

在本教程的开头我就说过，WIN64 有两个内核保护机制，KPP 和 DSE。KPP 阻止我们 PATCH 内核，DSE 拦截我们加载驱动。当然 KPP 和 DSE 并不是不可战胜的，WIN7X64 出来不久，FYYRE 就发布了破解内核的工具，后来有个叫做 Feryno 的人又公开了源代码，接下来我结合 FYYRE 的文档 Feryno 的源码进行讲解。

要实现突破 DSE 和 PatchGuard，必须修改两个文件，winload.exe 以及 ntoskrnl.exe。首先把 WINLOAD.EXE 扔到 IDA 里，看看要修改哪些地方：

```
.text:00000000004057E8 OslInitializeCodeIntegrity proc near      ; CODE XREF: OslpMain+61Cp
.text:00000000004057E8                                     ; DATA XREF: .pdata:00000000004B2168o
.text:00000000004057E8
.text:00000000004057E8 var_58             = qword ptr -58h
.text:00000000004057E8 var_50             = dword ptr -50h
.text:00000000004057E8 var_48             = dword ptr -48h
.text:00000000004057E8 var_38             = qword ptr -38h
.text:00000000004057E8 arg_8              = qword ptr 10h
.text:00000000004057E8 arg_18             = qword ptr 20h
.text:00000000004057E8
.text:00000000004057E8             mov     rax, rsp
.text:00000000004057EB             push   rbx
.text:00000000004057EC             push   rbp
.text:00000000004057ED             push   rdi
.text:00000000004057EE             push   r12
.text:00000000004057F0             push   r13
.text:00000000004057F2             sub     rsp, 50h
.text:00000000004057F6             xor     r13d, r13d
.text:00000000004057F9             mov     r12d, ecx
.text:00000000004057FC             lea     r8, [rax+18h]
.text:0000000000405800             lea     rcx, BlpApplicationEntry
.text:0000000000405807             lea     rdx, [rax+10h]
.text:000000000040580B             mov     [rax+20h], r13
.text:000000000040580F             mov     rdi, r13
.text:0000000000405812             mov     [rax-38h], r13
.text:0000000000405816             call   BlImgQueryCodeIntegrityBootOptions
```

需要修改的是加粗的那行（00000000004057E8）：

mov rax, rsp	48h, 8Bh, C4h
--------------	---------------

修改为：

mov al, 1	B0h, 01h
ret	C3h

这么做的用途是跳过对 BlImgQueryCodeIntegrityBootOptions 的调用，据我了解，此函数会判断 NTOSKRNL.EXE 的数字签名有效性（签名非法的话就拒绝加载 NTOSKRNL.EXE）。所以这是一个难缠的家伙，直接跳过。

