After I encrypt memory with CryptProtectMemory, can I move it around?

devblogs.microsoft.com/oldnewthing/20150413-00

April 13, 2015

Raymond Chen

A customer had a question about the the `CryptProtectMemory` function. After using it to encrypt a memory block, can the memory block be moved to another location and decrypted there? Or does the memory block have to be decrypted at the same location it was encrypted?

The answer is that the memory does not need to be decrypted at the same memory address at which it was encrypted. The address of the memory block is not used as part of the encryption key. You can copy or move the memory around, and as long as you don't tamper with the bytes, and you perform the decryption within the scope you specified, then it will decrypt.

That the buffer can be moved around in memory is obvious if the scope was specified as `CRYPTPROTECTMEMORY_CROSS_PROCESS` or `CRYPTPROTECTMEMORY_SAME_LOGON`, because those scopes encompass more than one process, so the memory will naturally have a different address in each process. The non-obvious part is that it also holds true for `CRYPTPROTECT-MEMORY_SAME_PROCESS`.

You can also decrypt the buffer multiple times. This is handy if you need to use the decrypted contents more than once, or if you want to hand out the encrypted contents to multiple clients, and leave each client to delay decrypting the data until immediately before they need it. (And then either re-encrypting or simply destroying the data after it is no longer needed in plaintext form.)

Today's Little Program demonstrates the ability to move encrypted data and to decrypt it more than once.

```
#include <windows.h>
#include <wincrypt.h>
#include <stdio.h> // horrors! mixing C and C++!

union MessageBuffer
{
  DWORD secret;
  char  buffer[CRYPTPROTECTMEMORY_BLOCK_SIZE];
};
static_assert(sizeof(DWORD) <= CRYPTPROTECTMEMORY_BLOCK_SIZE,
             "Need a bigger buffer");

int __cdecl main(int, char **)
{
  MessageBuffer message;

  // Generate a secret message into the buffer.
  message.secret = GetTickCount();

  printf("Shhh... the secret message is %u\n", message.secret);

  // Now encrypt the buffer.
  CryptProtectMemory(message.buffer, sizeof(message.buffer),
                     CRYPTPROTECTMEMORY_SAME_PROCESS);

  printf("You can't see it now: %u\n", message.secret);

  // Copy the buffer to a new location in memory.
  MessageBuffer copiedMessage;
  CopyMemory(copiedMessage.buffer, message.buffer,
             sizeof(copiedMessage.buffer));

  // Decrypt the copy (at a different address).
  CryptUnprotectMemory(copiedMessage.buffer,
                       sizeof(copiedMessage.buffer),
                       CRYPTPROTECTMEMORY_SAME_PROCESS);

  printf("Was the secret message %u?\n", copiedMessage.secret);

  SecureZeroMemory(copiedMessage.buffer, sizeof(copiedMessage.buffer));

  // Do it again!
  CopyMemory(copiedMessage.buffer, message.buffer,
             sizeof(copiedMessage.buffer));

  // Just to show that the original buffer is not needed,
  // let's destroy it.
  SecureZeroMemory(message.buffer, sizeof(message.buffer));

  // Decrypt the copy a second time.
  CryptUnprotectMemory(copiedMessage.buffer,
```

```
                        sizeof(copiedMessage.buffer),
                        CRYPTPROTECTMEMORY_SAME_PROCESS);

  printf("Was the secret message %u?\n", copiedMessage.secret);

  SecureZeroMemory(copiedMessage.buffer, sizeof(copiedMessage.buffer));

  return 0;
}
```

Bonus chatter: The enumeration values for the encryption scope are rather confusingly named and numbered. I would have called them

Old name	Old value	New name	New value
`CRYPTPROTECT-MEMORY_SAME_PROCESS`	0	`CRYPTPROTECT-MEMORY_SAME_PROCESS`	0
`CRYPTPROTECT-MEMORY_SAME_LOGON`	2	`CRYPTPROTECT-MEMORY_SAME_LOGON`	1
`CRYPTPROTECT-MEMORY_CROSS_PROCESS`	1	`CRYPTPROTECT-MEMORY_SAME_MACHINE`	2

I would have changed the name of the last flag to `CRYPTPROTECTMEMORY_SAME_MACHINE` for two reasons. First, the old name `CRYPTPROTECTMEMORY_CROSS_PROCESS` implies that the memory *must* travel to another process; *i.e.*, that if you encrypt with cross-process, then it must be decrypted by another process. Second, the flag name creates confusion when placed next to `CRYPTPROTECTMEMORY_SAME_LOGON`, because `CRYPTPROTECTMEMORY_SAME_LOGON` is also a cross-process scenario.

And I would have renumbered the values so that the entries are in a logical order: Higher numbers have larger scope than lower values.

Exercise: Propose a theory as to why the old names and values are the way they are.

Raymond Chen

Follow