DeathNote of Microsoft Windows Kernel

windows kernel internals



\$whoami

Peter

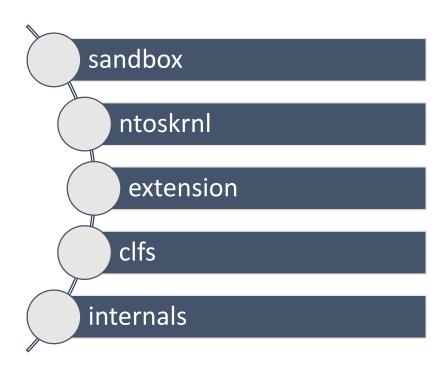
- @zer0mem
- Windows kernel research at KeenLab, Tencent
- pwn2own winner (2015 / 2016),
 pwnie nominee (2015)
- fuzzing focus : state
- wushu player

Daniel

- @long123king
- Windows kernel research at KeenLab, Tencent
- pwn2own winner (2016)
- fuzzing focus : data 'format'
- windbg guy



agenda



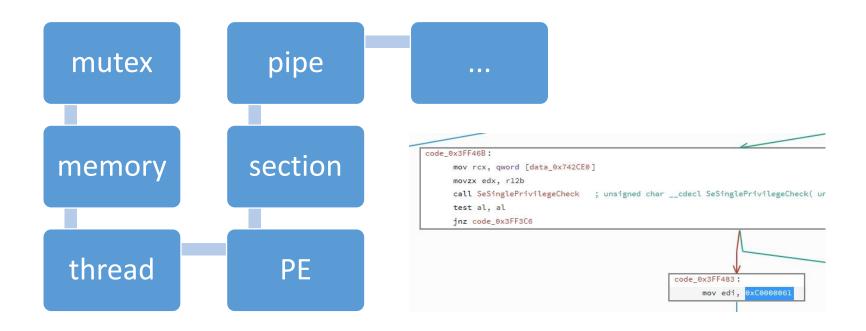


Sandbox

- limiting attack surface
 - potantional landscape for bugs
 - available methods for abusing it
- ACL vs access to various kernel objects
 - non ntos, non w32k drivers
 - various ntos objects
- w32k filtering
 - depends on sandboxed app needs
- w32k lockdown



Sandbox targets





... plus ?

- Nt*Transaction*
- Nt*Enlistment*
- Nt*Manager*

				AND THE PARTY OF T
S	Strings	Imports	Functions	
	Туре	Length	Value 🛦	
	Import	8	NtCommitTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtCreate <mark>Transaction</mark>	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtCreate <mark>Transaction</mark> Manager	ext-ms-win-ntos-tm-ll-1-0.dll
	Import	8	NtEnumerate <mark>Transaction</mark> Object	ext-ms-win-ntos-tm-ll-1-0.dll
	Import	8	NtFreeze <mark>Transaction</mark> s	ext-ms-win-ntos-tm-ll-1-0.dll
	Import	8	NtOpenTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtOpenTransactionManager	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtQueryInformationTransaction	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtQueryInformation <mark>Transaction</mark> Manager	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtRecover <mark>Transaction</mark> Manager	ext-ms-win-ntos-tm-ll-1-0.dll
	Import	8	NtRename <mark>Transaction</mark> Manager	ext-ms-win-ntos-tm-ll-1-0.dll
	Import	8	NtRollback <mark>Transaction</mark>	ext-ms-win-ntos-tm-ll-1-0.dll
	Import	8	NtRollforward <mark>Transaction</mark> Manager	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtSetInformation <mark>Transaction</mark>	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtSetInformation <mark>Transaction</mark> Manager	ext-ms-win-ntos-tm-l1-1-0.dll
	Import	8	NtThawTransactions	ext-ms-win-ntos-tm-l1-1-0.dll



what?

