

ورقات بيضاء في المختبر

WHITE PAPERS in ViTRO

(06/2020) #03



سلسلة

نظرة و تجربة



موضوع اليوم

CODE it & CRACK it

احد الأفكار التي تساهم بشكل جيد في تعلم الهندسة العكسية و تحليل البرمجيات هي القدرة على انشاء تطبيقات تنجز مهام معينة ثم تحليل الناتج النهائي داخل المنقح.

و بناءا على ذلك، سوف نبدأ بتجسيد هذه الفكرة بتطبيق المراحل التالية:

- بلورة فكرة التطبيق.
- ترجمتها الى أوامر.
- وأخيرا محاولة فهمها داخل المنقح.

بلورة فكرة التطبيق:

بعد التفكير في عدة طرق ارتأيت الى فكرة انجاز تطبيقين، التطبيق الأول يحتوي على خوارزمية بسيطة جدا تطلب ادخال معلومات التسجيل يكون فهمها سهل و في متناول الجميع.

و التطبيق الثاني يكون عبارة عن "حامل" للتطبيق الأول و يشغله في الذاكرة اعتمادا على هيكل نفسه «RunPE»

مع اعتماد تشفير بسيط جدا و واضح للتطبيق الأول «xor 1»
و وضع شرط تحققه يتيح عملية فك التشفير «GetDriveTypeA»

مخطط الفكرة:
لتوضيح المخطط قمت بتسمية التطبيق الأول: challenge
والتطبيق الثاني: blackbox



```
#include <windows.h>
#include <stdio.h>

#define TRIAL "Trial..."
#define PERSONAL "Personal..."
#define PROFESSIONAL "Professional..."
#define ENTERPRISE "Enterprise..."
#define SKILLED "Skilled..."
#define BAD "Invalid..."

int main(int argc, char *argv[]) {
    char Msg[20] = BAD;

    printf("[ReverseMe Challenge]\n[by YANiS]\n\nYour Registration Details: ");

    if (argc > 1) {
        DWORD len = 0 ;

        if (CryptStringToBinaryA(argv[1],
                                strlen(argv[1]),
                                CRYPT_STRING_BASE64 | CRYPT_STRING_BASE64HEADER,
                                NULL,
                                &len,
                                NULL,
                                NULL)) {
```

ترجمتها الى أوامر:

أوامر التطبيق Challenge:

```
if (len < 5) {  
    strcpy(Msg, BAD);  
}  
  
if ((len > 12) & (len < 21))  
    strcpy(Msg, PERSONAL);  
  
if ((len > 64) & (len < 66))  
    strcpy(Msg, SKILLED);  
  
if ((len > 29) & (len < 31))  
    strcpy(Msg, ENTERPRISE);  
  
if ((len > 6) & (len < 10))  
    strcpy(Msg, TRIAL);  
  
if ((len > 32) & (len < 42))  
    strcpy(Msg, PROFESSIONAL);  
} else  
    strcpy(Msg, BAD);  
}  
printf("%s\n", Msg);  
getchar();  
return 0;  
}
```

نلاحظ ان أوامر التطبيق واضحة، وهي تقوم بأخذ ما يتم تمريره
من المستخدم و معالجته من خلال دالة CryptStringToBinaryA
ثم تأخذ فقط طول الناتج ومقارنته بقيم ثابتة وعرض ناتج
المقارنة للمستخدم.

```
#include <windows.h>
```

```
unsigned char rawData[2560] = {  
    0x4C, 0x5B, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x91, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x2F, 0x75, 0x64, 0x79, 0x75, 0x01, 0x01, 0x01,  
    0xF1, 0x07, 0x01, 0x01, 0x01, 0x11, 0x01, 0x01, 0x01, 0x09, 0x01, 0x01, 0x01, 0x03, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x21, 0x01, 0x01, 0xE1,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x11  
};
```

ترجمتها الى أوامر:

أوامر التطبيق blackbox:

حفظ التطبيق challenge في جدول معطيات بعد تشفيره ب 1 xor

```

void RunPE(LPSTR pFileName, PVOID pRawData) {
    PIMAGE_DOS_HEADER pImageDosHeader;
    PIMAGE_NT_HEADERS pImageNtHeader;
    PIMAGE_SECTION_HEADER pImageSectionHeader;
    PROCESS_INFORMATION ProcessInfo;
    STARTUPINFOA StartupInfo;
    PCONTEXT pContext;
    PDWORD dwImageBase;
    LPVOID pImageBase;
    int iNumberOfSections;

    if (DRIVE_REMOVABLE == GetDriveTypeA(NULL)) {
        for (int x=0; x<2560; x++)
            rawData[x] = rawData[x] ^ 1;
    }

    pImageDosHeader = PIMAGE_DOS_HEADER(pRawData);

    if (pImageDosHeader->e_magic == IMAGE_DOS_SIGNATURE) {
        pImageNtHeader = PIMAGE_NT_HEADERS(DWORD(pRawData) + pImageDosHeader->e_lfanew);

        if (pImageNtHeader->Signature == IMAGE_NT_SIGNATURE) {
            RtlZeroMemory(&StartupInfo,
                sizeof(StartupInfo));
            RtlZeroMemory(&ProcessInfo,
                sizeof(ProcessInfo));

            if (CreateProcessA(pFileName,
                NULL,
                NULL,
                NULL,
                FALSE,
                CREATE_SUSPENDED,
                NULL,
                NULL,
                &StartupInfo,
                &ProcessInfo)) {

```

أوامر الـ RunPE:

آلية "الران بي" معروفة و مكشوفة من طرف برامج الحماية و هي مبنية على انشاء "عملية" انطلاقا من ملف تنفيذي مستهدف و إيقاف تنفيذها و استبدال محتوي هيكلها بمحتوي تنفيذي آخر ثم استئناف تنفيذ العملية.


```

pContext = PCONTEXT(VirtualAlloc(NULL,
    sizeof(pContext),
    MEM_COMMIT,
    PAGE_READWRITE));

pContext->ContextFlags = CONTEXT_FULL;
if (GetThreadContext(ProcessInfo.hThread,
    LPCONTEXT(pContext))) {
    ReadProcessMemory(ProcessInfo.hProcess,
        LPCVOID(pContext->Ebx + 8),
        LPVOID(&dwImageBase),
        4,
        NULL);

    pImageBase = VirtualAllocEx(ProcessInfo.hProcess,
        LPVOID(pImageNtHeader->OptionalHeader.ImageBase),
        pImageNtHeader->OptionalHeader.SizeOfImage,
        0x3000,
        PAGE_EXECUTE_READWRITE);

    if (pImageBase) {
        WriteProcessMemory(ProcessInfo.hProcess,
            pImageBase,
            pRawData,
            pImageNtHeader->OptionalHeader.SizeOfHeaders,
            NULL);

        for (iNumberOfSections = 0; iNumberOfSections < pImageNtHeader->FileHeader.NumberOfSections; iNumberOfSections++) {
            pImageSectionHeader = PIMAGE_SECTION_HEADER(DWORD(pRawData) + pImageDosHeader->e_lfanew + 248 + (iNumberOfSections * 40));
        }
    }
}

```

```

        WriteProcessMemory(ProcessInfo.hProcess,
                            LPVOID(DWORD(pImageBase) + pImageSectionHeader->VirtualAddress),
                            LPVOID(DWORD(pRawData) + pImageSectionHeader->PointerToRawData),
                            pImageSectionHeader->SizeOfRawData,
                            NULL);
    }

    WriteProcessMemory(ProcessInfo.hProcess,
                        LPVOID(pContext->Ebx + 8),
                        LPVOID(&pImageNtHeader->OptionalHeader.ImageBase),
                        4,
                        NULL);

    pContext->Eax = DWORD(pImageBase) + pImageNtHeader->OptionalHeader.AddressOfEntryPoint;
    SetThreadContext(ProcessInfo.hThread,
                      LPCONTEXT(pContext));

    ResumeThread(ProcessInfo.hThread);
}
}
}
}
VirtualFree(pRawData, 0, MEM_RELEASE);
}

```

و يتم استدعاء الدالة كما يلي:

```

void main() {
    SetConsoleTitleA("[BLACKBOX 2020]");
    RunPE("blackbox.exe", rawData);
}

```

نظرة داخل المنقح:

إشارة الى النقاط المهمة في التطبيق Challenge.

