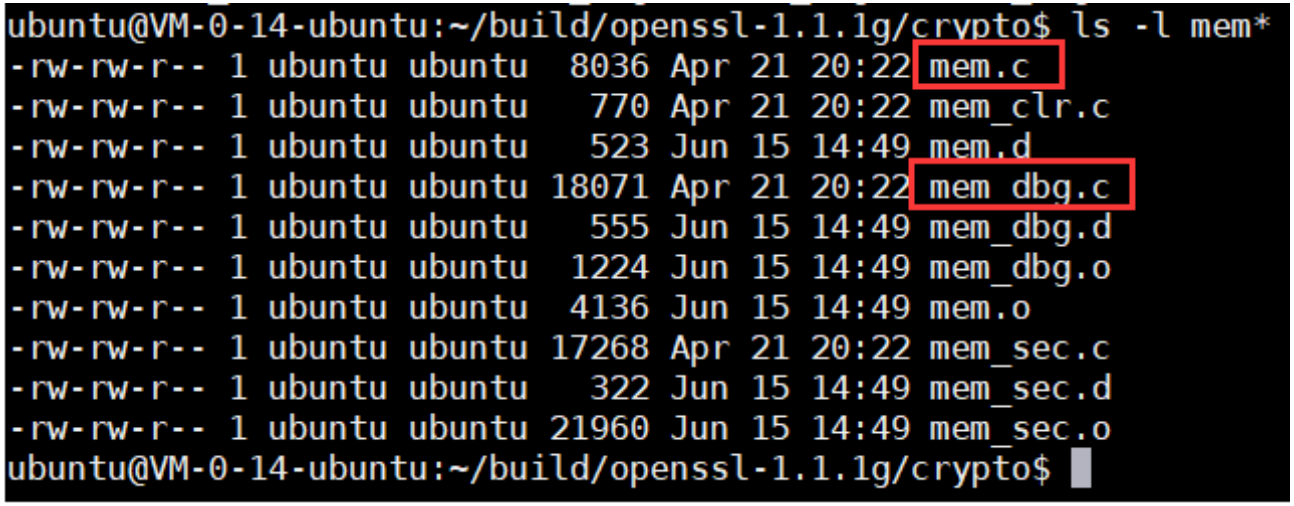
Linux (programming): 37 -- OpenSSL Library (memory allocation)

1, OpenSSL memory allocation

- When users use memory, the error they are prone to make is memory leak. When users call memory allocation and release functions, it is more difficult to find memory leaks. OpenSSL provides built-in memory allocation / release functions. If the user calls OpenSSL's memory allocation and release function completely, the memory leak point can be easily found
- When OpenSSL allocates memory, it maintains a memory allocation hash table inside it to store the allocated but not released memory information:
  - Add this information to the hash table when the user requests memory allocation
  - Delete this information when memory is released
  - When the user finds the memory leak point through the OpenSSL function, he only needs to query the hash table, and the user can handle the leaked memory through the OpenSSL callback function
- The OpenSSL code explained in this article belongs to version 1.1.1g, and the interface names may be different between different versions
- OpenSSL memory allocation and other functions for users to call are mainly implemented in crypto/mem.c, and its built-in allocation function is implemented in crypto/mem_dbg.c. Function call MEM in mem.c by default_ Implementation in DBG. C. If the user implements his own memory allocation function and the function to find the memory leak, he can call CRYPTO_set_mem_functions and CRYPTO_set_mem_debug_functions to set

