Documentation for mod_authnz_ibmdb2 db2-hash-routines scripts

Helmut K. C. Tessarek

8th January, 2019

mod_authnz_ibmdb2 is an Apache authentication module using IBM® DB2® as the backend database for storing user and group information. The module supports several encryption methods.

http://tessus.github.io/mod_authnz_ibmdb2

Date: 2019-01-08 14:55:52 -0500 Id: f43f87d

Contents

1. mod_authnz_ibmdb2

1.1. Building mod_authnz_ibmdb2 from a cloned repository

If you want to build the module from a cloned repository, autotools (autoconf, autoheader, automake) are required. The script `autogen.sh` needs autotools to create `configure` from scratch.

```
git clone https://github.com/tessus/mod_authnz_ibmdb2.git

cd mod_authnz_ibmdb2
./autogen.sh
./configure
make install
```

The `configure` script has a few options which are described in detail in subsection 1.3. Run `make install` as root or with `sudo`.

1.2. Building mod_authnz_ibmdb2 from a tarball

Download the latest tarball from:
`https://github.com/tessus/mod_authnz_ibmdb2/releases/latest`

```
tar -xzf mod_authnz_ibmdb2-X.Y.Z.tar.gz

cd mod_authnz_ibmdb2
./configure
make install
```

The `configure` script has a few options which are described in detail in subsection 1.3. Run `make install` as root or with `sudo`.

1.3. Configure options and details on building mod_authnz_ibmdb2

If you run ./configure as a user with a DB2 environment and apxs is in the path, there's nothing else to worry about.

However, you might have 2 versions of Apache installed and want to choose for which one the module is for, or you don't have the DB2 environment initialized.

There are options to specify the location of the DB2 home and the apxs utility:

```
--with-apxs=FILE                FILE is the pathname of the Apache tool

--with-IBM_DB2=DIR              DIR is the IBM DB2 instance or home
```

```
                                directory where the DB2 application
                                development headers and libraries are
                                located
```

By default man pages are installed automatically, but you can change this by using the following flag:

```
--disable-man-pages
```

During install the module can also be activated in the Apache config file httpd.conf:

```
--enable-activation
```

After successful configuration the module can be installed with:

```
make install
```

Be aware that this process needs root privileges.

If you decide to add the module manually, the following directive in your httpd.conf will do the trick:

```
LoadModule authnz_ibmdb2_module modules/mod_authnz_ibmdb2.so
```

1.4. Additional Apache configuration

The DB2 environment has to be set, before starting Apache. There are several ways to accomplish that:

- source the DB2 environment manually
- set the DB2 environment in the `apachectl` script
- set the DB2 environment in the `init.d` or `systemd` script

In fact, the only environment variable really necessary is `DB2INSTANCE`.

1.5. Description of the module

mod_authnz_ibmdb2 is an Apache authentication module using IBM DB2 as the backend database for storing user and group information. The module is designed for Apache 2.2.x and later and is based on the new authentication/authorization framework.

Here is a list of the new directives[1] that come with the module:

`AuthIBMDB2Database`	database name
`AuthIBMDB2Hostname`	database server hostname for uncataloged databases
`AuthIBMDB2Portnumber`	database instance port (default: 50000)
`AuthIBMDB2User`	user for connecting to the DB2 database
`AuthIBMDB2Password`	password for connecting to the DB2 database
`AuthIBMDB2UserTable`	name of the user table
`AuthIBMDB2GroupTable`	name of the group table
`AuthIBMDB2NameField`	name of the user column in the table (default: `username`)
`AuthIBMDB2GroupField`	name of the group column in the table (default: `groupname`)
`AuthIBMDB2PasswordField`	name of the password column in the table (default: `password`)
`AuthIBMDB2CryptedPasswords`	passwords are stored encrypted (default: `yes`)
`AuthIBMDB2KeepAlive`	connection kept open across requests (default: `yes`)
`AuthIBMDB2Authoritative`	lookup is authoritative (default: `yes`)
`AuthIBMDB2NoPasswd`	just check, if user is in usertable (default: `no`)
`AuthIBMDB2UserCondition`	restrict result set
`AuthIBMDB2GroupCondition`	restrict result set
`AuthIBMDB2UserProc`	stored procedure[2] for user authentication
`AuthIBMDB2GroupProc`	stored procedure[2] for group authentication
`AuthIBMDB2Caching`	user credentials are cached (default: `off`)
`AuthIBMDB2GroupCaching`	group information is cached (default: `off`)
`AuthIBMDB2CacheFile`	path to cache file (default: `/tmp/auth_cred_cache`)
`AuthIBMDB2CacheLifetime`	cache lifetime in seconds (default: `300`)

