Ghost in the BLF

A two-year journey of chasing in-the-wild LPE exploits in Windows CLFS

Quan Jin & Yingqi Shi & Guoxian Zhong

DBAPPSecurity WeBin Lab

About us

Quan Jin
@jq0904

Yingqi Shi
@Mas0nShi

Guoxian Zhong
@_p01arisZ

安恒信息
DAS-security 安全中国

安恒研究院
DAS-RESEARCH

卫兵实验室
WEBIN.LAB

About this talk

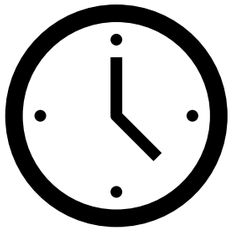
- Our journey of chasing in-the-wild LPE exploits in Windows CLFS
- Show a Pandora's box of Windows CLFS exploits
- Quote and compare some works already done by other vendors
- Show some samples that other vendors had missed
- Provide some insights on in-the-wild Windows LPE 0days trends

Previous CLFS researches

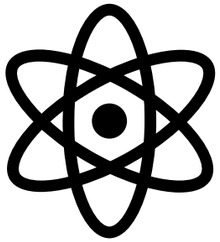
- DeathNote of Microsoft Windows Kernel - Keen Lab
- CLFS Internals - Alex Ionescu
- Attacking the Common Log File System – 360
- Windows CLFS and Five Exploits Used By Ransomware Operators - Kaspersky

Help in understanding this talk

Agenda



Start



Hunt

Analysis

Variant

Summary

Start

The 1st CLFS Exploit (1day)

- **Where did it come from**
  - The exploit was from **VirusTotal**

- **How we caught it**
  - It hit a rule we wrote for the an exploit technique (**Pipe Attribute**)

```
v25 = output;
v27 = pfnNtFsControlFile(a1, 0i64, 0i64, 0i64, status, 0x11003Ci64, data, size, output, cb_output)
```

- **Basic information**
  - Unknown CVE
  - **Compiled in September 2021, Caught in October 2021, “Patched” in September 2021**

https://i.blackhat.com/USA-22/Thursday/us-22-Jin-The-Journey-Of-Hunting-ITW-Windows-LPE-0day.pdf

pContainer

```
typedef struct _CLFS_CONTAINER_CONTEXT
{
    CLFS_NODE_ID cidNode;
    ULONGLONG cbContainer;
    CLFS_CONTAINER_ID cidContainer;
    CLFS_CONTAINER_ID cidQueue;
    union
    {
        CClfsContainer* pContainer;
        ULONGLONG ullAlignment;
    };
    CLFS_USN usnCurrent;
    CLFS_CONTAINER_STATE eState;
    ULONG cbPrevOffset;
    ULONG cbNextOffset;
} CLFS_CONTAINER_CONTEXT, *PCLFS_CONTAINER_CONTEXT;
```

NodeType = 0xC1FDF008
NodeSize = 0x30

Kernel pointer to CClfsContainer (!)

```
; const CClfsContainer::`vftable'
??_7CClfsContainer@@6B@ dq offset ?AddRef@CClfsContainer@@UEAAKXZ
                                  ; DATA XREF: CClfsBaseFilePer
                                  ; CClfsContainer::CClfsContai
                                  ; CClfsContainer::AddRef(void
                dq offset ?Release@CClfsContainer@@UEAAKXZ ; CClfsCon
                dq offset ?GetSListEntry@CClfsContainer@@UEAAPEAU_SLI
                dq offset ?Remove@CClfsContainer@@UEAAJXZ ; CClfsCont
```

https://securelist.com/windows-clfs-exploits-ransomware/111560/

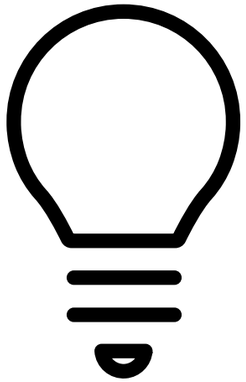
How it manipulates the pContainer

```
__int64 __fastcall CClfsLogFcbPhysical::FlushMetadata(CClfsLogFcbPhysical *pClfsLogFcbPhysical)
{
  // Omit ...

  v2 = CClfsBaseFile::AcquireClientContext(pClfsLogFcbPhysical->CClfsBaseFile, 0, &pCtxClient_);
  if ( v2 >= 0 && (pCtxClient = pCtxClient_) != 0i64 )
  {
    eState = pCtxClient_->eState;
    Flag = pClfsLogFcbPhysical->Flag;
    pCtxClient_->llCreateTime = pClfsLogFcbPhysical->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbPhysical->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbPhysical->ClientllWriteTime;
    pCtxClient->lsnOwnerPage = pClfsLogFcbPhysical->ClientlsnOwnerPage;
    pCtxClient->lsnArchiveTail = pClfsLogFcbPhysical->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbPhysical->ClientlsnBase.ullOffset;
    pCtxClient->lsnLast = pClfsLogFcbPhysical->ClientlsnLast.ullOffset;
    pCtxClient->lsnRestart = pClfsLogFcbPhysical->ClientlsnRestart.ullOffset;
    pCtxClient->cShadowSectors = pClfsLogFcbPhysical->ClientShadowSectors;
    pCtxClient->fAttributes = pClfsLogFcbPhysical->fAttributes;
    _eState = eState | 8; // 1-bit flip
    if ( (Flag & 0x10) == 0 )
        _eState = eState;
    pCtxClient->eState = _eState; // The 1st CLFS exploit write fake pContainer here
   // Omit ...
}
```

CLFS in April 2022

- **One CLFS CVE**
  - CVE-2022-24481
  - A “varaint” of “The 1st CLFS Exploit”, reported by us 
- **One in-the-wild CLFS CVE**
  - CVE-2022-24521
  - Reported by NSA and CrowedStrike
- **One slide**
  - Attacking the Common Log File System, BlackHat Asia 2022

What we think at April 2022



More exploits in CLFS may appear in the future

Hunt

How to catch them

- **Rethink: How we caught the 1st CLFS exploit**
  - It hit a rule we wrote for the **Pipe Attribute** exploit method, so the second may hit the rule again
- **One begets two, two begets three, three begets all things** 
  - Carefully research the first CLFS exploit
  - Research on multiple dimensions: **Code Similarity**, **PDB Path**, **Exploit Techniques**, **Strings**, ect.
- **Stay hungry, stay foolish**
  - Collect the samples disclosed by others, and carefully research them
  - Study any clues that can be found: every blog, every slide, every ioc, ect.

The 2nd CLFS Exploit (1day)

- **How we caught it**
  - It hit the rule we wrote for the **Pipe Attribute** exploit method (**again**)

- **Basic information**
  - Unknown CVE
  - **Compiled in April 2022, Caught in May 2022, Patched in April 2022**

- **The interesting part**
  - The author believed he was writing an exploit for CVE-2022-24521, but he was not
    - C:\Users\123\source\repos**cve_2022_24521** (1)**CVE_2022_24521**\x64\Release**CVE_2022_24521**_clfs.pdb

How it manipulates the pContainer

```
__int64 __fastcall CClfsLogFcbPhysical::FlushMetadata(CClfsLogFcbPhysical *pClfsLogFcbPhysical)
{
  // Omit ...

  v2 = CClfsBaseFile::AcquireClientContext(pClfsLogFcbPhysical->CClfsBaseFile, 0, &pCtxClient_);
  if ( v2 >= 0 && (pCtxClient = pCtxClient_) != 0i64 )
  {
    eState = pCtxClient_->eState;
    Flag = pClfsLogFcbPhysical->Flag;
    // The 2nd CLFS exploit write fake pContainer here, a.k.a Kaspersky Exploit #1
    pCtxClient_->llCreateTime = pClfsLogFcbPhysical->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbPhysical->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbPhysical->ClientllWriteTime;
    pCtxClient->lsnOwnerPage = pClfsLogFcbPhysical->ClientlsnOwnerPage;
    pCtxClient->lsnArchiveTail = pClfsLogFcbPhysical->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbPhysical->ClientlsnBase.ullOffset;
    pCtxClient->lsnLast = pClfsLogFcbPhysical->ClientlsnLast.ullOffset;
    pCtxClient->lsnRestart = pClfsLogFcbPhysical->ClientlsnRestart.ullOffset;
    pCtxClient->cShadowSectors = pClfsLogFcbPhysical->ClientShadowSectors;
    pCtxClient->fAttributes = pClfsLogFcbPhysical->fAttributes;
    _eState = eState | 8;

    // Omit ...
}
```

The 3rd CLFS Exploit (0day)

- **How we caught it**
  - It hit the rule we wrote for the **Pipe Attribute** exploit method (**again**)

- **Basic information**
  - **CVE-2022-24521**<- This is an in-the-wild 0day caught by NSA and CrowedStrike
  - **Compiled in October 2021, Caught in August 2022(too late), Patched in April 2022**
  - Lack of proper validation on **SignaturesOffset** of _CLFS_LOG_BLOCK_HEADER