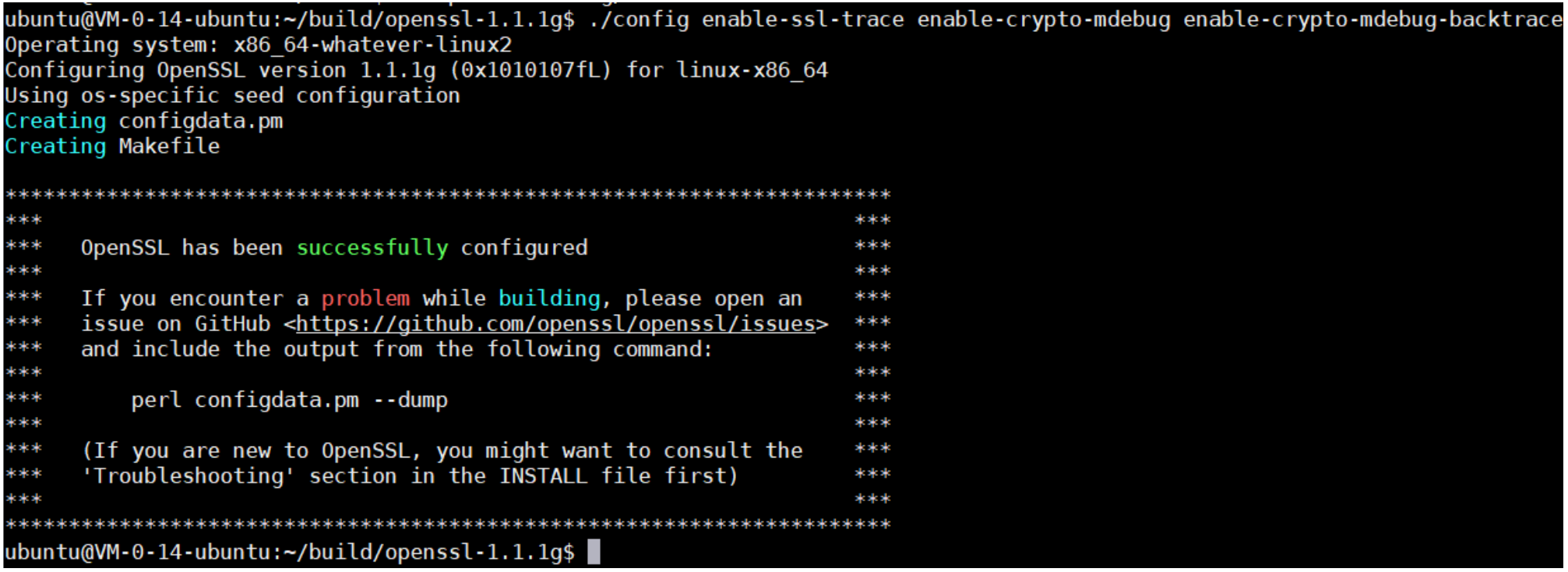


- Memory API reference document: https://www.openssl.org/docs/man1.1.1/man3/CRYPTO_mem_ctrl.html

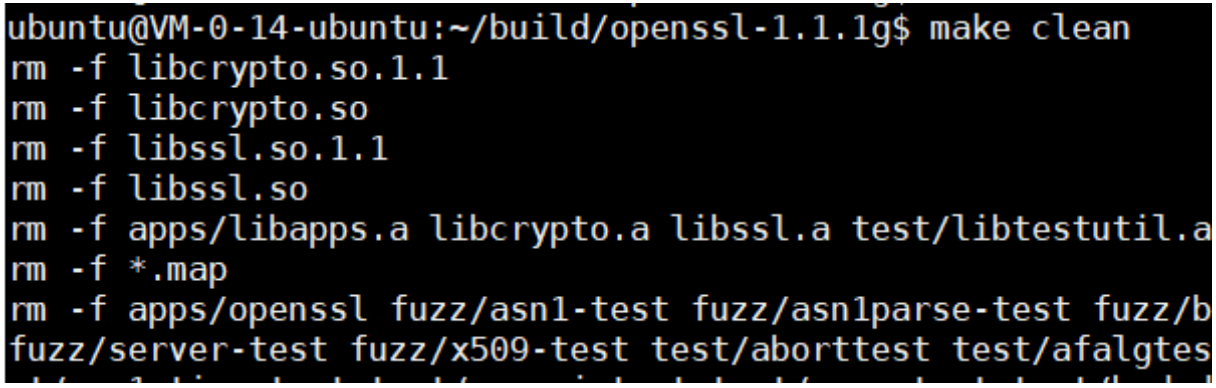
OPENSSL_NO_CRYPTO_MDEBUG macro definition

- There is an OpenSSL in OpenSSL_ NO_ CRYPTO_ The mdebug macro definition is used to output the memory usage. The default compilation is not open. You need to recompile the library. Add the parameter enable crypto mdebug enable crypto mdebug backtrace when executing config
- Without this macro, the following CRYPTO_xxx() series API may not be available
- For example, let's reconfigure, compile and install the openssl library

```
#Configuration
./config enable-ssl-trace enable-crypto-mdebug enable-crypto-mdebug-backtrace

#Clear previously compiled content
make clean

#Recompile
make

#Installation
sudo make install
```



```
ubuntu@VM-0-14-ubuntu:~/build/openssl-1.1.1g$ ./config enable-ssl-trace enable-crypto-mdebug enable-crypto-mdebug-backtrace
Operating system: x86_64-whatever-linux2
Configuring OpenSSL version 1.1.1g (0x1010107fL) for linux-x86_64
Using os-specific seed configuration
Creating configdata.pm
Creating Makefile

**********************************************************************
***                                                                ***
***   OpenSSL has been successfully configured                     ***
***                                                                ***
***   If you encounter a problem while building, please open an    ***
***   issue on GitHub <https://github.com/openssl/openssl/issues>  ***
***   and include the output from the following command:           ***
***                                                                ***
***       perl configdata.pm --dump                                ***
***                                                                ***
***   (If you are new to OpenSSL, you might want to consult the    ***
***   'Troubleshooting' section in the INSTALL file first)         ***
***                                                                ***
**********************************************************************
ubuntu@VM-0-14-ubuntu:~/build/openssl-1.1.1g$
```



```
ubuntu@VM-0-14-ubuntu:~/build/openssl-1.1.1g$ make clean
rm -f libcrypto.so.1.1
rm -f libcrypto.so
rm -f libssl.so.1.1
rm -f libssl.so
rm -f apps/libapps.a libcrypto.a libssl.a test/libtestutil.a
rm -f *.map
rm -f apps/openssl fuzz/asn1-test fuzz/asn1parse-test fuzz/b
fuzz/server-test fuzz/x509-test test/aborttest test/afalgtes
```

```
ubuntu@VM-0-14-ubuntu:~/build/openssl-1.1.1g$ make
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
    "-oMakefile" include/crypto/bn_conf.h.in > include/crypto/bn_conf.h
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
    "-oMakefile" include/crypto/dso_conf.h.in > include/crypto/dso_conf.h
/usr/bin/perl "-I." -Mconfigdata "util/dofile.pl" \
```

```
ubuntu@VM-0-14-ubuntu:~/build/openssl-1.1.1g$ sudo make install
make depend && make _build_libs
make[1]: Entering directory '/home/ubuntu/build/openssl-1.1.1g'
make[1]: Leaving directory '/home/ubuntu/build/openssl-1.1.1g'
make[1]: Entering directory '/home/ubuntu/build/openssl-1.1.1g'
make[1]: Nothing to be done for '_build_libs'.
make[1]: Leaving directory '/home/ubuntu/build/openssl-1.1.1g'
*** Installing runtime libraries
install libcrypto.so.1.1 -> /usr/local/lib/libcrypto.so.1.1
install libssl.so.1.1 -> /usr/local/lib/libssl.so.1.1
*** Installing development files
install ./include/openssl/aes.h -> /usr/local/include/openssl/aes.h
install ./include/openssl/asn1.h -> /usr/local/include/openssl/asn1
```

