Auth_db Module

Jan Janak
FhG Fokus

Jakob Schlyter

Edited by
Jan Janak

Auth_db Module

Edited by Jan Janak, and Jakob Schlyter,and Jan Janak

Copyright © 2002, 2003 FhG FOKUS

Revision History

Revision $Revision: 1.3 $ $Date: 2003/09/15 19:47:02 $

Table of Contents

1. User's Guide **1**
1.1. Overview 1
1.2. Dependencies 1
1.3. Exported Parameters 1
1.3.1. `db_url` (string) 1
1.3.2. `user_column` (string) 1
1.3.3. `domain_column` (string) 2
1.3.4. `password_column` (string) 2
1.3.5. `rpid_column` (string) 2
1.3.6. `calculate_ha1` (integer) 2
1.3.7. `password_column_2` (string) 3
1.3.8. `use_rpid` (integer) 3
1.4. Exported Functions 4
1.4.1. `www_authorize(realm, table)` 4
1.4.2. `proxy_authorize(realm, table)` 4
2. Developer's Guide **6**
3. Frequently Asked Questions **7**

List of Examples

1-1. db_url parameter usage 1
1-2. user_column usage 1
1-3. domain_column usage 2
1-4. password_column usage 2
1-5. rpid_column usage 2
1-6. calculate_ha1usage 3
1-7. password_column_2 usage 3
1-8. use_rpidusage 3
1-9. www_authorize usage 4
1-10. proxy_authorize usage 5

Chapter 1. User's Guide

1.1. Overview

This module contains all authentication related functions that need the access to the database. This module should be used together with auth module, it cannot be used independently because it depends on the module. Select this module if you want to use database to store authentication information like subscriber usernames and passwords. If you want to use radius authentication, then use auth_radius instead.

1.2. Dependencies

The module depends on the following modules (in the other words the listed modules must be loaded before this module):

- *auth* -- Generic authentication functions
- *database* -- Any database module (currently mysql, postgres, dbtext)

1.3. Exported Parameters

1.3.1. `db_url` (string)

This is URL of the database to be used. Value of the parameter depends on the database module used. For example for mysql and postgres modules this is something like sql://username:password@host:port/database. For dbtext module (which stores data in plaintex files) it is directory in which the database resides.

Default value is "sql://serro:47serro11@localhost/ser".

Example 1-1. `db_url` parameter usage

```
modparam("auth_db", "db_url", "sql://foo:bar@foobar.org/ser")
```

1.3.2. `user_column` (string)

This is the name of the column holding usernames. Default value is fine for most people. Use the parameter if you really need to change it.

Default value is "username".

Example 1-2. `user_column` usage

```
modparam("auth_db", "user_column", "user")
```

1.3.3. `domain_column` (string)

This is the name of the column holding domains of users. Default value is fine for most people. Use the parameter if you really need to change it.

Default value is "domain".

Example 1-3. `domain_column` usage

```
modparam("auth_db", "domain_column", "domain")
```

1.3.4. `password_column` (string)

This is the name of the column holding passwords. Passwords can be either stored as plain text or pre-calculated HA1 strings. HA1 strings are MD5 hashes of username, password, and realm. HA1 strings are more safe because the server doesn't need to know plaintext passwords and they cannot be obtained from HA1 strings.

Default value is "ha1".

Example 1-4. `password_column` usage

```
modparam("auth_db", "password_column", "password")
```

1.3.5. `rpid_column` (string)

This is the name of the column holding information for the Remote-Party-ID header field. Default value is fine for most people. Use the parameter if you really need to change it.

Default value is "rpid".

Example 1-5. `rpid_column` usage

```
modparam("auth_db", "rpid_column", "remote_party_id")
```

1.3.6. calculate_ha1 (integer)

This parameter tells server whether it should expect plaintext passwords in the database or HA1 string. If the parameter is set to 1 then the server will assume that the column pointed to by password_column contains plaintext passwords and it will calculate HA1 strings on the fly.

If the parameter is set to 0 then the server assumes that the database contains HA1 strings directly and will not calculate them. If username parameter of credentials contains also @domain (some user agents put domain in username parameter), then column pointed to by password_column_2 parameter will be used instead. This column should also contain HA1 strings but they should be calculated including the domain in the username parameter (as opposed to password_column which (when containing HA1 strings) should always contains HA1 strings calculated without domain in username.

This ensures that the authentication will always work when using pre-calculated HA1 string, not depending on the presence of the domain in username.

Default value of this parameter is 0.

Example 1-6. calculate_ha1usage

```
modparam("auth_db", "calculate_ha1", 1)
```

1.3.7. password_column_2 (string)

As described in the previous section this parameter contains name of column holding pre-calculated HA1 string that were calculated including the domain in the username. This parameter is used only when calculate_ha1 is set to 0 and user agent send a credentials containing the domain in the username.

Default value of the parameter is ha1b.

Example 1-7. password_column_2 usage

```
modparam("auth_db", "password_column_2", "ha1_2")
```

1.3.8. use_rpid (integer)

This parameter specifies whether the server should fetch a value for the Remote-Party-ID header field from the database.

If the parameter is set to 1 the server expects to find a value for this header in the column specified by the rpid_column parameter.

Default value of this parameter is 0.

Example 1-8. `use_rpid` usage

```
modparam("auth_db", "use_rpid", 1)
```

1.4. Exported Functions

1.4.1. `www_authorize(realm, table)`

The function verifies credencials according to RFC2617. If the credentials are verified sucessfully then the function will succeed and mark the credentials as authorized (marked credentials can be later used by some other functions). If the function was unable to verify the credentials for some reason then it will fail and the script should call `www_challenge` which will challenge the user again.

Meaning of the parameters is as follows:

- *realm* - Realm is a opaque string that the user agent should present to the user so he can decide what username and password to use. Usualy this is domain of the host the server is running on.

  If an empty string "" is used then the server will generate it from the request. In case of REGISTER requests To header field domain will be used (because this header field represents a user being registered), for all other messages From header field domain will be used.

- *table* - Table to be used to lookup usernames and passwords (usualy subscribers table).

Example 1-9. `www_authorize` usage

```
...
if (www_authorize("iptel.org", "subscriber")) {
    www_challenge("iptel.org", "1");
};
...
```

1.4.2. `proxy_authorize(realm, table)`

The function verifies credencials according to RFC2617. If the credentials are verified sucessfully then the function will succeed and mark the credentials as authorized (marked credentials can be later used by some other functions). If the function was unable to verify the credentials for some reason then it will fail and the script should call `proxy_challenge` which will challenge the user again.

Meaning of the parameters is as follows:

- *realm* - Realm is a opaque string that the user agent should present to the user so he can decide what username and password to use. Usualy this is domain of the host the server is running on.

If an empty string "" is used then the server will generate it from the request. From header field domain will be used as realm.

- *table* - Table to be used to lookup usernames and passwords (usualy subscribers table).

Example 1-10. proxy_authorize usage

```
...
if (!proxy_authorize("", "subscriber)) {
    proxy_challenge("", "1");  # Realm will be autogenerated
};
...
```

Chapter 2. Developer's Guide

To be done.

Chapter 3. Frequently Asked Questions

1. What is the meaning of life ?

42