Ring0 Crackme

deroko/ARTeam

Version 1.01 – October 2006

Keywords

Ring0

1. Abstract

This time, in new article, I'm not going to talk about any comercial protection, this tame I will talk about one interesting crackme from www.crackmes.de which was submitted by Ms-Rem. This crackme has a lots of nice features that make it very very interesting, some of them are IDT/SDT hooks, rootkit tricks to hide process, usage of \Device\PhysicalMemory, 2 process execution, APC, etc…

In this tutorial I'll cover most of stuff handled by crackme and show you my steps for bypassing it. Also you will see some neet tricks when it comes to softice anti J , also this will maybe help you to understand how SoftICE can coexist with drivers that hook IDT. Anyway, before we start I wanna say that this crackme is real master piece, and big tnx to Ms-Rem for this crackme.

Tools that I'm using here are:

- SoftICE
- IDA
- LiveKd
- wARK
- LordPE
- Tasm32 ß the best asm compiler ever

S verom u Boga, deroko/ARTeam

2. Know your enemy

Here is description of crackme published by Ms-Rem at www.crackmes.de:

```
This crackme use packing and ring0 tricks for hiding serial check code.
Crackme install driver dxapi32.sys.
Recommend to run this crackme on WMVare that will save you from problems with
fall of the system.
```

So we already know the name of driver, starting crackme.exe for the first time will endup with MessageBox informing you that you have to reboot your system(driver is installed), so we restart system and after that crackme runs without a problem. Since we are dealing with driver we have to check our SDT and IDT just to be sure that those are ok before we start SoftICE.

SDT hooks:

```
KeServiceDescriptorTable               : 0x80552B80
KeServiceDescriptorTalbe.ServiceTable  : 0x804DBD20
KeServiceDescriptorTable.ServiceLimit  : 0x0000011C

SSDT entry 0019 hooked by : 0xF728AC9C - NtClose
SSDT entry 00BA hooked by : 0xF728AD70 - NtReadVirtualMemory
SSDT entry 0115 hooked by : 0xF728AD02 - NtWriteVirtualMemory

[!] Hooked Native Apis : 3
```

IDT hooks are only seen when crackme is loaded:

```
01h   0008:F728A7BC     Interrupt   32 bit      00    1
03h   0008:F728A78A     Interrupt   32 bit      03    1
```

Next thing is to analyze dxapi32.sys and get as much as possible informations about it. So we will move our efforts to new driver, entry point of driver seems normal, sets handlers for IRP_MJ_CREATE and IRP_MJ_WRITE, hooks SDT and creates hook check thread:

```
.text:00011744                call    native_api_services
.text:00011749                mov     edi, eax
.text:0001174B                test    edi, edi
.text:0001174D                jnz     short loc_11791
.text:0001174F                mov     esi, [ebp+DriverObject]
.text:00011752                push    esi             ; DriverObject
.text:00011753                call    create_device
.text:00011758                call    HookNativeApis
.text:0001175D                push    edi             ; StartContext
.text:0001175E                push    offset StartRoutine ; StartRoutine
.text:00011763                push    edi             ; ClientId
.text:00011764                push    edi             ; ProcessHandle
.text:00011765                push    edi             ; ObjectAttributes
.text:00011766                push    1F03FFh         ; DesiredAccess
.text:0001176B                lea     eax, [ebp+ThreadHandle]
.text:0001176E                push    eax             ; ThreadHandle
.text:0001176F                call    ds:PsCreateSystemThread

...

.text:00011784                mov     eax, offset CreateWriteHandler
.text:00011789                mov     [esi+48h], eax  ; IRP_MJ_WRITE
.text:0001178C                mov     [esi+38h], eax  ; IRP_MJ_CREATA
```

After that driver entrypoint is finished. We will examine most important parts of this initialization process.

```
.text:00010DDE native_api_services proc near
.text:00010DDE                 mov     ecx, ds:NtBuildNumber
.text:00010DE4                 movzx   ecx, word ptr [ecx]
.text:00010DE7                 xor     eax, eax
.text:00010DE9                 cmp     ecx, 893h
.text:00010DEF                 jz      short win_2k
.text:00010DF1                 cmp     ecx, 0A28h
.text:00010DF7                 jz      short win_xp
.text:00010DF9                 mov     eax, 0C0000002h
.text:00010DFE                 retn
.text:00010DFF
.text:00010DFF
.text:00010DFF win_xp:
.text:00010DFF                 mov     dword_11B24, 7Ah
.text:00010E09                 mov     NtClose_ServiceNum, 19h
.text:00010E13                 mov     NtWVM_ServiceNumber, 115h
.text:00010E1D                 mov     NtRVM_ServiceNumber, 0BAh
.text:00010E27                 mov     dword_11B20, 84h
.text:00010E31                 mov     dword_11ACC, 4
.text:00010E3B                 mov     NtTerminateProcess_d, 101h
.text:00010E45                 mov     dword_11ADC, 88h
.text:00010E4F                 retn
.text:00010E50
.text:00010E50
.text:00010E50 win_2k
.text:00010E50                 mov     dword_11B24, 6Ah
.text:00010E5A                 mov     NtClose_ServiceNum, 18h
.text:00010E64                 mov     NtWVM_ServiceNumber, 0F0h
.text:00010E6E                 mov     NtRVM_ServiceNumber, 0A4h
.text:00010E78                 mov     dword_11B20, 9Ch
.text:00010E82                 mov     dword_11ACC, 8
.text:00010E8C                 mov     NtTerminateProcess_d, 0E0h
.text:00010E96                 mov     dword_11ADC, 0A0h
.text:00010EA0                 retn
.text:00010EA0 native_api_services endp
```

Depending on Build number driver will retrieve service numbers which will be used as index into KeServiceDescriptorTable to hook APIs. You may esily determine which is which native API by disassembling ntdll.dll and watching which number is passsed in eax before sysenter is used or int 2eh on win2k systems. (or you may write program to do this for you, it is very simple, check ntapi.asm supplied with this document), now we will examine hooking procedure of SDT:

```
.text:00010F24 HookNativeApis  proc near
.text:00010F24
.text:00010F24 old_cr0         = dword ptr -4
.text:00010F24
.text:00010F24                 push    ecx
.text:00010F25                 push    esi
.text:00010F26                 push    edi
.text:00010F27                 cli
.text:00010F28                 mov     eax, cr0
.text:00010F2B                 mov     [esp+0Ch+old_cr0], eax
.text:00010F2F                 and     eax, 0FFFEFFFFh
.text:00010F34                 mov     cr0, eax
.text:00010F37                 mov     ecx, ds:KeServiceDescriptorTable
.text:00010F3D                 mov     edx, [ecx]
.text:00010F3F                 mov     eax, NtClose_ServiceNum
.text:00010F44                 shl     eax, 2
.text:00010F47                 mov     edx, [eax+edx]
.text:00010F4A                 mov     Old_NtClose, edx
.text:00010F50                 mov     ecx, [ecx]
.text:00010F52                 mov     dword ptr [eax+ecx], offset NtCloseHook
.text:00010F59                 mov     ecx, ds:KeServiceDescriptorTable
.text:00010F5F                 mov     edx, [ecx]
.text:00010F61                 mov     eax, NtWVM_ServiceNumber
.text:00010F66                 shl     eax, 2
```

```
.text:00010F69                  mov     edx, [eax+edx]
.text:00010F6C                  mov     Old_NtWriteVirtualMemory, edx
.text:00010F72                  mov     ecx, [ecx]
.text:00010F74                  mov     dword ptr [eax+ecx], offset hook_NtWriteVirtualMemory
.text:00010F7B                  mov     ecx, ds:KeServiceDescriptorTable
.text:00010F81                  mov     eax, NtRVM_ServiceNumber
.text:00010F86                  mov     edx, [ecx]
.text:00010F88                  shl     eax, 2
.text:00010F8B                  mov     edx, [eax+edx]
.text:00010F8E                  mov     Old_NtReadVirtualMemory, edx
.text:00010F94                  mov     ecx, [ecx]
.text:00010F96                  push    0               ; Remove
.text:00010F98                  push    offset NotifyRoutine ; NotifyRoutine
.text:00010F9D                  mov     dword ptr [eax+ecx], offset Hook_NrReadVirtualMemory
.text:00010FA4                  call    PsSetCreateProcessNotifyRoutine
.text:00010FA9                  mov     eax, ds:KeServiceDescriptorTable
.text:00010FAE                  mov     eax, [eax+8]    ; number of native APIS
.text:00010FB1                  shl     eax, 2          ; multiply by 4
.text:00010FB4                  push    206B6444h       ; Tag
.text:00010FB9                  push    eax             ; NumberOfBytes
.text:00010FBA                  push    0               ; PoolType
.text:00010FBC                  mov     size_native_apis, eax
.text:00010FC1                  call    ds:ExAllocatePoolWithTag
.text:00010FC7                  mov     ecx, size_native_apis
.text:00010FCD                  mov     edx, ds:KeServiceDescriptorTable
.text:00010FD3                  mov     native_apis_pool, eax
.text:00010FD8                  mov     esi, [edx]      ; array of native APIs
.text:00010FDA                  mov     edi, eax        ; allocated pool
.text:00010FDC                  mov     eax, ecx
.text:00010FDE                  shr     ecx, 2
.text:00010FE1                  rep movsd               ; copy array of native APIs to aloocated pool
.text:00010FE3                  mov     ecx, eax        ; this also copies hooked Native APIs
.text:00010FE5                  and     ecx, 3
.text:00010FE8                  rep movsb
.text:00010FEA                  mov     eax, [esp+0Ch+old_cr0] ; restore WP in cr0
.text:00010FEE                  mov     cr0, eax
.text:00010FF1                  sti
.text:00010FF2                  pop     edi
.text:00010FF3                  pop     esi
.text:00010FF4                  pop     ecx
.text:00010FF5                  retn
.text:00010FF5 HookNativeApis   endp
```