2, Relevant data structure

struct app_mem_info_st

```
/*-
 * For application-defined information (static C-string `info')
 * to be displayed in memory leak list.
 * Each thread has its own stack.  For applications, there is
 *   OPENSSL_mem_debug_push("...")     to push an entry,
 *   OPENSSL_mem_debug_pop()     to pop an entry,
 */
struct app_mem_info_st {
```

```
    CRYPTO_THREAD_ID threadid;
    const char *file;
    int line;
    const char *info;
    struct app_mem_info_st *next; /* tail of thread's stack */
    int references;
} APP_INFO;
```

```
/* memory-block description */
struct mem_st {
    void *addr;
    int num;
    const char *file;
    int line;
    CRYPTO_THREAD_ID threadid;
    unsigned long order;
    time_t time;
    APP_INFO *app_info;
#ifndef OPENSSL_NO_CRYPTO_MDEBUG_BACKTRACE
    void *array[30];
    size_t array_siz;
#endif
};
```

- OpenSSL memory allocation data structure is an internal data structure defined in crypto / MEM_ In DBG. C
- The meanings are as follows:
    - addr: address to allocate memory
    - num: size of allocated memory
    - File: file to allocate memory
    - Line: line number of allocated memory
    - Thread: the thread ID of the allocated memory

- order: memory allocation times
- Time: memory allocation time
- app_info: used to store user application information. It is a linked list, in which file, line number, thread ID and other information are stored
- References: number of references

3, Interface Overview

- OpenSSL provides the following memory allocation interfaces: OpenSSL memory allocation is performed by OPENSSL_xx API processing, these are usually macros, they add standard C language__ FILE_ And__ LINE_ Parameter and call low level CRYPTO_xxx API. Some functions do not add those parameters, but exist for consistency

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

 int OPENSSL_malloc_init(void)

 void *OPENSSL_malloc(size_t num)
 void *OPENSSL_zalloc(size_t num)
 void *OPENSSL_realloc(void *addr, size_t num)
 void OPENSSL_free(void *addr)
 char *OPENSSL_strdup(const char *str)
 char *OPENSSL_strndup(const char *str, size_t s)
 size_t OPENSSL_strlcat(char *dst, const char *src, size_t size);
 size_t OPENSSL_strlcpy(char *dst, const char *src, size_t size);
 size_t OPENSSL_strnlen(const char *str, size_t maxlen);
 void *OPENSSL_memdup(void *data, size_t s)
 void *OPENSSL_clear_realloc(void *p, size_t old_len, size_t num)
 void OPENSSL_clear_free(void *str, size_t num)
 void OPENSSL_cleanse(void *ptr, size_t len);

 unsigned char *OPENSSL_hexstr2buf(const char *str, long *len);
 char *OPENSSL_buf2hexstr(const unsigned char *buffer, long len);
```