- **PDB Path**
  - C:\Users\User\Desktop\2\x64\Release\exploit.pdb

How it manipulates the pContainer

- CClfsBaseFilePersisted::WriteMetadataBlock

```
// The 3rd exploit write fake pContainer here
ClfsEncodeBlock(RecoderHeader, *(RecoderHeader + 4) << 9, *(RecoderHeader + 2), 0x10u, 1u);

v10 = CClfsContainer::WriteSector(*(this + 0x13), *(this + 0x14), 0i64, *(*(this + 6) + 24 * v8),
    *(RecoderHeader + 4), &v23);

if ( v7 )
{
    ClfsDecodeBlock(RecoderHeader, *(RecoderHeader + 4), *(RecoderHeader + 2), 0x10u, &v21);
    v17 = (this + 0x1C0);
    do
    {
        if ( *v17 && CClfsBaseFile::AcquireContainerContext(this, v6, &containerContext) >= 0 )
        {
            containerContext->pContainer = *v17; // won't enter here
            CClfsBaseFile::ReleaseContainerContext(this, &containerContext);
        }
        ++v6;
        ++v17;
    }
    while ( v6 < 0x400 );
}
```

https://www.freebuf.com/articles/network/339537.html

The 4th CLFS Exploit (0day)

- **How we caught it**
  - It hit several rules we wrote, include the **Pipe Attribute** rule (**again**)

- **Basic information**
  - **CVE-2022-37969** <- This is an in-the-wild 0day caught by us
  - **Compiled in August 2022, Caught in August 2022, Patched in September 2022**

- **New functions for arbitrary write**
  - clfs!CClfsContainer::Release -> **nt!SeSetAccessStateGenericMapping**
  - clfs!CClfsContainer::Remove -> **clfs!ClfsEarlierLsn**

https://www.zscaler.com/blogs/security-research/technical-analysis-windows-clfs-zero-day-vulnerability-cve-2022-37969-part2-exploit-analysis

The 4th CLFS Exploit (0day)

```
__int64 __fastcall CClfsBaseFilePersisted::AllocSymbol(
        CClfsBaseFilePersisted *this,
        unsigned int a2,
        void **pNewSymbol)
{
  // Omit ...
  Size = a2;
  BaseLogRecord = CClfsBaseFile::GetBaseLogRecord(this);
  BaseLogRecord_ = BaseLogRecord;
  if ( !BaseLogRecord )
    return 0xC01A000Di64;
  pClfsLogBlockHeader = *(*(v6 + 0x30) + 0x30i64);
  *pNewSymbol = 0i64;
  cbSymbolZone = BaseLogRecord->rgContainers[0x3FC];
  if ( &BaseLogRecord->rgContainers[0x400] + cbSymbolZone + Size > (&pClfsLogBlockHeader->MajorVersion
                                                                   + pClfsLogBlockHeader->SignaturesOffset) )
    return 0xC0000023i64;
  NewSymbol = &BaseLogRecord->rgContainers[0x400] + cbSymbolZone;
  memset(NewSymbol, 0, Size); // The 4th exploit set fake pContainer here
  BaseLogRecord_->rgContainers[0x3FC] += Size;
  result = 0i64;
  *pNewSymbol = NewSymbol;
  return result;
}
```

The 5th CLFS Exploit (1day)

- **How we caught it**
  - It hit several rules we wrote, include the **Pipe Attribute** rule (**again**)
- **Basic information**
  - Unknown CVE
  - **Compiled in September 2022, Caught in October 2022, Patched in September 2022**
- **New exploit technique**
  - Hijack calls in **CClfsBaseFilePersisted::CheckSecureAccess**
  - **clfs!ClfsMgmtDeregisterManagedClient, nt!PoFxProcessorNotification, nt!RtlClearBit**

https://securelist.com/windows-clfs-exploits-ransomware-september-2022/111584/

How it manipulates the pContainer

```
__int64 __fastcall CClfsLogFcbPhysical::FlushMetadata(CClfsLogFcbPhysical *pClfsLogFcbPhysical)
{
  // Omit ...

  v2 = CClfsBaseFile::AcquireClientContext(pClfsLogFcbPhysical->CClfsBaseFile, 0, &pCtxClient_);
  if ( v2 >= 0 && (pCtxClient = pCtxClient_) != 0i64 )
  {
    eState = pCtxClient_->eState;
    Flag = pClfsLogFcbPhysical->Flag;
    pCtxClient_->llCreateTime = pClfsLogFcbPhysical->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbPhysical->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbPhysical->ClientllWriteTime;
    pCtxClient->lsnOwnerPage = pClfsLogFcbPhysical->ClientlsnOwnerPage;
    pCtxClient->lsnArchiveTail = pClfsLogFcbPhysical->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbPhysical->ClientlsnBase.ullOffset;
    pCtxClient->lsnLast = pClfsLogFcbPhysical->ClientlsnLast.ullOffset;
    pCtxClient->lsnRestart = pClfsLogFcbPhysical->ClientlsnRestart.ullOffset;
    pCtxClient->cShadowSectors = pClfsLogFcbPhysical->ClientShadowSectors;
    // The 5th CLFS exploit write fake pContainer here, a.k.a Kaspersky Exploit #2
    pCtxClient->fAttributes = pClfsLogFcbPhysical->fAttributes;
    _eState = eState | 8;

    // Omit ...
}
```

The 6th CLFS Exploit (1day)

- **How we caught it**
  - It hit several rules we wrote, include the **Pipe Attribute** rule (**again**)
- **Basic information**
  - Unknown CVE
  - **Compiled in October 2022, Caught in November 2022, Patched in October 2022**
- **Exploit technique**
  - Same with the 5th CLFS exploit
  - Kaspersky had talked about this case in their blogs (Exploit #3)
    - https://securelist.com/windows-clfs-exploits-ransomware-october-2022/111591/

How it manipulates the pContainer

```
__int64 __fastcall CClfsLogFcbPhysical::FlushMetadata(CClfsLogFcbPhysical *pClfsLogFcbPhysical)
{
  // Omit ...

  v2 = CClfsBaseFile::AcquireClientContext(pClfsLogFcbPhysical->CClfsBaseFile, 0, &pCtxClient_);
  if ( v2 >= 0 && (pCtxClient = pCtxClient_) != 0i64 )
  {
    eState = pCtxClient_->eState;
    Flag = pClfsLogFcbPhysical->Flag;
    pCtxClient_->llCreateTime = pClfsLogFcbPhysical->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbPhysical->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbPhysical->ClientllWriteTime;
    pCtxClient->lsnOwnerPage = pClfsLogFcbPhysical->ClientlsnOwnerPage;
    pCtxClient->lsnArchiveTail = pClfsLogFcbPhysical->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbPhysical->ClientlsnBase.ullOffset;
    pCtxClient->lsnLast = pClfsLogFcbPhysical->ClientlsnLast.ullOffset;
    // The 6th CLFS exploit write fake pContainer here, a.k.a Kaspersky Exploit #3
    pCtxClient->lsnRestart = pClfsLogFcbPhysical->ClientlsnRestart.ullOffset;
    pCtxClient->cShadowSectors = pClfsLogFcbPhysical->ClientShadowSectors;
    pCtxClient->fAttributes = pClfsLogFcbPhysical->fAttributes;
    _eState = eState | 8;

    // Omit ...
}
```

Two different exploit techniques

- **Developer (A)**
  - Hijack calls in **CClfsBaseFilePersisted::RemoveContainer**
  - Using **nt!XmXchgOp** and **nt!HalpDmaPowerCriticalTransitionCallback** to achieve primitive
  - Example: the **1st** CLFS exploit, (part of) the **3rd** CLFS exploits

- **Developer (B)**
  - Hijack calls in **CClfsLogFcbPhysical::CloseContainers**
  - Using **clfs!ClfsSetEndOfLog** or **nt!SeSetAccessStateGenericMapping** to achieve write primitive
  - Using the exploit template of CVE-2021-26868
  - Example: the **2nd** CLFS exploit, the **4th** CLFS exploit, the **5th** CLFS exploit, the **6th** CLFS exploit, the **10th** CLFS exploit (later)

The 7th CLFS Exploit (0day)

- **How we caught it**
  - We connected to this exploit after we hunt CVE-2023-28252 itw 0day

- **Basic information**
  - CVE-2023-23376 <- This is an in-the-wild 0day caught by Microsoft
  - **The core exploit was compiled in January 2023, the sample was compiled in February 2023**
  - **Caught by us in March 2023(too late), Patched in February 2023**

- **Exploit technique**
  - Kaspersky had talked about this case in their blogs (Exploit #4)
    - https://securelist.com/windows-clfs-exploits-ransomware-cve-2023-23376/111593/

The 8th CLFS Exploit (0day)

- **How we caught it**
  - It hit a simple rule we wrote for common CLFS exploit strings
- **Basic information**
  - CVE-2023-28252 <- This is an in-the-wild 0day caught by us
  - **Compiled in March 2023, Caught in March 2023, Patched in April 2023**
- **Exploit technique**
  - Kaspersky had talked about this case in their blogs (Exploit #5)
    - https://securelist.com/windows-clfs-exploits-ransomware-cve-2023-28252/111601/