接下来修改 ntoskrnl.exe。第一个地方是 SepInitializeCodeIntegrity:

```

PAGE:00000001403EAA60 SepInitializeCodeIntegrity proc near
; CODE XREF: SepInitializationPhase1+231p
PAGE:00000001403EAA60
PAGE:00000001403EAA60 arg_0          = qword ptr 8
PAGE:00000001403EAA60
PAGE:00000001403EAA60          mov     [rsp+arg_0], rbx
PAGE:00000001403EAA65          push   rdi
PAGE:00000001403EAA66          sub     rsp, 20h
PAGE:00000001403EAA6A          xor     ebx, ebx
PAGE:00000001403EAA6C          cmp     cs:InitIsWinPEMode, bl
PAGE:00000001403EAA72          jnz     loc_1403EAB0C
PAGE:00000001403EAA78          xor     eax, eax
PAGE:00000001403EAA7A          mov     cs:g_CiEnabled, 1
PAGE:00000001403EAA81          lea     edi, [rbx+6]
PAGE:00000001403EAA84          mov     cs:g_CiCallbacks, rax
PAGE:00000001403EAA8B          mov     cs:qword_14021EE48, rax
PAGE:00000001403EAA92          mov     cs:qword_14021EE50, rax
PAGE:00000001403EAA99          mov     rax, cs:qword_1402A8120
PAGE:00000001403EAAA0          cmp     rax, rbx
PAGE:00000001403EAAA3          jz      short loc_1403EAAF7
PAGE:00000001403EAAA5          cmp     [rax+98h], rbx
PAGE:00000001403EAAAC          jz      short loc_1403EAAEE
PAGE:00000001403EAAAE          mov     rcx, [rax+98h]
PAGE:00000001403EAB5          lea     rdx, ??_C@_OBJ@KFBEEJ@DISABLE_INTEGRITY_CHECKS?$AA@NNGAKEGL@
PAGE:00000001403EABBC          call    SepIsOptionPresent
PAGE:00000001403EABAC          mov     rcx, cs:qword_1402A8120
PAGE:00000001403EAB0C          lea     rdx, ??_C@_OM@LNFBGLD@TESTSIGNING?$AA@NNGAKEGL@
PAGE:00000001403EABCF          mov     rcx, [rcx+98h]
PAGE:00000001403EABD6          cmp     eax, ebx
PAGE:00000001403EABD8          cmovnz  edi, ebx
PAGE:00000001403EABDB          call    SepIsOptionPresent
PAGE:00000001403EABE0          cmp     eax, ebx
PAGE:00000001403EABE2          mov     rax, cs:qword_1402A8120
PAGE:00000001403EABE9          jz      short loc_1403EABEE
PAGE:00000001403EABEB          or      edi, 8
PAGE:00000001403EABEE
PAGE:00000001403EABEE loc_1403EABEE: ; CODE XREF: SepInitializeCodeIntegrity+4Cj
PAGE:00000001403EABEE                                     ;
SepInitializeCodeIntegrity+89j
PAGE:00000001403EABEE          cmp     rax, rbx
PAGE:00000001403EABF1          jz      short loc_1403EABF7
PAGE:00000001403EABF3          lea     rbx, [rax+30h]
PAGE:00000001403EABF7

```

```

PAGE:00000001403EAAF7 loc_1403EAAF7:
; CODE XREF: SepInitializeCodeIntegrity+43j
PAGE:00000001403EAAF7
SepInitializeCodeIntegrity+91j
PAGE:00000001403EAAF7          lea     r8, g_CiCallbacks
PAGE:00000001403EAAFE          mov     rdx, rbx
PAGE:00000001403EAB01          mov     ecx, edi
PAGE:00000001403EAB03          call   CiInitialize
PAGE:00000001403EAB08          mov     ebx, eax
PAGE:00000001403EAB0A          jmp     short loc_1403EAB12
PAGE:00000001403EAB0C
PAGE:00000001403EAB0C loc_1403EAB0C:
; CODE XREF: SepInitializeCodeIntegrity+12j
PAGE:00000001403EAB0C          mov     cs:g_CiEnabled, bl
PAGE:00000001403EAB12
PAGE:00000001403EAB12 loc_1403EAB12:
; CODE XREF: SepInitializeCodeIntegrity+AAj
PAGE:00000001403EAB12          mov     eax, ebx
PAGE:00000001403EAB14          mov     rbx, [rsp+28h+arg_0]
PAGE:00000001403EAB19          add     rsp, 20h
PAGE:00000001403EAB1D          pop     rdi
PAGE:00000001403EAB1E          retn
PAGE:00000001403EAB1E SepInitializeCodeIntegrity endp

```

需要修改的是加粗的那行（00000001403EAA72）：

jnz loc_1403EAB0C	0Fh, 85h, 94h, 00h, 00h, 00h
-------------------	------------------------------

修改为：

nop	90h
jmp loc_1403EAB0C	E9h, 94h, 00h, 00h, 00h

这样做的目的是跳过“是否为 WINPE 模式”的判断，强行转入系统是 WINPE 模式的处理过程（loc_1403EAB0C）。因为如果是 WINPE 模式的话就不会校验驱动的数字签名。