As you may see by looking at this code, crackme uses cr0 to wipe Write protection. Then it will hook NtWriteVirtualMemory, NtReadVirtualMemory and NtClose, also it will use PsSetCreateProcessNotifyRoutine to define callback that will be called whenever process in system is created/deleted. (process is object, so objects are created and deleted in windows J). After that whole list of API pointers from KeServiceDescriptorTable including hooked ones is copied to new nonpaged pool allocated before copying occurs. This copying occurs so SDT can't be restored easily, well not without a little bit of patching. Remember this PsSetCreateProcessNotifyRoutine, because you will understand why it is used later on… Also note two variables : native_apis_pool and size_native_apis where native_apis_pool is buffer with hooked SDT and size_native_apis is size of this pool (number of pointers * 4 obtained from KeServiceDescriptorTable):

```
typedef struct ServiceDescriptorEntry {
        unsigned int *ServiceTableBase;
        unsigned int *ServiceCounterTableBase;
        unsigned int NumberOfServices;
        unsigned char *ParamTableBase;
} SSDT_Entry;
```

Hooks of NtWrite/ReadVirtualMemory are used to deny write/read from/to protected process but NtClose is used as checker if driver is running or not:

```
.text:00010C9C NtCloseHook:
.text:00010C9C                 call    PsGetCurrentProcessId
.text:00010CA1                 push    eax
.text:00010CA2                 call    check_runing_process
.text:00010CA7                 xor     ecx, ecx
.text:00010CA9                 cmp     dword ptr [esp+4], 0FE12F2E5h
.text:00010CB1                 setz    cl
.text:00010CB4                 test    eax, ecx
.text:00010CB6                 jz      short loc_10CC0
.text:00010CB8                 mov     eax, 0EFFFD27Fh
.text:00010CBD                 retn    4
.text:00010CC0 loc_10CC0:
.text:00010CC0                 jmp     Old_NtClose
```

Whenever NtClose is called with argument 0FE12F2E5h driver will return 0EFFFD27Fh , but if driver isn't present then STATUS_INVALID_HANDLE will be returned, so this is used as check if driver is running J

Next what we are going to analyze is Thread created by driver startup code:

```
.text:00011596 ; void __stdcall StartRoutine(PVOID)
.text:00011596 StartRoutine    proc near
.text:00011596
.text:00011596 Object          = _KTIMER ptr -3Ch
.text:00011596 var_14          = qword ptr -14h
.text:00011596 var_C           = dword ptr -0Ch
.text:00011596 var_8           = dword ptr -8
.text:00011596 var_4           = dword ptr -4
.text:00011596
.text:00011596                 push    ebp
.text:00011597                 mov     ebp, esp
.text:00011599                 sub     esp, 3Ch
.text:0001159C                 push    ebx
.text:0001159D                 push    1               ; Type
.text:0001159F                 lea     eax, [ebp+Object]
.text:000115A2                 push    eax             ; Timer
.text:000115A3                 call    ds:KeInitializeTimerEx
.text:000115A9                 xor     ebx, ebx
.text:000115AB                 push    ebx             ; Dpc
.text:000115AC                 push    3E8h            ; Period
.text:000115B1                 xor     eax, eax
.text:000115B3                 push    eax
.text:000115B4                 xor     ecx, ecx
.text:000115B6                 push    ecx             ; DueTime
.text:000115B7                 lea     eax, [ebp+Object]
.text:000115BA                 push    eax             ; Timer
.text:000115BB                 call    ds:KeSetTimerEx
.text:000115C1                 cmp     byte_11AC0, bl
.text:000115C7                 jnz     loc_11696
.text:000115CD                 push    esi
.text:000115CE                 push    edi
.text:000115CF                 jmp     short loc_115D3
.text:000115D1 ; ---------------------------------------------------------------------------
.text:000115D1
.text:000115D1 loc_115D1:
.text:000115D1                 xor     ebx, ebx
.text:000115D3
.text:000115D3 loc_115D3:
.text:000115D3                 push    ebx             ; Timeout
.text:000115D4                 push    ebx             ; Alertable
.text:000115D5                 push    ebx             ; WaitMode
.text:000115D6                 push    ebx             ; WaitReason
.text:000115D7                 lea     eax, [ebp+Object]
.text:000115DA                 push    eax             ; Object
.text:000115DB                 call    ds:KeWaitForSingleObject
.text:000115E1                 cli
.text:000115E2                 mov     eax, cr0
.text:000115E5                 mov     [ebp+var_4], eax
.text:000115E8                 and     eax, 0FFFEFFFFh
.text:000115ED                 mov     cr0, eax
.text:000115F0                 mov     eax, ds:KeServiceDescriptorTable
.text:000115F5                 mov     ecx, size_native_apis
```

```
.text:000115FB                  mov     edi, [eax]
.text:000115FD                  mov     esi, native_apis_pool
.text:00011603                  mov     eax, ecx
.text:00011605                  shr     ecx, 2
.text:00011608                  rep movsd
.text:0001160A                  mov     ecx, eax
.text:0001160C                  and     ecx, 3
.text:0001160F                  rep movsb
.text:00011611                  mov     eax, [ebp+var_4]
.text:00011614                  mov     cr0, eax
.text:00011617                  sti
.text:00011618                  sidt    [ebp+var_14]
.text:0001161C                  mov     ebx, dword ptr [ebp+var_14+2]
.text:0001161F                  mov     si, [ebx+18h]
.text:00011623                  rol     esi, 10h
.text:00011626                  mov     si, [ebx+1Eh]
.text:0001162A                  ror     esi, 10h
.text:0001162D                  mov     [ebp+var_C], esi
.text:00011630                  mov     eax, cr4
.text:00011633                  and     eax, 0FFFFFFF7h
.text:00011636                  mov     cr4, eax
.text:00011639                  mov     si, [ebx+8]
.text:0001163D                  rol     esi, 10h
.text:00011640                  mov     si, [ebx+0Eh]
.text:00011644                  ror     esi, 10h
.text:00011647                  mov     [ebp+var_8], esi
.text:0001164A                  xor     eax, eax
.text:0001164C                  cmp     [ebp+var_8], offset new_int1_handle
.text:00011653                  setnz   al
.text:00011656                  xor     ecx, ecx
.text:00011658                  cmp     [ebp+var_C], offset new_int3_handle
.text:0001165F                  setnz   cl
.text:00011662                  xor     ebx, ebx
.text:00011664                  or      eax, ecx
.text:00011666                  xor     ecx, ecx
.text:00011668                  cmp     runin_processes_pid_ptr, ebx
.text:0001166E                  setnz   cl
.text:00011671                  test    eax, ecx
.text:00011673                  jz      short loc_11688
.text:00011675                  call    ds:IoGetCurrentProcess
.text:0001167B                  push    10h             ; size_t
.text:0001167D                  push    ebx             ; int
.text:0001167E                  push    eax             ; void *
.text:0001167F                  call    ds:memchr
.text:00011685                  add     esp, 0Ch
.text:00011688
.text:00011688 loc_11688:
.text:00011688                  cmp     byte_11AC0, bl
.text:0001168E                  jz      loc_115D1
.text:00011694                  pop     edi
.text:00011695                  pop     esi
.text:00011696
.text:00011696 loc_11696:
.text:00011696                  pop     ebx
.text:00011697                  leave
.text:00011698                  retn    4
.text:00011698 StartRoutine     endp
```

Here you may see how timer is set, but also note how, when certain amount of time passes it will overwrite SDT with saved copy of hooked SDT. Very nice, so if you restore SDT it will again hook it, and you will have to patch a few places to be able to dump parts of crackme memory. Patching isn't hard here, write driver that will nook rep movsd/rep movsb here:

```
.text:00011608                  rep movsd
.text:0001160A                  mov     ecx, eax
.text:0001160C                  and     ecx, 3
.text:0001160F                  rep movsb
```

You maight wanna check anti-msrem.asm driver for complete source (not very advanced driver, only simple patching J).

After patching of this 2 instructions crackme won't run anymore, it will start but due to check with NtClose it won't run.

We also know that there are hooks of int1 and int3 so we may locate them in driver by using this simple formula:

Int1_address – driverbase = 0000078Ah now go to that RVA in IDA and you will find int1 and int3 handles:

```
.text:0001078A new_int3_handle proc near
.text:0001078A                 pusha
.text:0001078B                 mov     di, 30h
.text:0001078F                 mov     fs, di
.text:00010792                 mov     eax, [esp+20h]  ; EIP
.text:00010796                 dec     eax             ; address of int3h in process
.text:00010797                 push    eax
.text:00010798                 call    handle_int3h
.text:0001079D                 test    eax, eax
.text:0001079F                 jnz     short __old_int3_handle
.text:000107A1                 mov     di, 3Bh
.text:000107A5                 mov     fs, di
.text:000107A8                 popa
.text:000107A9                 dec     byte ptr [esp+0] ; eip points to int3h
.text:000107AC                 iret
.text:000107AD __old_int3_handle:
.text:000107AD                 mov     di, 3Bh
.text:000107B1                 mov     fs, di
.text:000107B4                 popa
.text:000107B5                 jmp     old_int3_handle
.text:000107B5 new_int3_handle endp
```

And int1:

```
.text:000107BC new_int1_handle proc near
.text:000107BC
.text:000107BC arg_4           = dword ptr  8
.text:000107BC
.text:000107BC                 pusha
.text:000107BD                 push    fs
.text:000107BF                 mov     di, 30h
.text:000107C3                 mov     fs, di
.text:000107C6                 push    eax
.text:000107C7                 call    check_int1_process
.text:000107CC                 test    eax, eax
.text:000107CE                 jz      short __old_int1_handle
.text:000107D0                 mov     eax, [esp+2Ch]  ; eflags
.text:000107D4                 and     eax, 0FFFFFEFFh ; wipe TRAP flag if used
.text:000107D9                 or      eax, 10000h     ; set Resume Flag
.text:000107DE                 mov     [esp+2Ch], eax  ; set eflags
.text:000107E2                 xor     eax, eax
.text:000107E4                 mov     dr7, eax        ; wipe dr7
.text:000107E7                 mov     dr0, eax        ; wipe dr0
.text:000107EA                 pop     fs
.text:000107EC                 popa
.text:000107ED                 iret
.text:000107EE __old_int1_handle:
.text:000107EE                 pop     fs
.text:000107F0                 popa
.text:000107F1                 jmp     old_int1_handle
.text:000107F1 new_int1_handle endp
```

Int1 handle is not complicated to understand it will wipe dr7 and dr0 and Trap Flag from eflags if int1 handler is called from crackme, otherwiese it will call old handler. Not much to talk about it, really simple.

On other hand int3 is much much more interesting because it has APC delivery in it. Really kewl.