Analysis

When we analyzed the 1st exploit

- **Completely unfamiliar with Windows CLFS**
  - At that time(**October 2022**), we hadn't come across CLFS vulnerabilities

- **Very few reference materials**
  - DeathNote of Microsoft Windows Kernel, KeenLab, 2016
  - CLFS Internals, Alex Ionescu, 2021

- **Lots of reverse engineering and debugging**
  - BLF file format is very complex when you don’t have an 010 Editor template
  - But if you have, that’s another story

010 Editor Template for BLF

Name	Value	Start	Size	Color
struct CLFS_LOG_BLOCK_HEADER Header		0h	70h	Fg: Bg:
struct CLFS_CONTROL_RECORD Record		70h	E0h	Fg: Bg:
struct ControlBlock		0h	150h	Fg: Bg:
struct ControlBlockShadow		400h	70h	Fg: Bg:
struct BaseBlock		800h	13A5h	Fg: Bg:
struct CLFS_LOG_BLOCK_HEADER Header		800h	70h	Fg: Bg:
struct RecordWrap		870h	1335h	Fg: Bg:
struct CLFS_BASE_RECORD_HEADER BaseRecordHeader		870h	1335h	Fg: Bg:
struct CLFS_METADATA_RECORD_HEADER RecordHeader		870h	8h	Fg: Bg:
GUID IdLog[16]	{906F30CB-1D35-11ED-B...	878h	10h	Fg: Bg:
ULONGLONG ClientSymbolTable[11]		888h	58h	Fg: Bg:
ULONGLONG ContainerSymbolTable[11]		8E0h	58h	Fg: Bg:
ULONGLONG SecuritySymbolTable[11]		938h	58h	Fg: Bg:
DWORD NextContainer	0h	990h	4h	Fg: Bg:
DWORD NextClient	1h	994h	4h	Fg: Bg:
DWORD FreeContainers	0h	998h	4h	Fg: Bg:
DWORD ActiveContainers	0h	99Ch	4h	Fg: Bg:
DWORD FreeContainersCount	0h	9A0h	4h	Fg: Bg:
DWORD BusyContainers	0h	9A4h	4h	Fg: Bg:
DWORD ClientArray[124]		9A8h	1F0h	Fg: Bg:
DWORD ContainerArray[1024]		B98h	1000h	Fg: Bg:
DWORD SymbolZone	1114Bh	1B98h	4h	Fg: Bg:
DWORD Sector	0h	1B9Ch	4h	Fg: Bg:
USHORT Unused	0h	1BA0h	2h	Fg: Bg:
enum CLFS_LOG_STATE LogState	3h	1BA2h	1h	Fg: Bg:
UCHAR Usn	1h	1BA3h	1h	Fg: Bg:
UCHAR Clients	1h	1BA4h	1h	Fg: Bg:

IDA C header file for CLFS

```
__int64 __fastcall CClfsLogFcbVirtual::FlushMetadata(CClfsLogFcbVirtual *pClfsLogFcbVirtual, __int64 a2, __int64 a3)
{
  __int64 *v4; // rax
  __int64 v5; // rdx
  __int64 v6; // rbx
  _CLFS_CLIENT_CONTEXT *pCtxClient; // [rsp+50h] [rbp+10h] BYREF
  char v9; // [rsp+58h] [rbp+18h] BYREF

  pCtxClient = 0i64;
  LOBYTE(a3) = pClfsLogFcbVirtual->m_ClientId;
  v4 = (pClfsLogFcbVirtual->m_PhysicalFcb->GetClientBaseLsn)(pClfsLogFcbVirtual->m_PhysicalFcb, &v9, a3);
  LOBYTE(v5) = pClfsLogFcbVirtual->m_ClientId;
  v6 = *v4;
  (pClfsLogFcbVirtual->m_PhysicalFcb->AcquireClientContext)(pClfsLogFcbVirtual->m_PhysicalFcb, v5, &pCtxClient);
  if ( pCtxClient )
  {
    pCtxClient->llCreateTime = pClfsLogFcbVirtual->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbVirtual->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbVirtual->ClientllWriteTime;
    pCtxClient->lsnArchiveTail = pClfsLogFcbVirtual->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbVirtual->ClientlsnBase.ullOffset;
    pCtxClient->lsnLast = pClfsLogFcbVirtual->ClientlsnLast.ullOffset;
    pCtxClient->lsnRestart = pClfsLogFcbVirtual->ClientlsnRestart.ullOffset;
    pCtxClient->lsnPhysicalBase = v6;
    (pClfsLogFcbVirtual->m_PhysicalFcb->ReleaseClientContext)(pClfsLogFcbVirtual->m_PhysicalFcb, &pCtxClient);
    return (pClfsLogFcbVirtual->m_PhysicalFcb->FlushMetadata)(pClfsLogFcbVirtual->m_PhysicalFcb);
  }
  else
  {
    if ( *&WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (*(*&WPP_GLOBAL_Control + 44i64) & 0x8000000) != 0 )
      WPP_SF_sdid(*(*&WPP_GLOBAL_Control + 24i64));
    return 0xC01A000Di64;
  }
}
```

Identify different CLFS Exploits

- **Identify whether a sample is 0day, 1day or nday**
  - The simplest method is to create multiple reproduction environments(0day/1day/nday)

- **Identify the root cause of a vulnerability**
  - Reverse engineering
  - Debugging
  - Use 010 Editor template to parse malicious BLF log file (to find abnormal values)

- **Classify an exploit under a specific CVE**
  - Based on the analysis, categorize the new exploit into historical database

Some interesting exploits

- **Samples with packing and anti-debugging**
  - **CryptOne** -> CVE-2023-23376
  - **Themida** -> CVE-2023-28252

- **Samples with code obfuscation**
  - `6132342df727e2e9a4ac1310f6c8d6c5280bb723b3eb3f193555698a54a3be82` <- CVE-2023-28252

- **Sample that doesn't work properly (unable to execute or BSOD)**
  - `8ed4c3977a4a56308afbf114299cc34ff9a58c9c51558cc3bb3316392ed1235b` <- BSOD, 28252
  - `C236c957815d70d58af6320b02d4fc353f87459bbb2a1e6f1bbd77e5ce8cacac` <- unable to execute, 28252

The 9th CLFS Exploit (nday)

- **How we caught it**
  - It hit a simple rule we wrote for common CLFS exploit strings (**again**)

- **Basic information**
  - Unknown CVE
  - **Compiled in November 2023, Caught in November 2023, Patched in January 2022**

- **This exploit is distinguished from developer (A) and developer (B)**
  - Winlogon.exe will crash after executing the exploit <- Injected low-quality shellcode
  - The exploit is not compatible with Windows 11

How it manipulates the pContainer

```
__int64 __fastcall CClfsLogFcbPhysical::FlushMetadata(CClfsLogFcbPhysical *pClfsLogFcbPhysical)
{
  // Omit ...

  v2 = CClfsBaseFile::AcquireClientContext(pClfsLogFcbPhysical->CClfsBaseFile, 0, &pCtxClient_);
  if ( v2 >= 0 && (pCtxClient = pCtxClient_) != 0i64 )
  {
    eState = pCtxClient_->eState;
    Flag = pClfsLogFcbPhysical->Flag;
    // The 9th CLFS exploit write fake pContainer here
    pCtxClient_->llCreateTime = pClfsLogFcbPhysical->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbPhysical->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbPhysical->ClientllWriteTime;
    pCtxClient->lsnOwnerPage = pClfsLogFcbPhysical->ClientlsnOwnerPage;
    pCtxClient->lsnArchiveTail = pClfsLogFcbPhysical->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbPhysical->ClientlsnBase.ullOffset;
    pCtxClient->lsnLast = pClfsLogFcbPhysical->ClientlsnLast.ullOffset;
    pCtxClient->lsnRestart = pClfsLogFcbPhysical->ClientlsnRestart.ullOffset;
    pCtxClient->cShadowSectors = pClfsLogFcbPhysical->ClientShadowSectors;
    pCtxClient->fAttributes = pClfsLogFcbPhysical->fAttributes;
    _eState = eState | 8;
    // Omit ...
}
```

The 10th CLFS Exploit (1day)

- **How we caught it**
  - It hit the rule we wrote for the **Pipe Attribute** exploit method (**again**)

- **Basic information**
  - Unknown CVE
  - Code obfuscation
  - **Compiled in November 2022, Caught in November 2023, Patched in September 2022**

- **Exploit technique**
  - **Same with the 2nd CLFS exploit**, it should be the same author

Look for the root cause

- **Q: How to quickly locate**
  - **Reverse thinking**
  - Set hardware breakpoint on the allocate memory area (e.g. **VirtualAlloc**)

