NAME

pwquality - Documentation of the libpwquality API

SYNOPSIS

```
#include <pwquality.h>

pwquality_settings_t *pwquality_default_settings(void);
void pwquality_free_settings(pwquality_settings_t *pwq);

int pwquality_read_config(pwquality_settings_t *pwq, const char *cfgfile,
    void **auxerror);

int pwquality_set_option(pwquality_settings_t *pwq, const char *option);
int pwquality_set_int_value(pwquality_settings_t *pwq, int setting, int value);
int pwquality_set_str_value(pwquality_settings_t *pwq, int setting,
    const char *value);
int pwquality_get_int_value(pwquality_settings_t *pwq, int setting, int *value);
int pwquality_get_str_value(pwquality_settings_t *pwq, int setting, const char **value);

int pwquality_generate(pwquality_settings_t *pwq, int entropy_bits,
    char **password);

int pwquality_check(pwquality_settings_t *pwq, const char *password,
    const char *oldpassword, const char *user, void **auxerror);

const char *pwquality_strerror(char *buf, size_t len, int errcode, void *auxerror);
```

DESCRIPTION

Function **pwquality_default_settings()** allocates and returns default pwquality settings to be used in other library calls. The allocated opaque structure has to be freed with the **pwquality_free_settings()** call.

The **pwquality_read_config()** parses the configuration file (if *cfgfile* is NULL then the default one). If *auxerror* is not NULL it also possibly returns auxiliary error information that must be passed into **pwquality_strerror()** function.

New in 1.3.0:

The library first tries to parse all **.conf* configuration files from *<cfgfile>.d* directory if it exists. Order of parsing determines what values will be in effect - the latest wins.

Function **pwquality_set_option()** is useful for setting the options as configured on a pam module command line in form of *opt=val*.

Getter and setter functions for the individual integer and string setting values are: **pwquality_set_int_value()**, **pwquality_set_str_value()**, **pwquality_get_int_value()**, and **pwquality_get_str_value()**. In case of the string getter the caller must copy the string before another calls that can manipulate the *pwq* settings object.

The **pwquality_generate()** function generates a random password of *entropy_bits* entropy and checks it according to the settings. The **password* is allocated on the heap by the library. The *entropy_bits* value is adjusted to fit within the **PWQ_MIN_ENTROPY_BITS** and **PWQ_MAX_ENTROPY_BITS** range before generating a password.

The **pwquality_check()** function checks the *password* according to the settings. It returns either score (value between 0 and 100), negative error number, and possibly also auxiliary error information that must be passed into the **pwquality_strerror()** function. The *oldpassword* is optional and can be NULL. The *user* is used for checking the *password* against the user name and potentially other **passwd**(5) information and can be NULL. The *auxerror* can be NULL - in that case the auxiliary error information is not returned. However if it is non-NULL not passing the returned **auxerror* into **pwquality_strerror()** can lead to memory leaks.

The score of a password depends on the value of the setting **PWQ_SETTING_MIN_LENGTH**. If it is set higher, the score for the same passwords will be lower.

Function **pwquality_strerror()** translates the *errcode* and *auxerror* auxiliary data into a localized text message. If *buf* is NULL the function uses an internal static buffer which makes the function non-reentrant in that case. The returned pointer is not guaranteed to point to the *buf*. The function deallocates eventual *auxerror* data passed into it, thus it must not be called twice with the same *auxerror* data.

RETURN VALUES

In general the functions which return **int** return 0 as success value and negative values as concrete **PWQ_ERROR** error code. **pwquality_strerror()** does not allocate data and so it cannot fail.

The returned positive or zero score from **pwquality_check()** should not be used for rejection of passwords, it should be used only as approximate indicator of entropy present in the password with values such as 0-30 being low, 30-60 medium, and 60-100 high.

EXAMPLE

Typical use of the libpwquality API:

```
#include <pwquality.h>

...

    pwquality_settings_t *pwq;
    int rv;
    void *auxerror;
    char buf[PWQ_MAX_ERROR_MESSAGE_LEN];

    pwq = pwquality_default_settings();
    if (pwq == NULL) {
        fprintf(stderr, "Error: %s\n", pwquality_strerror(buf, sizeof(buf), PWQ_ERROR_MEM_ALLOC, NULL));
        return -1;
    }

    if ((rv=pwquality_read_config(pwq, NULL, &auxerror)) != 0) {
        pwquality_free_settings(pwq);
        fprintf(stderr, "Error: %s\n", pwquality_strerror(buf, sizeof(buf), rv, auxerror));
        return -1;
    }

    rv = pwquality_check(pwq, buf, NULL, user, &auxerror);
    pwquality_free_settings(pwq);

    if (rv >= 0) {
        fprintf(stderr, "Password entropy score is: %d\n", rv);
    } else {
        fprintf(stderr, "Password is rejected with error: %s\n", pwquality_strerror(buf, sizeof(buf), rv, auxerror));
    }
```

SEE ALSO

pwquality.conf(5)

AUTHORS

Tomas Mraz <tmraz@redhat.com>