Auth Module

Jan Janak
FhG Fokus

Juha Heinanen
Song Networks

Edited by

Jan Janak

Auth Module

Edited by Jan Janak, and Juha Heinanen,and Jan Janak

Revision History

Revision $Revision: 1.2.2.2 $ $Date: 2003/08/28 02:15:22 $

Table of Contents

List of Examples

Chapter 1. User’s Guide

1.1. Overview

This is a generic module that itself doesn’t provide all functions necessary for authentication but provides functions that are needed by all other authentication related modules (so called authenticatio backends).

We decided to break the authentication code into several modules because there are now more than one backends (currently database authentication and radius are supported). This allows us to create separate packages so uses can install and load only required functionality. This also allows us to avoid unecessary dependencies in the binary packages.

1.2. Dependencies

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

- *sl* -- Stateless replies

1.3. Exported Parameters

1.3.1. `secret` (string)

Default value is randomly generated string.

Example 1-1. Setting secret module parameter

```
modparam("auth", "secret", "johndoessecretphrase")
```

1.3.2. `nonce_expire` (integer)

Nonces have limited lifetime. After a given period of time nonces will be considered invalid. This is to protect replay attacks. Credentials containing a stale nonce will be not authorized, but the user agent will be challenged again. This time the challenge will contain `stale` parameter which will indicate to the client that it doesn’t have to disturb user by asking for username and password, it can recalculate credentials using existing username and password.

The value is in seconds and default value is 300 seconds.

Example 1-2. nonce_expire example

```
modparam("auth", "nonce_expire", 600)   # Set nonce_expire to 600s
```

1.3.3. `rpid_prefix` (string)

Prefix to be added to Remote-Party-ID header field just before the URI returned from either radius or database.

Default value is "".

Example 1-3. rpid_prefix

```
modparam("auth", "rpid_prefix", "Whatever <")
```

1.3.4. `rpid_suffix` (string)

Suffix to be added to Remote-Party-ID header field after the URI returned from either radius or database.

Default value is ";party=calling;id-type=subscriber;screen=yes".

Example 1-4. rpid_suffix

```
modparam("auth", "rpid_suffix", "@1.2.3.4>")
```

1.4. Exported Functions

1.4.1. `www_challenge(realm, qop)`

The function challenges a user agent. It will generate a WWW-Authorize header field containing a digest challenge, it will put the header field into a response generated from the request the server is processing and send the reply. Upon reception of such a reply the user agent should compute credentials and retry the request. For more information regarding digest authentication see RFC2617.

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 (beucase this header field represents a user being registered), for all other messages From header field domain will be used.

- *qop* - Value of this parameter can be either "1" or "0". When set to 1 then the server will put qop parameter in the challenge. When set to 0 then the server will not put qop parameter in the challenge. It is strongly recommended to use qop parameter, hovewere there are still some user agents that cannot handle qop parameter properly so we made this optional. On the other hand there are still some user agents that cannot handle request without qop parameter too.

Example 1-5. www_challenge usage

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

1.4.2. `proxy_challenge(realm, qop)`

The function challenges a user agent. It will generate a Proxy-Authorize header field containing a digest challenge, it will put the header field into a response generated from the request the server is processing and send the reply. Upon reception of such a reply the user agent should compute credentials and retry the request. For more information regarding digest authentication see RFC2617.

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.

- *qop* - Value of this parameter can be either "1" or "0". When set to 1 then the server will put qop parameter in the challenge. When set to 0 then the server will not put qop parameter in the challenge. It is strongly recommended to use qop parameter, hovewere there are still some user agents that cannot handle qop parameter properly so we made this optional. On the other hand there are still some user agents that cannot handle request without qop parameter too.

Example 1-6. proxy_challenge usage

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

1.4.3. `consume_credentials()`

This function removes previously authorized credentials from the message being processed by the server. That means that the downstream message will not contain credentials there were used by this server. This ensures that

the proxy will not reveal information about credentials used to downstream elements and also the message will be a little bit shorter. The function must be called after `www_authorize` or `proxy_authorize`.

Example 1-7. consume_credentials example

```
...
if (www_authorize("", "subscriber)) {
    consume_credentials();
};
...
```

1.4.4. `is_rpid_user_e164()`

The function checks if the SIP URI received from the database or radius server and will potentially be used in Remote-Party-ID header field contains an E164 number (+ followed by up to 15 decimal digits) in its user part. Check fails, if no such SIP URI exists (i.e. radius server or database didn't provide this information).

Example 1-8. is_rpid_user_e164 usage

```
...
if (is_rpid_user_e164()) {
    # do something here
};
...
```

1.4.5. `append_rpid_hf`

Appends to the message a Remote-Party-ID header that contains header 'Remote-Party-ID: ' followed by the saved value of the SIP URI received from the database or radius server followed by the value of module parameter radius_rpid_suffix. The function does nothing if no saved SIP URI exists.

Example 1-9. append_rpid_hf

```
...
append_rpid_hf();  # Append Remote-Party-ID header field
...
```

Chapter 2. Developer’s Guide

To be done.

Chapter 3. Frequently Asked Questions

1. What is the meaning of life ?

42