Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'CRYPTO_secure_free.3ossl' command

$ man CRYPTO_secure_free.3ossl

OPENSSL_SECURE_MALLOC(3ossl) OpenSSL OPENSSL_SECURE_MALLOC(3ossl)

NAME

CRYPTO_secure_malloc_init, CRYPTO_secure_malloc_initialized,

CRYPTO_secure_malloc_done, OPENSSL_secure_malloc, CRYPTO_secure_malloc,

OPENSSL_secure_zalloc, CRYPTO_secure_zalloc, OPENSSL_secure_free,

CRYPTO_secure_free, OPENSSL_secure_clear_free,

CRYPTO_secure_clear_free, OPENSSL_secure_actual_size,

CRYPTO_secure_allocated, CRYPTO_secure_used - secure heap storage

SYNOPSIS

```
#include <openssl/crypto.h>

int CRYPTO_secure_malloc_init(size_t size, size_t minsize);

int CRYPTO_secure_malloc_initialized();

int CRYPTO_secure_malloc_done();

void *OPENSSL_secure_malloc(size_t num);
void *CRYPTO_secure_malloc(size_t num, const char *file, int line);

void *OPENSSL_secure_zalloc(size_t num);
```

```
void *CRYPTO_secure_zalloc(size_t num, const char *file, int line);

void OPENSSL_secure_free(void* ptr);
void CRYPTO_secure_free(void *ptr, const char *, int);

void OPENSSL_secure_clear_free(void* ptr, size_t num);
void CRYPTO_secure_clear_free(void *ptr, size_t num, const char *, int);

size_t OPENSSL_secure_actual_size(const void *ptr);

int CRYPTO_secure_allocated(const void *ptr);
size_t CRYPTO_secure_used();
```

DESCRIPTION

In order to help protect applications (particularly long-running servers) from pointer overruns or underruns that could return arbitrary data from the program's dynamic memory area, where keys and other sensitive information might be stored, OpenSSL supports the concept of a "secure heap." The level and type of security guarantees depend on the operating system. It is a good idea to review the code and see if it addresses your threat model and concerns.

If a secure heap is used, then private key BIGNUM values are stored there. This protects long-term storage of private keys, but will not necessarily put all intermediate values and computations there.

CRYPTO_secure_malloc_init() creates the secure heap, with the specified "size" in bytes. The "minsize" parameter is the minimum size to allocate from the heap or zero to use a reasonable default value. Both "size" and, if specified, "minsize" must be a power of two and "minsize" should generally be small, for example 16 or 32. "minsize" must be less than a quarter of "size" in any case.

CRYPTO_secure_malloc_initialized() indicates whether or not the secure heap as been initialized and is available.

CRYPTO_secure_malloc_done() releases the heap and makes the memory unavailable to the process if all secure memory has been freed. It can take noticeably long to complete.

OPENSSL_secure_malloc() allocates "num" bytes from the heap. If CRYPTO_secure_malloc_init() is not called, this is equivalent to calling OPENSSL_malloc(). It is a macro that expands to CRYPTO_secure_malloc() and adds the "__FILE__" and "__LINE__" parameters.

OPENSSL_secure_zalloc() and CRYPTO_secure_zalloc() are like OPENSSL_secure_malloc() and CRYPTO_secure_malloc(), respectively, except that they call memset() to zero the memory before returning.

OPENSSL_secure_free() releases the memory at "ptr" back to the heap. It must be called with a value previously obtained from OPENSSL_secure_malloc(). If CRYPTO_secure_malloc_init() is not called, this is equivalent to calling OPENSSL_free(). It exists for consistency with OPENSSL_secure_malloc() , and is a macro that expands to CRYPTO_secure_free() and adds the "__FILE__" and "__LINE__" parameters..

OPENSSL_secure_clear_free() is similar to OPENSSL_secure_free() except that it has an additional "num" parameter which is used to clear the memory if it was not allocated from the secure heap. If CRYPTO_secure_malloc_init() is not called, this is equivalent to calling OPENSSL_clear_free().

OPENSSL_secure_actual_size() tells the actual size allocated to the pointer; implementations may allocate more space than initially

requested, in order to "round up" and reduce secure heap fragmentation.

OPENSSL_secure_allocated() tells if a pointer is allocated in the
secure heap.

CRYPTO_secure_used() returns the number of bytes allocated in the
secure heap.

RETURN VALUES

CRYPTO_secure_malloc_init() returns 0 on failure, 1 if successful, and
2 if successful but the heap could not be protected by memory mapping.

CRYPTO_secure_malloc_initialized() returns 1 if the secure heap is
available (that is, if CRYPTO_secure_malloc_init() has been called, but
CRYPTO_secure_malloc_done() has not been called or failed) or 0 if not.

OPENSSL_secure_malloc() and OPENSSL_secure_zalloc() return a pointer
into the secure heap of the requested size, or "NULL" if memory could
not be allocated.

CRYPTO_secure_allocated() returns 1 if the pointer is in the secure
heap, or 0 if not.

CRYPTO_secure_malloc_done() returns 1 if the secure memory area is
released, or 0 if not.

OPENSSL_secure_free() and OPENSSL_secure_clear_free() return no values.

SEE ALSO

OPENSSL_malloc(3), BN_new(3)

HISTORY

The OPENSSL_secure_clear_free() function was added in OpenSSL 1.1.0g.

The second argument to CRYPTO_secure_malloc_init() was changed from an int to a size_t in OpenSSL 3.0.

COPYRIGHT