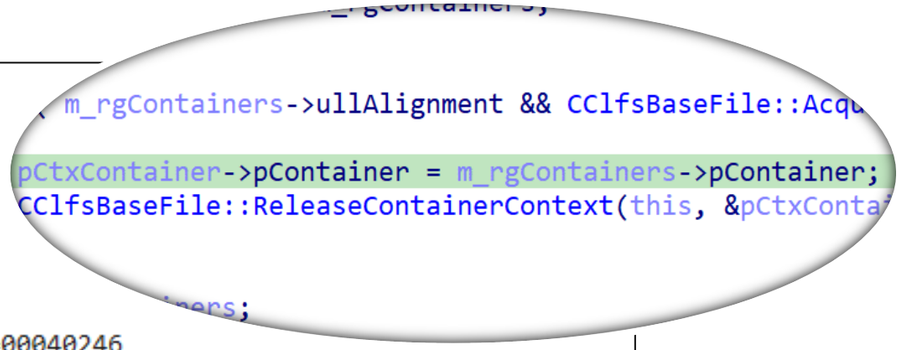
```
.text:00000000004B62A2                  mov     r9d, 4          ; flProtect
.text:00000000004B62A8                  mov     r8d, 3000h      ; flAllocationType
.text:00000000004B62AE                  mov     edx, 1000000h   ; dwSize
.text:00000000004B62B3                  mov     ecx, 40000000h  ; lpAddress
.text:00000000004B62B8                  call    cs:VirtualAlloc
```

```
0: kd>  ba r8 0x40000000
0: kd> g
Breakpoint 0 hit
0033:00007ff7`158252c0 c78424e807000000000000 mov dword ptr [rsp+7E8h],0
1: kd> k
 # Child-SP          RetAddr               Call Site
00 0000005a`e78d4960 000001eb`5c46cc80     0x00007ff7`158252c0
01 0000005a`e78d4968 00007ffc`1c404290     0x000001eb`5c46cc80
02 0000005a`e78d4970 00000000`00000000     0x00007ffc`1c404290
1: kd> g
Breakpoint 0 hit
CLFS!CClfsLogFcbPhysical::CloseContainers+0x70:
fffff802`46b49870 488b4008        mov     rax,qword ptr [rax+8]
```

Look for the root cause

- **Where the pContainer changed**
  - CClfsBaseFilePersisted::**WriteMetadataBlock**

```
Breakpoint 1 hit
rax=ffff80071a57a414 rbx=0000000000000001 rcx=0000000040000000
rdx=0000000000000000 rsi=ffffb68f0f61d1c8 rdi=ffffb68f0f61d000
rip=fffff8052d96c071 rsp=ffffde8052daf410 rbp=ffffde8052daf668
 r8=ffffde8052daf340  r9=0000000000000001 r10=0000000000000000
r11=ffffde8052daf300 r12=ffffb68f13060101 r13=0000000000000002
r14=ffff80071a579000 r15=0000000000000000
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00040246
CLFS!CClfsBaseFilePersisted::WriteMetadataBlock+0x201:
fffff805`2d96c071 48894818        mov     qword ptr [rax+18h],rcx ds:002b:ffff8007`1a57a42c=0000000000000000
1: kd> k
 # Child-SP          RetAddr               Call Site
00 ffffde80`52daf410 fffff805`2d96b9e0     CLFS!CClfsBaseFilePersisted::WriteMetadataBlock+0x201
01 ffffde80`52daf4a0 fffff805`2d9434ef     CLFS!CClfsBaseFilePersisted::FlushImage+0x40
02 ffffde80`52daf4e0 fffff805`2d97b1e0     CLFS!CClfsLogFcbPhysical::FlushMetadata+0xef
03 ffffde80`52daf530 fffff805`2d995bfa     CLFS!CClfsLogFcbPhysical::AllocContainer+0x180
04 ffffde80`52daf5d0 fffff805`2d99096c     CLFS!CClfsLogFcbVirtual::AllocContainer+0x5a
05 ffffde80`52daf610 fffff805`2d96e445     CLFS!CClfsRequest::AllocContainer+0x22c
```



Look for the root cause

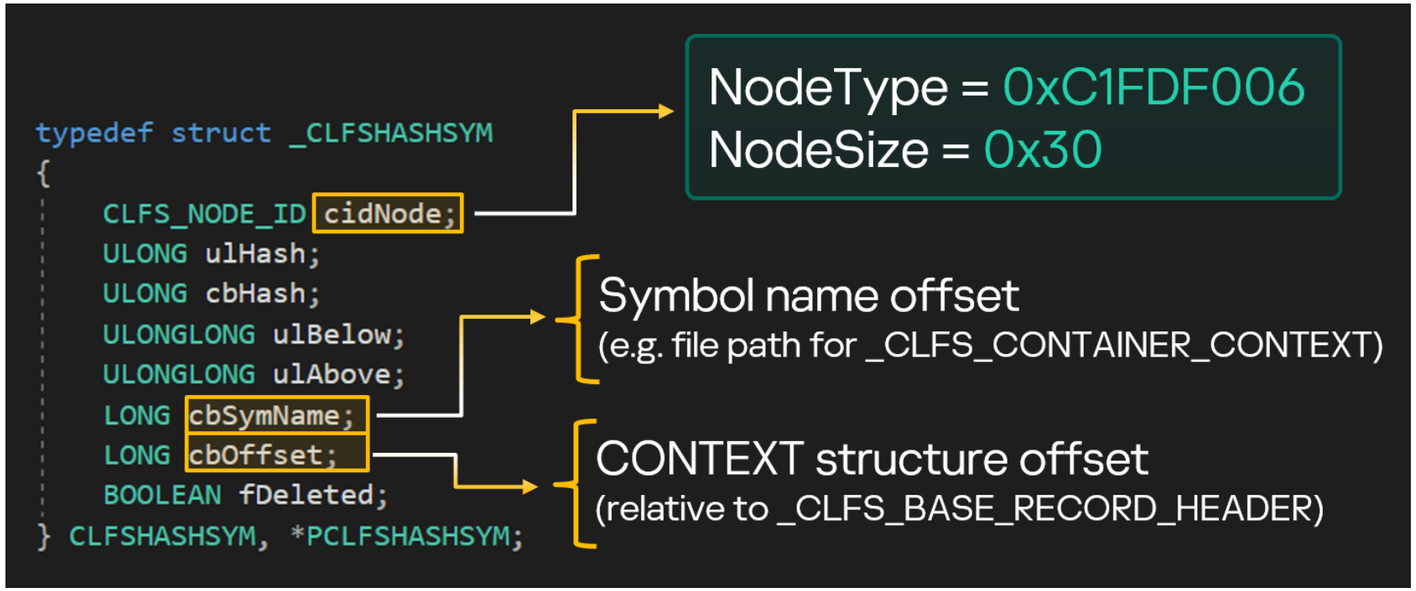
```
NTSTATUS CClfsBaseFilePersisted::WriteMetadataBlock(CClfsBaseFilePersisted *this, uint a2, bool a3)
{
    // Omit ...
    for (unsigned int i = 0; i < 0x400; ++i)
    {
        pCtxContainer = nullptr;
        rc = CClfsBaseFile::AcquireContainerContext(this, i, &pCtxContainer);
        if (rc >= 0) {
            pCtxContainer->pContainer = nullptr;
            this->m_rgContainers[i] = pCtxContainer->pContainer; // Backup to class instance
            CClfsBaseFile::ReleaseContainerContext(this, &pCtxContainer);
        } else { /* Omit */ }
    }
    // Omit ...
    for (unsigned int i = 0; i < 0x400; ++i)
    {
      if (CClfsBaseFile::AcquireContainerContext(this, i, &pCtxContainer) >= 0 )
      {
        pCtxContainer->pContainer = this->m_rgContainers[i]; // Restore to File Common Block
        CClfsBaseFile::ReleaseContainerContext(this, &pCtxContainer);
      }
    }
}
```

Look for the root cause

```
NTSTATUS CClfsBaseFile::AcquireContainerContext(CClfsBaseFile *this, UINT cid, PCLFS_CONTAINER_CONTEXT
*ppCtxContainer)
{
    NTSTATUS rc;
    // Omit ...
    if (precHdr)
    {
        UINT v12 = precHdr->rgContainers[cid];

        if (v12)
        {
            rc = CClfsBaseFile::GetSymbol(this, v12, cid, ppCtxContainer);
        }
        else
        {
            rc = STATUS_INVALID_HANDLE;
        }

    }
    // Omit ...
    return rc;
}
```