In int3 handler there is procedure which I named handle_int3:

```
.text:000106CE handle_int3h    proc near
.text:000106CE int3_eip        = dword ptr  8
.text:000106CE
.text:000106CE                 push    esi
.text:000106CF                 call    PsGetCurrentProcessId
.text:000106D4                 mov     esi, eax
.text:000106D6                 push    esi
.text:000106D7                 call    check_runing_process
.text:000106DC                 test    eax, eax
.text:000106DE                 jnz     short __good_process
.text:000106E0                 inc     eax
.text:000106E1                 jmp     short __not_my_process_0
.text:000106E3
.text:000106E3 __good_process:
.text:000106E3                 push    ebx
.text:000106E4                 push    esi
.text:000106E5                 call    __check_runnin_processes
.text:000106EA                 mov     esi, eax
.text:000106EC                 xor     ebx, ebx
.text:000106EE                 cmp     esi, ebx
.text:000106F0                 jz      short __not_my_conditions
.text:000106F2                 push    edi
.text:000106F3                 push    206B6444h       ; Tag
.text:000106F8                 push    10h             ; NumberOfBytes
.text:000106FA                 push    ebx             ; PoolType
.text:000106FB                 call    ds:ExAllocatePoolWithTag
.text:00010701                 push    ebx             ; State
.text:00010702                 mov     edi, eax        ; KEVENT type
.text:00010704                 push    ebx             ; Type
.text:00010705                 push    edi             ; Event
.text:00010706                 call    ds:KeInitializeEvent
.text:0001070C                 push    ebx             ; zero...
.text:0001070D                 push    [esp+0Ch+int3_eip] ; address of int3h
.text:00010711                 push    edi             ; PKEVENT
.text:00010712                 push    [esi+proc_struct.user_apc] ; user procedure to be executed
.text:00010715                 push    [esi+proc_struct.ptr_ethread] ; pointer to KTHREAD
.text:00010718                 call    insert_apc      ; APC initialization routine = 1315474ch
.text:0001071D                 push    ebx             ; Timeout
.text:0001071E                 push    ebx             ; Alertable
.text:0001071F                 push    ebx             ; WaitMode
.text:00010720                 push    ebx             ; WaitReason
.text:00010721                 push    edi             ; Object
.text:00010722                 call    ds:KeWaitForSingleObject
.text:00010728                 push    ebx             ; Tag
.text:00010729                 push    edi             ; P
.text:0001072A                 call    ds:ExFreePoolWithTag
.text:00010730                 pop     edi
.text:00010731                 jmp     short loc_10739
.text:00010733 __not_my_conditions:
.text:00010733                 mov     large ds:0, ebx
.text:00010739
.text:00010739 loc_10739:
.text:00010739                 xor     eax, eax
.text:0001073B                 pop     ebx
.text:0001073C
.text:0001073C __not_my_process_0:
.text:0001073C                 pop     esi
.text:0001073D                 retn    4
.text:0001073D handle_int3h    endp
```

And insert_apc:

```
.text:00010626 insert_apc      proc near
.text:00010626 pkthread        = dword ptr  8
.text:00010626 user_procedure  = dword ptr  0Ch
.text:00010626 EVENT_ptr       = dword ptr  10h
.text:00010626 int3h_eip       = dword ptr  14h
.text:00010626 set_to_zero     = dword ptr  18h
.text:00010626
.text:00010626                 push    ebp
.text:00010627                 mov     ebp, esp
```

```
.text:00010629                  push    edi
.text:0001062A                  push    206B6444h       ; Tag
.text:0001062F                  push    30h             ; NumberOfBytes
.text:00010631                  push    0               ; PoolType
.text:00010633                  call    ds:ExAllocatePoolWithTag
.text:00010639                  push    [ebp+EVENT_ptr] ; IN PVOID NormalContext
.text:0001063C                  mov     edi, eax
.text:0001063E                  push    1               ; KPROCESSOR_MODE ApcMode (kernel user)
.text:00010640                  push    [ebp+user_procedure] ; user APC routine
.text:00010643                  push    0               ; RundownRoutine
.text:00010645                  push    offset kernel_apc_routine ; kernel routine
.text:0001064A                  push    0               ; KAPC_ENVIRONMENT
.text:0001064C                  push    [ebp+pkthread]  ; PKTHREAD
.text:0001064F                  push    edi             ; PAPC
.text:00010650                  call    KeInitializeApc
.text:00010655                  push    0               ; mode
.text:00010657                  push    [ebp+set_to_zero] ; systemparam2
.text:0001065A                  push    [ebp+int3h_eip] ; systemparam1
.text:0001065D                  push    edi             ; PAPC
.text:0001065E                  call    KeInsertQueueApc
.text:00010663                  pop     edi
.text:00010664                  pop     ebp
.text:00010665                  retn    14h
.text:00010665 insert_apc       endp ; sp = -30h
```

To get as much as pobbile informations what is what I used LiveKd to dump live memory and to analyze addresses, that's how I found user APC procedure first. But later I saw that it could be located much easier (althou this is easy enough J)

To recognize processes (2 of them) driver is using one struct which I named proc_struct:

```
00000000 proc_struct     struc ; (sizeof=0x24)
00000000 Next            dd ?
00000004 unknown_0       dd ?
00000008 hiden_pid       dd ?                    ; pid of dialog process
0000000C apc_pid         dd ?                    ; pid of sleeping process
00000010 unknown_1       dd ?
00000014 ptr_eprocess    dd ?                    ; e process of hiden_pid
00000018 ptr_ethread     dd ?                    ; e thread of APC process
0000001C user_apc        dd ?                    ; user APC address
00000024 proc_struct     ends
```

Reason why EPROCESS is stored here is because process is hiden. And when process is hiden there is no other way to find it's eprocess, well at least PsLookupProcessByProcessId won't find it. But hiden processes can be located via SwapContext because windows is using thread scheduling, so yep hiden process will run as long as it's threads are visible J The reason why EPROCESS is stored is because crackme adopts ring0 WriteProcessMemory via KeAttachProcess, and without saved EPROCESS there will be no chnace for crackme to conect to target process. APC process (I named it like that) is 2nd process of crackme which will loop with (from virtual.dll):

```
CODE:1315698E loc_1315698E:
CODE:1315698E                   push    0FFFFFFFFh      ; bAlertable
CODE:13156990                   push    0FFFFFFFFh      ; dwMilliseconds
CODE:13156992                   call    SleepEx
CODE:13156997                   jmp     short loc_1315698E
```

APC will be executed and SleepEx will return, but again it will fall into SleepEx loop, now you may also realize why there is PsSetCreateProcessNotifyRoutine. Simple, this 2nd process will never end, so you have to terminate it somehow and that's what Notify routine is doing. Here is small explanation from DDK:

PsSetCreateProcessNotifyRoutine adds a driver-supplied callback routine to, or removes it from, a list of routines to be called whenever a process is created or deleted.

```
NTSTATUS
  PsSetCreateProcessNotifyRoutine(
    IN PCREATE_PROCESS_NOTIFY_ROUTINE  NotifyRoutine,
    IN BOOLEAN  Remove
    );
```

And prototype of NotifyRoutine is:

```
VOID
(*PCREATE_PROCESS_NOTIFY_ROUTINE) (
    IN HANDLE  ParentId,
    IN HANDLE  ProcessId,
    IN BOOLEAN  Create
    );
```

The *ParentId* and *ProcessId* parameters identify the process, and the *Create* parameter indicates whether the process was created (TRUE) or deleted (FALSE).

So now driver knows when process is terminated and can terminate sleeping APC child. Lets see that routine (you may locate it by looking at hook_native_apis folowing references):

```
.text:00010EA2 ; void __stdcall NotifyRoutine(HANDLE,HANDLE,BOOLEAN)
.text:00010EA2 NotifyRoutine   proc near
.text:00010EA2
.text:00010EA2 ObjectAttributes= OBJECT_ATTRIBUTES ptr -20h
.text:00010EA2 CLIENT_ID       = CLIENT_ID ptr -8
.text:00010EA2 Process_ID      = dword ptr  0Ch
.text:00010EA2 create          = dword ptr  10h
.text:00010EA2
.text:00010EA2                 push    ebp
.text:00010EA3                 mov     ebp, esp
.text:00010EA5                 sub     esp, 20h
.text:00010EA8                 push    ebx
.text:00010EA9                 xor     ebx, ebx
.text:00010EAB                 cmp     byte ptr [ebp+create], bl
.text:00010EAE                 jnz     short loc_10F1F
.text:00010EB0                 push    [ebp+Process_ID]
.text:00010EB3                 call    check_runing_process
.text:00010EB8                 test    eax, eax
.text:00010EBA                 jz      short loc_10F1F
.text:00010EBC                 push    [ebp+Process_ID]
.text:00010EBF                 call    __check_runnin_processes
.text:00010EC4                 cmp     eax, ebx
.text:00010EC6                 jz      short loc_10ECD
.text:00010EC8                 mov     ecx, [eax+proc_struct.apc_pid]
.text:00010ECB                 jmp     short loc_10ED8
.text:00010ECD loc_10ECD:
.text:00010ECD                 push    [ebp+Process_ID]
.text:00010ED0                 call    find_proc_struct
.text:00010ED5                 mov     ecx, [eax+proc_struct.hiden_pid]
.text:00010ED8
.text:00010ED8 loc_10ED8:
.text:00010ED8                 push    eax
.text:00010ED9                 mov     [ebp+ObjectAttributes.Length], 18h
.text:00010EE0                 mov     [ebp+ObjectAttributes.RootDirectory], ebx
.text:00010EE3                 mov     [ebp+ObjectAttributes.Attributes], ebx
.text:00010EE6                 mov     [ebp+ObjectAttributes.ObjectName], ebx
.text:00010EE9                 mov     [ebp+ObjectAttributes.SecurityDescriptor], ebx
.text:00010EEC                 mov     [ebp+ObjectAttributes.SecurityQualityOfService], ebx
.text:00010EEF                 mov     [ebp+CLIENT_ID.UniqueProcess], ecx
.text:00010EF2                 mov     [ebp+CLIENT_ID.UniqueThread], ebx
```

```
.text:00010EF5                  call    sub_109DC
.text:00010EFA                  lea     eax, [ebp+CLIENT_ID]
.text:00010EFD                  push    eax             ; ClientId
.text:00010EFE                  lea     eax, [ebp+ObjectAttributes]
.text:00010F01                  push    eax             ; ObjectAttributes
.text:00010F02                  push    1F0FFFh         ; DesiredAccess
.text:00010F07                  lea     eax, [ebp+create]
.text:00010F0A                  push    eax             ; ProcessHandle
.text:00010F0B                  call    ds:ZwOpenProcess
.text:00010F11                  test    eax, eax
.text:00010F13                  jnz     short loc_10F1F
.text:00010F15                  push    ebx             ; ExitStatus
.text:00010F16                  push    [ebp+create]    ; ProcessHandle
.text:00010F19                  call    ds:ZwTerminateProcess
.text:00010F1F
.text:00010F1F loc_10F1F:
.text:00010F1F                  pop     ebx
.text:00010F20                  leave
.text:00010F21                  retn    0Ch
.text:00010F21 NotifyRoutine    endp
```

What is going here? There are 3 procedures which have similar names __check_runing_process, find_proc_struct etc… common for them is that those are used to locate proc_struct for certain process by PID. I have presented partialy reversed proc_struct so all that is done here is something like this:

First it checks if process is terminated, if so then it checks if terminated processes is protected process, if one of them is protected and is is deleted then NtTerminateProcess is used to terminate other process. This will happen when you termiante child process (visible one from Task Manager or Process Explorer) or you click on ‘X’ in dialog box when Notify routine will kill Sleeping child process.