```
int OPENSSL_hexchar2int(unsigned char c);

void *CRYPTO_malloc(size_t num, const char *file, int line)
void *CRYPTO_zalloc(size_t num, const char *file, int line)
void *CRYPTO_realloc(void *p, size_t num, const char *file, int line)
void CRYPTO_free(void *str, const char *, int)
char *CRYPTO_strdup(const char *p, const char *file, int line)
char *CRYPTO_strndup(const char *p, size_t num, const char *file, int line)
void *CRYPTO_clear_realloc(void *p, size_t old_len, size_t num,
                           const char *file, int line)
void CRYPTO_clear_free(void *str, size_t num, const char *, int)

void CRYPTO_get_mem_functions(
        void *(**m)(size_t, const char *, int),
        void *(**r)(void *, size_t, const char *, int),
        void (**f)(void *, const char *, int))
int CRYPTO_set_mem_functions(
        void *(*m)(size_t, const char *, int),
        void *(*r)(void *, size_t, const char *, int),
        void (*f)(void *, const char *, int))

void CRYPTO_get_alloc_counts(int *m, int *r, int *f)

int CRYPTO_set_mem_debug(int onoff)

env OPENSSL_MALLOC_FAILURES=... <application>
env OPENSSL_MALLOC_FD=... <application>

int CRYPTO_mem_ctrl(int mode);

int OPENSSL_mem_debug_push(const char *info)
int OPENSSL_mem_debug_pop(void);
```

```
int CRYPTO_mem_debug_push(const char *info, const char *file, int line);
int CRYPTO_mem_debug_pop(void);

int CRYPTO_mem_leaks(BIO *b);
int CRYPTO_mem_leaks_fp(FILE *fp);
int CRYPTO_mem_leaks_cb(int (*cb)(const char *str, size_t len, void *u),
                        void *u);
```

- Next, I will introduce the functions. Since these functions are summarized by querying documents by myself, there are some errors in the function, parameter and return value introduction. You can test the functions before using these API s

4, Function introduction 1

OPENSSL_malloc_init

```
int OPENSSL_malloc_init(void);

//The source code is defined as follows:
/* No longer needed, so this is a no-op */
#define OPENSSL_malloc_init() while(0) continue
```

- The function does not perform any operations and does not need to be called. Included for compatibility with earlier versions of OpenSSL

OPENSSL_malloc

```
void *OPENSSL_malloc(size_t num);

//The source code is defined as follows:
# define OPENSSL_malloc(num) \
        CRYPTO_malloc(num, OPENSSL_FILE, OPENSSL_LINE)
```

- Function: equivalent to malloc() function of C language, used to apply for memory
- Parameters: size of requested memory
- Return value:
    - Success: return newly allocated memory
    - Failed: NULL returned

OPENSSL_zalloc

```
void *OPENSSL_zalloc(size_t num);

//The source code is defined as follows:
# define OPENSSL_zalloc(num) \
        CRYPTO_zalloc(num, OPENSSL_FILE, OPENSSL_LINE)
```

- Function: it is also equivalent to the malloc() function of C language, which is used to apply for memory, but when it applies for memory, it sets all memory to 0
- The internal call is actually CRYPTO_malloc() function
- Parameters: size of requested memory
- Return value:
    - Success: return newly allocated memory
    - Failed: NULL returned

OPENSSL_realloc

```
void *OPENSSL_realloc(void *addr, size_t num);

//The source code is defined as follows:
# define OPENSSL_realloc(addr, num) \
        CRYPTO_realloc(addr, num, OPENSSL_FILE, OPENSSL_LINE)
```

- Function: also equivalent to realloc() function of C language, reallocate memory to a pointer that has applied for memory
- Parameters:
    - addr: original address space

    - num: size of reapplication
- Return value:
    - Success: address of newly allocated memory
    - Failed: NULL returned

OPENSSL_free

```
void OPENSSL_free(void *addr);

//The source code is defined as follows:
# define OPENSSL_free(addr) \
        CRYPTO_free(addr, OPENSSL_FILE, OPENSSL_LINE)
```

- Function: also equivalent to the free() function of C language, used to release the specified memory
- Parameter: address of memory to be released

OPENSSL_cleanse

```
void OPENSSL_cleanse(void *ptr, size_t len);
```

- Function: equivalent to C language memset(prt,0,len), use 0 to fill in ptr of len size
- Parameters:
    - ptr: memory start address to be set
    - len: number of padding 0
- If memory is a mapping of files, use OpenSSL carefully_ cleanse(). If the storage controller uses write compression, it is possible that the sensitive tail bytes will continue to exist after zeroing because the zero block will be compressed. If the storage controller uses loss equalization, the old sensitive data will not be overwritten; instead, the block of 0 will be written to the new physical location

OPENSSL_clear_realloc

```
void *OPENSSL_clear_realloc(void *p, size_t old_len, size_t num);

//The source code is defined as follows:
```

```
# define OPENSSL_clear_realloc(addr, old_num, num) \
        CRYPTO_clear_realloc(addr, old_num, num, OPENSSL_FILE, OPENSSL_LINE)
```

- Function: with OPENSSL_realloc is similar in that it also releases the original memory of p, but before releasing p, it sets all the memory referred to by p to 0, old_len is the original memory size of p, and then a new block of memory is reallocated for p. the memory size is num, and the new memory is returned
- Parameters:
  - p: Original memory address
  - nold_len: the size of the original memory address
  - num: size of the reallocated memory address
- Return value:
  - Success: address of newly allocated memory
  - Failed: NULL returned

OPENSSL_clear_free

```
void OPENSSL_clear_free(void *str, size_t num);