Look for the root cause

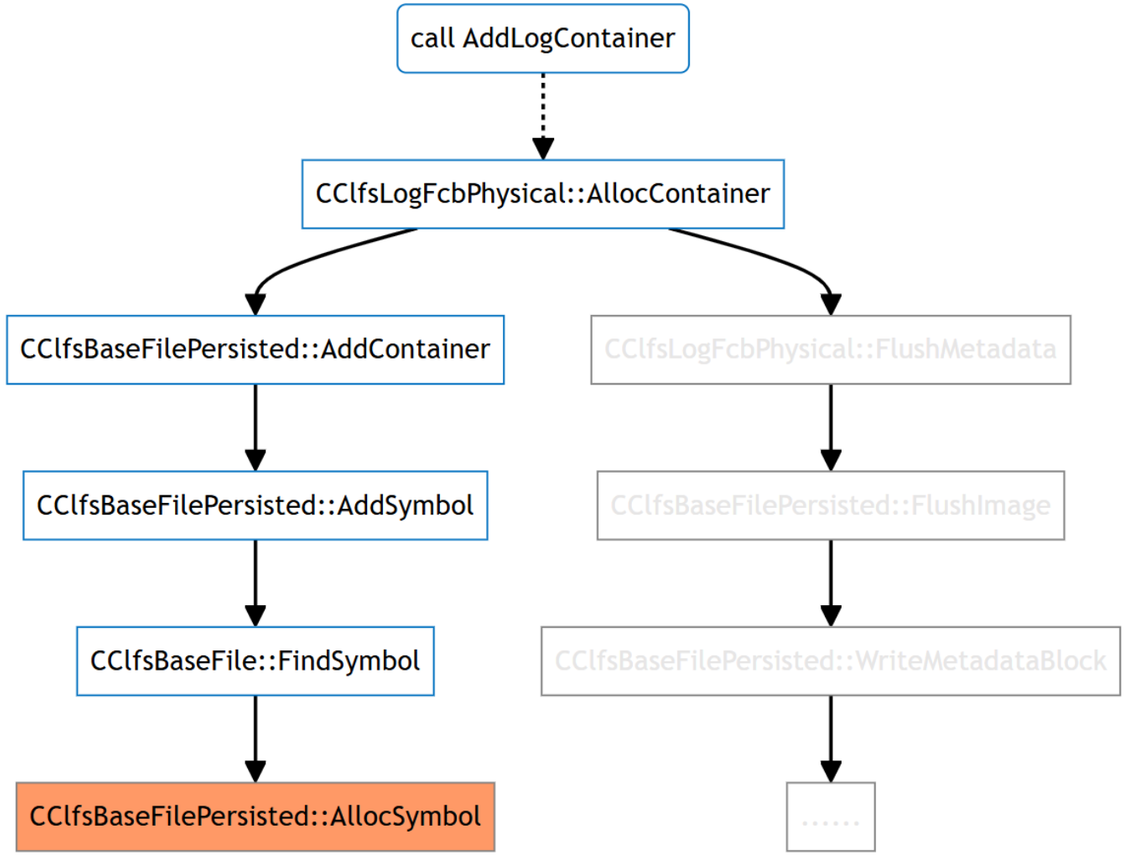
- **Where the Container Symbol Changed**
  - CClfsBaseFile::**GetSymbol**
  - CClfsBaseFilePersisted::**AddSymbol**
  - ……



https://securelist.com/windows-clfs-exploits-ransomware/111560/

Look for the root cause

- **Where the Container Symbol Changed**
  - Allocate symbol when calling **AddLogContainer**



Look for the root cause

```
NTSTATUS CClfsBaseFilePersisted::AllocSymbol(CClfsBaseFilePersisted *this, unsigned int szAllocate, PCLFSHASHSYM
*ppSym)
{
    PCLFS_BASE_RECORD_HEADER precHdr = CClfsBaseFile::GetBaseLogRecord(this);
    if (!precHdr)
        return STATUS_LOG_METADATA_CORRUPT;

    PCLFS_LOG_BLOCK_HEADER pblockHdr = this->m_rgBlocks[ClfsMetaBlockGeneral].pbImage;
    *ppSym = nullptr;

    pSymbolZoneEnd = (char *)precHdr + sizeof(CLFS_BASE_RECORD_HEADER) + precHdr->cbSymbolZone;

    char *pBlockEnd = (char*)pblockHdr + pblockHdr->SignaturesOffset;

    if (pSymbolZoneEnd + szAllocate > pBlockEnd)
        return STATUS_BUFFER_TOO_SMALL;

    memset(pSymbolZoneEnd, 0, szAllocate);
    precHdr->cbSymbolZone += szAllocate;

    *ppSym = pSymbolZoneEnd; // 0x8200 + 0x70 + 0x1338 + 0x3C = 0x95E4 (In File Common Block)

    return STATUS_SUCCESS;
}
```

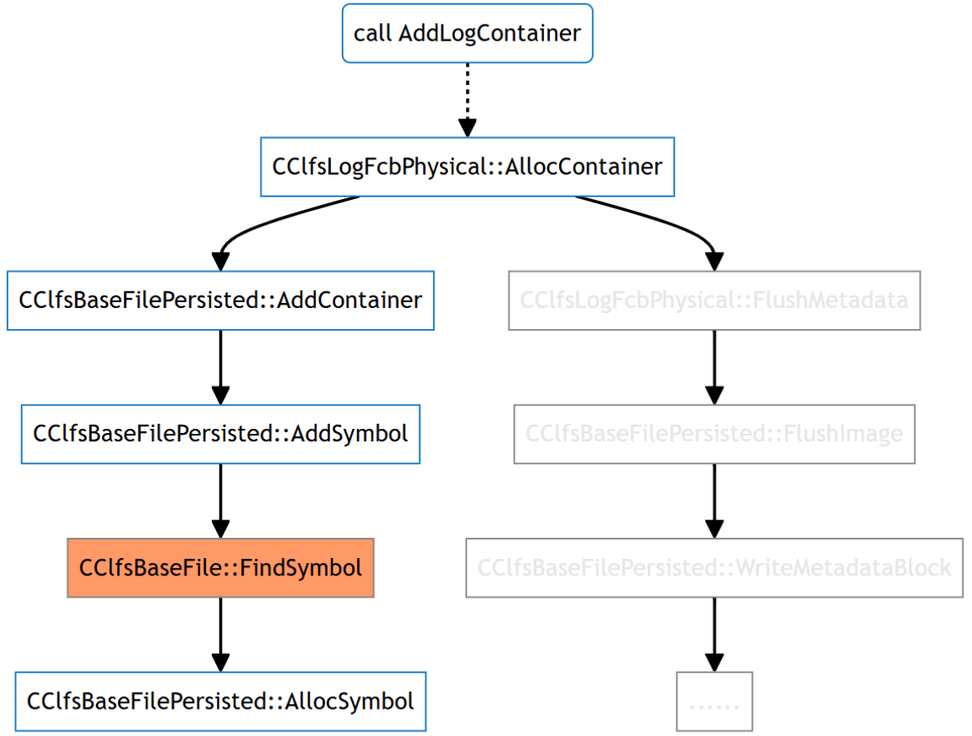
Look for the root cause

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
9590	00	00	00	00	00	00	00	00	3C	00	00	00	00	00	00	00	........<.......
95A0	00	00	43	01	02	00	00	00	06	F0	FD	C1	30	00	00	00	..C.....ðýÁ0...
95B0	16	00	D2	02	B8	00	00	00	00	00	00	00	00	00	00	00	..Ò.¸...........
95C0	00	00	00	00	00	00	00	00	F0	13	00	00	68	13	00	00	........ð...h...
95D0	00	00	00	00	00	00	00	00	07	F0	FD	C1	88	00	00	00	........ðýÁˆ...
95E0	00	00	00	00	40	9C	00	00	00	00	00	00	00	00	00	00	....@œ..........
95F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	10	04	....
9600	00	00	00	00	D4	13	00	00	A4	13	00	00	00	00	00	00	....Ô...¤.......
9610	00	00	00	00	00	00	00	40	00	00	00	40	00	00	00	40	.......@...@...@
9620	00	00	00	40	01	00	00	00	A4	13	00	00	01	00	00	00	...@....¤.......
9630	00	00	00	00	00	00	00	40	00	00	00	40	00	00	00	40	.......@...@...@
9640	00	00	00	40	00												
9650	00	00	00	00	00												
9660	5C	00	3F	00	3F												
9670	73	00	65	00	72												
9680	6C	00	69	00	63												
9690	67	00	2E	00	62												

> BaseRecordHeader		8270h	1338h	struct CLFS_BASE_RECORD_HE…
⌄ ClientContext		95A8h	0h	struct
⌄ Context[0]		95D8h	88h	struct CLFS_CLIENT_CONTEXT
> cidNode		95D8h	8h	struct CLFS_NODE_ID
cidClient	0h	95E0h	2h	CLFS_CLIENT_ID
fAttributes	VIRTUAL (0h)	95E2h	2h	enum FILE_ATTRIBUTES
cbFlushThreshold	9C40h	95E4h	4h	ULONG
cShadowSectors	0h	95E8h	4h	ULONG

