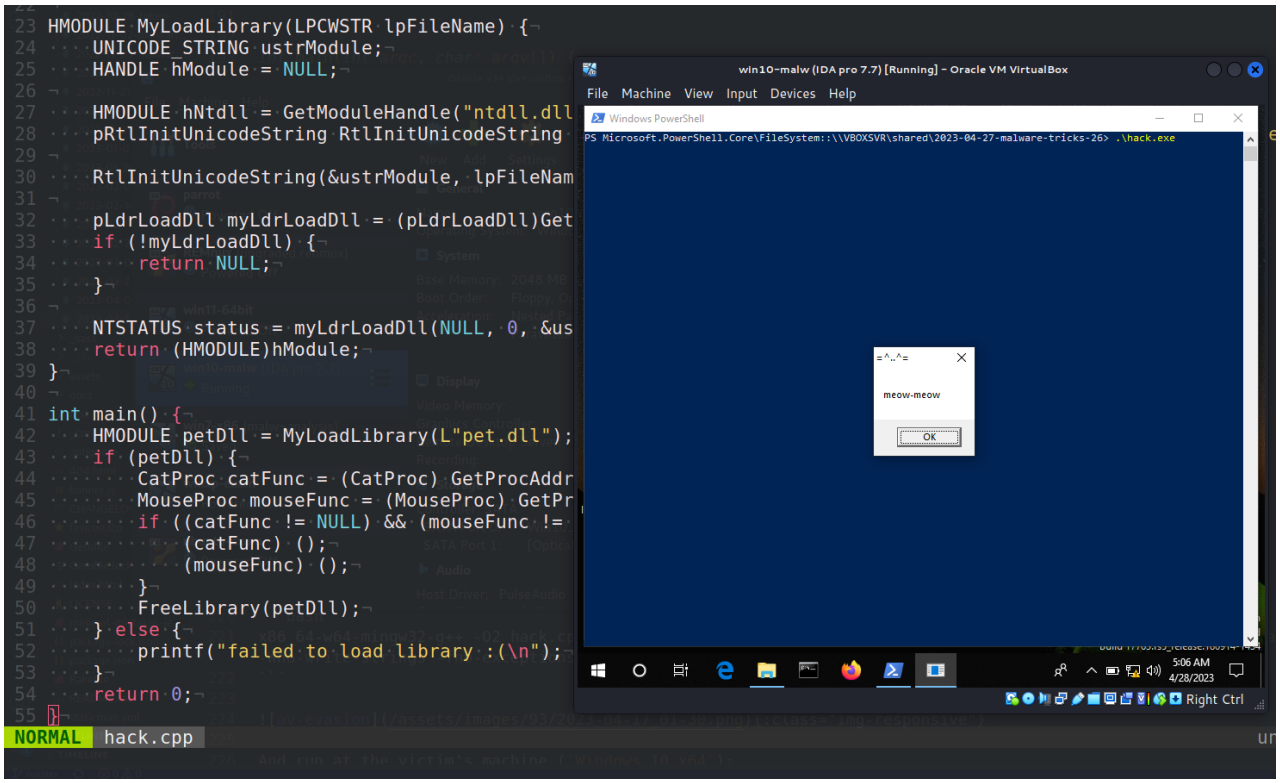
Malware development trick - part 27: WinAPI LoadLibrary implementation. Simple C++ example.

cocomelonc.github.io/malware/2023/04/27/malware-tricks-27.html

April 27, 2023

2 minute read

Hello, cybersecurity enthusiasts and white hackers!



Today, I just want to focus my research on another malware development trick: it's also helpful to AV evasion in some cases and scenarios. Like previous posts with GetModuleHandle and GetProcAddress implementations, what about my own `LoadLibrary` implementation? Let's try to do it.