Now we have to examine CreateWrite handler in driver:

```
.text:00011414 CreateWriteHandler proc near            ; DATA XREF: start+B6o
.text:00011414
.text:00011414 var_5C           = dword ptr -5Ch
.text:00011414 user_buffer_local= dword ptr -58h
.text:00011414 ms_exc           = CPPEH_RECORD ptr -18h
.text:00011414 Irp              = dword ptr  0Ch
.text:00011414
.text:00011414                  push    4Ch
.text:00011416                  push    offset unk_119B8
.text:0001141B                  call    __SEH_prolog
.text:00011420                  xor     ecx, ecx
.text:00011422                  xor     esi, esi
.text:00011424                  mov     ebx, [ebp+Irp]
.text:00011427                  mov     eax, [ebx+60h]  ; io_stack_location
.text:0001142A                  mov     [ebx+IRP.IoStatus.Information], ecx
.text:0001142D                  cmp     byte ptr [eax], 4 ; isl_MajorFunction
.text:00011430                  jnz     short __create_request
.text:00011432                  mov     eax, [eax+IO_STACK_LOCATION.Parameters.Write.Length]
.text:00011435                  mov     esi, [ebx+IRP.UserBuffer]
.text:00011438                  cmp     eax, 40h
.text:0001143B                  jnz     short __wrong_write_size
.text:0001143D                  mov     [ebp+ms_exc.disabled], ecx
.text:00011440                  push    10h
.text:00011442                  pop     ecx
.text:00011443                  lea     edi, [ebp+user_buffer_local]
.text:00011446                  rep movsd
.text:00011448                  lea     eax, [ebp+user_buffer_local]
.text:0001144B                  push    eax
.text:0001144C                  call    main_user_request_handler
.text:00011451                  mov     esi, eax
.text:00011453                  mov     [ebp+var_5C], esi
.text:00011456                  mov     [ebx+IRP.IoStatus.Information], 40h
.text:0001145D
.text:0001145D loc_1145D:
.text:0001145D                  or      dword ptr [ebp-4], 0FFFFFFFFh
```

```
.text:00011461                 jmp     short __create_request
.text:00011463
.text:00011463 loc_11463:
.text:00011463                 xor     eax, eax
.text:00011465                 inc     eax
.text:00011466                 retn
.text:00011467 loc_11467:
.text:00011467                 mov     esp, [ebp-18h]
.text:0001146A                 mov     ebx, [ebp+Irp]
.text:0001146D                 and     [ebx+IRP.IoStatus.Information], 0
.text:00011471                 mov     esi, 0C0000006h
.text:00011476                 jmp     short loc_1145D
.text:00011478 __wrong_write_size:
.text:00011478                 mov     esi, 0C0000022h
.text:0001147D                 mov     [ebx+IRP.IoStatus.Information], ecx
.text:00011480
.text:00011480 __create_request:
.text:00011480                 mov     [ebx+IRP.IoStatus.anonymous_0.Status], esi
.text:00011483                 xor     dl, dl          ; PriorityBoost
.text:00011485                 mov     ecx, ebx        ; Irp
.text:00011487                 call    ds:IofCompleteRequest
.text:0001148D                 mov     eax, esi
.text:0001148F                 call    __SEH_epilog
.text:00011494                 retn    8
.text:00011494 CreateWriteHandler endp
```

First CreateWriteHandler will check why is this driver called. Create or Write, if it is Write then you have copying of passed buffer to local variable and then that buffer is passed to main_user_request_handler:

```
.text:000112B6 main_user_request_handler proc
.text:000112B6
.text:000112B6 user_buffer     = dword ptr  8
.text:000112D3                 mov     eax, [esp+user_buffer]
.text:000112D7                 mov     ecx, [eax]
.text:000112D9                 dec     ecx
.text:000112DA                 cmp     ecx, 9          ; switch 10 cases
.text:000112DD                 ja      short loc_1134B ; default
.text:000112DF                 jmp     ds:off_11351[ecx*4] ; switch jump
```

From here you may see that buffer passed to driver via WriteFile is of maximum size 40h and 1st 4 bytes are used as Service ID, used later on with switch/case in main_user_request_handler.

Oki, we have basic knowledge of driver structure and how and what it is doing.

3. dumping virtual.exe and virtual.dll

If you trace crackme.exe and focs.dll a little bit you may notice how both of them are actually loaders for virtual.exe and virtual.dll, this extraction ocurs before driver protection starts so you may dump regions from crackme. For this I'll go fast on this subject because it isn't very complex to dump those regions.

Load crackme.exe in debugger, BPX on VirtualAlloc, and watch how sections + pe header are being copied to it, after it is copied put process into infinite loop and dump region at 400000h, now all you have to do to get valid exe is to set RAW of section = RVA of section, now simple redirect process to ExitProcess and you have crackme dumped. (there is a lot more to fix!!!!).
Apply same trick to focs.dll and dump virtual.dll from it.

Good now you have virtual.exe and virtual.dll ready to be disassembled. First we try our dumped crackme.exe and it is stoped due to int3h in it, we know that there are int1/3 hooks so we also know that crackme will act on some specific way when int 3 occurs in it. It will deliver APC to child process and then it will wait to be signaled and return to process.

Now lets examine virtual.dll:

```
CODE:13157189                push    offset aDbgbreakpoint ; "DbgBreakPoint"
CODE:1315718E                push    offset aNtdll_dll ; "ntdll.dll"
CODE:13157193                call    GetModuleHandleA_0
CODE:13157198                push    eax
CODE:13157199                call    GetProcAddress_0
CODE:1315719E                mov     ds:DbgBreakPoint, eax
CODE:131571A3                mov     ds:byte_1315A994, 68h
CODE:131571AA                mov     ds:dword_1315A995, offset hook_DbgBreakPoint
CODE:131571B4                mov     ds:byte_1315A999, 0C3h
CODE:131571BB                push    offset NumberOfBytesWritten
CODE:131571C0                push    6
CODE:131571C2                push    offset byte_1315A994
CODE:131571C7                mov     eax, ds:DbgBreakPoint
CODE:131571CC                push    eax
CODE:131571CD                push    0FFFFFFFFh
CODE:131571CF                call    WriteProcessMemory
CODE:131571D4                xor     eax, eax
CODE:131571D6                call    mess_with_callgate
CODE:131571DB                call    sub_13155520
CODE:131571E0                call    sub_131563FC
CODE:131571E5                call    install_driver
CODE:131571EA                push    offset NumberOfBytesWritten
CODE:131571EF                push    0
CODE:131571F1                push    0
CODE:131571F3                push    offset ThreadStart
CODE:131571F8                push    0
CODE:131571FA                push    0
CODE:131571FC                call    CreateThread
CODE:13157201                mov     ds:hThread, eax
```

Here you see concept of hooking some APIs in current process, interesting also procedures are mess_with_callgate which in this paticilar case does nothing, only Sleep for 10h miliseconds, but in same procedure there is call to mess with gdt and to install callgate which isn't called.

After that we have install_driver, this procedure will perform search in system32\drivers\ folder looking for dxapi32.sys, if there is such file then crackme assumes that driver is installed and it will continue with execution, otherwise MessageBox will popup informing you that driver is installed and that you should reboot your system.

Next is CreateThread and new thread does nothing. It will call mess_with_callgate which will Sleep only and loop like that untill process isn't terminated.

Very interesting procedure is here:

```
CODE:131573BA                call    new_process
CODE:131573BF                test    al, al
CODE:131573C1                jz      short loc_131573CE
CODE:131573C3
CODE:131573C3 loc_131573C3:
CODE:131573C3                push    0FFFFFFFFh
CODE:131573C5                push    0FFFFFFFFh
CODE:131573C7                call    SleepEx
CODE:131573CC                jmp     short loc_131573C3
```

new_process procedure is responsible for creation of new process and also it will inform driver about parent/child pid. So lets see how it works.

```
CODE:13156758 new_process     proc near               ; CODE XREF: CODE:131573BAp
CODE:13156758
CODE:13156758 var_7C          = dword ptr -7Ch
CODE:13156758 var_78          = dword ptr -78h
CODE:13156758 var_74          = dword ptr -74h
CODE:13156758 var_70          = dword ptr -70h
CODE:13156758 var_6C          = dword ptr -6Ch
CODE:13156758 var_68          = dword ptr -68h
CODE:13156758 Buffer          = dword ptr -64h
CODE:13156758 dwProcessId     = dword ptr -60h
CODE:13156758 hObject         = dword ptr -5Ch
CODE:13156758 hProcess        = dword ptr -58h
CODE:13156758 StartupInfo     = _STARTUPINFOA ptr -48h
CODE:13156758 NumberOfBytesWritten= dword ptr -4
CODE:13156758
CODE:13156758                 push    ebp
CODE:13156759                 mov     ebp, esp
CODE:1315675B                 add     esp, 0FFFFFF84h
CODE:1315675E                 push    ebx
CODE:1315675F                 push    esi
CODE:13156760                 push    edi
CODE:13156761                 xor     eax, eax
CODE:13156763                 mov     [ebp+var_74], eax
CODE:13156766                 mov     [ebp+var_78], eax
CODE:13156769                 mov     [ebp+var_7C], eax
CODE:1315676C                 mov     [ebp+var_6C], eax
CODE:1315676F                 mov     [ebp+var_70], eax
CODE:13156772                 mov     [ebp+var_68], eax
CODE:13156775                 xor     eax, eax
CODE:13156777                 push    ebp
CODE:13156778                 push    offset j_unknown_libname_10_72
CODE:1315677D                 push    dword ptr fs:[eax]
CODE:13156780                 mov     fs:[eax], esp
CODE:13156783                 push    offset aZwvdmcontrol ; "ZwVdmControl"
CODE:13156788                 push    offset aNtdll_dll_0 ; "ntdll.dll"
CODE:1315678D                 call    GetModuleHandleA_0
CODE:13156792                 push    eax             ; hModule
CODE:13156793                 call    GetProcAddress_0
CODE:13156798                 mov     ebx, eax
CODE:1315679A                 lea     eax, [ebp+NumberOfBytesWritten]
CODE:1315679D                 push    eax             ; lpNumberOfBytesRead
CODE:1315679E                 push    0Ch             ; nSize
CODE:131567A0                 lea     eax, [ebp+Buffer]
CODE:131567A3                 push    eax             ; lpBuffer
CODE:131567A4                 mov     edi, ebx
CODE:131567A6                 push    edi             ; lpBaseAddress
CODE:131567A7                 push    0FFFFFFFFh      ; hProcess
CODE:131567A9                 call    ReadProcessMemory
CODE:131567AE                 cmp     [ebp+Buffer], 64152917h   <-- signature bytes
CODE:131567B5                 jz      __2nd_process
```

Here we see that crackme checks if it is parent or child process by checking bytes in NtVdmControl, if no matching bytes were found, we are father process and it is time to create child process:

```
CODE:131567BB                 lea     eax, [ebp+StartupInfo]
CODE:131567BE                 mov     edx, 44h
CODE:131567C3                 call    sub_13146004
CODE:131567C8                 mov     [ebp+StartupInfo.cb], 44h
CODE:131567CF                 lea     eax, [ebp+hProcess]
CODE:131567D2                 push    eax             ; lpProcessInformation
CODE:131567D3                 lea     eax, [ebp+StartupInfo]
CODE:131567D6                 push    eax             ; lpStartupInfo
CODE:131567D7                 push    0               ; lpCurrentDirectory
CODE:131567D9                 push    0               ; lpEnvironment
CODE:131567DB                 push    4               ; dwCreationFlags
CODE:131567DD                 push    0               ; bInheritHandles
CODE:131567DF                 push    0               ; lpThreadAttributes
CODE:131567E1                 push    0               ; lpProcessAttributes
CODE:131567E3                 lea     edx, [ebp+var_68]
```

```
CODE:131567E6                   xor     eax, eax
CODE:131567E8                   call    @ParamStr
CODE:131567ED                   mov     eax, [ebp+var_68]
CODE:131567F0                   call    unknown_libname_21
CODE:131567F5                   push    eax             ; lpCommandLine
CODE:131567F6                   push    0               ; lpApplicationName
CODE:131567F8                   call    CreateProcessA
CODE:131567FD                   mov     [ebp+Buffer], 64152917h  <-- signature bytes
CODE:13156804                   call    GetCurrentProcessId
CODE:13156809                   mov     esi, eax
CODE:1315680B                   mov     [ebp+dwProcessId], esi   <-- buffer + 4 = parent PID
CODE:1315680E                   lea     edx, [ebp+var_70]
CODE:13156811                   mov     eax, esi
CODE:13156813                   call    @Sysutils@IntToStr$qqri_0
CODE:13156818                   mov     ecx, [ebp+var_70]
CODE:1315681B                   lea     eax, [ebp+var_6C]
CODE:1315681E                   mov     edx, offset aFocs ; "FOCS"
CODE:13156823                   call    @System@@LStrCat3$qqrv
CODE:13156828                   mov     eax, [ebp+var_6C]
CODE:1315682B                   call    unknown_libname_21
CODE:13156830                   push    eax             ; lpName
CODE:13156831                   push    0               ; bInitialState
CODE:13156833                   push    0FFFFFFFFh      ; bManualReset
CODE:13156835                   push    0               ; lpEventAttributes
CODE:13156837                   call    CreateEventA
CODE:1315683C                   mov     esi, eax
CODE:1315683E                   push    2               ; dwOptions
CODE:13156840                   push    0               ; bInheritHandle
CODE:13156842                   push    0               ; dwDesiredAccess
CODE:13156844                   lea     eax, [ebp+hObject]
CODE:13156847                   push    eax             ; lpTargetHandle
CODE:13156848                   mov     eax, [ebp+hProcess]
CODE:1315684B                   push    eax             ; hTargetProcessHandle
CODE:1315684C                   push    esi             ; hSourceHandle
CODE:1315684D                   push    0FFFFFFFFh      ; hSourceProcessHandle
CODE:1315684F                   call    DuplicateHandle
CODE:13156854                   lea     eax, [ebp+NumberOfBytesWritten]
CODE:13156857                   push    eax             ; lpNumberOfBytesWritten
CODE:13156858                   push    0Ch             ; nSize
CODE:1315685A                   lea     eax, [ebp+Buffer]
CODE:1315685D                   push    eax             ; lpBuffer
CODE:1315685E                   push    edi             ; lpBaseAddress
CODE:1315685F                   mov     eax, [ebp+hProcess]
CODE:13156862                   push    eax             ; hProcess
CODE:13156863                   call    WriteProcessMemory
CODE:13156868                   mov     eax, [ebp-54h]
CODE:1315686B                   push    eax             ; hThread
CODE:1315686C                   call    ResumeThread
CODE:13156871                   push    0FFFFFFFFh      ; dwMilliseconds
CODE:13156873                   push    esi             ; hHandle
CODE:13156874                   call    WaitForSingleObject
CODE:13156879                   push    esi             ; hObject
CODE:1315687A                   call    CloseHandle
CODE:1315687F                   xor     ebx, ebx
CODE:13156881                   jmp     loc_13156999
```

As you may see father will tell child it's PID, and will continue with initialization, now observe how child is acting, it will collect PIDs of both processes and inform driver about them using service ID : 7, also it will send to driver address of APC procedure and will enter into infinite loop with SleepEx allowing in such way APC to "interupt" it's sleeping state.

```
CODE:13156886
CODE:13156886 __2nd_process:
CODE:13156886                   mov     eax, [ebp+dwProcessId]
CODE:13156889                   push    eax             ; dwProcessId
CODE:1315688A                   push    0               ; bInheritHandle
CODE:1315688C                   push    1F0FFFh         ; dwDesiredAccess
CODE:13156891                   call    OpenProcess
CODE:13156896                   mov     ds:pHandle, eax
CODE:1315689B                   push    0               ; hTemplateFile
CODE:1315689D                   push    0               ; dwFlagsAndAttributes
CODE:1315689F                   push    3               ; dwCreationDisposition
CODE:131568A1                   push    0               ; lpSecurityAttributes
CODE:131568A3                   push    0               ; dwShareMode
```

```
CODE:131568A5                 push    40000000h       ; dwDesiredAccess
CODE:131568AA                 push    offset a_RootSystem0_4 ;"\\\\.\\Root#SYSTEM#0000#{“...
CODE:131568AF                 call    CreateFileA
CODE:131568B4                 mov     ds:hDevice, eax
CODE:131568B9                 mov     eax, [ebp+dwProcessId]
CODE:131568BC                 mov     ds:parent_pid, eax
CODE:131568C1                 call    GetCurrentProcessId
CODE:131568C6                 mov     ds:current_pid, eax
CODE:131568CB                 mov     ds:Buffer, 7
CODE:131568D5                 mov     eax, offset dword_1315A900
CODE:131568DA                 mov     ds:dword_1315A8C4, eax
CODE:131568DF                 mov     eax, offset APC_userprocedure
CODE:131568E4                 mov     ds:apc_procedure_addr, eax
CODE:131568E9                 push    0               ; lpOverlapped
CODE:131568EB                 lea     eax, [ebp+NumberOfBytesWritten]
CODE:131568EE                 push    eax             ; lpNumberOfBytesWritten
CODE:131568EF                 push    40h             ; nNumberOfBytesToWrite
CODE:131568F1                 push    offset Buffer   ; lpBuffer
CODE:131568F6                 mov     eax, ds:hDevice
CODE:131568FB                 push    eax             ; hFile
CODE:131568FC                 call    WriteFile_0
CODE:13156901                 push    0               ; hTemplateFile
CODE:13156903                 push    0               ; dwFlagsAndAttributes
CODE:13156905                 push    3               ; dwCreationDisposition
CODE:13156907                 push    0               ; lpSecurityAttributes
CODE:13156909                 push    0               ; dwShareMode
CODE:1315690B                 push    80000000h       ; dwDesiredAccess
CODE:13156910                 lea     edx, [ebp+var_7C]
CODE:13156913                 xor     eax, eax
CODE:13156915                 call    @ParamStr
CODE:1315691A                 mov     eax, [ebp+var_7C]
CODE:1315691D                 lea     edx, [ebp+var_78]
CODE:13156920                 call    unknown_libname_89
CODE:13156925                 mov     ecx, [ebp+var_78]
CODE:13156928                 lea     eax, [ebp+var_74]
CODE:1315692B                 mov     edx, offset dword_13156A34
CODE:13156930                 call    @System@@LStrCat3$qqrv
CODE:13156935                 mov     eax, [ebp+var_74]
CODE:13156938                 call    unknown_libname_21
CODE:1315693D                 push    eax             ; lpFileName
CODE:1315693E                 call    CreateFileA
CODE:13156943                 mov     ebx, eax
CODE:13156945                 cmp     ebx, 0FFFFFFFFh
CODE:13156948                 jnz     short loc_13156963
CODE:1315694A                 mov     eax, [ebp+hObject]
CODE:1315694D                 push    eax             ; hEvent
CODE:1315694E                 call    SetEvent
CODE:13156953                 mov     eax, [ebp+hObject]
CODE:13156956                 push    eax             ; hObject
CODE:13156957                 call    CloseHandle
CODE:1315695C                 push    0FFFFFFFFh      ; dwMilliseconds
CODE:1315695E                 call    Sleep
CODE:13156963
CODE:13156963 loc_13156963:
CODE:13156963                 mov     ecx, offset unk_1315A8BC
CODE:13156968                 xor     edx, edx        ; lDistanceToMove
CODE:1315696A                 mov     eax, ebx        ; hFile
CODE:1315696C                 call    sub_131547D8
CODE:13156971                 mov     ds:dword_1315A924, eax
CODE:13156976                 push    ebx             ; hObject
CODE:13156977                 call    CloseHandle
CODE:1315697C                 mov     eax, [ebp+hObject]
CODE:1315697F                 push    eax             ; hEvent
CODE:13156980                 call    SetEvent
CODE:13156985                 mov     eax, [ebp+hObject]
CODE:13156988                 push    eax             ; hObject
CODE:13156989                 call    CloseHandle
CODE:1315698E
CODE:1315698E loc_1315698E:
CODE:1315698E                 push    0FFFFFFFFh      ; bAlertable
CODE:13156990                 push    0FFFFFFFFh      ; dwMilliseconds
CODE:13156992                 call    SleepEx
CODE:13156997                 jmp     short loc_1315698E
CODE:13156999 ; ---------------------------------------------------------------------------
CODE:13156999
CODE:13156999 loc_13156999:
```

```
CODE:13156999                   xor     eax, eax
CODE:1315699B                   pop     edx
CODE:1315699C                   pop     ecx
CODE:1315699D                   pop     ecx
CODE:1315699E                   mov     fs:[eax], edx
CODE:131569A1                   push    offset loc_131569BB
CODE:131569A6
CODE:131569A6 loc_131569A6:
CODE:131569A6                   lea     eax, [ebp+var_7C]
CODE:131569A9                   mov     edx, 6
CODE:131569AE                   call    unknown_libname_12
CODE:131569B3                   retn
CODE:131569B3 new_process       endp ; sp = -90h
```