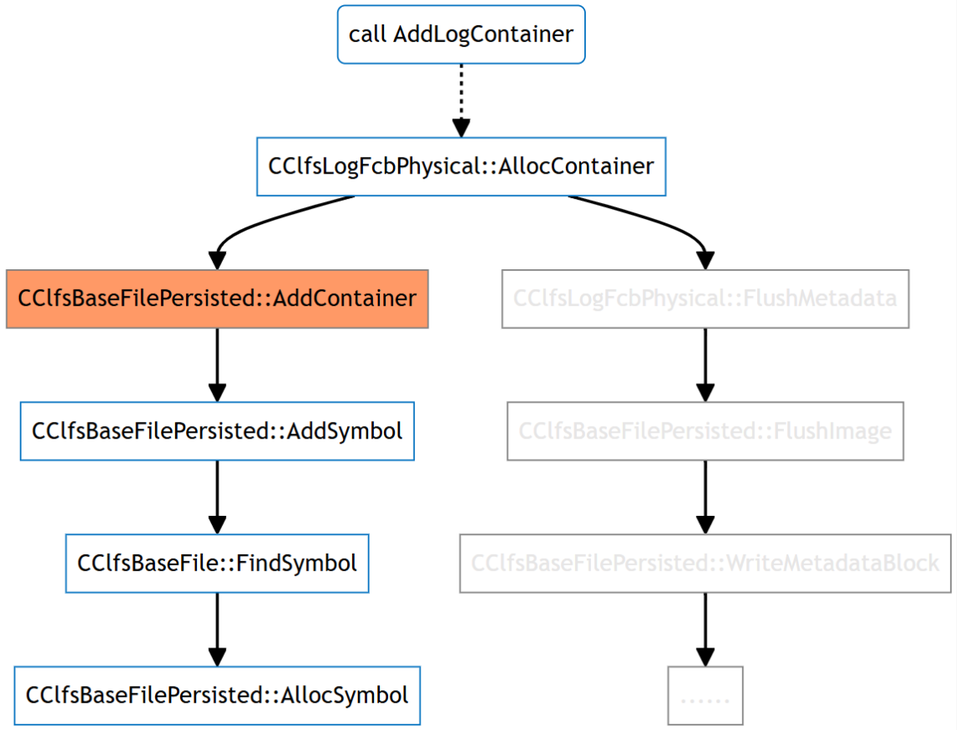
Look for the root cause

```
NTSTATUS CClfsBaseFile::FindSymbol(PUNICODE_STRING a1, PCLFSHASHTBL pHashTbl, char a3, int a4, PCLFSHASHSYM *ppHashSym)
{
    // Omit ...
    ULONG cbAlloc = szSym + cbHashSym + 0x30;
    PCLFSHASHSYM pNewHashSym = nullptr;
    NTSTATUS rc =
        pBaseFile->AllocSymbol(pBaseFile, cbAlloc,
                &pNewHashSym);
    if (rc < 0)
        return rc;
    // Omit ...
    pNewHashSym->cidNode.cType = 0xC1FDF006;
    pNewHashSym->cidNode.cbNode = 0x30;
    pNewHashSym->cbOffset =
         (ULONG)((char *)pNewHashSym -
                 (char *)precHdr + 0x30);
    pNewHashSym->cbSymName =
        (ULONG)((char *)pSymName - (char *)precHdr);

     *ppHashSym = pNewHashSym;

    return STATUS_SUCCESS;
}
```



Look for the root cause

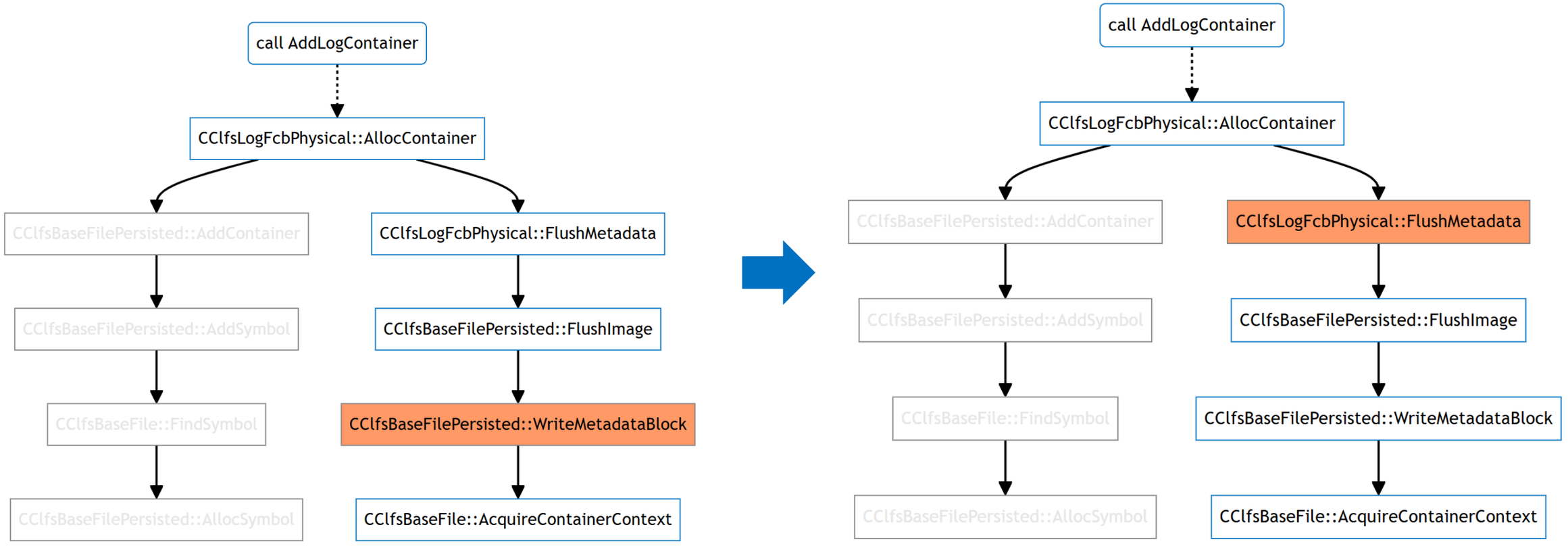
```
NTSTATUS CClfsBaseFilePersisted::AddContainer(PUNICODE_STRING a1, ...)
{
    // Omit ...

    NTSTATUS rc = CClfsBaseFilePersisted::AddSymbol(this, a1,
        &this->m_symContainerTbl, 0x30u, &v31, &offset);

    // Omit ...

    PCLFSHASHSYM pSymContainer =
         CClfsBaseFile::OffsetToAddr(this, offset);

    if (!pSymContainer)
        return STATUS_INSUFFICIENT_RESOURCES;

    // Omit ...

    precHdr->rgContainers[i] = pSymContainer->cbOffset;

    // Omit ...

    return STATUS_SUCCESS;
}
```



Look for the root cause

```
NTSTATUS CClfsBaseFile::AcquireContainerContext(CClfsBaseFile *this, UINT cid, PCLFS_CONTAINER_CONTEXT
*ppCtxContainer)
{
    NTSTATUS rc;
    // Omit ...
    if (precHdr)
    {
        UINT v12 = precHdr->rgContainers[cid];

        if (v12)
        {
            rc = CClfsBaseFile::GetSymbol(this, v12, cid, ppCtxContainer);
        }
        else
        {
            rc = STATUS_INVALID_HANDLE;
        }

    }
    // Omit ...
    return rc;
}
```

Look for the root cause



How it manipulates the pContainer

```
__int64 __fastcall CClfsLogFcbPhysical::FlushMetadata(CClfsLogFcbPhysical *pClfsLogFcbPhysical)
{
  // Omit ...

  v2 = CClfsBaseFile::AcquireClientContext(pClfsLogFcbPhysical->CClfsBaseFile, 0, &pCtxClient_);
  if ( v2 >= 0 && (pCtxClient = pCtxClient_) != 0i64 )
  {
    eState = pCtxClient_->eState;
    Flag = pClfsLogFcbPhysical->Flag;
    pCtxClient_->llCreateTime = pClfsLogFcbPhysical->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbPhysical->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbPhysical->ClientllWriteTime;
    pCtxClient->lsnOwnerPage = pClfsLogFcbPhysical->ClientlsnOwnerPage;
    pCtxClient->lsnArchiveTail = pClfsLogFcbPhysical->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbPhysical->ClientlsnBase.ullOffset;
    // The 10th CLFS exploit write fake pContainer here
    pCtxClient->lsnLast = pClfsLogFcbPhysical->ClientlsnLast.ullOffset;       // high 4 bytes
    pCtxClient->lsnRestart = pClfsLogFcbPhysical->ClientlsnRestart.ullOffset; //  low 4 bytes
    pCtxClient->cShadowSectors = pClfsLogFcbPhysical->ClientShadowSectors;
    pCtxClient->fAttributes = pClfsLogFcbPhysical->fAttributes;
    _eState = eState | 8;
    // Omit ...
}
```

How it manipulates the pContainer

∨ hashSym	
> cIdNode	
ulHash	D6527D4h
cbHash	60h
ullBelow	0h
ullAbove	0h
cbSymbolName	13D4h
cbOffset	13A4h
fDeleted	0h
∨ ctx	
> cidNode	
cbContainer	4000000040000000h
cidContainer	1h
cidQueue	1000h
∨ Ptr	
pContainer	40000000h
ullAlignment	40000000h
usnCurrent	40000000h
eState	ClfsContainerInactive (2h)
cbPrevOffset	0h
cbNextOffset	0h