LoadLibrary

`LoadLibrary` is a Windows API function that allows you to load a dynamic-link library (`DLL`) module into the address space of the calling process. The function takes the name of the DLL as an argument and returns a handle to the loaded module. If the function fails, it returns `NULL`:

```
HMODULE LoadLibraryA(
  LPCSTR lpLibFileName
);
```

`lpFileName` - A null-terminated string that specifies the name of the module (either a `.dll` or `.exe` file).

practical example

First of all, create our own `DLL`. For example something like this (`pet.c`):

```
/*
pet.dll - DLL example for LoadLibrary
*/

#include <windows.h>
#pragma comment (lib, "user32.lib")

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  ul_reason_for_call, LPVOID lpReserved)
{
  switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
      break;
    case DLL_PROCESS_DETACH:
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
  }
  return TRUE;
}

__declspec(dllexport) int _cdecl Cat() {
  MessageBox(NULL, "meow-meow", "=^..^=", MB_OK);
  return 1;
}

__declspec(dllexport) int _cdecl Mouse() {
  MessageBox(NULL, "squeak-squeak", "<:3()~", MB_OK);
  return 1;
}
```

Then, create our application which load this `DLL`. Simple implementation of `LoadLibrary` logic is looks like this:

```
/*
 * hack.c - LoadLibrary implementation. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/tutorial/2023/04/27/malware-tricks-27.html
*/
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include <winternl.h>

typedef int (__cdecl *CatProc)();
typedef int (__cdecl *MouseProc)();

typedef NTSTATUS(NTAPI *pLdrLoadDll) (
    PWCHAR PathToFile,
    ULONG Flags,
    PUNICODE_STRING ModuleFileName,
    PHANDLE ModuleHandle
);

typedef VOID (NTAPI *pRtlInitUnicodeString)(PUNICODE_STRING DestinationString, PCWSTR
SourceString);

HMODULE MyLoadLibrary(LPCWSTR lpFileName) {
    UNICODE_STRING ustrModule;
    HANDLE hModule = NULL;

    HMODULE hNtdll = GetModuleHandle("ntdll.dll");
    pRtlInitUnicodeString RtlInitUnicodeString =
(pRtlInitUnicodeString)GetProcAddress(hNtdll, "RtlInitUnicodeString");

    RtlInitUnicodeString(&ustrModule, lpFileName);

    pLdrLoadDll myLdrLoadDll =
(pLdrLoadDll)GetProcAddress(GetModuleHandle("ntdll.dll"), "LdrLoadDll");
    if (!myLdrLoadDll) {
        return NULL;
    }

    NTSTATUS status = myLdrLoadDll(NULL, 0, &ustrModule, &hModule);
    return (HMODULE)hModule;
}

int main() {
    HMODULE petDll = MyLoadLibrary(L"pet.dll");
    if (petDll) {
        CatProc catFunc = (CatProc) GetProcAddress(petDll, "Cat");
        MouseProc mouseFunc = (MouseProc) GetProcAddress(petDll, "Mouse");
        if ((catFunc != NULL) && (mouseFunc != NULL)) {
            (catFunc) ();
            (mouseFunc) ();
        }
```

```
        FreeLibrary(petDll);
    } else {
        printf("failed to load library :(\n");
    }
    return 0;
}
```

This implementation utilizes the undocumented `LdrLoadDll` function:

```
NTSYSAPI
NTSTATUS
NTAPI
LdrLoadDll(
  IN PWCHAR               PathToFile OPTIONAL,
  IN ULONG                Flags OPTIONAL,
  IN PUNICODE_STRING      ModuleFileName,
  OUT PHANDLE             ModuleHandle );
```

which is part of the `ntdll.dll` library. Using the `LdrLoadDll` function, we can manually load a library by providing the necessary parameters.