//The source code is defined as follows:
# define OPENSSL_clear_free(addr, num) \
        CRYPTO_clear_free(addr, num, OPENSSL_FILE, OPENSSL_LINE)
```

- Function: with OpenSSL_ The function of free is similar. It is also used to release memory. However, it clears all memory values before releasing the memory referred to in str
- Its internal call OpenSSL_ Clean, then call CRYPTO_free
- Parameters:
  - str: memory address to release
  - num: size of memory address to be released

Demo case

- **below mem_test1.c Corresponding Github The links are:**https://github.com/dongyusheng/csdn-code/tree/master/openssl/mem

```c
//mem_test1.c
#include <stdio.h>
#include <string.h>
#include <openssl/crypto.h>

#define STIRNG_LEN 5

int main()
{
    //1. Request a memory size of 4
    char *str = (char *)OPENSSL_malloc(4);
    if(str == NULL)
    {
        printf("OPENSSL_malloc error\n");
        return -1;
    }
    printf("OPENSSL_malloc success\n\n");

    //2. Set all memory to 0
    OPENSSL_cleanse(str, 4);
    printf("Set all memory to 0\n\n");

    //3. Copy the string to str and print
    memcpy(str, "abc", 4);
    printf("str: %s\n\n", str);

    //4. Set all the original memory to 0, and then reapply a piece of memory
    str = OPENSSL_clear_realloc(str, 4, 10);
    if(str == NULL)
    {
        printf("OPENSSL_clear_realloc error\n");
        return -1;
```

```
    }
    printf("OPENSSL_clear_realloc success\n\n");

    //5. Set all memory to 0
    OPENSSL_cleanse(str, 4);
    printf("Set all memory to 0\n\n");

    //6. Copy the string to str and print
    memcpy(str, "Github", 7);
    printf("str: %s\n\n", str);

    //7. Free memory
    OPENSSL_free(str);

    return 0;
}
```

- The compilation test is as follows:

```
gcc -o mem_test1 mem_test1.c -lssl -lcrypto
```

```
ubuntu@VM-0-14-ubuntu:~/code/openssl$ gcc -o mem_test1 mem_test1.c -lssl -lcrypto
ubuntu@VM-0-14-ubuntu:~/code/openssl$ ./mem_test1
OPENSSL_malloc success

Set all memory to 0

str: abc

OPENSSL_clear_realloc success

Set all memory to 0

str: Github

ubuntu@VM-0-14-ubuntu:~/code/openssl$
```

5, Function introduction 2

OPENSSL_strlcpy

```
size_t OPENSSL_strlcpy(char *dst, const char *src, size_t siz);
```

- Function: similar to the copy function of C language, copy the content referred to in src to dst, and the copy size is siz
- Parameters:
  - dst: destination address
  - src: source address
  - siz: copy size

OPENSSL_strlcat

```
size_t OPENSSL_strlcat(char *dst, const char *src, size_t siz);
```

- Function: splice src to the back of dts, where does siz start splicing from dst

- Parameters:
  - dst: destination address
  - src: source address
  - siz: how many positions of dst start splicing

OPENSSL_strnlen

```
size_t OPENSSL_strnlen(const char *str, size_t maxlen);
```

- To be continued

6, Function introduction 3

- To be continued

OPENSSL_strdup

```
char *OPENSSL_strdup(const char *str);

//Implementation in source code:
# define OPENSSL_strdup(str) \
        CRYPTO_strdup(str, OPENSSL_FILE, OPENSSL_LINE)
```

OPENSSL_strndup

```
char *OPENSSL_strndup(const char *str, size_t s);

//Implementation in source code:
# define OPENSSL_strndup(str, n) \
        CRYPTO_strndup(str, n, OPENSSL_FILE, OPENSSL_LINE)
```

OPENSSL_memdup

```
void *OPENSSL_memdup(void *data, size_t s);

//Implementation in source code:
# define OPENSSL_memdup(str, s) \
        CRYPTO_memdup((str), s, OPENSSL_FILE, OPENSSL_LINE)
```

7, Function introduction 4

- When the following functions are tested, they are expected to be inconsistent with the document description. Don't try them until they are updated

OPENSSL_hexstr2buf

```
unsigned char *OPENSSL_hexstr2buf(const char *str, long *len);
```

- Function: parse str into hexadecimal string and return a pointer to the parsed value
- The returned string is inside the function using OPENSSL_malloc(), so OpenSSL should be used after use_ Free () to release it
- The colon between the hexadecimal "bytes" of two characters will be ignored. Odd number errors in hexadecimal numbers
- Parameters:
    - str: string to convert
    - len:
- Return value:
    - Success: return hex string
    - Failed: NULL returned

OPENSSL_buf2hexstr

```
char *OPENSSL_buf2hexstr(const unsigned char *buffer, long len);
```

- Function: with OPENSSL_hexstr2buf, in contrast, is used to convert hexadecimal strings to strings
- The returned string is inside the function using OPENSSL_malloc(), so OpenSSL should be used after use_ Free () to release it
- Parameters:
    - buffer: stored hexadecimal string

    - len:
- Return value:
    - Success: return the string after quasi conversion
    - Failed: NULL returned
    - NULL if len is 0

OPENSSL_hexchar2int