Variant

Why is variant analysis important

17/40 in-the-wild 0-days from 2022 are variants of previously known bugs.

https://github.com/maddiestone/ConPresentations/blob/master/Zer0Con2023.YearInReview.pdf

How we conduct variant analysis

- **Condition # 1**
  - The vulnerability itself never disappeared, however sometimes a simple check was added in a recent patch, causing the exploit to no longer work
  - Typical example: **The “patch” of 1st CLFS Exploit** -> **CVE-2022-24481**

- **Condition # 2**
  - The developers seemed to understand the root cause and fixed the vulnerability, but they either didn't fix it properly or make a mistake
  - Typical example # 1: **The patch of CVE-2022-24481** -> **CVE-2022-35803**
  - Typical example # 2: **The patch of CVE-2023-23376** -> **CVE-2022-28252**

Variant Case # 1: CVE-2022-24481

- How the 1st CLFS Exploit use the vulnerability

```
__int64 __fastcall CClfsLogFcbPhysical::FlushMetadata(CClfsLogFcbPhysical *pClfsLogFcbPhysical)
{
  // Omit ...

  v2 = CClfsBaseFile::AcquireClientContext(pClfsLogFcbPhysical->CClfsBaseFile, 0, &pCtxClient_);
  if ( v2 >= 0 && (pCtxClient = pCtxClient_) != 0i64 )
  {
    eState = pCtxClient_->eState;
    Flag = pClfsLogFcbPhysical->Flag;
    pCtxClient_->llCreateTime = pClfsLogFcbPhysical->ClientllCreateTime;
    pCtxClient->llAccessTime = pClfsLogFcbPhysical->ClientllAccessTime;
    pCtxClient->llWriteTime = pClfsLogFcbPhysical->ClientllWriteTime;
    pCtxClient->lsnOwnerPage = pClfsLogFcbPhysical->ClientlsnOwnerPage;
    pCtxClient->lsnArchiveTail = pClfsLogFcbPhysical->ClientlsnArchiveTail.ullOffset;
    pCtxClient->lsnBase = pClfsLogFcbPhysical->ClientlsnBase.ullOffset;
    pCtxClient->lsnLast = pClfsLogFcbPhysical->ClientlsnLast.ullOffset;
    pCtxClient->lsnRestart = pClfsLogFcbPhysical->ClientlsnRestart.ullOffset;
    pCtxClient->cShadowSectors = pClfsLogFcbPhysical->ClientShadowSectors;
    pCtxClient->fAttributes = pClfsLogFcbPhysical->fAttributes;
    _eState = eState | 8; // 1-bit flip
    if ( (Flag & 0x10) == 0 )
        _eState = eState;
    pCtxClient->eState = _eState; // The 1st CLFS exploit write pContainer here
    // Omit ...
}
```

Variant Case # 1: CVE-2022-24481

- **How microsoft “patch” this vulnerability**
  - The patch **checked the value of Client Context Offset** to make sure it **couldn't be less than 0x1368**, the Client Context Offset in the 1st exploit was **0x2B5**

```
__int64 __fastcall CClfsBaseFile::GetSymbol(
        CClfsBaseFile *this,
        unsigned int offset,
        char a3,
        struct _CLFS_CLIENT_CONTEXT **a4)
{
  unsigned int v8; // ebx
  BOOLEAN v10; // r15
  struct _CLFS_CLIENT_CONTEXT *v11; // rax
  unsigned int v12; // [rsp+20h] [rbp-38h]

  v8 = 0;
  v12 = 0;
  if ( offset < 0x1368 )                    // patch
    return 0xC01A000Di64;
  *a4 = 0i64;
  v10 = ExAcquireResourceSharedLite(*((PERESOURCE *)this + 4), 1u);
  v11 = (struct _CLFS_CLIENT_CONTEXT *)CClfsBaseFile::OffsetToAddr(this);
```

Variant Case # 1: CVE-2022-24481

- **How to bypass the check**
  - What if we construct a Client Context Offset that is **greater than 0x1368**, and make the Client Context Offset point to a forged Container Context?

```
pClientContext = Base Log Record + 0x10000(New Offset)
               =    0x800 + 0x70 + 0x10000(New Offset) = 0x10870
                                                      = addr of a forged Container Context
```

```
1:07E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 01
1:0800h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1:0820h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1:0840h: 06 F0 FD C1 30 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1:0860h: 30 00 01 00 00 00 01 00 00 00 00 00 00 00 00 00 08 F0 FD C1 30 00 00 00 00 00 08 00 00 00 00 00
1:0880h: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
1:08A0h: 5C 00 3F 00 3F 00 5C 00 43 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00 5C 00 75 00 73 00 65 00
1:08C0h: 72 00 5C 00 41 00 70 00 70 00 44 00 61 00 74 00 61 00 5C 00 4C 00 6F 00 63 00 61 00 6C 00 5C 00
1:08E0h: 54 00 65 00 6D 00 70 00 5C 00 77 00 63 00 74 00 46 00 43 00 35 00 2E 00 74 00 6D 00 70 00 00 00
```

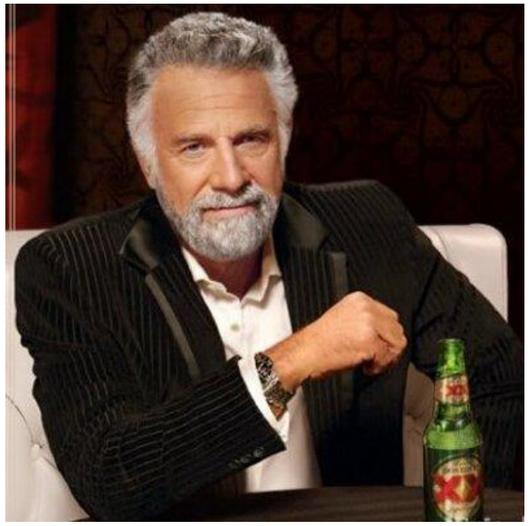
offset = 0x10000

Make 1-bit flip great again

```
1: kd> .formats ffffd78f`c8725f30 // origin pContainer
Evaluate expression:
  Hex:     ffffd78f`c8725f30

  Binary:  11111111 11111111 11010111 10001111 11001000 0111 0 010 01011111 00110000
```

```
1: kd> .formats ffffd78f`c87a5f30 // fake pContainer
Evaluate expression:
  Hex:     ffffd78f`c87a5f30

  Binary:  11111111 11111111 11010111 10001111 11001000 0111 1 010 01011111 00110000
```

Variant Case # 1: CVE-2022-24481



Variant Case # 2: CVE-2022-35803

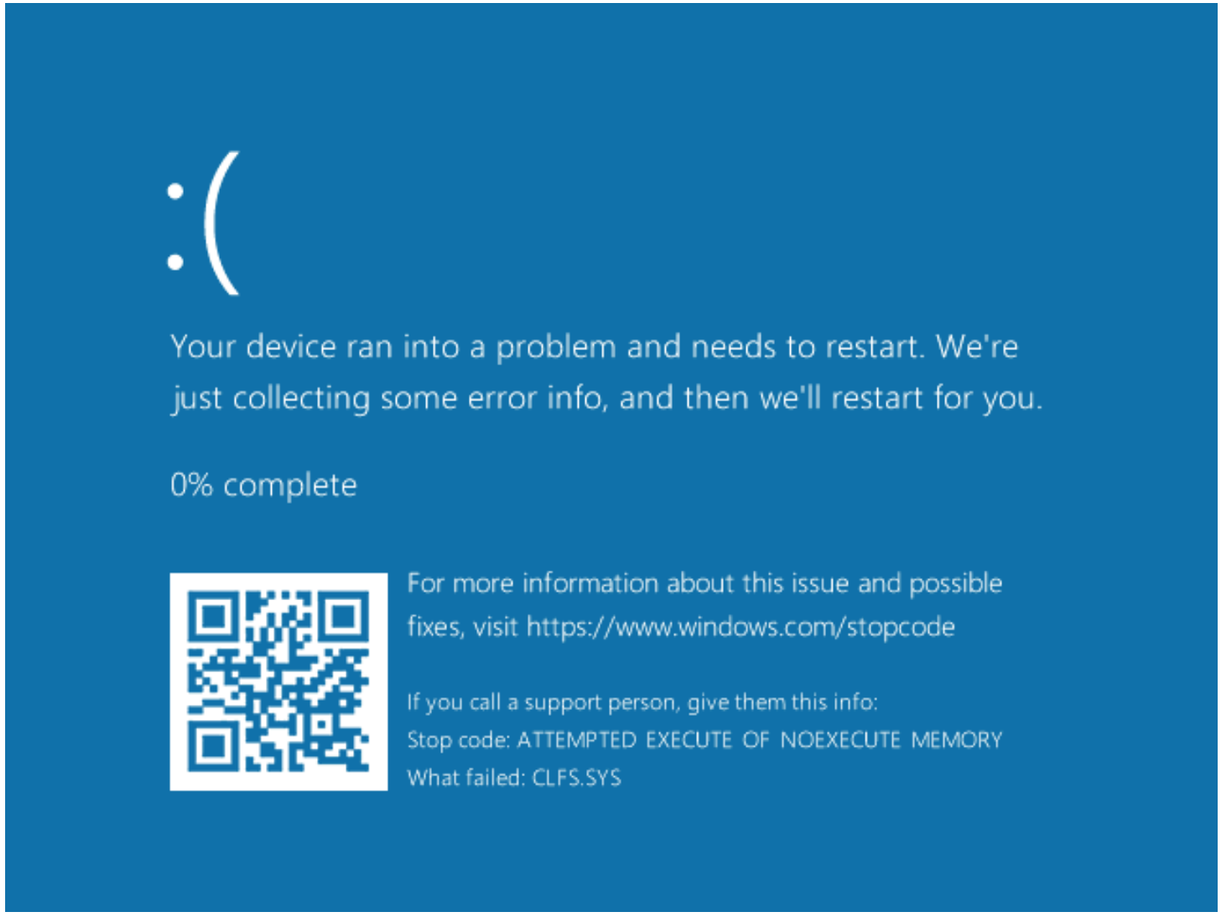
- **The patch of CVE-2022-24481**
  - In the April 2022 patch, CLFS driver added a new function **CClfsBaseFile::ValidateRgOffsets**
    - The way of directly modifying the **Client Context Offset** has been fixed
    - Still no check for **cidNode.cType**