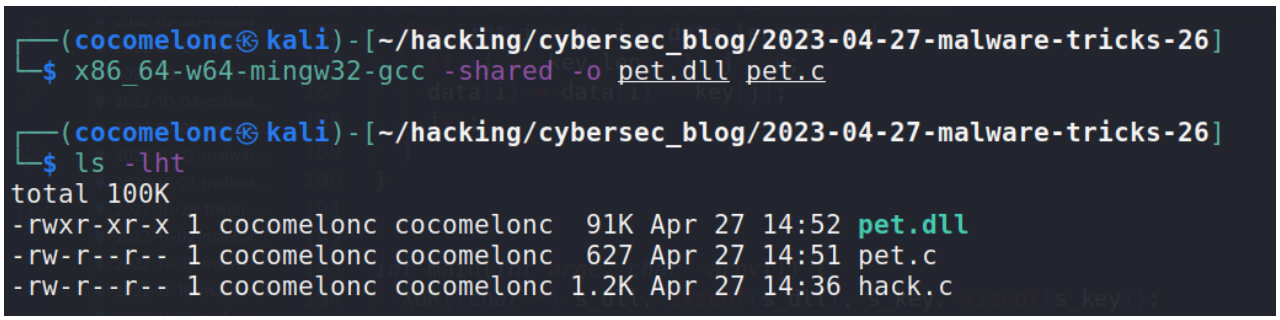
Ok, can we use our `DLL` in other applications by loading it with `MyLoadLibrary`? Yes, we can. Also, as you can see, we can getting the function addresses using `GetProcAddress`.

Also, we use the `RtlInitUnicodeString` function pointer instead of the direct function call.

demo

Ok, let's go to see everything in practice. First of all compile our `pet.c`:

```
x86_64-w64-mingw32-gcc -shared -o pet.dll pet.c
```



Then, compile our "malware" application (`hack.cpp`):