```
int OPENSSL_hexchar2int(unsigned char c);
```

- Function: converts the character specified by the parameter to the equivalent hexadecimal
- Parameters: characters to convert
- Return value:
    - Success: return character hexadecimal number
    - Failed: Return - 1

8, Function introduction 5

CRYPTO_set_mem_functions

```
int CRYPTO_set_mem_functions(
        void *(*m)(size_t, const char *, int),
        void *(*r)(void *, size_t, const char *, int),
        void (*f)(void *, const char *, int));
```

- Function: OpenSSL uses OpenSSL by default_ malloc(),OPENSSL_realloc(),OPENSSL_free(), but this function allows developers to design their own "memory allocation, memory reallocation, memory release" functions, and then pass the pointer to this function
- Note: crypto is allowed in the library_ set_ mem_ Functions () to replace some functions, it is recommended to replace all at once. And it's dangerous to change only the malloc() implementation
- Parameters:
    - m: Custom memory allocation function pointer. If NULL is passed, OpenSSL will be used by default_ Malloc() to allocate memory
    - r: Custom memory reallocation function pointer. If NULL is passed, OpenSSL will also be used by default_ Realloc() reallocates memory
    - f: Custom memory release function pointer. If NULL is passed, OpenSSL will also be used by default_ Free() to free memory

- Return value:
  - Success: Return 1
  - Failed: return 0 (almost always due to allocation issues)

CRYPTO_get_mem_functions

```
void CRYPTO_get_mem_functions(
         void *(**m)(size_t, const char *, int),
         void *(**r)(void *, size_t, const char *, int),
         void (**f)(void *, const char *, int));
```

- Function: fill the function pointer of "memory allocation, memory reallocation, memory release" of the current program into the pointer indicated by the corresponding parameter
- Parameters:
  - m: Pointer passed by the developer to save the function pointer of the current memory allocation
  - r: Pointer passed by the developer to save the function pointer of current memory reallocation
  - f: Pointer passed by the developer to save the function pointer of current memory release

9, Function introduction 6

CRYPTO_mem_ctrl

```
int CRYPTO_mem_ctrl(int mode);
```

- Function: this function is mainly used to control whether to record memory information during memory allocation. If you do not record memory information, you will not be able to find memory leaks
- mode parameter:
  - CRYPTO_MEM_CHECK_ON: turn on memory recording
  - CRYPTO_MEM_CHECK_OFF: turn off memory recording
- Return value:
  - Success: returns the value of the previous pattern
  - Failed: Return - 1

CRYPTO_set_mem_debug

```
int CRYPTO_set_mem_debug(int onoff);
```

- Features: the default implementation can include some debugging features (if enabled at build time). By keeping a list of all memory allocations, this increases some overhead and removes it from the list when it is free. This is most useful for identifying memory leaks. This function turns this trace on or off
- In order to have an effect, any allocation function (such as OpenSSL) must be called_ This function is called before malloc ().
- Parameters:
  - CRYPTO_MEM_CHECK_ENABLE: enable debugging
  - CRYPTO_MEM_CHECK_DISABLE: turn off debugging function
- Return value:
  - Success: Return 1
  - Failed: return 0 (almost always due to allocation issues)

10, Function introduction 7

OPENSSL_mem_debug_push

```
int OPENSSL_mem_debug_push(const char *info);
```

- Function: when checking memory, it may be useful to store additional context about the operation being performed. For example, identify field names when resolving complex data structures. This function appends an identity string to the allocation stack
- Parameter: identification string. This must be a global or other static string
- Return value:
  - Success: Return 1
  - Failed: return 0

OPENSSL_mem_debug_pop

```
int OPENSSL_mem_debug_pop(void);
```

- Function: with OPENSSL_mem_debug_push() instead, removes the identity state from the stack
- Return value:

- Success: Return 1
- Failed: return 0

11, Function introduction 8

CRYPTO_mem_leaks

```
int CRYPTO_mem_leaks(BIO *b);
```

- Function: at the end of the program, this function can be called to generate a report of all "leaked" memory and write it to the BIO specified in the parameter
- Parameters: specified BIO
- return:
  - No memory leaks: Return 1
  - Memory leak: return 0
  - Function call failed: Return - 1

CRYPTO_mem_leaks_fp

```
int CRYPTO_mem_leaks_fp(FILE *fp);
```

- Function: with crypto_ mem_ Like the leaks function, this function can be called at the end of the program to generate a report of all "leaked" memory, but it is written to a file stream
- Parameters: file pointers
- return:
  - No memory leaks: Return 1
  - Memory leak: return 0
  - Function call failed: Return - 1

CRYPTO_mem_leaks_cb