Kernel Transaction Manager

- Purpose
 - The Kernel Transaction Manager (KTM) enables the development of applications that use transactions. The transaction engine itself is within the kernel, but transactions can be developed for kernel- or user-mode transactions, and within a single host or among distributed hosts.
 - The KTM is used to implement Transactional NTFS (TxF) and Transactional Registry (TxR). TxF allows transacted file system operations within the NTFS file system. TxR allows transacted registry operations. KTM enables client applications to coordinate file system and registry operations with a transaction.



tm.sys

- simple object state
- few syscalls available
- not much code involved
- however interestingly interconnected

- Results:
 - 1 nullptr deref
 - 1 exploitable vulnerability

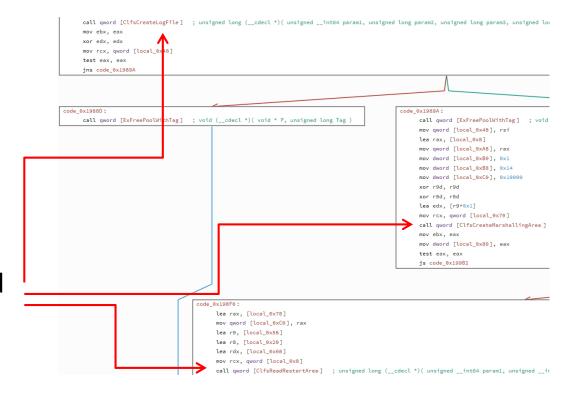


tm indirection

tm.sys simple purpose driver

 but interesting module involved at backend

• CLFS.sys





CLFS.sys

Purpose

- The Common Log File System (CLFS) API provides a high-performance, general-purpose 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.

Where applicable

- You can use CLFS for data and event management and to develop server and enterprise applications.
- For data management, you can use CLFS with the following:
 - Database systems
 - Messaging, such as store-and-forward systems
 - Online transactional processing (OLTP) systems
 - Other kinds of transactional systems



CLFS.sys

- well integrated to transactions and more!
- c++ code base
- serve fair attack surface

• ... but not at appcontainer or untrusted level ...

• or does it?



NtCreateTransactionManager

- depends on CLFS
- use CLFS for its checkpoints

- therefore implies :
 - Opens CLFS
 - *PARSE* CLFS
 - interact with CLFS

lets try it out!

CLFS - data fuzzing I.

- i am not fan of data fuzzing in kernel
 - as i am strongly against data parsing at kernel at all:)
- lets do quick probe, that i am ok with :
 - mutate randomly file
 - results = 0
 - cool for me, i am not much interested anyway
 - get back to original idea!



CLFS - state fuzzing

- approach 1.
 - RE clfs.sys
 - go to ioctl
 - .. ah too lazy to do that from scratch ...
- approach 2.
 - go trough msdn docs
 - understand how those api works
 - callstack necessary to suceed to call one or another api
 - implement that logic to Qilin (our internal fuzzer)
 - mess with logic in Qilin little bit



bugz++

- after first dry run of fuzzer in 15 min first crashes
 - ... wtf
- but ddos only
- eliminate that
- another bugz apear
- now time to rethink .. data fuzzing really so bad idea afterall ?



CLFS - data fuzzing II.

RE where & how are data parsed

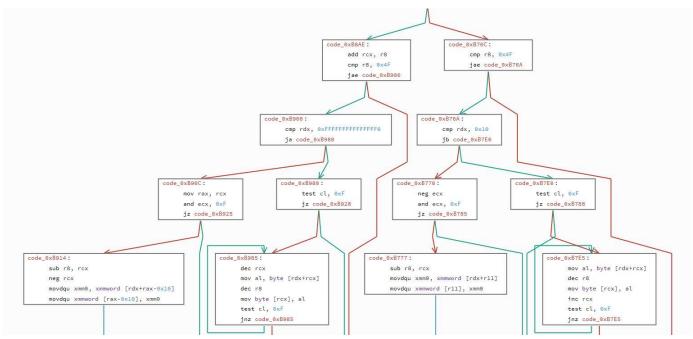
EntryPoint : ClfsCreateLogFile

- ouch ... magic .. dummy fuzz proof
 - I. crc
 - II. rellocation required



CLFS - lets fuzz more seriously

 too lazy to re-implement existing code, but is it even necesary?