So you have basic knowledge how crackme is working. I'll talk only about steps that I've taken to unpack this crackme, now I have relized that some steps weren't necessery but what the hack…

By looking at user APC procedure stored in virtual.dll I noticed this part of code which will call driver:

```
CODE:1315461C attach_to_process_r0 proc near
CODE:1315461C NumberOfBytesWritten= dword ptr -44h
CODE:1315461C Buffer          = dword ptr -40h
CODE:1315461C var_3C          = dword ptr -3Ch
CODE:1315461C var_38          = dword ptr -38h
CODE:1315461C var_34          = dword ptr -34h
CODE:1315461C
CODE:1315461C                   add     esp, 0FFFFFFBCh
CODE:1315461F                   mov     [esp+44h+Buffer], 0Ah
CODE:13154627                   mov     [esp+8], edx
CODE:1315462B                   mov     [esp+12], eax
CODE:1315462F                   mov     [esp+10h], ecx
CODE:13154633                   push    0               ; lpOverlapped
CODE:13154635                   lea     eax, [esp+48h+NumberOfBytesWritten]
CODE:13154639                   push    eax             ; lpNumberOfBytesWritten
CODE:1315463A                   push    40h             ; nNumberOfBytesToWrite
CODE:1315463C                   lea     eax, [esp+50h+Buffer]
CODE:13154640                   push    eax             ; lpBuffer
CODE:13154641                   mov     eax, ds:hDevice
CODE:13154646                   push    eax             ; hFile
CODE:13154647                   call    WriteFile_0
CODE:1315464C                   add     esp, 44h
CODE:1315464F                   retn
CODE:1315464F attach_to_process_r0 endp
```

As you may see it is using Service ID 0Ah to tell driver something, and soon we will find out what, but we have to switch to driver disassembly now and take a look at switch/case in main_user_request_handler here:

```
.text:00011331                  push    dword ptr [eax+0Ch] ; case 0x9
.text:00011334                  push    dword ptr [eax+8] ; destination
.text:00011337                  push    dword ptr [eax+4] ; source
.text:0001133A                  call    PsGetCurrentProcessId
.text:0001133F                  push    eax
.text:00011340                  call    find_proc_struct
.text:00011345                  push    eax             ; proc_struct
.text:00011346                  call    copy_memory_from_r0
```

And we enter into copy_memory_from_r0:

```
.text:0001083C ; int __stdcall copy_memory_from_r0(int proc_struct,int source,int destination,SIZE_T
NumberOfBytes)
.text:0001083C copy_memory_from_r0 proc near            .text:0001083C
.text:0001083C proc_struct      = dword ptr  8
.text:0001083C source           = dword ptr  0Ch
.text:0001083C destination      = dword ptr  10h
```

```
.text:0001083C NumberOfBytes   = dword ptr  14h
.text:0001083C
.text:0001083C                 push    ebp
.text:0001083D                 mov     ebp, esp
.text:0001083F                 push    ebx
.text:00010840                 push    esi
.text:00010841                 push    edi
.text:00010842                 push    206B6444h       ; Tag
.text:00010847                 push    [ebp+NumberOfBytes] ; NumberOfBytes
.text:0001084A                 push    0               ; PoolType
.text:0001084C                 call    ds:ExAllocatePoolWithTag
.text:00010852                 mov     ecx, [ebp+NumberOfBytes]
.text:00010855                 mov     esi, [ebp+source]
.text:00010858                 mov     ebx, eax
.text:0001085A                 mov     eax, ecx
.text:0001085C                 shr     ecx, 2
.text:0001085F                 mov     edi, ebx        ; from esi to newly allocated nonpaged pool
.text:00010861                 rep movsd
.text:00010863                 mov     ecx, eax
.text:00010865                 mov     eax, [ebp+proc_struct]
.text:00010868                 and     ecx, 3
.text:0001086B                 rep movsb
.text:0001086D                 push    [eax+proc_struct.ptr_eprocess] ; e process of hiden_pid
.text:00010870                 call    KeAttachProcess
.text:00010875                 cli
.text:00010876                 mov     eax, cr0
.text:00010879                 mov     [ebp+source], eax ; save cr0
.text:0001087C                 and     eax, 0FFFEFFFFh ; wipe WP protection
.text:00010881                 mov     cr0, eax
.text:00010884                 mov     ecx, [ebp+NumberOfBytes]
.text:00010887                 mov     edi, [ebp+destination]
.text:0001088A                 mov     eax, ecx
.text:0001088C                 shr     ecx, 2
.text:0001088F                 mov     esi, ebx
.text:00010891                 rep movsd
.text:00010893                 mov     ecx, eax
.text:00010895                 and     ecx, 3
.text:00010898                 rep movsb
.text:0001089A                 mov     eax, [ebp+source]
.text:0001089D                 mov     cr0, eax
.text:000108A0                 sti
.text:000108A1                 call    KeDetachProcess
.text:000108A6                 push    0               ; Tag
.text:000108A8                 push    ebx             ; P
.text:000108A9                 call    ds:ExFreePoolWithTag
.text:000108AF                 pop     edi
.text:000108B0                 pop     esi
.text:000108B1                 pop     ebx
.text:000108B2                 pop     ebp
.text:000108B3                 retn    10h
.text:000108B3 copy_memory_from_r0 endp ; sp = -4
```

copy_memory_from_r0 is custom implementation of WriteProcessMemory. So APC procedure once it locates code that should be copied to hiden process will tell driver what to copy and where. Here you see usage of KeAttachProcess, but maybe some can ask why it is first copying data to nonpaged pool instead of direct copy? Answer is simple, KeAttachProcess WILL change value of cr3 which has physical address of page directory entry. When it happens source address of 2nd process won't be visible to us, and copying from nonexisitng address will result in page_fault, and remember what page_fualt in ring0 means.

So that's it about static disassembly, now we are moving to our activities with SoftICE, it is much easier when you can check you theory in practice. But still SoftICE cann't be loaded due to hooks in IDT. Now comes my workaround to make softice work with this crackme without any sideefects on it's execution.

4. SoftICE comes to rescue

I'm usally not impressed by my ideas but I really loved this one, and I was thinking about making SoftICE to work with this crackme for a 2-3 hours, finaly I set down and wrote everything on paper and started writing my driver to enable presence of SoftICE.

In driver there is only one procedure responsbile for interupt hooking which is called to hook and unhook interupts:

```
.text:000103FE hook_interupt   proc near
.text:000103FE
.text:000103FE var_C           = qword ptr -0Ch
.text:000103FE var_4           = dword ptr -4
.text:000103FE int_vector      = dword ptr  8      <-- arg0
.text:000103FE int_handler     = dword ptr  0Ch    <-- arg1
.text:000103FE arg_8           = word ptr  10h     <-- arg2
...
.text:0001042D                 cli
.text:0001042E                 call    sub_1039C   <-- real hook occurs here
.text:00010433                 sti
.text:00010434                 mov     eax, [ebp+var_4]  <-- old handle
.text:00010437                 pop     edi
.text:00010438                 pop     esi
.text:00010439                 pop     ebx
.text:0001043A                 leave
.text:0001043B                 retn    0Ch
.text:0001043B hook_interupt   endp
```

This procedure is realtively simple and is invoked from here, and also from Unhook_interupts:

```
.text:000107F8 hookints_drs    proc near
.text:000107F8                 push    0EEh
.text:000107FD                 push    offset new_int3_handle
.text:00010802                 push    3
.text:00010804                 call    hook_interupt
.text:00010809                 push    8Eh
.text:0001080E                 push    offset new_int1_handle
.text:00010813                 push    1
.text:00010815                 mov     old_int3_handle, eax
.text:0001081A                 call    hook_interupt
.text:0001081F                 mov     old_int1_handle, eax
.text:00010824                 cli
.text:00010825                 mov     eax, cr4
.text:00010828                 and     eax, 0FFFFFFF7h ; DE bit wiped out
.text:0001082B                 mov     cr4, eax
.text:0001082E                 xor     eax, eax
.text:00010830                 mov     dr7, eax        ; dr7 to 0 (no local nor global bpms)
.text:00010833                 mov     dr1, eax        ; dr1 and dr0 are also zeroed
.text:00010836                 mov     dr0, eax
.text:00010839                 sti
.text:0001083A                 retn
.text:0001083A hookints_drs    endp
```

As you may see hook interupt will return address of old handler, and hook it with passed one, so since interupt hooking isn't nuclear physics we may detour this procedure to our driver, and we are also going to detour int handlers in dxapi32.sys so both of them will point to SoftICE handlers.