```
int CRYPTO_mem_leaks_cb(int (*cb)(const char *str, size_t len, void *u),
                        void *u);
```

- Function: same as the above two functions, but it is not written to the given BIO, and calls the callback function specified by parameter 1 for each output string, and passes the string, length and user data u to the function parameter specified by parameter 1
- Parameters:
  - cb: callback function, the parameter str is the string in the memory leak report, and the parameter len is the length of the string
  - u: The last parameter value of cb callback function
- return:
  - No memory leaks: Return 1
  - Memory leak: return 0
  - Function call failed: Return - 1

Demo case

- The following demo program applies for a memory, but it is not released at last, and then calls. CRYPTO_mem_leaks_fp()Function to output memory leak information to a file
- **below mem_test2.c Corresponding Github The links are:**https://github.com/dongyusheng/csdn-code/tree/master/openssl/mem

```
//mem_test2.c
#include <stdio.h>
#include <openssl/crypto.h>

int main()
{
    //1. Allocate memory
    char *str = (char*)OPENSSL_malloc(10);
    if(str == NULL)
    {
        printf("OPENSSL_malloc() error\n");
        return -1;
    }
    printf("OPENSSL_malloc() success\n");

    //2. The program does not release memory and ends the program directly
```

```
    //3. Write the memory leakage information to the file
    //Open the file in a mode. The file can only be written. If the file exists, the written data is attached to the
    FILE *fp = fopen("mem_test2.txt", "a");
    int ret = CRYPTO_mem_leaks_fp(fp);
    if(ret == 1)
    {
        printf("No memory leaks\n");
    }
    else if(ret == 0)
    {
        printf("There is a memory leak, please read mem_test2.txt\n");
    }
    else
    {
        printf("OPENSSL_mem_leaks_fp() error\n");
        fclose(fp);
        return -1;
    }
    fclose(fp);

    return 0;
}
```

- The compilation test is as follows. This function cannot be found during the test. I don't know why, but it exists in openssl/crypto.h

```
gcc -o mem_test2 mem_test2.c -lssl -lcrypto
```

```
ubuntu@VM-0-14-ubuntu:~/code/openssl$ gcc -o mem_test2 mem_test2.c -lssl -lcrypto
mem_test2.c: In function ‘main’:
mem_test2.c:22:15: warning: implicit declaration of function ‘CRYPTO_mem_leaks_fp’; did you mean ‘C
     int ret = CRYPTO_mem_leaks_fp(fp);
               ^~~~~~~~~~~~~~~~~~~
               CRYPTO_num_locks
/tmp/ccyd2gnx.o: In function `main':
mem_test2.c:(.text+0x6f): undefined reference to `CRYPTO_mem_leaks_fp'
collect2: error: ld returned 1 exit status
ubuntu@VM-0-14-ubuntu:~/code/openssl$
```

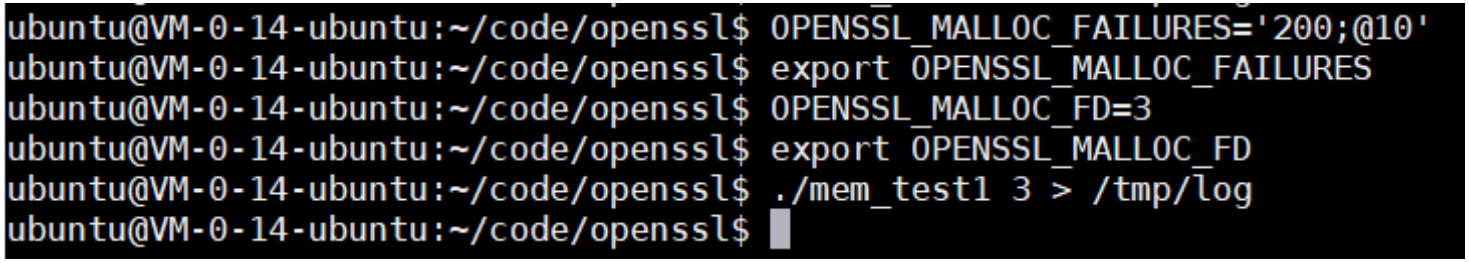
12, Function introduction 9

```
void CRYPTO_get_alloc_counts(int *m, int *r, int *f);
```

- If the library is built using the crypto mdebug option, you can also use this function and two environment variables OPENSSL_MALLOC_FAILURES and OPENSSL_MALLOC_FD
- If OpenSSL is built with the configuration option enabled, crypto mdebug is particularly suitable
- Function: the function sets the CRYPTO in the program_ malloc(),CRYPTO_realloc(),CRYPTO_ The number of calls to free() is filled in the pointer specified by the parameter
- Parameters:
  - m: User manual input, used to save crypto in the program_ The number of calls to malloc (). If it is NULL, the corresponding value is not stored
  - r: User manual input, used to save crypto in the program_ The number of calls to realloc(). If it is NULL, the corresponding value is not stored
  - f: User manual input, used to save crypto in the program_ The number of calls to free(). If it is NULL, the corresponding value is not stored
- Environment variable:
  - OPENSSL_MALLOC_FAILURES: this variable controls the frequency of allocation failures. It is a set of semicolon separated fields, each of which is a count (default is zero) and an optional atsign and percent (default is 100). If the count is zero, it will last forever. For example, 100;@25 or 100@0;0@25 Means that the first 100 assignments pass, and then all other assignments (until the program exits or crashes) have a 25% chance of failing
  - OPENSSL_MALLOC_FD: if the variable is resolved to a positive integer, it is treated as an open file descriptor and all allocated records are written to the descriptor. If the allocation fails and the platform supports the allocation, the backtracking is written to the descriptor. This is useful because

malloc can fail but cannot be checked, and problems only occur later. The following classic shell syntax example shows how to use it (not on all platforms)