[1] see Appendix A
[2] see Appendix C

If `AuthIBMDB2Authoritative` is `Off`, then iff the user is not found in the database, let other authentication modules try to find the user. Default is `On`.

If `AuthIBMDB2KeepAlive` is `On`, then the server instance will keep the IBM DB2 server connection open. In this case, the first time the connection is made, it will use the current set of Host, User, and Password settings. Subsequent changes to these will not affect this server, so they should all be the same in every htaccess file. If you need to access multiple IBM DB2 servers for this authorization scheme from the same web server, then keep this setting `Off` – this will open a new connection to the server every time it needs one. The values of the database and various tables and fields are always used from the current `.htaccess` file settings.

If `AuthIBMDB2NoPasswd` is `On`, then any password the user enters will be accepted as long as the user exists in the database.
Setting this also overrides the setting for `AuthIBMDB2PasswordField` to be the same as `AuthIBMDB2NameField` (so that the SQL statements still work when there is no password at all in the database, and to remain backward-compatible with the default values for these fields.)

For groups, we use the same `AuthIBMDB2NameField` as above for the user ID, and `AuthIBMDB2GroupField` to specify the group name.
`AuthIBMDB2GroupTable` specifies the table to use to get the group info. It defaults to the value of `AuthIBMDB2UserTable`. If you are not using groups, you do not need a `groupname` field in your database, obviously.

The optional directives `AuthIBMDB2UserCondition` and
`AuthIBMDB2GroupCondition` can be used to restrict queries made against the User and Group tables. The value for each of these should be a string that you want added to the end of the where-clause when querying each table. For example, if your user table has an `active` integer field and you only want users to be able to login, if that field is 1, you could use a directive like this:
`AuthIBMDB2UserCondition active=1`

If `AuthIBMDB2UserProc` is set, the named stored procedure[3] is responsible for returning the password of the user in question to the module. It must return exactly one value and row - the password. If set, `AuthIBMDB2UserTable`, `AuthIBMDB2NameField`, `AuthIBMDB2PasswordField`, `AuthIBMDB2UserCondition` are ignored.
If `AuthIBMDB2NoPasswd` is `On`, then the username has to be returned instead of the password. The stored procedure must have the following parameter format:

```
CREATE PROCEDURE user_procedure_name ( IN VARCHAR, OUT VARCHAR )
```

If `AuthIBMDB2GroupProc` is set, the named stored procedure[4] is responsible for returning the groups the user in question belongs to. It must return an open cursor to the result set. If set, `AuthIBMDB2GroupTable`, `AuthIBMDB2NameField`, `AuthIBMDB2GroupField`, `AuthIBMDB2GroupCondition` are ignored. The stored procedure must have the following parameter format:

```
CREATE PROCEDURE group_procedure_name ( IN VARCHAR )
```

If `AuthIBMDB2Caching` ist set to `On`, the user credentials are cached in a file defined in `AuthIBMDB2CacheFile` and expires after `AuthIBMDB2CacheLifetime` seconds.

If `AuthIBMDB2GroupCaching` ist set to `On`, the group information is cached in a cache file that is named like the file specified in `AuthIBMDB2CacheFile` but with the extension `.grp`. The cache expires after `AuthIBMDB2CacheLifetime` seconds.