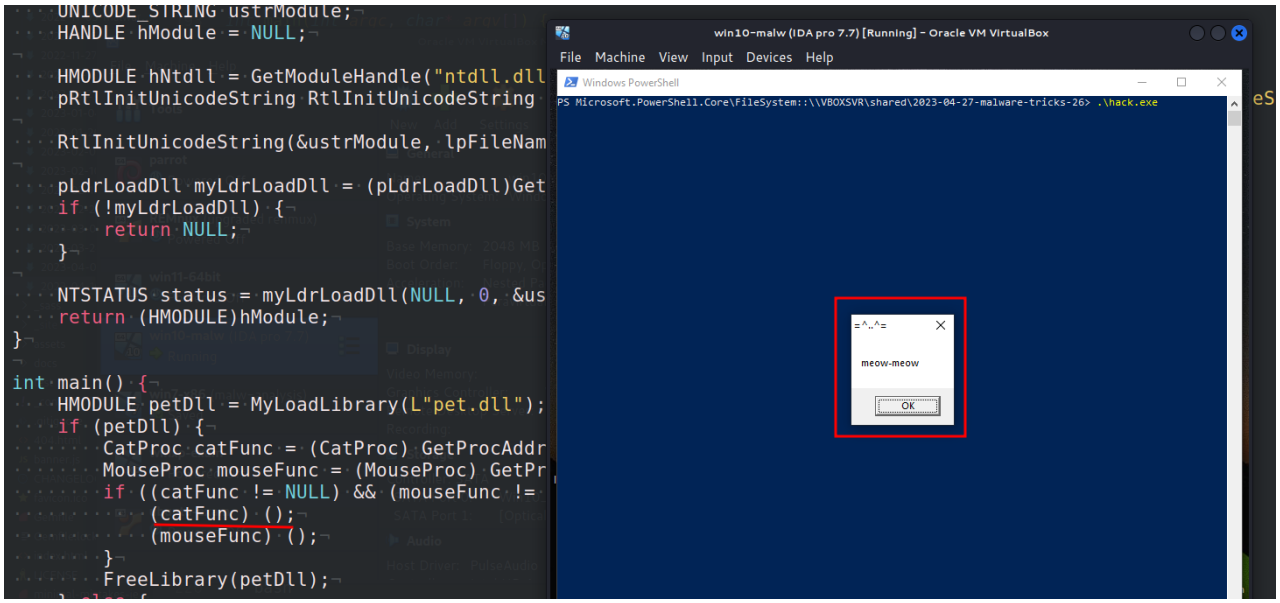
```
x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

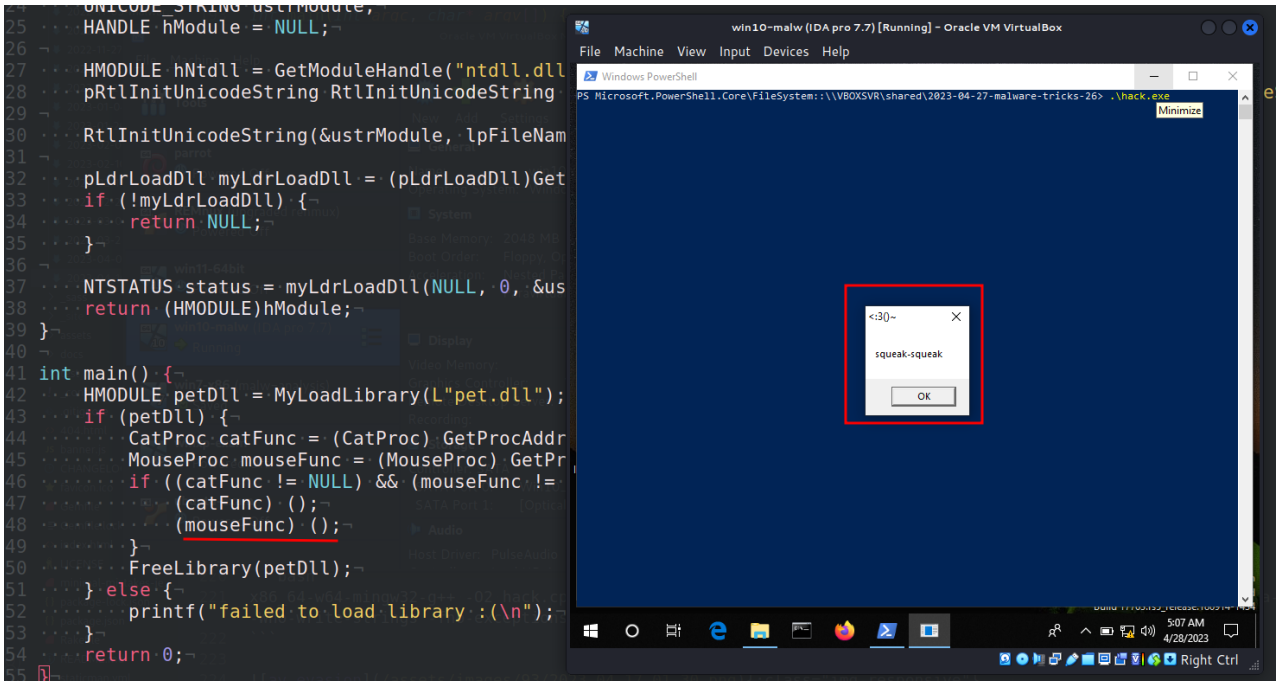
```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-04-27-malware-tricks-26]
└─$ x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
In file included from hack.cpp:9:
/usr/share/mingw-w64/include/winternl.h:1122:14: warning: 'void RtlUnwind(PVOID, PVOID, PEXCEPTION_RECORD, PVOID)' redeclared without dllimport attribute: previous dllimport ignored [-Wattributes]
 1122 |   VOID NTAPI RtlUnwind (PVOID TargetFrame,PVOID TargetIp,PEXCEPTION_RECORD ExceptionRecord,PVOID ReturnValue);
      |              ^~~~~~~~

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-04-27-malware-tricks-26]
└─$ ls -l
total 140
-rw-r--r-- 1 cocomelonc cocomelonc  1591 Apr 27 15:21 hack.cpp
-rwxr-xr-x 1 cocomelonc cocomelonc 39936 Apr 27 15:21 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc   627 Apr 27 14:51 pet.c
-rwxr-xr-x 1 cocomelonc cocomelonc 92775 Apr 27 14:52 pet.dll
```

Finally, run it at the victim's machine (`Windows 10 x64`):

```
.\hack.exe
```





As you can see, everything is worked perfectly! =^..^=

I hope this post spreads awareness to the blue teamers of this interesting malware dev technique, and adds a weapon to the red teamers arsenal.

LoadLibrary
GetProcAddress
GetModuleHandle
RtlInitUnicodeString
source code in github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine