A trip to nt!KiSystemCall64

Uma viagem ao submundo do Windows Internals

By Matheus Santos - _int0x80

Abstract: Este artigo explora o funcionamento interno do sistema operacional Windows, concentrando-se especificamente no manipulador de chamadas do sistema nt!KiSystemCall64. Explorando como chamadas do sistema são mecanismos cruciais que concedem ao sistema operacional solicitar serviços do kernel, permitindo operações que o próprio sistema operacional não pode executar diretamente, como a criação de arquivos no disco rígido. O artigo investiga a distinção entre Modo Usuário e Modo Kernel, ilustrando como os syscalls preenchem a lacuna de comunicação entre esses dois modos. Por meio de análise detalhada e desmontagem do manipulador KiSystemCall64, o artigo examina a transição do modo de usuário para o modo kernel, o papel da instrução swapgs e o uso de registros específicos de modelo (MSRs) no processo syscall. O estudo fornece insights sobre a estrutura KUSER_SHARED_DATA, destacando sua importância na redução das mudanças de contexto. As descobertas contribuem para uma compreensão mais profunda dos componentes internos do Windows, oferecendo um guia básico para iniciantes neste campo.

Keywords: Windows Internals, Syscalls, Windows Kernel

Introdução

Quando comecei minha jornada de estudos sobre o NT Internals uma das curiosidades que eu mais tinha era de saber como o sistema manipulava e lidava com as Syscalls(Chamadas de sistema). Hoje, irei abordar especificamente sobre o manipulador de chamada de sistemas, o **nt!KiSystemCall64**. Para aqueles que estão iniciando e aprendendo aos poucos não se preocupem, irei detalhar de forma breve e rápida sobre as chamadas de sistema para que todos possam acompanhar-me neste artigo.

O que são as chamadas de sistema?

As chamadas de sistema são um mecanismo fornecido pelo sistema operacional para a realização de algum serviço que o sistema operacional por si só não pode realizar. Por exemplo, criar um arquivo no disco rígido. Quando um usuário clica em “Criar um novo arquivo de texto” no Windows, por trás, o que acontece é uma syscall(Chamada de sistema) que o Sistema Operacional irá enviar para que o Kernel realize o serviço. De certa forma, isso existe para proteger a máquina de atividades que poderiam causar danos se o usuário viesse a operar manualmente. Pensando neste sentido, teremos que abordar sobre anéis de proteção do processador **Ring 0** ao **Ring 3**.

User-Mode Vs Kernel-Mode

Quando falamos em User-Mode e Kernel-Mode, estamos se referindo aos anéis de proteção do processador. Estes anéis existem como forma de controle de privilégio do que pode ser acessado e operado. O Sistema Operacional atua em Ring 3, que é um anel de proteção sem ou com muitos poucos privilégios. Portanto, é por este motivo que o sistema operacional não pode simplesmente criar um arquivo no disco rígido. O Kernel por sua vez, atua em Ring 0 que possui o maior privilégio dos anéis. Ele consegue interferir e interagir com outros hardwares da máquina diretamente. O mecanismo de Syscall existe exatamente para realizar essa comunicação entre Ring3(Sistema operacional) e Ring0(Kernel). Quando uma syscall é realizada, uma interrupção de software é gerada e o controle do programa passa a ser controlado pelo Kernel, que por sua vez realiza a tarefa solicitada e devolve o controle do programa para o Ring3(Sistema Operacional).