CLFS - lets fuzz more seriously

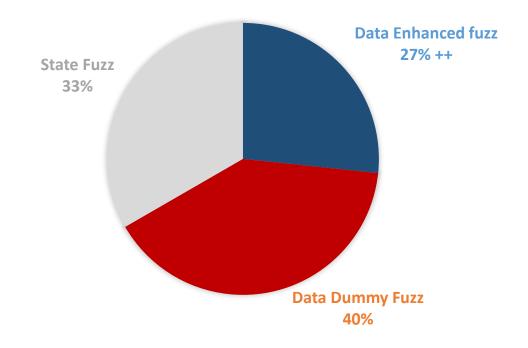
too lazy to implement crc & rellocations

```
checkReturn
NTSTATUS
ClfsEncodeBlock fn(
    void* block,
    size t size,
                                                  void
    size_t type,
                                                  DoPoC()
    size_t gran = 0x10,
    bool crc32 = true
decltype(&ClfsEncodeBlock_fn) ClfsEncodeBlock =
                                                      CMemFromFileResource clfs("c:/windows/system32/drivers/clfs.sys");
                                                      CMemFromFileResource crc("m_rgCrcTable");
ClfsDecodeBlock fn(
                                                      printf("\n table : %i", crc->size());
    void* block,
    size_t size,
                                                      memcpy(clfs->get<char>() + 0x1D4B0, crc->get<char>(), crc->size());
    size_t type,
    size_t gran,
                                                      clfsEncodeBlock = reinterpret_cast<decltype(&ClfsEncodeBlock_fn)>(clfs->get<char>() + 0x9d88);
    size t* res
                                                      ClfsDecodeBlock = reinterpret cast<decLtype(&ClfsDecodeBlock fn)>(clfs->get<char>() + 0x9d88 - 0x0
decltype(&ClfsDecodeBlock_fn) ClfsDecodeBlock =
                                                      DWORD protect;
                                                      if (FALSE == VirtualProtect(clfs->get<char>(), clfs->size(), PAGE_EXECUTE_READWRITE, &protect())
```



CLFS { state, dummy, enhanced }

CLFS FUZZING STRATEGIES => RESULTS





CLFS internals

... under the hood ...



BLF (Base Log File) Format

+0x0000		1
+0x0000	Control Record	[2 Sectors], contains layout, extend and truncate information.
+0x0400	Control Record Shadow	[2 Sectors], shadow copy of control record
+0x0800		
	Base Log Record	[0x3D Sectors], contains client and container information.
+0x8200	Base Log Record Shadow	[0x3D Sectors], shadow copy of base log record.
+0xFC00	Truncate Record	[1 Sector], contains truncate information.
+0xFE00	Truncate Record Shadow	[1 Sector], shadow copy of truncate record.
•		

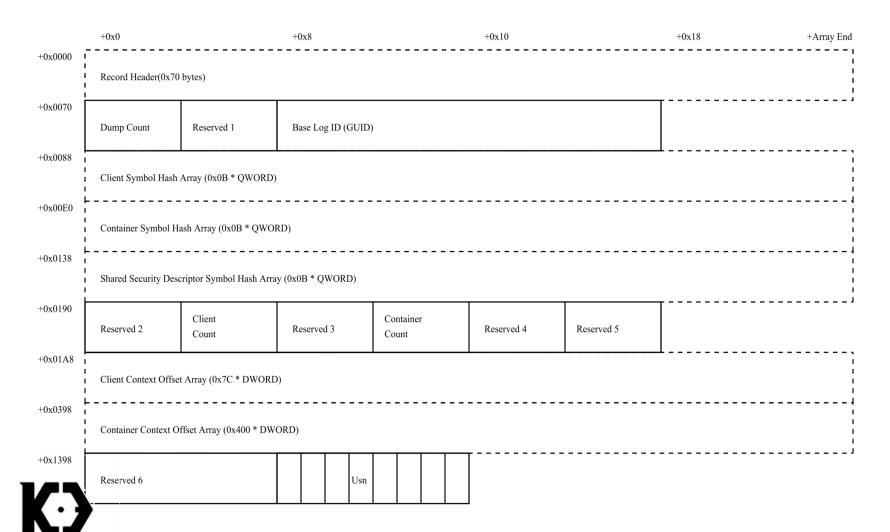
Record Header

	+0x0	+0x2	+0x3	+0x4	+0x6	+0x8	+0xC +0x10
+0x00	Magic	Fixup Upper Byte	Stream Index	Number of Sectors	Number of Sectors Copy	Reserved 1	Checksum
+0x10	Format Version			Reserved 2		Current LSN	
+0x20	Next LSN					Record Offset Array (0x10 * DWOR	D)
+0x30							
+0x40							
+0x50				 			
+0x60				 		Fixup Offset	Reserved 3
				I I			