第二个地方（在公开的符号里，这个函数没有名称，很明显是微软把它从符号表里抹掉了，但 FYYRE 表示，此函数的名字是 KiInitializePatchGuard）：

```

INIT:0000000140561340 sub_140561340 proc near
; CODE XREF: KiFilterFiberContext+FFp
INIT:0000000140561340          ; KiFilterFiberContext+187p
INIT:0000000140561340
INIT:0000000140561340 var_F78      = qword ptr -0F78h
INIT:0000000140561340 var_F70      = qword ptr -0F70h
INIT:0000000140561340 var_F68      = qword ptr -0F68h
INIT:0000000140561340 var_F60      = qword ptr -0F60h

```

```

INIT:0000000140561340 var_F58      = dword ptr -0F58h
...
INIT:0000000140561340 var_48      = byte ptr -48h
INIT:0000000140561340 arg_0       = dword ptr  8
INIT:0000000140561340 arg_8       = dword ptr 10h
INIT:0000000140561340 arg_10      = dword ptr 18h
INIT:0000000140561340 arg_18      = qword ptr 20h
INIT:0000000140561340
INIT:0000000140561340      mov     [rsp+arg_10], r8d
INIT:0000000140561345      mov     [rsp+arg_8], edx
INIT:0000000140561349      mov     [rsp+arg_0], ecx
INIT:000000014056134D      push    rbx
INIT:000000014056134E      push    rbp
INIT:000000014056134F      push    rsi
INIT:0000000140561350      push    rdi
INIT:0000000140561351      push    r12
INIT:0000000140561353      push    r13
INIT:0000000140561355      push    r14
INIT:0000000140561357      push    r15
INIT:0000000140561359      sub     rsp, 0F58h
INIT:0000000140561360      xor     edi, edi
INIT:0000000140561362      cmp     cs:InitSafeBootMode, edi
INIT:0000000140561368      jz      short loc_140561371
INIT:000000014056136A      mov     al, 1
INIT:000000014056136C      jmp     loc_1405640D9
INIT:0000000140561371
INIT:0000000140561371 loc_140561371:  ; CODE XREF: sub_140561340+28j
INIT:0000000140561371      lea     rbx, FsRtlUninitializeSmallMcb
INIT:0000000140561378      lea     rdx, [rsp+0F98h+var_E40]
INIT:0000000140561380      mov     rcx, rbx
INIT:0000000140561383      call    RtlPcToFileHeader
INIT:0000000140561388      cmp     rax, rdi
INIT:000000014056138B      jz      loc_1405640D7
INIT:0000000140561391      mov     rcx, [rsp+0F98h+var_E40]
INIT:0000000140561399      call    RtlImageNtHeader
INIT:000000014056139E      cmp     rax, rdi
INIT:00000001405613A1      jz      loc_1405640D7
...
INIT:00000001405640D9 loc_1405640D9:  ; CODE XREF: sub_140561340+2Cj
INIT:00000001405640D9      ; sub_140561340+9C36j
INIT:00000001405640D9      add     rsp, 0F58h
INIT:00000001405640E0      pop     r15
INIT:00000001405640E2      pop     r14
INIT:00000001405640E4      pop     r13

```

INIT:00000001405640E6	pop	r12
INIT:00000001405640E8	pop	rdi
INIT:00000001405640E9	pop	rsi
INIT:00000001405640EA	pop	rbp
INIT:00000001405640EB	pop	rbx
INIT:00000001405640EC	retn	
INIT:00000001405640EC	endp	

需要修改的是加粗的那行（0000000140561368）：

jz short loc_140561371	74h, 07h
------------------------	----------

修改为：

nop	90h
nop	90h

这样做的目的是跳过“是否为安全模式”的判断，强制转入系统是安全模式的处理过程。因为如果是安全模式，就不会初始化 PatchGuard（fyyre 在她的博客里解释说：PatchGuard does not initialize if we boot into safe mode.）。