```
# Set two environment variables
OPENSSL_MALLOC_FAILURES='200;@10'
export OPENSSL_MALLOC_FAILURES
OPENSSL_MALLOC_FD=3
export OPENSSL_MALLOC_FD

# Input the running information of the program into the file
...app invocation... 3>/tmp/log$$
```



Demo case

- The following shows the call of this function
- **below mem_test3.c Corresponding Github The links are:**https://github.com/dongyusheng/csdn-code/tree/master/openssl/mem

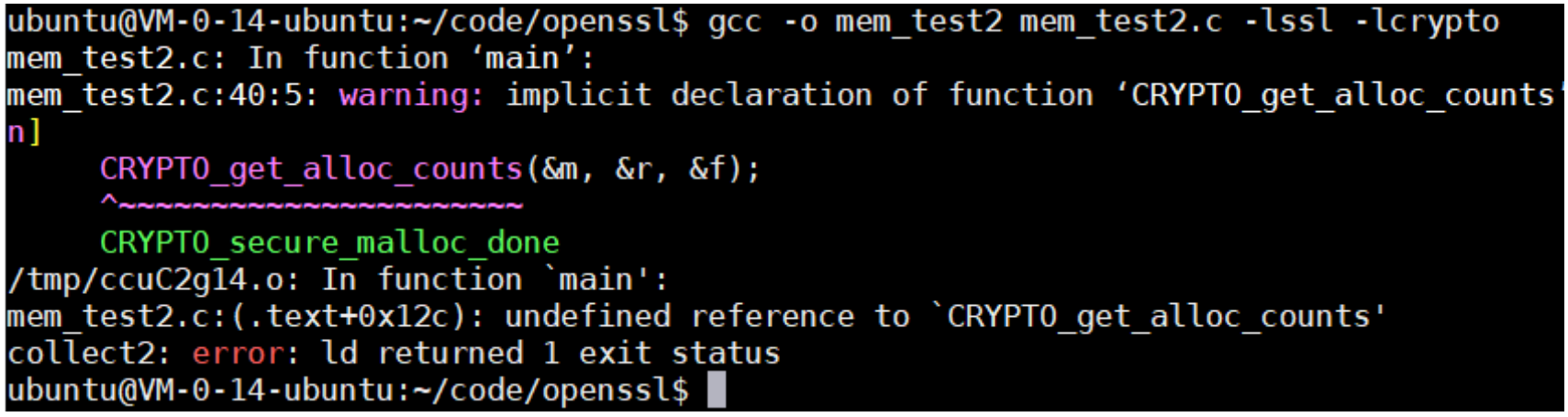
```
//mem_test3.c
#include <stdio.h>
#include <openssl/crypto.h>

int main()
{
    //1. Allocate memory
```

```
    char *str1 = (char*)OPENSSL_malloc(10);
    if(str1 == NULL)
    {
        printf("OPENSSL_malloc() error\n");
        return -1;
    }
    printf("OPENSSL_malloc() success\n");

    //2. Allocate memory
    char *str2 = (char*)OPENSSL_malloc(10);
    if(str2 == NULL)
    {
        printf("OPENSSL_malloc() error\n");
        return -1;
    }
    printf("OPENSSL_malloc() success\n");

    //3. Reallocate memory for str1
    str1 = OPENSSL_realloc(str1, 20);
    if(str1 == NULL)
    {
        printf("OPENSSL_realloc() error\n");
        return -1;
    }
    printf("OPENSSL_realloc() success\n");

    //4. Release two memory
    OPENSSL_free(str1);
    OPENSSL_free(str2);

    //5. Print memory usage information
    int m, r, f;
```

```
    CRYPTO_get_alloc_counts(&m, &r, &f);
    printf("information:\n\t, malloc count: %d\n\t, realloc count: %d\n\t, free count: %d\n",m, r, f);

    return 0;
}
```

- The compilation test is as follows. Since I did not use the crypto mdebug option when installing the OpenSSL library, I failed to call this function

```
gcc -o mem_test3 mem_test3.c -lssl -lcrypto
```



Keywords: OpenSSL github C SSL

Added by **wafflestomper** on *Tue, 16 Jun 2020 06:52:37 +0300*

Popular Keywords

Java - 1794

Database - 814

Attribute - 785

Programming - 772

Python - 717

Javascript - 697

less - 639

Spring - 623

network - 612

xml - 600

JSON - 577

Android - 574

github - 562

Linux - 505

PHP - 495

MySQL - 448

SQL - 401

encoding - 373

Mobile - 334

Windows - 324