في الصورة التالية لاحظ النقاط المهمة (المحدد بأسهم):

- فحص اذا تم ادخال معلومات التسجيل عن طريق الـ Command line (السهم 01)

- استدعاء الدالة CryptStringToBinaryA (السهم 02)

- فحص القيم المرجعة بعد المعالجة (CryptStringToBinaryA) ومنها توجيه التنفيذ حسب

تطابقها بالقيم الثابتة موضوع المقارنة (الأسهم 03)

نظرة داخل المنقح:

إشارة الى النقاط المهمة في التطبيق blackbox.

في الصورة التالية لاحظ النقاط المهمة (المحددة بأسهم):

- استدعاء دالة `GetDriveTypeA` مع تمرير قيمة `NULL` كباراميتر للحصول على نوع السواقة التي تم

تشغيل التطبيق عليها (السهم 01)

- فحص القيمة المرجعة ومقارنتها بالقيمة الثابتة 2 والتي تمثل `RIVE_REMOVABLE` (السهم 02)

- أوامرك التشفير xor بالقيمة الثابتة 1 (الأسهم 03 و 04)


```

;-- section..text:
;-- eip:
403: fcn.00401000 (LPCSTR lpApplicationName, int32_t arg_ch);
; var LPSTARTUPINFOA lpStartupInfo @ ebp-0x64
; var LPPROCESS_INFORMATION lpProcessInformation @ ebp-0x20
; var HANDLE hThread @ ebp-0x1c
; var LPVOID *lpBuffer @ ebp-0x10
; var LPVOID var_ch @ ebp-0xc
; var signed int var_8h @ ebp-0x8
; var int32_t var_4h @ ebp-0x4
; arg LPCSTR lpApplicationName @ ebp+0x8
; arg int32_t arg_ch @ ebp+0xc
push     ebp                                ; [00] -r-x section size 4096 named .text
mov      ebp, esp
sub      esp, 0x64
push     ebx
push     edi
xor      edi, edi
push     edi                                ; LPCSTR lpRootPathName
call     dword [GetDriveTypeA]             ; 0x402028 ; UINT GetDriveTypeA(LPCSTR lpRootPathName)
cmp      eax, 2                             ; 2
jne      0x401027

```

```

xor      eax, eax

```

```

xor      byte [eax + section..data], 1 ; 0x403000
inc      eax
cmp      eax, 0xa00                        ; 2560
jnl      0x401018

```

```

mov      ebx, dword [arg_ch]
mov      eax, 0x5a4d                        ; 'MZ'
cmp      word [ebx], ax
jne      0x401182

```

```

push     esi
mov      esi, dword [ebx + 0x3c]
add      esi, ebx
cmp      dword [esi], 0x4550
jne      0x401181

```

```

push     0x44                                ; eflags ; size_t n
lea      eax, [lpStartupInfo]
push     edi                                ; int c
push     eax                                ; void *s
call     sub.msvcrt.dll_memset              ; void *memset(void *s, int c, size_t n)
push     0x10                                ; 16 ; size_t n
lea      eax, [lpProcessInformation]
push     edi                                ; int c
push     eax                                ; void *s
call     sub.msvcrt.dll_memset              ; void *memset(void *s, int c, size_t n)
add      esp, 0x18
lea      eax, [lpProcessInformation]
push     eax                                ; LPPROCESS_INFORMATION lpProcessInformation
lea      eax, [lpStartupInfo]
push     eax                                ; LPSTARTUPINFOA lpStartupInfo
push     edi                                ; LPCSTR lpCurrentDirectory
push     edi                                ; LPVOID lpEnvironment
push     4                                  ; 4 ; DWORD dwCreationFlags
push     edi                                ; BOOL bInheritHandles
push     edi                                ; LPSECURITY_ATTRIBUTES lpThreadAttributes
push     edi                                ; LPSECURITY_ATTRIBUTES lpProcessAttributes
push     edi                                ; LPSTR lpCommandLine
push     dword [lpApplicationName]          ; LPCSTR lpApplicationName
call     dword [CreateProcessA]             ; 0x402024 ; BOOL CreateProcessA(LPCSTR lpApplicationName, LPS...
test     eax, eax
je       0x401181