经过这三处修改，DSE 和 PatchGuard 就废了。但修改完文件还要重新计算文件的 checksum 才行，否则系统是无法启动的。以上所有步骤用编程手段来实现就是：读取 PE 文件到 buffer -> 根据特征码搜索要修改的位置 -> 修改特定位置的内容 -> 重新计算 checksum -> 把修改过的 buffer 输出为文件。核心代码如下：

```

; //字节补丁
patch_bytes:
    push    rbx rsi rdi
    push    rcx
    pop     rsi
    cld
    lodsb                                ; load size of bytes to find
    movzx   ebx, al
    lea     rdi, [file_buf]
    lea     r8, [rdi+rdx]                ; end of file

align 10h
patch_bytes_L0:
    push    rsi rdi
    mov     ecx, ebx
    repz    cmpsb
    pop     rdi rsi
    jz      patch_bytes_L4
    scasb                                ; rdi+1 in 1-byte instruction
    lea     rax, [rdi+rbx*1]

```

```

        cmp        rax, r8
        jbe        patch_bytes_L0
; end of file reached
        jmp        patch_bytes_L9

patch_bytes_L4:
; matching bytes found, patch them
        add        rsi, rbx
        lodsb
; load size of bytes to be written
        movzx      ecx, al
        repz movsb
patch_bytes_L9:
        pop        rdi rsi rbx
        ret

; //重新计算 checksum
reconstruct_crc:
; in: ecx = size of file

struct    IMAGE_DOS_HEADER
        e_magic                rw        1        ; Magic number
        e_cblp                rw        1        ; Bytes on last page
of file
        e_cp                  rw        1        ; Pages in file
        e_crlc                rw        1        ; Relocations
        e_cparhdr            rw        1        ; Size of header in paragraphs
        e_minalloc            rw        1        ; Minimum extra paragraphs
needed
        e_maxalloc            rw        1        ; Maximum extra paragraphs
needed
        e_ss                  rw        1        ; Initial (relative)
SS value
        e_sp                  rw        1        ; Initial SP value
        e_csum                rw        1        ; Checksum
        e_ip                  rw        1        ; Initial IP value
        e_cs                  rw        1        ; Initial (relative)
CS value
        e_lfarlc            rw        1        ; File address of relocation
table
        e_ovno                rw        1        ; Overlay number
        e_res                rw        4        ; Reserved words
        e_oemid                rw        1        ; OEM identifier (for
e_oeminfo)
        e_oeminfo            rw        1        ; OEM information; e_oemid
specific