[3] see Appendix C.1
[4] see Appendix C.2

1.6. Examples

First create the two tables within DB2:

```
CREATE TABLE WEB.USERS (
    USERNAME VARCHAR(40) NOT NULL,
    PASSWORD VARCHAR(40) );

ALTER TABLE WEB.USERS
    ADD PRIMARY KEY (USERNAME);

CREATE TABLE WEB.GROUPS (
    USERNAME  VARCHAR(40) NOT NULL,
    GROUPNAME VARCHAR(40) NOT NULL );

ALTER TABLE WEB.GROUPS
    ADD PRIMARY KEY (USERNAME, GROUPNAME);
```

Then you will have to insert records into the two tables:

`bcrypt` is a User Defined Function that is explained in the db2-hash-routines part of this documentation.

```
INSERT INTO WEB.USERS (username, password)
    VALUES ('test', bcrypt('testpwd'));
INSERT INTO WEB.GROUPS (username, groupname)
    VALUES ('test', 'admin');
```

Then add the following lines to your httpd.conf:

```
<Directory "/var/www/my_test_dir">
    AuthName                    "DB2 Authentication"
    AuthType                    Basic
    AuthBasicProvider           ibmdb2

    AuthIBMDB2User              db2inst1
    AuthIBMDB2Password          ibmdb2
    AuthIBMDB2Database          auth
    AuthIBMDB2UserTable         web.users
    AuthIBMDB2NameField         username
    AuthIBMDB2PasswordField     passwd

    AuthIBMDB2CryptedPasswords  On
    AuthIBMDB2KeepAlive         On
    AuthIBMDB2Authoritative     On
    AuthIBMDB2NoPasswd          Off
```

```
    AuthIBMDB2GroupTable        web.groups
    AuthIBMDB2GroupField        groupname

    require                     group admin
    AllowOverride               None
</Directory>
```

If you want to use stored procedures and caching, the directives would look like this:

```
<Directory "/var/www/my_test_dir">
    AuthName                     "DB2 Authentication"
    AuthType                     Basic
    AuthBasicProvider            ibmdb2

    AuthIBMDB2User               db2inst1
    AuthIBMDB2Password           ibmdb2
    AuthIBMDB2Database           auth
    AuthIBMDB2UserProc           user_sp
    AuthIBMDB2GroupProc          group_sp

    AuthIBMDB2Caching            On
    AuthIBMDB2GroupCaching       On

    require                      group admin
    AllowOverride                None
</Directory>
```

2. db2-hash-routines

2.1. Building the library and registering the UDFs and SPs

Login as the instance user and run the script

Linux and AIX	`./makertn`
Win32	`makertn.bat`

The `makertn` script detects the DB2 instance directory and locates `apr-1-config` and `apu-1-config` automatically. If for some reason the script cannot set either one of the necessary variables, they have to be set manually. Uncomment and change the following variables in the `makertn` script.

```
DB2PATH=
APRPATH=
APUPATH=
```

Set `DB2PATH` to the directory where DB2 is accessed. This is usually the instance home directory.
Set APRPATH to where apr-1-config is located.
Set APUPATH to where apu-1-config is located.

The UDFs and SPs are written in ANSI C and should compile on all platforms.

The only requirements are APR and APR-util. You can get APR and APR-util at `http://apr.apache.org/`

To register the UDFs and SPs, connect to your database and run the script:

```
db2 -td@ -f register.ddl
```

2.2. Description of the UDFs and SPs

This library delivers the following routines[5]:

`bcrypt`
`sha256_hex`
`sha1_hex`
`sha256`
`sha512`
`php_md5`
`apr_md5`
`apr_crypt`
`apr_sha1`
`apr_sha256`
`validate_pw`

The `php_md5` routine is compatible to the PHP md5 function.
The `sha256_hex` routine returns a sha256 64-character hexadecimal hash.
The `sha1_hex` routine returns a sha1 40-character hexadecimal hash.
The `apr_md5`, `apr_crypt`, `apr_sha1` and `bcrypt` routines are compatible to the functions used in Apache's `htpasswd` utility.
The `apr_sha256` routine returns the identifier {`SHA256`} plus the base64 encoded sha256 hash.
The `sha256` and `sha512` functions return glib2's crypt hashes (if supported).

`validate_pw` can be used to validate a password against a hash.

On systems with glibc2, the `validate_pw` routine will also validate hashes of the form `$id$salt$encrypted`. The following values of `id` are supported:

ID	Method
1	MD5
2a	Blowfish (not in mainline glibc; added in some Linux distributions)
5	SHA-256 (since glibc 2.7)
6	SHA-512 (since glibc 2.7)

Note: In win32 environments `apr_crypt` returns the output of `bcrypt`, if available. If `bcrypt` is not available, the output of `apr_md5` is returned.

[5] see Appendix B for a reference of the UDFs and SPs

3. scripts

3.1. Description of the scripts

There are four scripts to import the users and groups from already existing user and/or group files into DB2. They are written in php, so you should have the php cli binary in your `/usr/local/bin` directory.

The script `sync_pwds` is for syncing the system users with a table within your DB2 database.

You have to change the settings in the `config.php` file for your environment.

Here is a table of the relation between the directives for the *mod_authnz_ibmdb2* module and the settings in the `config.php` file:

config.php	module directive
`$dbname = "auth";`	AuthIBMDB2Database
`$dbuser = "db2inst1";`	AuthIBMDB2User
`$dbpwd = "db2inst1";`	AuthIBMDB2Password
`$usertable = "users";`	AuthIBMDB2UserTable
`$grouptable = "groups";`	AuthIBMDB2GroupTable
`$namefield = "username";`	AuthIBMDB2NameField
`$passwordfield = "password";`	AuthIBMDB2PasswordField
`$groupfield = "groupname";`	AuthIBMDB2GroupField

Attention: The scripts were developed on Linux, therefore they will only work on systems where the `/etc/passwd`, the `/etc/shadow`, the `/etc/group` and the `/etc/gshadow` are in the same format as on Linux systems.

Note: `user_imp` and `group_imp` will work on all systems, because these scripts don't rely on above mentioned files.

3.2. Examples

If the settings in the `config.php` are as above and you execute the `./user_etc_imp` script following happens:

All users (except system users like root or mail) are imported from the linux box into the table `users` in the database `auth`. The table `users` has `username` as the columnname

for the users and `password` as the columnname for the passwords.

To import users from an existing htpasswd users file, just run the script

```
./user_imp <path-to-userfile>
```

To import group information from an existing Apache group file, run the script

```
./group_imp <path-to-groupfile>
```

4. GIT access

The git repositories can be cloned from github with the following instruction set:

```
git clone https://github.com/tessus/mod_authnz_ibmdb2.git
git clone https://github.com/tessus/mod_auth_ibmdb2.git
git clone https://github.com/tessus/db2-hash-routines.git
```

You can also browse the repositories via the web:

mod_authnz_ibmdb2	`https://github.com/tessus/mod_authnz_ibmdb2`
mod_auth_ibmdb2	`https://github.com/tessus/mod_auth_ibmdb2`
db2-hash-routines	`https://github.com/tessus/db2-hash-routines`

5. FAQ

Q: IBM's Websphere plugin and mod_auth(nz)_ibmdb2 seem to break each other. What can I do?
A: mod_auth(nz)_ibmdb2 has to be loaded after the Websphere plugin.

Q: Which versions of DB2 are supported?
A: All DB2 versions currently supported by IBM. I've tested the module with all versions since DB2 UDB v7.x, but older versions should work as well.

Q: What is the difference between mod_auth_ibmdb2 and mod_authnz_ibmdb2?
A: mod_authnz_ibmdb2 is based on the new authentication backend provider scheme of Apache 2.2. This module will only work for Apache 2.2 and later. mod_auth_ibmdb2 works for Apache 2.0.x and 1.x.

Q: What platforms are supported?
A: All POSIX platforms. I've compiled and tested the module on Linux and IBM AIX. Since the modules are using the APR libraries now, they can be compiled on Windows as well.