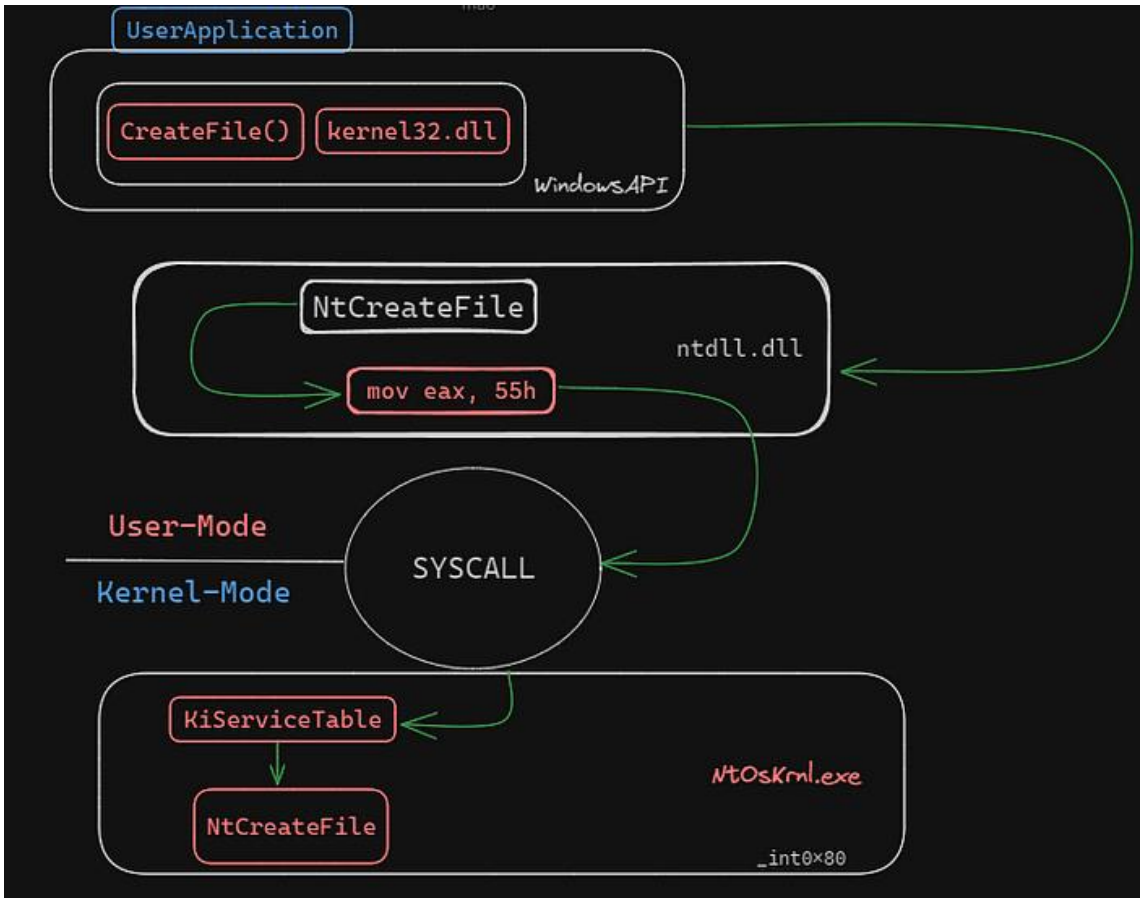

Figura 1 - NtCreateFile syscall

nt!KiSystemCall64 – Manipulador de syscall do Windows

Este é o manipulador de syscall do Windows, ele é responsável pela entrada do modo Kernel quando uma syscall é realizada.

Quando uma chamada de sistema é realizada, a instrução "syscall" é executada. Vamos olhar no WinDBG o stub da "**NtCreateFile**" para fins de entendimento.

```
lkd> u ntdll!NtCreateFile
ntdll!NtCreateFile:
00007ffe`7756db40 4c8bd1          mov     r10,rcx
00007ffe`7756db43 b855000000      mov     eax,55h
00007ffe`7756db48 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`7756db50 7503            jne     ntdll!NtCreateFile+0x15 (00007ffe`7756db55)
00007ffe`7756db52 0f05            syscall
00007ffe`7756db54 c3              ret
00007ffe`7756db55 cd2e            int     2Eh
00007ffe`7756db57 c3              ret
```

Figura 2 - Stub NtCreateFIle

Analisando a instrução SYSCALL, podemos ver que ela pega o valor salvo em **EAX (0x55h)** e salta para o manipulador de chamadas do sistema que é salvo no registro **MSR IA32_LSTAR** (do manual da Intel) que é apropriadamente denominado **nt!KiSystemCall64Shadow**.

Este é o manipulador **Kernel RIP Syscall** em modo longo (somente 64 bits). Quando as instruções SYSCALL são executadas, o código salta para a rotina do modo kernel cujo endereço é apontado por um **Model Specific Register (MSR)**. MSRs são registros especiais, específicos da CPU, que devem ser acessados por meio de instruções da CPU **rdmsr** (leitura) e **wrmsr** (gravação) por meio de um índice. Para x64, os três valores que procuramos são:

***IA32_STAR (0xC0000081)**: Segmentos Ring 0 e Ring 3 + SYSCALL EIP:*

00–31 = SYSCALL EIP

32–47 = base do segmento do kernel

48–63 = base do segmento do usuário.

IA32_LSTAR (0xC0000082): O RIP do kernel para SYSCALL em modo longo (software de 64 bits).

IA32_CSTAR (0xC0000083): O RIP do kernel para SYSCALL em modo de compatibilidade.

IA32_SFMASK (0xC0000084): Os 32 bits inferiores são a máscara do sinalizador SYSCALL. Se um bit for definido, o bit correspondente em EFLAGS será apagado.