```

e_res2		rw	10	; Reserved words
e_lfanew	rd	1		; File address of new exe
header				
ends				
IMAGE_DOS_SIGNATURE	=		'MZ'	
struct	IMAGE_FILE_HEADER			
Machine		rw	1	
NumberOfSections	rw	1		
TimeDateStamp		rd	1	
PointerToSymbolTable	rd	1		
NumberOfSymbols		rd	1	
SizeOfOptionalHeader	rw	1		
Characteristics		rw	1	
ends				
IMAGE_SIZEOF_FILE_HEADER	=		sizeof. IMAGE_FILE_HEADER	; = 20
IMAGE_FILE_MACHINE_AMD64	=		8664h	; AMD64 (K8)
struct	IMAGE_DATA_DIRECTORY			
VirtualAddress		rd	1	
Size		rd	1	
ends				
IMAGE_NUMBEROF_DIRECTORY_ENTRIES	=		16	
struct	IMAGE_OPTIONAL_HEADER64			
; Standard fields.				
Magic		rw	1	
MajorLinkerVersion	rb	1		
MinorLinkerVersion	rb	1		
SizeOfCode	rd	1		
SizeOfInitializedData		rd	1	
SizeOfUninitializedData		rd	1	
AddressOfEntryPoint	rd	1		
BaseOfCode	rd	1		
; NT additional fields.				
ImageBase	rq	1		
SectionAlignment	rd	1		
FileAlignment		rd	1	
MajorOperatingSystemVersion	rw	1		
MinorOperatingSystemVersion	rw	1		
MajorImageVersion	rw	1		
MinorImageVersion	rw	1		
MajorSubsystemVersion		rw	1	
MinorSubsystemVersion		rw	1	

```

        Win32VersionValue          rd      1
        SizeOfImage                 rd      1
        SizeOfHeaders               rd      1
        CheckSum                    rd      1
        Subsystem                   rw      1
        DllCharacteristics           rw      1
        SizeOfStackReserve           rq      1
        SizeOfStackCommit            rq      1
        SizeOfHeapReserve            rq      1
        SizeOfHeapCommit             rq      1
        LoaderFlags                  rd      1
        NumberOfRvaAndSizes          rd      1
;    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
        DataDirectory                IMAGE_DATA_DIRECTORY
                                   rb
                                   (IMAGE_NUMBEROF_DIRECTORY_ENTRIES-1)*(sizeof. IMAGE_DATA_DIRECTORY)
ends
IMAGE_SIZEOF_NT_OPTIONAL64_HEADER = 240
IMAGE_NT_OPTIONAL_HDR64_MAGIC = 020Bh

struct    IMAGE_NT_HEADERS64
        Signature                    rd      1
        FileHeader                   IMAGE_FILE_HEADER
        OptionalHeader               IMAGE_OPTIONAL_HEADER64
ends
IMAGE_NT_SIGNATURE = 'PE'

        lea    rdx, [file_buf]
        cmp    [rdx+IMAGE_DOS_HEADER.e_magic], IMAGE_DOS_SIGNATURE
        jnz    reconstruct_crc_L9
        mov    eax, [rdx+IMAGE_DOS_HEADER.e_lfanew]
        add    rax, rdx
        cmp    [rax+IMAGE_NT_HEADERS64.Signature], IMAGE_NT_SIGNATURE
        jnz    reconstruct_crc_L9
        cmp    [rax+IMAGE_NT_HEADERS64.FileHeader.Machine], IMAGE_FILE_MACHINE_AMD64
        jnz    reconstruct_crc_L9
        cmp
        [rax+IMAGE_NT_HEADERS64.OptionalHeader.Magic], IMAGE_NT_OPTIONAL_HDR64_MAGIC
        jnz    reconstruct_crc_L9

; CRC
; erase original file crc
        and    [rax+IMAGE_NT_HEADERS64.OptionalHeader.CheckSum], 0

```



```

; ecx = size
; rdx = file_buf

    mov     r10d, ecx
    shr     ecx, 1
    xor     r9d, r9d
    xor     r8d, r8d
calculate_checksum:
    mov     r9w, [rdx]
    add     r8d, r9d
    mov     r9w, r8w
    shr     r8d, 16
    add     r8d, r9d
    add     rdx, 2
    loop    calculate_checksum
    add     r8d, r10d
    mov     [rax+IMAGE_NT_HEADERS64.OptionalHeader.CheckSum], r8d
; Checksum done

reconstruct_crc_L9:
    ret

```

;//总体实现：相当于 main 函数

;//基本流程：read_system_file -> patch_bytes -> reconstruct_crc -> write_system_file

start:

```

    push    rbx
    sub     rsp, 8*(4+2)

    lea     rcx, [file0]
    call    read_system_file
    or      eax, eax
    mov     ebx, eax                                ; size of file
    jz      exit_failed
    mov     edx, ebx                                ; size of the whole file
    lea     rcx, [file0ed]                          ; pointer to patching data
    call    patch_bytes
    mov     edx, ebx
    lea     rcx, [file3ed]
    call    patch_bytes
    mov     edx, ebx
    lea     rcx, [file3ed]
    call    patch_bytes
    mov     edx, ebx
    lea     rcx, [file3ed]
    call    patch_bytes