Q: Why isn't there a binary release for .....?
A: I don't have a development environment for every operating system. Furthermore I don't think that binary releases make sense for Unix style operating systems.

Q: What is the package db2-hash-routines for?
A: This package contains User Defined Functions and Stored Procedures to generate and validate hashes in DB2.

Q: How do I get support?
A: Please submit a ticket at the Issues Tracker (hosted by github).

6. Links

6.1. Official mod_auth(nz)_ibmdb2 website

http://tessus.github.io/mod_authnz_ibmdb2

6.2. Support Requests

https://github.com/tessus/mod_authnz_ibmdb2/issues

6.3. PHP scripts to import users/groups

https://github.com/tessus/usr-grp-import-scripts/archive/master.zip

6.4. developerWorks article

mod_auth_ibmdb2: A novel authentication method for Apache
http://www.ibm.com/developerworks/db2/library/techarticle/dm-0407tessarek/

6.5. GIT repositories

https://github.com/tessus/mod_authnz_ibmdb2
https://github.com/tessus/mod_auth_ibmdb2
https://github.com/tessus/db2-hash-routines
https://github.com/tessus/usr-grp-import-scripts

A. directives and default values

directive	default value
AuthIBMDB2Database	–
AuthIBMDB2Hostname	–
AuthIBMDB2Portnumber	50000
AuthIBMDB2User	–
AuthIBMDB2Password	–
AuthIBMDB2UserTable	–
AuthIBMDB2GroupTable	–
AuthIBMDB2NameField	username
AuthIBMDB2GroupField	groupname
AuthIBMDB2PasswordField	password
AuthIBMDB2CryptedPasswords	yes
AuthIBMDB2KeepAlive	yes
AuthIBMDB2Authoritative	yes
AuthIBMDB2NoPasswd	no
AuthIBMDB2UserCondition	–
AuthIBMDB2GroupCondition	–
AuthIBMDB2UserProc	–
AuthIBMDB2GroupProc	–
AuthIBMDB2Caching	off
AuthIBMDB2GroupCaching	off
AuthIBMDB2CacheFile	/tmp/auth_cred_cache
AuthIBMDB2CacheLifetime	300

B. UDF and SP reference

B.1. bcrypt

```
>>-BCRYPT--(--expression--)----------------------------------><

>>-BCRYPT--(--expression--,--hash--)--------------------------><
```

bcrypt algorithm. The `bcrypt` routine is compatible to the function used in Apache's `htpasswd` utility.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(60). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', bcrypt('testpwd'))

2)
    SELECT bcrypt( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    ------------------------------------------------------------
    $2y$05$2jb66aPElSkNLT1t8e6dQepuCY2BP3JnYUh0xeV9r1PEoOGyOLkym

      1 record(s) selected.

3)
    CALL bcrypt('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : $2y$05$WYSu1X6PVA0Ra.aPSjrdv.S6hOp.AYSnNRT521rmLRjD4Mj9
    UY6ve

      Return Status = 0
```

B.2. sha256_hex

```
>>-SHA256_HEX--(--expression--)--------------------------------><

>>-SHA256_HEX--(--expression--,--hash--)----------------------><
```

SHA256 algorithm. The `sha256_hex` routine returns a 64-character hexadecimal hash.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(64). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', sha256_hex('testpwd'))

2)
    SELECT sha256_hex('testpwd') FROM SYSIBM.SYSDUMMY1

    1
    ----------------------------------------------------------------
    a85b6a20813c31a8b1b3f3618da796271c9aa293b3f809873053b21aec501087

      1 record(s) selected.

3)
    CALL sha256_hex('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : a85b6a20813c31a8b1b3f3618da796271c9aa293b3f809873053b21
    aec501087

      Return Status = 0
```

B.3. sha1_hex

```
>>-SHA1_HEX--(--expression--)---------------------------------><

>>-SHA1_HEX--(--expression--,--hash--)------------------------><
```

SHA1 algorithm. The `sha1_hex` routine returns a 40-character hexadecimal hash.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(40). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', sha1_hex('testpwd'))

2)
    SELECT sha1_hex('testpwd') FROM SYSIBM.SYSDUMMY1

    1
    ----------------------------------------
    98ef0758e6aac6f9a9e1197548c8190b72c9581d

      1 record(s) selected.