Control Record

	+0x0	+0x4	+0x8		+0xC		+0x10
+0x00	Record Header(0x70 bytes)						
+0x70	Dump Count		Magic (0xC1F50	C1F500005F1C)			
+0x80	Reserved 1	Extend Phase	Extend Block Index	Extend Block Index Shadow	Block After l	Sectors Extend	
+0x90	File Sectors Before Extend	Delta File Sectors	Truncate Type		Trunc Field 1	Trunc Field 2	Truncate Field 3
+0xA0	Reserved 2		Reserved 3				
+0xB0	Reserved 4		Actual Records Count Reserved 5				
+0xC0	Array of Record Parameters(0x18 * c	ount)				 	

Base Log Record



Container Record

	+0x0		+0x4	+0x8	+0xC $+0x10$	
+0x00	Record Head	der(0x70 bytes)			 	
+0x70	Virtual LSN			Undo Next LSN		
+0x80	Previous LS	N		Record Size	Next Record Size	
+0x90	Flag	Data Offset	Record Type			



Symbol Header

	+0x0	+0x4	+0x8	+0xC	+0x10
+0x00	Type (0xC1FDF006)	Size	Checksum	Name Offset	
+0x10	Reserved 1		Reserved 2		
+0x20	Block Name Offset	Block Attribute Offset	Reserved 3		



Client Context

	+0x0	+0x4	+0x8		+0xC	+0x10
+0x00	Type (0xC1FDF007)	Size	Stream Index	Reserved 1	Flush Threshold	
+0x10	Reserved 2 (0x28 Bytes)					1 1 1 1
+0x38	Reserved LSN 1		Base LSN			
+0x48	Last Flushed LSN		Last LSN			
+0x58	Reserved LSN 2		Reserved LSN	3		
+0x68	Reserved 3 (0x20 Bytes)					
						;

Container Context

	+0x0	+0x4	+0x8	+0xC	+0x10
+0x00	Type (0xC1FDF008)	Size	Container File Size	Reserved 1	
+0x10	Physical Container Index	Logical Container Index	Reserved LSN 1		
+0x20	Stream Count	Flag	Reserved 2		



CClfsBaseFilePersisted::ReadImage

ControlDogord

CClfcRocoFiloPorcietod

	CClisBaseFilePersisted	1	ControlRecord	
+0x28	records_count		Actual Records Count	+0xB8
+0x30	records_params_ptr		Array of Record Parameters	+0xC0
+0x38	records_bitmap_ptr			
+0x90	sector_size			
145				

Record Parameter

	+0x0	+0x4	+0x8	+0xC +	0x10
+0x00	Buffer Pointer		Size	Offset	
+0x10	ID				

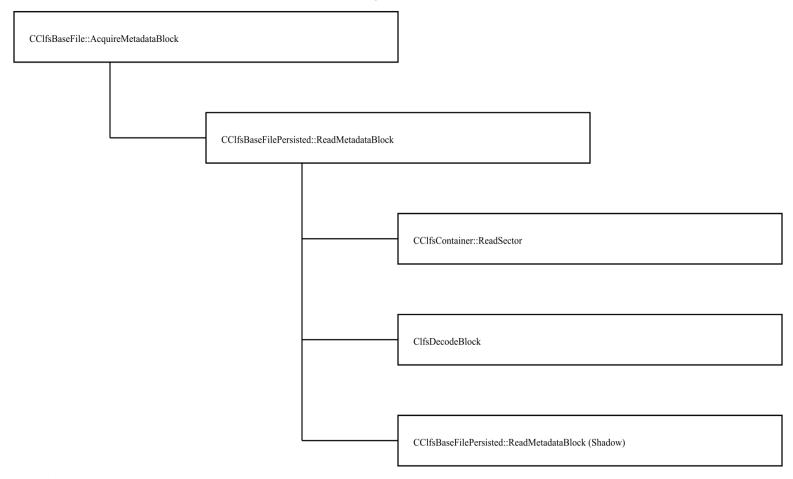


CClfsBaseFile::GetBaseLogRecord

```
CClfsBaseFile::GetBaseLogRecord(CClfsBaseFile* this)
               xor
                       eax, eax
                      ax, [rcx+28h]
               cmp
               jz
                       short locret 1C00335DB
                      rcx, [rcx+30h]
               mov
                      rcx, [rcx+30h]
               mov
               test
                      rcx, rcx
               jz
                      short locret_1C00335DB
                      eax, [rcx+28h]
               mov
               add
                      rax, rcx
locret 1C00335DB:
               retn
```