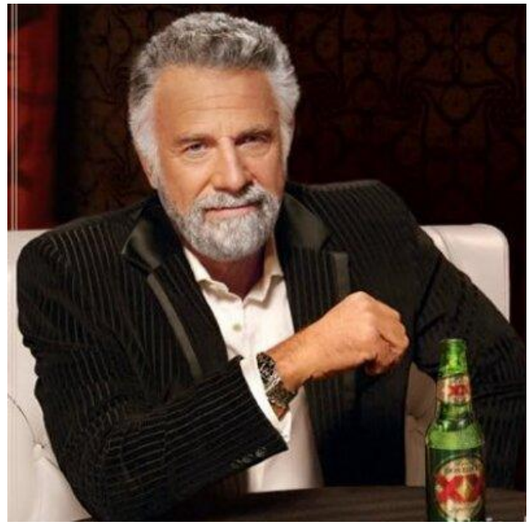
- **How to bypass the check**
  - Set the **cidNode.cType** of Client Context to **0xC1FDF008** (the type of Container Context)

```
+-------------+  +0x00 <--- pClientContext
|             |               cidNode.cType = 0xC1FDF008 (the type of Container Context)
+-------------+  +0x30 <--- Symbol of Container Context
|             |
+-------------+  +0x60 <--- pContainerContext
|             |
+-------------+  +0x78 <--- pContainerContext->pContainer
|             |               (overlapped with) pClientContext->eState
```

https://blog.northseapwn.top/2022/11/11/Windows-Kernel-Exploit-CVE-2022-35803-in-Common-Log-File-System/index.html

Variant Case # 2: CVE-2022-35803





Summary

Hunt 10 itw CLFS exploits in 2 years

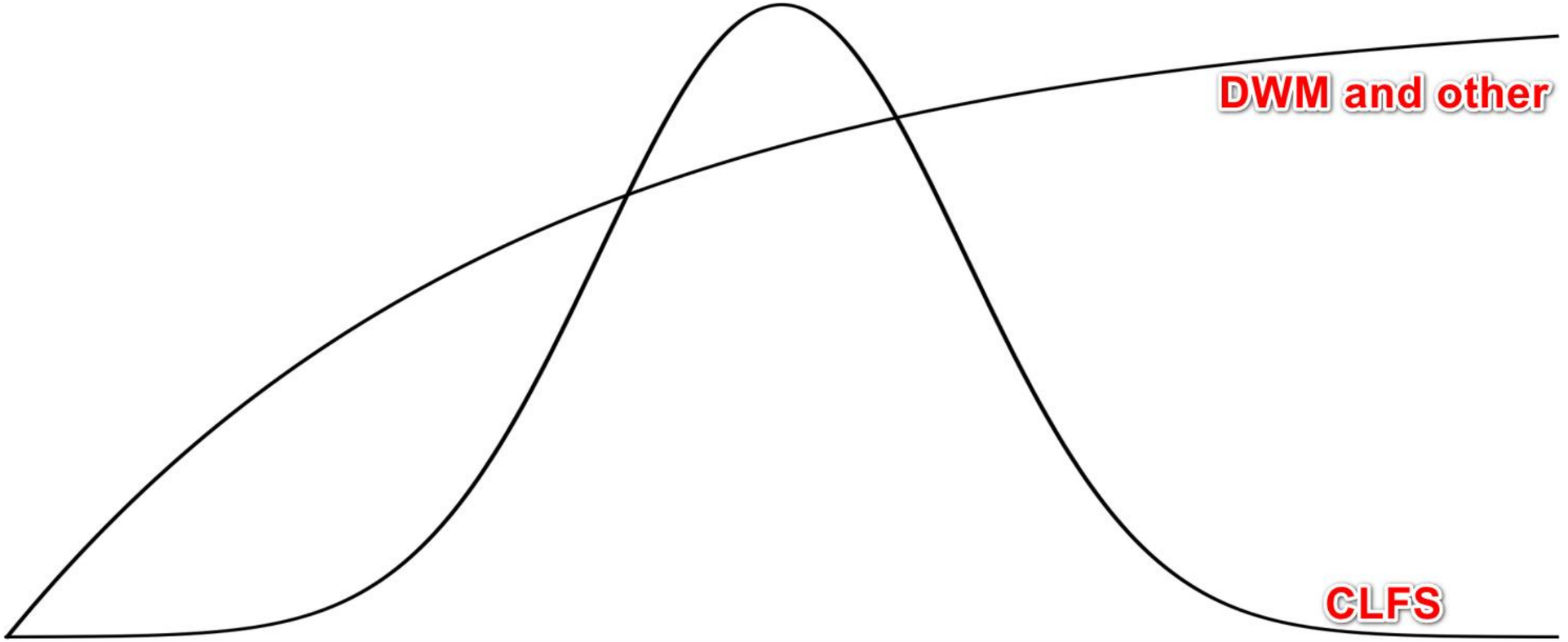
Exploit Name	CVE	Kaspersky Alias	SHA-256
The 1st Exploit			86a8f267cf0f51c032f7b1777eb1e51f7cd1badf3f3894e2557a3f571fca9f3d
The 2nd Exploit		Exploit #1	f94998b90a28c678e4ed6bdf851f339e02a58369435b20ad62858e0ea5bc8eba
The 3rd Exploit	CVE-2022-24521		eb3452c64970f805f1448b78cd3c05d851d758421896edd5dfbe68e08e783d18
The 4th Exploit	CVE-2022-37969		0a478f8d4f5f203e100a2a6c56a4e71a062ec463eb68c3f833fd74b3070af482
The 5th Exploit		Exploit #2	234541906b3c50d907b6f7668632b57f0cb43002b4a8241eca1f4c412898c586
The 6th Exploit		Exploit #3	9a676c29863d06a1344b7b983b9f8c15978ca9914542bec1c20c1c5e4985c529
The 7th Exploit	CVE-2023-23376	Exploit #4	eecb4b46b140258887fde5cc95552359aad259a9ddc0d7801e2b7949108be15a
The 8th Exploit	CVE-2023-28252	Exploit #5	018c464676b4a71be83bc073f482e94a4850e9c24abe4c4ed1285258ca95a21e
The 9th Exploit			cd882d0dae4d0734c00a7838d810cab22d9207721fe3cdd4ac7ec3adf9555ea2
The 10th Exploit			a37c848d279e68b7ff01c97c07baf7f33727abc6e1ee79348ff22597b259e9d4

and more ...

Get 4 CVEs in this journey

Apr 12, 2022	Jinquan with DBAPPSecurity Threat Intelligence Center	Windows Common Log File System Driver Elevation of Privilege Vulnerability	CVE-2022-24481
Sep 13, 2022	Zscaler ThreatLabz with Zscaler CrowdStrike Genwei Jiang with Mandiant, FLARE OTF Quan Jin with DBAPPSecurity	Windows Common Log File System Driver Elevation of Privilege Vulnerability	CVE-2022-37969
Sep 13, 2022	Mahendra Mishra of Microsoft's Windows Servicing and Delivery Group xi4oyu and Quan Jin with DBAPPSecurity WeBin Lab	Windows Common Log File System Driver Elevation of Privilege Vulnerability	CVE-2022-35803
Apr 11, 2023	Quan Jin with DBAPPSecurity WeBin Lab Genwei Jiang with Mandiant Boris Larin (oct0xor) with Kaspersky	Windows Common Log File System Driver Elevation of Privilege Vulnerability	CVE-2023-28252

Trends on itw Windows LPE Exploits



Acknowledgements

Xuecheng Li(@lxi4oyu)

Siyuan Liu(@4nsw3r123)

Mingjia Liu(@cyberestro)

Genwei Jiang(@binjo)

Thanks!