3)
    CALL sha1_hex('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : 98ef0758e6aac6f9a9e1197548c8190b72c9581d

      Return Status = 0
```

B.4. sha256

```
>>-SHA256--(--expression--+---------+--)----------------------><
                         '-,--salt-'

>>-SHA256--(--expression--+---------+--,--hash--)--------------><
                         '-,--salt-'
```

SHA256 algorithm. The `sha256` routine returns a glibc2's crypt hash. If the system's crypt does not support sha-256, an `SQLSTATE 39702` is returned.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.
An optional salt can be specified, which must be a eight-character string chosen from the set [a-z-Z0-9./]. If the salt is not exactly eight characters long, an `SQLSTATE 39703` is returned. If the salt contains invalid characters, an `SQLSTATE 39704` is returned.

The result of the function is CHAR(55). The result can be null; if one of the arguments is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', sha256('testpwd'))

2)
    SELECT sha256( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    ------------------------------------------------------
    $5$S.LqPR7Z$273zPncMdmJ0dE1WdLldWVBmaHSDUDl8/tW8At8Hc0A

      1 record(s) selected.

3)
    CALL sha256('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : $5$vSDCZr2d$rfh.aDopE5l3lm26AwwcIYnuVdV7/9QBACWukqYyV3/
```

```
      Return Status = 0

4)
    SELECT sha256('testpwd', '12345678') FROM SYSIBM.SYSDUMMY1

    1
    ----------------------------------------------------------
    $5$12345678$.oVAnOr/.FK8fYNiFPvoXPQvEOT9Calecygw6K9wIb9

      1 record(s) selected.

5)
    CALL sha256('testpwd', '12345678', ?)

    Value of output parameters
    --------------------------
    Parameter Name  : HASH
    Parameter Value : $5$12345678$.oVAnOr/.FK8fYNiFPvoXPQvEOT9Calecygw6K9wIb9

    Return Status = 0
```

B.5. sha512

```
>>-SHA512--(--expression--+---------+--)----------------------><
                         '-,--salt-'

>>-SHA512--(--expression--+---------+--,--hash--)--------------><
                         '-,--salt-'
```

SHA512 algorithm. The `sha512` routine returns a glibc2's crypt hash. If the system's crypt does not support sha-512, an `SQLSTATE 39702` is returned.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.
An optional salt can be specified, which must be a eight-character string chosen from the set [a-z-Z0-9./]. If the salt is not exactly eight characters long, an `SQLSTATE 39703` is returned. If the salt contains invalid characters, an `SQLSTATE 39704` is returned.

The result of the function is CHAR(98). The result can be null; if one of the arguments is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', sha512('testpwd'))

2)
    SELECT sha512( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    ------------------------------------------------------------------------
    ----------------------
    $6$cD33haq7$dl.RqEaLamlesTPVzSIQr4N1MY3BsVZ76VS8qNte0IOIWO2XorMg8U797KKOFGm
    X8dJhT3WuF6p17HmvvoQ6Q/

      1 record(s) selected.

3)
    CALL sha512('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
```

```
  Parameter Value : $6$1W.m9JN1$Dh.VPl7vy.igGaeDUdDWw6ZlDOxufwDWmOukpOYknPt
djxiSM2yzWBkzHffalb/2axNHPqEi9UUzXUbSm4LGa/

  Return Status = 0

4)
SELECT sha512('testpwd', '12345678') FROM SYSIBM.SYSDUMMY1

1
--------------------------------------------------------------------------
------------------------
$6$12345678$tlHrypdWTz6FqubBpgL/ePlxr4lZuQ8OK1zfV6zWUmGJSz.5kGWwQGjg69Qm1Bm
3.DvILruqA61o3EHsxSoko1

  1 record(s) selected.

5)
CALL sha512('testpwd', '12345678', ?)

  Value of output parameters
  --------------------------
  Parameter Name  : HASH
  Parameter Value : $6$12345678$tlHrypdWTz6FqubBpgL/ePlxr4lZuQ8OK1zfV6zWUmG
JSz.5kGWwQGjg69Qm1Bm3.DvILruqA61o3EHsxSoko1

  Return Status = 0
```

B.6. php_md5

```
>>-PHP_MD5--(--expression--)----------------------------------><

>>-PHP_MD5--(--expression--,--hash--)-------------------------><
```

MD5 hash. The `php_md5` routine is compatible to the PHP md5 function.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(32). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', php_md5('testpwd'))

2)
    SELECT php_md5( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    --------------------------------
    342df5b036b2f28184536820af6d1caf

      1 record(s) selected.

3)
    CALL php_md5('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : 342df5b036b2f28184536820af6d1caf

      Return Status = 0
```

B.7. apr_md5

```
>>-APR_MD5--(--expression--)----------------------------------><

>>-APR_MD5--(--expression--,--hash--)-------------------------><
```

Seeded MD5 hash. The `apr_md5` routine is compatible to the function used in Apache's `htpasswd` utility.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(37). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', apr_md5('testpwd'))

2)
    SELECT apr_md5( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    ------------------------------------
    $apr1$GfVmOTyJ$n7F1Vkwl/kX8MLgTJq1lp1

      1 record(s) selected.

3)
    CALL apr_md5('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : $apr1$GfVmOTyJ$n7F1Vkwl/kX8MLgTJq1lp1

      Return Status = 0
```

B.8. apr_crypt

```
>>-APR_CRYPT--(--expression--)--------------------------------><

>>-APR_CRYPT--(--expression--,--hash--)-----------------------><
```

Unix crypt. The `apr_crypt` routine is compatible to the function used in Apache's `htpasswd` utility.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(13). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', apr_crypt('testpwd'))

2)
    SELECT apr_crypt( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    -------------
    cqs7uOvz8KBlk

      1 record(s) selected.

3)
    CALL apr_crypt('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : cqs7uOvz8KBlk

      Return Status = 0
```

B.9. apr_sha1

```
>>-APR_SHA1--(--expression--)---------------------------------><

>>-APR_SHA1--(--expression--,--hash--)------------------------><
```

SHA1 algorithm. The `apr_sha1` routine is compatible to the function used in Apache's `htpasswd` utility.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(33). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', apr_sha1('testpwd'))

2)
    SELECT apr_sha1( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    ---------------------------------
    {SHA}mO8HWOaqxvmp4Rl1SMgZC3LJWBO=

      1 record(s) selected.

3)
    CALL apr_sha1('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : {SHA}mO8HWOaqxvmp4Rl1SMgZC3LJWBO=

      Return Status = 0
```

B.10. apr_sha256

```
>>-APR_SHA256--(--expression--)-------------------------------><

>>-APR_SHA256--(--expression--,--hash--)----------------------><
```

SHA256 algorithm. The `apr_sha256` routine returns the identifier {`SHA256`} plus the base64 encoded sha256 hash.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4096 bytes.

The result of the function is CHAR(52). The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    INSERT INTO USERS (username, password)
       VALUES ('test', apr_sha256('testpwd'))

2)
    SELECT apr_sha256( 'testpwd' ) FROM SYSIBM.SYSDUMMY1

    1
    ----------------------------------------------------
    {SHA256}qFtqIIE8Maixs/NhjaeWJxyaopOz+AmHMFOyGuxQEIc=

      1 record(s) selected.

3)
    CALL apr_sha256('testpwd', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : HASH
      Parameter Value : {SHA256}qFtqIIE8Maixs/NhjaeWJxyaopOz+AmHMFOyGuxQEIc=

      Return Status = 0
```

B.11. validate_pw

```
>>-VALIDATE_PW--(--password--,--hash--)------------------------><

>>-VALIDATE_PW--(--password--,--hash--,--is_valid--)-----------><
```

This routine can be used to validate a password against a hash.