```

```

push     4                                  ; 4 ; DWORD flProtect
push     0x1000                             ; DWORD flAllocationType
push     4                                  ; 4 ; SIZE_T dwSize
push     edi                                ; LPVOID lpAddress
call     dword [VirtualAlloc]               ; 0x402020 ; LPVOID VirtualAlloc(LPVOID lpAddress, SIZE_T dwSi...
mov      ebx, eax
push     ebx                                ; LPCONTEXT lpContext
mov      dword [ebx], 0x10007
push     dword [hThread]                    ; HANDLE hThread
call     dword [GetThreadContext]           ; 0x40201c ; BOOL GetThreadContext(HANDLE hThread, LPCONTEXT l...
test     eax, eax
je       0x401175

```

```

push     edi                                ; SIZE_T *lpNumberOfBytesRead
push     4                                  ; 4 ; SIZE_T nSize
lea      eax, [lpBuffer]
push     eax                                ; LPVOID lpBuffer
mov      eax, dword [ebx + 0xa4]
add      eax, 8
push     eax                                ; LPCVOID lpBaseAddress
push     dword [lpProcessInformation]        ; HANDLE hProcess
call     dword [ReadProcessMemory]          ; 0x402000 ; BOOL ReadProcessMemory(HANDLE hProcess, LPCVOID l...
push     0x40                                ; '0' ; 64 ; DWORD flProtect
push     0x3000                             ; DWORD flAllocationType
push     dword [esi + 0x50]                  ; SIZE_T dwSize
push     dword [esi + 0x34]                  ; LPVOID lpAddress
push     dword [lpProcessInformation]        ; HANDLE hProcess
call     dword [VirtualAllocEx]             ; 0x402018 ; LPVOID VirtualAllocEx(HANDLE hProcess, LPVOID lpA...
mov      dword [var_ch], eax
cmp      eax, edi
je       0x401175

```

```

push     edi
push     dword [esi + 0x54]
mov      edi, dword [WriteProcessMemory] ; 0x402014
push     dword [arg_ch]
push     eax
push     dword [lpProcessInformation]
call     edi
and      dword [var_8h], 0
xor      eax, eax
cmp      ax, word [esi + 6]
jae      0x401146

```

```

and      dword [var_4h], eax

```

```

mov      ecx, dword [arg_ch]
mov      eax, dword [var_4h]
mov      edx, dword [ecx + 0x3c]
add      eax, ecx
lea      eax, [eax + edx + 0xf8]
mov      edx, dword [eax + 0x14]
push     0
mov      eax, dword [eax + 0x10]
mov      eax, dword [eax + 0xc]
add      eax, dword [var_ch]
add      edx, ecx
push     edx
push     eax
push     dword [lpProcessInformation]
call     edi
movzx    eax, word [esi + 6]
inc      dword [var_8h]
add      dword [var_4h], 0x28                ; 40
cmp      dword [var_8h], eax
jnl      0x40110d

```

```

push     0
push     4                                  ; 4
lea      eax, [esi + 0x34]
push     eax
mov      eax, dword [ebx + 0xa4]
add      eax, 8
push     eax
push     dword [lpProcessInformation]
call     edi
mov      eax, dword [esi + 0x28]
add      eax, dword [var_ch]
push     ebx                                ; const CONTEXT *lpContext
mov      eax, [ebx + 0xb0], eax
push     dword [hThread]                    ; HANDLE hThread
call     dword [SetThreadContext]          ; 0x402010 ; BOOL SetThreadContext(HANDLE hThread, const CONTE...
xor      edi, edi

```

```

push     dword [hThread]                    ; HANDLE hThread
call     dword [ResumeThread]              ; 0x40200c ; DWORD ResumeThread(HANDLE hThread)
mov      ebx, dword [arg_ch]

```

```

pop      esi

```

```

push     0x8000                             ; DWORD dwFreeType
push     edi                                ; SIZE_T dwSize
push     ebx                                ; LPVOID lpAddress
call     dword [VirtualFree]               ; 0x402008 ; BOOL VirtualFree(LPVOID lpAddress, SIZE_T dwSize,...
pop      edi
pop      ebx
leave
ret

```