



Название:	Чужая память
Категория:	Реверс-инжиниринг
Уровень:	Средняя
Очки:	400
Описание:	Чужая память, как мёд, всегда "вкуснее"
Теги:	Антиотладка, Внедрение данных в процесс
Автор:	ROP

Прохождение:

Смотрим, что делает исполняемый файл через декомпилятор.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    PROCESSENTRY32 pe; // [rsp+40h] [rbp-40h] BYREF
    LPVOID lpBaseAddress; // [rsp+170h] [rbp+F0h]
    HANDLE hProcess; // [rsp+178h] [rbp+F8h]
    HANDLE hSnapshot; // [rsp+180h] [rbp+100h]
    unsigned int i; // [rsp+188h] [rbp+108h]
    int v9; // [rsp+18Ch] [rbp+10Ch]

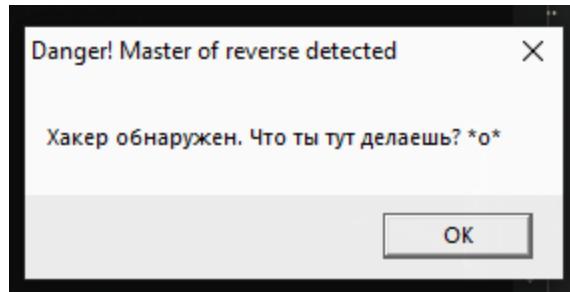
    sub_401970(argc, argv, envp);
    hSnapshot = CreateToolhelp32Snapshot(2u, 0);
```

```
if ( hSnapshot == (HANDLE)-1i64 )
    return 1;
pe.dwSize = 304;
v9 = 0;
if ( Process32First(hSnapshot, &pe) )
{
    while ( 1 )
    {
        if ( !strcmp(pe.szExeFile, "ida64.exe") )
        {
            MessageBoxA(0i64, &Text, "Danger! Master of reverse detected", 0);
            goto LABEL_11;
        }
        if ( !strcmp(pe.szExeFile, "ida.exe") )
        {
            MessageBoxA(0i64, &byte_405070, "Danger! Master of reverse detecte
d", 0);
            goto LABEL_11;
        }
        if ( !strcmp(pe.szExeFile, "notepad.exe") )
            break;
        if ( !Process32Next(hSnapshot, &pe) )
            goto LABEL_11;
    }
    v9 = 1;
}
LABEL_11:
CloseHandle(hSnapshot);
if ( !v9 )
    return 1;
hProcess = OpenProcess(0x1F0FFFu, 0, pe.th32ProcessID);
if ( !hProcess )
    return 1;
lpBaseAddress = VirtualAllocEx(hProcess, 0i64, 4ui64, 0x3000u, 0x40u);
if ( lpBaseAddress )
{
```

```
for ( i = 0; i <= 0x5C; ++i )
    byte_404020[i] -= 48;
if ( WriteProcessMemory(hProcess, lpBaseAddress, byte_404020, 0x5Dui6
4, 0i64)
    && CreateRemoteThread(hProcess, 0i64, 0i64, (LPTHREAD_START_ROUTI
NE)lpBaseAddress, 0i64, 0, 0i64) )
{
    VirtualFreeEx(hProcess, lpBaseAddress, 0i64, 0x8000u);
    CloseHandle(hProcess);
    return 0;
}
else
{
    VirtualFreeEx(hProcess, lpBaseAddress, 0i64, 0x8000u);
    CloseHandle(hProcess);
    return 1;
}
}
else
{
    CloseHandle(hProcess);
    return 1;
}
}
```

Данный код "перебирает" запущенные процессы и ищет `notepad.exe`. Затем он выделяет 0x8000 байт памяти в этом процессе и записывает дешифрованный массив `byte_404020` в неё. В конце создаётся поток, который начинает выполняться с этой памяти.

Если запущены процессы `ida64.exe` или `ida.exe` выводятся предупреждения и программа завершается.



Проверки можно обойти, пропатчив имя процессов `ida64.exe` и `ida.exe` на другое.

Затем попадаем на код, что запустит поток в `notepad.exe`.