O Windows possui duas instruções que podem ser utilizadas para executar a syscall. São elas a instrução "syscall" representada pelo opcode "**0f05**" e a interrupção "**int 2Eh**" que possui o opcode "**cd2e**". Ambas são utilizadas para causar a interrupção e alcançar o manipulador de syscall do Windows o "**nt!KiSystemCall64**". Quando uma dessas instruções são executadas um teste é executado pelo próprio stub da syscall, como podemos ver na imagem abaixo:

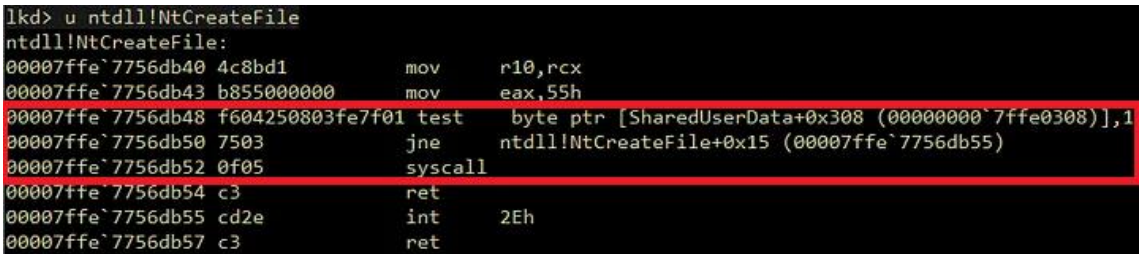

Figura 3 - Descobrimento da arquitetura do S.O

O sistema acessa a estrutura **KUSER_SHARED_DATA** que possui um endereço fixo mapeado para os processos no user-mode, sendo ele "**0xfffff78000000000**". Durante a instrução **test**, um deslocamento com offset de **0x308** permite que alcance o membro "SystemCall" da estrutura e testa se o valor é 1, se for o sistema é baseado em x64. Ou seja, ele executa a instrução "syscall". Caso não, ele pula para o endereço "**00007ffe7756db55**" que possui a instrução "**int 2Eh**". Ou seja, em sistemas baseado em **x86**, a instrução para gerar a interrupção é a "**int 2Eh**". Na arquitetura **x64**, a instrução é "syscall".

De acordo com a própria documentação da Microsoft:

`SystemCall`

No AMD64, esse valor será inicializado para um valor diferente de zero se o sistema operar com uma exibição alterada do mecanismo de chamada de serviço do sistema.

Quando a instrução de syscall é executada, o valor de **IA32_LSTAR** é recuperado e colocado no registrador **RIP(x64)** do Kernel. Chamamos isso de **Kernel RIP Syscall.** Após isso o modo Kernel passa a executar o manipulador de syscall o **nt!KiSystemCall64.**

Para vias de complemento:

A estrutura **KUSER_SHARED_DATA** é uma estrutura utilizada para que o Kernel compartilhe informações com os processos do lado do user-mode, evitando assim uma troca de contexto de user-mode para kernel-mode constante. Você pode procurar ver mais detalhes sobre essa estrutura na documentação da Microsoft em KUSER_SHARED_DATA.

Desmontagem do manipulador de syscall:

Desmontando o manipulador **KiSystemCall64**, iremos ver que a primeira instrução a ser executada é "**swapgs**".

Figura 4 - Swapgs Instruction

Na verdade, existem dois manipuladores SysCall diferentes, com e sem a palavra-chave 'Shadow'. A "Sombra" vem do recurso "**Kernel Virtual Address Shadow**" que visa corrigir o bug Meltdown .

A `swapgs` instrução privilegiada é usada para alterar/trocar o valor atual do registrador base **GS** pelo valor residente no endereço **MSR C0000102H** (**IA32_KERNEL_GS_BASE**). Para ser mais claro, o valor do registro base **GS** é igual ao valor contido no **MSR IA32_GS_BASE**. Em sistemas **Windows x64**, os valores são:

- **IA32_KERNEL_GS_BASE**—Ponteiro para a região de controle do processador (**PCR**) atual, especificamente a região de controle do processador do kernel (**KPCR**)
- **IA32_GS_BASE**—Ponteiro para thread de execução atual **TEB**

Assim, no modo longo 64, o segmento **GS** sempre aponta para o thread atual **TEB**, no modo usuário, enquanto no modo kernel aponta para o **PCR** do processador atual.

A próxima instrução `mov qword ptr gs:[10h],rsp` salva o ponteiro da pilha do user-mode.

Conclusão:

Este artigo detalhou o funcionamento interno do manipulador de syscall nt!KiSystemCall64, oferecendo uma visão abrangente sobre a transição de modos e o papel de instruções específicas. Espero que este artigo tenha contribuido de alguma forma para quem está iniciando os estudos do Windows Internals. Qualquer dúvida ou sugestões podem me escrever um email: int0x80.c@gmail.com. Desde já, agradeço a todos que chegaram até aqui.