This is accomplished by detouring crackme int 1/3 handlers to handlers installed by SoftICE, on other hand SoftICE handler look like this:

```
        push    _KiTrapXX
        jmp     __softice
```

So if SoftICE doesn't handle int1/int3 or other trap/fault it will call KiTrapXX that is pushed on stack, now comes my idea. But this time we are going to use smart approach on hooking driver because one mistake and you will get BSOD (not from fact that driver wanna kill you but because SoftICE installs some int3h in some native APIs to keep track of something – exports + KeServiceDescriptorTable!? Wrongly handled int3h will BSOD system)…

Oki I will rip 5 bytes from dxapi32.sys int1/int3 handler and will detour them to SoftICE handler, but I will also change:

```
        push    _KiTrapXX       >>>>>>>>>>>>>>  push    __stolen_bytesint1/3>>>
        jmp     __softice       >>>>>>>>>>>>>>  jmp     __softice             V
                                                                              V
__stolen_bytes1/3:      pusha   <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                        mov     di, 30h
                        mov     fs, di
                        nop
                        ...
                        push    __bytes_after_hook
                        retn
```

Maybe disassembly from softice will make more sense?

```
:idt
Int   Type     Sel:Offset     Attributes Symbol/Owner
IDTbase=8003F400  Limit=07FF
0000  IntG32   0008:804D8BFF  DPL=0  P   _KiTrap00
0001  IntG32   0008:F728A7BC  DPL=3  P   dxapi32!.text+04BC
0002  TaskG    0058:00000000  DPL=0  P   _KiTrap02
0003  IntG32   0008:F728A78A  DPL=3  P   dxapi32!.text+048A
0004  IntG32   0008:804D92E0  DPL=3  P   _KiTrap04
...
:idt 3
0003  IntG32   0008:F728A78A  DPL=3  P   dxapi32!.text+048A
:u f728a78a
0008:F728A78A  JMP      _F8AB14EC        <-- detour to SoftICE
0008:F728A78F  MOV       FS,DI
0008:F728A792  MOV       EAX,[ESP+20]
0008:F728A796  DEC       EAX
0008:F728A797  PUSH      EAX
0008:F728A798  CALL     _F728A6CE
0008:F728A79D  TEST      EAX,EAX
0008:F728A79F  JNZ      _F728A7AD
...
:u f8ab14ec
0008:F8AB14EC  PUSH      F8AB14BE        <-- 5bytes + jmp back to dxapi32.sys
0008:F8AB14F1  JMP      _F87B770D        <-- jmp __SoftICE
0008:F8AB14F6  MOV       EAX,[ESP+04]
0008:F8AB14FA  MOV       EDI,[F8AB12A4]
0008:F8AB1500  LEA       EDI,[EAX*8+EDI]
0008:F8AB1503  MOVZX     EAX,WORD PTR [EDI+06]
0008:F8AB1507  SHL       EAX,10
0008:F8AB150A  MOV       AX,[EDI]
...
:u f8ab14be
0008:F8AB14BE  PUSHAD                    <-- stolen bytes overwriten by
0008:F8AB14BF  MOV       DI,0030         <-- hook to SoftICE
0008:F8AB14C3  NOP
```

```
0008:F8AB14C4  NOP
0008:F8AB14D7  NOP
0008:F8AB14D8  NOP                          <-- BPX here (or any nop) if you
0008:F8AB14D9  NOP                              wanna debug dxapi32.sys int3
0008:F8AB14DA  NOP                              handler
0008:F8AB14DB  NOP
0008:F8AB14DC  PUSH      F728A78F           <-- jmp to rest of code
0008:F8AB14E1  RET
...
:u f728a78f
0008:F728A78F  MOV       FS,DI              <-- dxapi32.sys int3h handle
0008:F728A792  MOV       EAX,[ESP+20]
0008:F728A796  DEC       EAX
0008:F728A797  PUSH      EAX
0008:F728A798  CALL     _F728A6CE
0008:F728A79D  TEST      EAX,EAX
0008:F728A79F  JNZ      _F728A7AD
0008:F728A7A1  MOV       DI,003B
0008:F728A7A5  MOV       FS,DI
0008:F728A7A8  POPAD
0008:F728A7A9  DEC       BYTE PTR [ESP]
0008:F728A7AC  IRETD
0008:F728A7AD  MOV       DI,003B
0008:F728A7B1  MOV       FS,DI
0008:F728A7B4  POPAD
0008:F728A7B5  JMP       [_KiTrap03]     <-- KiTrap03 handle
```

Looks nice J So SoftICE handler will always execute first, giving chance to SoftICE to handle single steping and step overs without a problem, now I can also debug int3 handle of dxapi32.sys, isn't it nice. Also I have to hook intterupt_hook so I can control when crackme wants to hook and to unhook interupt.

If hooking is performed then write to SoftICE hook addresses of detoured handlers, and return address of KiTrap01 or KiTrap03 depending which int is being hooked, if unhooking is performed then write to SoftICE handler (push) KiTrap01 and KiTrap03 and in restore SoftICE interupts in IDT. If we don't return KiTrap01/KiTrap03 during this hooking then there is chance of falling into deadlock…

Example -> driver hooks IDT and takes old handler from there -> driver handle is hooked to point to SoftICE -> int3 -> softICE handler takes over control and it doesn't handle it -> driver takes control and decides that this is not int 3h in hiden process so it will call old handle -> but wait!? Old handle is softice handle which doesn't handle int3h so it calls again dxapi32.sys handle and so on, so on…

For detailed implementation check anti-msrem1.asm driver, it works like charm, and thanks to it I could better undertsand my target.

5. Removing int3h from crackme.exe

As I mentioned earlier, crackme will use APC in second process to handle int3h in hiden process. To get better understand of what is happening here I have used i3here on in softice to break when int3h occurs (remember SoftICE handles int3 first with anti-msrem1.asm driver loaded), next thing for me was to set bpx at copy_memory_from_r0 in Write handler and to observer what is going on. Maybe I could analyze better APC procedure but I really liked my idea about SoftICE presence and I've decided to give it a shot.

I have set breaks and watched what is copied, where and how.
What I saw was more then enough to get right conclusion. First crackme will overwrite all stolen code block with int3h, it is logical step because only int3h will force it's restoring, but after all blocks are overwriten with int3h, then real code will be copied over one of int3h blocks. This copying is used so we never get fully reconstructed code in crackme. Ms-Rem was very nice and has included real serial with this crackme so we can log all places where stolen codes are. Anyway, real serial wasn't necessery because by int3h buffer overwriting we could find all places where stolen code is. Without real key we could locate all bufferes but then we should manualy redirect execution to those places and get real code or else better analyze APC procedure in virtual.dll.

First to get all buffers I hooked copy_memory_from_r0 (check log.asm driver) and logged where are all addresses:

```
00000000    0.00000000  Destination - 0x00452325
00000001    0.00003297  Destination - 0x00452365
00000002    0.00005364  Destination - 0x0045242C
00000003    0.00007347  Destination - 0x00452486
00000004    0.00009303  Destination - 0x00452548
00000005    0.00011286  Destination - 0x004525BD
00000006    0.00013270  Destination - 0x00452684
00000007    0.00015253  Destination - 0x004526DF
00000008    0.00017237  Destination - 0x00452722
00000009    0.00019192  Destination - 0x0045276D
00000010    0.00021148  Destination - 0x004527B4
00000011    0.00023103  Destination - 0x0045281B
00000012    0.00025087  Destination - 0x004528BF
```

My hook was stored here:

```
.text:00010884                mov     ecx, [ebp+NumberOfBytes]
.text:00010887                mov     edi, [ebp+destination]
.text:0001088A                mov     eax, ecx    <-- hook here
.text:0001088C                shr     ecx, 2      <-- and here
.text:0001088F                mov     esi, ebx
.text:00010891                rep movsd
.text:00010893                mov     ecx, eax
.text:00010895                and     ecx, 3
.text:00010898                rep movsb
```

Here we have more then enough for hooking(5 bytes), we have size of copied block, and we have also destination buffer in EDI and source buffer in EBX, so now I can log all buffers and also dump them into files and later simply apply those dumps to dumped crackme and it is unpacked.

Again I will suggest to check my log.asm driver, it isn't very complex so you will be able to understand it easily. That's pretty much about dumping and fixing.

6. Process hiding from ring3

Even if target is fully deprotected, there are still some more stuff left to fix, one of them is to disable process hiding from ring3. I will sum it here, but for detaild explanation about this trick you should refer to crazylord's paper in Phrack named "Playing with windows /dev/(k)mem" [1].

First open your target in IDA and find reference to ZwOpenSection, you will see also how privileges for \Device\PhysicalMemory are being adjusted so it can be opened and modified, all of this is documented in paper I mentioned.

Then it enters into GDT modification and installs this callgate:

```
CODE:004518CE                   mov     eax, offset call_gate_procedure
CODE:004518D3                   mov     edx, [ebx]
CODE:004518D5                   mov     [edx], ax
CODE:004518D8                   shr     eax, 10h
CODE:004518DB                   mov     edx, [ebx]
CODE:004518DD                   mov     [edx+6], ax
CODE:004518E1                   mov     eax, [ebx]
CODE:004518E3                   mov     word ptr [eax+4], 0EC00h
CODE:004518E9                   xor     eax, eax
CODE:004518EB                   mov     dword ptr ds:callgate_descriptor, eax
CODE:004518F0                   mov     ax, [ebx]
CODE:004518F3                   sub     ax, word ptr ds:lpBaseAddress
CODE:004518FA                   sub     ax, word ptr [ebp+var_C]
CODE:004518FE                   mov     word ptr ds:callgate_descriptor+4, ax
```

As you may see in this code it writes address of call_gate_procedure to free slot in GDT and also stores Selector into callgate_descriptor+4. When we are calling callgate only thing that is relevant in it is Selector while address is not important. By executing far call to that callgate CPU will automaticaly redirect execution to address sotred in GDT.

Next thing we are going to examine is call_gate_procedure:

```
CODE:004516A4 call_gate_procedure:
CODE:004516A4                   cli
CODE:004516A5                   pusha
CODE:004516A6                   pushf
CODE:004516A7                   mov     di, 30h
CODE:004516AB                   mov     fs, di
CODE:004516AE                   call    ds:callgate_current
CODE:004516B4                   mov     di, 3Bh
CODE:004516B8                   mov     fs, di
CODE:004516BB                   popf
CODE:004516BC                   popa
CODE:004516BD                   sti
CODE:004516BE                   retf
```

If you observe this code you may see how it is acting like callgate dispatcher and will execute procedure that is stored in variable callgate_current, there are only 3 calls to callgate dispatcher, also cli and sti are used to disable task switching while we are in callgate.

