%s\"", argv[i]);
      }

      if (pam_get_user(pamh, (char **)&user, "login: ") !=
PAM_SUCCESS)
      {
            printf("pam_sm_authenticate: pam_get_user
failed");
            syslog(LOG_AUTHPRIV | LOG_ERR,
                  "pam_sm_authenticate: pam_get_user
failed");
            return PAM_USER_UNKNOWN;
      }

      snprintf(buf, sizeof(buf), "%s's Password: ", user);
      prompt.msg = buf;
      if (pam_get_item(pamh, PAM_CONV, (void **)&conv) !=
PAM_SUCCESS
          || conv->conv(1, (struct pam_message **)msg,
                  &response, conv->appdata_ptr) !=
PAM_SUCCESS
          || response == NULL
          || response[0].resp == NULL)
      {
            printf("pam_sm_authenticate: cannot get user
response");
            syslog(LOG_AUTHPRIV | LOG_ERR,
                  "pam_sm_authenticate: cannot get user
response");
            return PAM_AUTHINFO_UNAVAIL;
      }
      strlcpy(buf, response[0].resp, sizeof(buf));
      free(response[0].resp);
      pass = buf;


      ret = check_passwd(mapfile, user, pass);

#if DEBUG
      printf("<== pam_sm_authenticate(): %s\n",
            (ret == PAM_SUCCESS) ? "succeeded" : "failed");
#endif

      return ret;
}

int pam_sm_acct_mgmt(pam_handle_t * pamh, int flags, int
argc, const char **argv)
{
      int i;
      int ret;
      char *mapfile, *user;
      mapfile = user = NULL;

      for (i = 0; i < argc; i++)
      {
```

(continued on back cover)

```
            if (strncmp(argv[i], "mapfile=", 8) == 0)
            {
                  mapfile = (char *)argv[i] + 8;
                  continue;
            }

            printf("pam_sm_authenticate: unknown argument
'%s'",argv[i]);
            syslog(LOG_AUTHPRIV | LOG_ERR,
                   "pam_sm_authenticate: unknown "
                   "argument \"%s\"", argv[i]);
      }

      if(pam_get_item( pamh, PAM_USER, (void*) &user ) !=
PAM_SUCCESS)
      {
            printf("pam_sm_authenticate: pam_get_item
failed");
            syslog(LOG_AUTHPRIV | LOG_ERR,
                   "pam_sm_authenticate: pam_get_user
failed");
            return PAM_USER_UNKNOWN;
      }

      if (!mapfile)
      {
            mapfile = getenv("PAM_MAP");
            if (!mapfile)
            {
                  mapfile = MAPFILE;
            }
      }

      ret = check_acct(mapfile, user);
      if (ret != PAM_SUCCESS)
      {
            syslog(LOG_AUTHPRIV | LOG_ERR,
                   "could not identify user (from
check_acct(%s))"
                   ,user);
            return ret;
      }

      return PAM_SUCCESS;
}
```

SYBASE®

Sybase Incorporated
Worldwide Headquarters
One Sybase Drive
Dublin CA, 94568 USA
T 1.800.8.SYBASE
F 1.925.234.5678
www.sybase.com