The two input arguments can be character strings that are either a CHAR or VARCHAR not exceeding 4096 bytes (password) and 120 bytes (hash). The second parameter (hash) must not be empty, otherwise an `SQLSTATE 39701` is returned.

The result of the routine is an INTEGER. If the password is valid, 1 is returned. If the password is not valid, 0 is returned. The result can be null; if the argument is null, the result is the null value.

Examples:

```
1)
    SELECT validate_pw('testpwd', 'cqs7uOvz8KBlk') FROM SYSIBM.SYSDUMMY1"

    1
    -----------
              1

      1 record(s) selected.

2)
    CALL validate_pw('testpwd', 'cqs7uOvz8KBlk', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : IS_VALID
      Parameter Value : 1

      Return Status = 0

3)
    CALL validate_pw('testpwd', '0123456789abcdef', ?)

      Value of output parameters
      --------------------------
      Parameter Name  : IS_VALID
```

```
Parameter Value : 0

Return Status = 0
```

C. Stored Procedure Support

Stored procedures can minimize the network traffic and with regard to the authentication module configuration they can ease the administration. The module supports two types of stored procedures: one for user authentication and one for group authentication.

For the following 2 sections we use these 3 tables:

```
CREATE TABLE WEB.USERS (
    USERNAME VARCHAR(40) NOT NULL,
    PASSWORD VARCHAR(40) );

ALTER TABLE WEB.USERS
    ADD PRIMARY KEY (USERNAME);

CREATE TABLE WEB.GROUPS (
    GROUPNAME VARCHAR(40) NOT NULL,
    ACTIVE    INTEGER     NOT NULL );

ALTER TABLE WEB.GROUPS
    ADD PRIMARY KEY (GROUPNAME);

CREATE TABLE WEB.MAPPING (
    USERNAME  VARCHAR(40) NOT NULL,
    GROUPNAME VARCHAR(40) NOT NULL );

ALTER TABLE WEB.MAPPING
    ADD PRIMARY KEY (USERNAME, GROUPNAME)
    ADD FOREIGN KEY (USERNAME) REFERENCES WEB.USERS (USERNAME)
    ADD FOREIGN KEY (GROUPNAME) REFERENCES WEB.GROUPS (GROUPNAME);
```

C.1. user authentication

The stored procedure for user authentication is responsible for returning the password of the user in question to the module. It must return exact one value - the password. If `AuthIBMDB2NoPasswd` is `On`, then the username has to be returned instead of the password.

The stored procedure must have the following parameter format:

```
CREATE PROCEDURE user_procedure_name ( IN VARCHAR, OUT VARCHAR )
```

Example:

```
CREATE PROCEDURE user_sp
(IN v_username VARCHAR(40), OUT v_password VARCHAR(40))
LANGUAGE SQL
BEGIN
  SELECT password INTO v_password FROM web.users
  WHERE username = v_username;
END@
```

If `AuthIBMDB2NoPasswd` is `On`, then the stored procedure would have to look like this:

```
CREATE PROCEDURE user_sp
(IN v_username VARCHAR(40), OUT v_password VARCHAR(40))
LANGUAGE SQL
BEGIN
  SELECT username INTO v_password FROM web.users
  WHERE username = v_username;
END@
```

C.2. group authentication

The stored procedure for group authentication is responsible for returning the groups the user in question belongs to. It must return an open cursor to the result set.

The stored procedure must have the following parameter format:

```
CREATE PROCEDURE group_procedure_name ( IN VARCHAR )
```

Example

```
CREATE PROCEDURE group_sp
(IN v_username VARCHAR(40))
LANGUAGE SQL
DYNAMIC RESULT SETS 1
BEGIN
  DECLARE res CURSOR WITH RETURN FOR
  SELECT m.groupname FROM web.groups g, web.mapping m
  WHERE m.groupname = g.groupname AND
        m.username = v_username AND
        g.active = 1;

  OPEN res;
END@
```

Date: 2019-01-08 14:55:52 -0500 Id: f43f87dab2b8fed8a9e6968cdda81b8c3f7d7a14