Before we enter into process hiding techinque we are going to see also how it gets needed offsets for EPROCESS struct and also some addresses from maped ntoskrnl.exe here:

```
CODE:00451D50 win_xp:
CODE:00451D50                 mov     ds:ActiveProcessLinks, 88h
CODE:00451D5A                 mov     ds:UniqueProcessId, 84h
CODE:00451D64                 mov     ds:ImageFileName, 174h
CODE:00451D6E                 mov     ds:InheritedFromUniqueProcessId, 14Ch
CODE:00451D78                 mov     ds:SeAuditProcessCreationInfo, 1F4h
CODE:00451D82
CODE:00451D82 loc_451D82:
CODE:00451D82                 mov     eax, offset aNtoskrnl_exe ; "ntoskrnl.exe"
CODE:00451D87                 call    getntoskrnlbase
CODE:00451D8C                 mov     ds:real_ntoskrnl_base, eax
CODE:00451D91                 push    1               ; dwFlags
CODE:00451D93                 push    0               ; hFile
CODE:00451D95                 push    offset aNtoskrnl_exe ; "ntoskrnl.exe"
CODE:00451D9A                 call    LoadLibraryExA_0
CODE:00451D9F                 mov     ds:mapped_ntoskrnl_base, eax
CODE:00451DA4                 mov     eax, offset aMmgetphysicala ; "MmGetPhysicalAddress"
CODE:00451DA9                 call    find_exports
CODE:00451DAE                 mov     ds:MmGetPhysicalAddress, eax
CODE:00451DB3                 mov     eax, offset aMmisaddressval ; "MmIsAddressValid"
CODE:00451DB8                 call    find_exports
CODE:00451DBD                 mov     ds:MmIsAddressValid, eax
CODE:00451DC2                 mov     eax, offset aIogetcurrentpr ; "IoGetCurrentProcess"
CODE:00451DC7                 call    find_exports
CODE:00451DCC                 mov     ds:IoGetCurrentProcess, eax
CODE:00451DD1                 mov     eax, offset aKe386setioacce ; "Ke386SetIoAccessMap"
CODE:00451DD6                 call    find_exports
CODE:00451DDB                 mov     ds:Ke386SetIoAccessMap, eax
CODE:00451DE0                 mov     eax, offset aKe386queryioac ; "Ke386QueryIoAccessMap"
CODE:00451DE5                 call    find_exports
CODE:00451DEA                 mov     ds:Ke386QueryIoAccessMap, eax
CODE:00451DEF                 mov     eax, offset aKe386iosetacce ; "Ke386IoSetAccessProcess"
CODE:00451DF4                 call    find_exports
CODE:00451DF9                 mov     ds:Ke386IoSetAccessProcess, eax
CODE:00451DFE                 mov     eax, offset aExallocatepool ; "ExAllocatePool"
CODE:00451E03                 call    find_exports
CODE:00451E08                 mov     ds:ExAllocatePool, eax
CODE:00451E0D                 mov     eax, offset aExfreepool ; "ExFreePool"
CODE:00451E12                 call    find_exports
CODE:00451E17                 mov     ds:ExFreePool, eax
CODE:00451E1C                 mov     ds:dword_455C14, esi
CODE:00451E22                 mov     eax, ds:dword_455C14
CODE:00451E27                 sub     eax, 1
```

Here you may see how it stores offsets to important part of EPROCESS depending on it if it is win2k or XP, also it will locate address of ntoskrnl.exe in memory, and addresses of important exports of ntoskrnl.exe, non of them is rebased according to base of real ntoskrnl.exe because only one of them is used and that is IoGetCurrentProcess but this one only is only retriving EPROCESS of current process so it doesn't require rebeasing. I mean it can be called from ntosknl.exe mapped in crackme.

Oki now lets examine calls to callgate, there is only 3 of them, and those are called from here:

```
CODE:00451A48 locate_process  proc near
CODE:00451A48
CODE:00451A48 var_8           = dword ptr -8
CODE:00451A48 var_4           = dword ptr -4
CODE:00451A48
CODE:00451A48                 add     esp, 0FFFFFFF8h
CODE:00451A4B                 mov     dword ptr [esp], offset EPROCESS
CODE:00451A52                 mov     edx, esp
CODE:00451A54                 mov     eax, offset IoGetCurrentProcess_callgate
CODE:00451A59                 call    call_callgate
CODE:00451A5E                 mov     eax, [esp+4]
CODE:00451A62                 pop     ecx
CODE:00451A63                 pop     edx
CODE:00451A64                 retn
CODE:00451A64 locate_process  endp
```

First we receive EPROCESS of, I suppose of SMSS.exe, only parentless process is SMSS.exe created by ntoskrnl.exe, for more refer to Barnaby Jack's aritcle about "Step Into The Ring0" [2].

```
CODE:00451A24 IoGetCurrentProcess_callgate:
CODE:00451A24                 mov     ebx, eax        ; stack location with args in ring3
CODE:00451A26                 call    ds:IoGetCurrentProcess
CODE:00451A2C                 mov     edx, [ebx]      ; Arguments buffer
CODE:00451A2E                 mov     esi, [edx+eprocess_struct.ep_activeprocesslinks]
CODE:00451A31                 mov     edi, [edx+eprocess_struct.ep_InheritedFromUniqueProcessId]
CODE:00451A34
CODE:00451A34 __cycle:
CODE:00451A34                 mov     ecx, [edi+eax]
CODE:00451A37                 test    ecx, ecx
CODE:00451A39                 jz      short __exit_callgate_0
CODE:00451A3B                 mov     eax, [esi+eax]  ; next EPROCESS in list entry
CODE:00451A3E                 sub     eax, esi        ; negative offset to start of EPROCESS
CODE:00451A40                 jmp     short __cycle
CODE:00451A42 __exit_callgate_0:
CODE:00451A42                 mov     [ebx+4], eax    ; save EPROCESS
CODE:00451A45                 retn
```

Next one is for locating EPROCESS of given PID, but this time PID is current process:

```
CODE:00451FEC findprocess_callgate:
CODE:00451FEC                 mov     ebx, [eax]      ; EPROCESS struct
CODE:00451FEE                 mov     ecx, [eax+4]    ; Current process ID
CODE:00451FF1                 push    eax             ; save arguments list
CODE:00451FF2                 mov     eax, [ebx]      ; eax = EPROCESS
CODE:00451FF4                 mov     esi, [ebx+eprocess_struct.ep_activeprocesslinks]
CODE:00451FF7                 mov     edi, [ebx+eprocess_struct.ep_uniqueprocessid]
CODE:00451FFA
CODE:00451FFA __find_my_eprocess:
CODE:00451FFA                 mov     edx, [edi+eax]  ; get process ID from EPROCESS
CODE:00451FFD                 cmp     edx, ecx        ; same as PID of process that we are
CODE:00451FFF                 jz      short __found_my_eprocess ; looking for
CODE:00452001                 mov     eax, [esi+eax]  ; ActiveProcessLinks.Flink
CODE:00452004                 sub     eax, esi        ; start of EPROCESS
CODE:00452006                 cmp     eax, [ebx]      ; is it same EPROCESS as from where
CODE:00452008                 jz      short __exit_find_eprocess ;we have started
CODE:0045200A                 jmp     short __find_my_eprocess
CODE:0045200C
CODE:0045200C __found_my_eprocess:
CODE:0045200C                 pop     edx
CODE:0045200D                 mov     [edx+8], eax
CODE:00452010                 retn
CODE:00452011 __exit_find_eprocess:
CODE:00452011                 pop     edx
CODE:00452012                 mov     dword ptr [edx+8], 0
CODE:00452019                 retn
```

Unlinking occurs in 3rd callgate:

```
CODE:00452040 erase_process_callgate:
CODE:00452040                 push    eax
CODE:00452041                 mov     eax, [eax+4]    ; my EPROCESS
CODE:00452044                 push    eax
CODE:00452045                 call    ds:MmIsAddressValid
CODE:0045204B                 test    eax, eax
CODE:0045204D                 jz      short __wrong_address
CODE:0045204F                 pop     eax
CODE:00452050                 mov     ebx, [eax]      ; struct offset
CODE:00452052                 mov     ecx, [eax+4]    ; EPROCESS
CODE:00452055                 mov     esi, [ebx+eprocess_struct.ep_activeprocesslinks]
CODE:00452058                 mov     edx, [esi+ecx]  ; EDX next EPROCESS
CODE:0045205B                 add     esi, 4
CODE:0045205E                 mov     edi, [esi+ecx]  ; ESI previous EPROCESS
CODE:00452061                 mov     [edx+LIST_ENTRY.Blink], edi ; unlink EPROCESS
CODE:00452064                 mov     [edi+LIST_ENTRY.Flink], edx ; unlink EPROCESS
```

```
CODE:00452066                   retn
CODE:00452067
CODE:00452067 __wrong_address:
CODE:00452067                   pop     eax
CODE:00452068                   retn
```

All you can do now to make process vivible is to patch rotuines that open \Device\PhysicalMemory at this place:

```
CODE:004528D5                   call    preapare_for_callgate ; patch here
CODE:004528DA                   call    GetCurrentProcessId
CODE:004528DF                   call    unlink_process  ; and here
```

Now we are finaly done and we have removed all anti-debug from this crackme, really nice and chalanging crackme. Real master piece.

7. Conslusion

I hope that someone will benefit from this essay, I don't have much to say in this chapter except that I had really fun solving this crackme. I hope you have learnt something new by reading this solution.

8. References

[1] Playing with Windows /dev/(k)mem, crazylord,
http://www.phrack.org/archives/59/p59-0x10.txt
[2] Remote Windows Kernel Exploitation – Step Into the Ring 0, Barnaby Jack,
http://research.eeye.com/html/papers/

S verom u Boga, deroko/ARTeam

9. Greetings

I wish to tank all the ARTeam members for sharing their knowledge, to 29a virus writing group for one of the best e-zines, Ms-Rem for this crackme… and of course you for reading this article.

http://cracking.accessroot.com