```
.text:0000000004017F0 loc_4017F0:
.text:0000000004017F0 mov    rdx, [rbp+110h+lpBaseAddress]
.text:0000000004017F7 mov    rax, [rbp+110h+Process]
.text:0000000004017FE mov    [rsp+190h+lpThreadId], 0 ; lpThr
.text:000000000401807 mov    [rsp+190h+dwCreationFlags], 0 ;
.text:00000000040180F mov    qword ptr [rsp+190h+f1Protect], 0
.text:000000000401818 mov    r9, rdx           ; lpStartAddress
.text:00000000040181B mov    r8d, 0            ; dwStackSize
.text:000000000401821 mov    edx, 0            ; lpThreadAttrib
.text:000000000401826 mov    rcx, rax           ; hProcess
.text:00000000040182D mov    rax, cs:_imp_CreateRemoteThread
.text:00000000040183B call   rax : _imp_CreateRemoteThread
.text:000000000401832 test   rax, rax
.text:000000000401835 jnz    short loc_401877

40_389) 00000BF0 0000000004017F0: main:loc_4017F0 (Synchronized with RIP)

48 8B 95 F0 00 00 00 48 BB 85 F8 00 00 00 48 C7 H.....H.....H.
44 24 30 00 00 00 00 C7 44 24 28 00 00 00 00 48 D$0....$(...H.
C7 44 24 20 00 00 00 00 49 89 D1 41 B8 00 00 ..$.....I.....
08 BA 00 00 00 00 48 89 C1 48 88 05 18 7A 00 00 .....H.....z...
FF D0 48 85 C0 75 40 48 BB 95 F0 00 00 00 48 88 .....@H.....H.
85 F8 00 00 00 41 B9 00 80 00 00 41 B8 00 00 00 .....A.....A....
```

В `lpBaseAddress` будет адрес нужной памяти в `notepad.exe`.



Присоединяемся к процессу и переходим по нужному адресу.

IDA View-RIP

```
debug041:0000028E30770000 ; Segment permissions: Read/Write/Execute
debug041:0000028E30770000 debug041 segment byte public 'CODE' use64
debug041:0000028E30770000 assume cs:debug041
debug041:0000028E30770000 ;org 28E30770000h
debug041:0000028E30770000 assume es:ntdll_dll, ss:ntdll_dll, ds:ntdll_dll, fs:nothing, gs:nothing
debug041:0000028E30770000 lea    rbx, unk_28E3077002E
debug041:0000028E30770007 mov    rcx, 0
debug041:0000028E3077000E mov    rcx, 0
debug041:0000028E30770015 mov    rdi, 2Fh ; /*
debug041:0000028E3077001C loc_28E3077001C:
debug041:0000028E3077001C sub    byte ptr [rbx+rcx], 80h ; CODE XREF: debug041:0000028E30770026!
debug041:0000028E30770020 inc    rcx
debug041:0000028E30770023 cmp    rcx, rdi
debug041:0000028E30770026 jnz    short loc_28E3077001C
debug041:0000028E30770028 nop
debug041:0000028E3077002B ; UNKNOWN 0000028E30770000: debug041:0000028E30770000 (Synchronized with RIP)
```

Представляем байты как код, создаём точку останова на первой инструкции, выполняем процесс далее и выполняем код `task.exe` далее.

IDA View-RIP

```
RAX debug041:0000028E30770000 ;org 28E30770000h
RDX debug041:0000028E30770000 assume es:ntdll_dll, ss:ntdll_dll, ds:ntdll_dll, fs:nothing, gs:nothing
RIP debug041:0000028E30770000 lea    rbx, unk_28E3077002E
R9 debug041:0000028E30770007 mov    rcx, 0
R8 debug041:0000028E3077000E mov    rcx, 0
R10 debug041:0000028E30770015 mov    rdi, 2Fh ; /*
R11 debug041:0000028E3077001C loc_28E3077001C:
R12 debug041:0000028E3077001C sub    byte ptr [rbx+rcx], 80h ; CODE XREF: debug041:0000028E30770026!
R13 debug041:0000028E30770020 inc    rcx
R14 debug041:0000028E30770023 cmp    rcx, rdi
R15 debug041:0000028E30770026 jnz    short loc_28E3077001C
R16 debug041:0000028E30770028 nop
R17 debug041:0000028E3077002B ; UNKNOWN 0000028E3077001C: debug041:loc_28E3077001C (Synchronized with RIP)
```

General registers

RAX 0000028E30770000	debug041:0000028E30770000
RBX 0000000000000000	...
RCX 0000000000000000	...
RDX 0000028E30770000	debug041:0000028E30770000
RSI 0000000000000000	...
RDI 0000000000000000	...
RBP 0000000000000000	...
RSP 000000CA0610F7F8	Stack[0000086C]:0000000000000000
RIP 0000028E30770000	Debug041:0000028E30770000
R8 0000000000000000	...
R9 0000028E30770000	debug041:0000028E30770000
R10 0000000051C60EE000	...
R11 FFFFFFFFFFFFFF	...
R12 0000000000000000	...
R13 0000000000000000	...
R14 0000000000000000	...
R15 0000000000000000	...
EFL 00000247	...

Выполняем весь этот код.

Далее будет дешифрованный флаг.