```

```
    mov     ecx, ebx                                ; size of the whole file
    call    reconstruct_crc
    mov     edx, ebx                                ; size of the whole file
    lea     rcx, [file0n]
    call    write_system_file
    cmp     eax, ebx
    jnz     exit_delete_file0n
    lea     rcx, [file1]
    call    read_system_file
    or      eax, eax
    mov     ebx, eax
    jnz     L0
    lea     rcx, [file2]
    call    read_system_file
    or      eax, eax
    mov     ebx, eax
    jz      exit_failed
L0:    mov     edx, ebx
    lea     rcx, [file1ed]
    call    patch_bytes
    mov     edx, ebx
    lea     rcx, [file2ed]
    call    patch_bytes
    mov     edx, ebx
    lea     rcx, [file3ed]
    call    patch_bytes
    mov     edx, ebx
    lea     rcx, [file3ed]
    call    patch_bytes
    mov     edx, ebx
    lea     rcx, [file3ed]
    call    patch_bytes
    mov     ecx, ebx
    call    reconstruct_crc
    mov     edx, ebx
    lea     rcx, [file1n]
    call    write_system_file
    cmp     eax, ebx
    jz      exit_success
;      jmp     exit_delete_file1n_file0n

exit_delete_file1n_file0n:
    lea     rcx, [file1n]
    call    [DeleteFileA]
```

```
exit_delete_file0n:
    lea     rcx, [file0n]
    call    [DeleteFileA]

exit_failed:
    lea     rbx, [msg_failed]
    jmp     exit_msg

exit_success:
    lea     rbx, [msg_success]

exit_msg:
;STD_OUTPUT_HANDLE          = -11
;INVALID_HANDLE_VALUE       = -1
;
    mov     rcx, STD_OUTPUT_HANDLE
    push    STD_OUTPUT_HANDLE
    pop     rcx
    call    [GetStdHandle]
    push    rax
    pop     rcx
if INVALID_HANDLE_VALUE = -1
    inc     rax
else
    cmp     rax, INVALID_HANDLE_VALUE
end if
    jz      exit

    and     qword [rsp + 8*(4+0)], 0
    lea     r9, [rsp + 8*(4+1)]
    mov     r8d, msg_size
;    lea     rdx, [rbx]
    push    rbx
    pop     rdx
;    lea     rcx, [rcx] ; already set
    call    [WriteFile]

exit:
    xor     eax, eax
    add     rsp, 8*(4+2)
    pop     rbx
    ret
```

光有打过补丁的 winload.exe 和 ntoskrnl.exe 还不行，还需要建立新的 BCD 入口，把 PEAUTH 服务设为手动，防止在登陆时蓝屏：

```
@ECHO OFF

ECHO.
ECHO Creating patched copies of winload, ntkrnlmp/ntoskrnl...
ECHO.
patch.exe

ECHO.
ECHO Creating BCD Entry...
ECHO.
set ENTRY_GUID={46595952-454E-4F50-4747-554944FFFFFF}
bcdedit -create %ENTRY_GUID% -d "KPP & DSE Disabled" -application OSLOADER
bcdedit -set %ENTRY_GUID% device partition=%SYSTEMDRIVE%
bcdedit -set %ENTRY_GUID% osdevice partition=%SYSTEMDRIVE%
bcdedit -set %ENTRY_GUID% systemroot \Windows
bcdedit -set %ENTRY_GUID% path \Windows\system32\freeload.exe
bcdedit -set %ENTRY_GUID% kernel goodkrnl.exe
bcdedit -set %ENTRY_GUID% recoveryenabled 0
bcdedit -set %ENTRY_GUID% nx OptOut
bcdedit -set %ENTRY_GUID% nointegritychecks 1
bcdedit -set %ENTRY_GUID% testsigning 1
bcdedit -displayorder %ENTRY_GUID% -addlast
bcdedit -timeout 5
bcdedit -default %ENTRY_GUID%

ECHO.
ECHO Setting PEAUTH service to manual... (avoid BSOD at login screen)
ECHO.
sc config peauth start= demand

ECHO.
ECHO Complete!
ECHO.
PAUSE
shutdown /r /t 0
```

修改成功后重启，进入新建的内核启动项，就能进行任意内核 PATCH 而不用担心蓝屏了。如果此时运行 WIN64AST，会检查出内核项异常：

课后作业：尝试把汇编源代码改成 C 语言版本（附件里的“一键破解 PGDS”仅仅是把汇编写的 EXE 和 BAT 封装在一起而已，请大家用 VC 把汇编 EXE 的内容真正实现一次）。