CClfsBaseFile::AcquireMetadataBlock



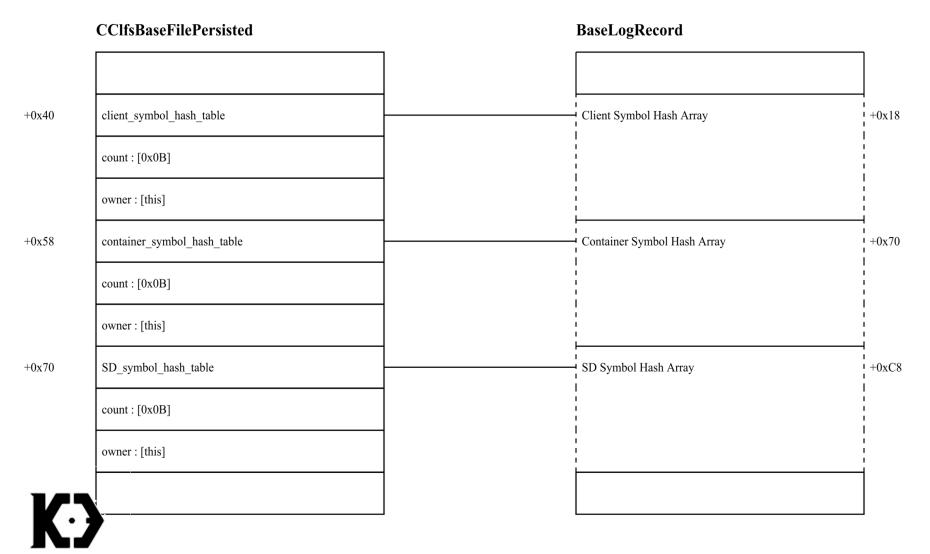


Use of AcquireMetadataBlock

CClfsBaseFile::GetControlRecord CClfsBaseFile::AcquireMetadataBlock CClfsBaseFilePersisted::AcquireTruncateContext CClfsBaseFile::AcquireMetadataBlock



CClfsBaseFilePersisted::OpenImage

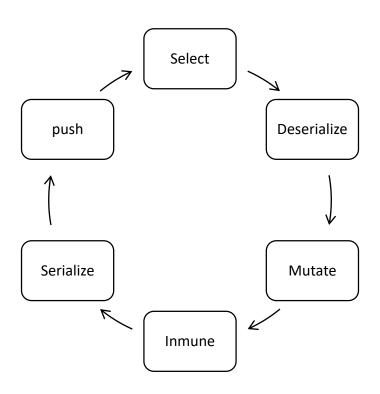


Symbol Hash Function

```
int64 ClfsHashPJW(const struct UNICODE STRING *a1)
    unsigned int v1 = 0, v4 = 0, v6;
    PWSTR wchar_buffer = a1->Buffer;
    const struct UNICODE STRING *v3 = a1;
    if ( a1->Length & 0xFFFE ){
        do{
            int v5 = 0x10 * v1 + RtlUpcaseUnicodeChar(*wchar buffer);
            v6 = v5 \& 0xF0000000;
            if ( v5 & 0xF0000000 )
                v5 ^= v6 >> 0x18;
            v1 = v6 ^ v5;
            ++wchar buffer;
            ++v4;
        while ( v4 < (unsigned int)v3->Length >> 1 );
    return v1;
}
```



- If you know the target well enough, you can fuzz it well.
- Since now, we know:
 - BLF file format
 - Control Record
 - Base Log Record
 - Symbol Header
 - Client Context
 - Container Context
 - Container Record
- Clfs.sys has its own logic to parse these formats, is it robust enough?





```
class CControlRecord : public CFormatBase<CControlRecord>
   virtual bool serialize(ostream & out) const override;
    virtual bool deserialize(istream & in) override;
    virtual bool mutate() override;
    •••••
};
class CBaseLogRecord : public CFormatBase<CBaseLogRecord>
   virtual bool serialize(ostream & out) const override;
    virtual bool deserialize(istream & in) override;
    virtual bool mutate() override;
```



```
bool CCLFSFormat::deserialize(istream & in)
   m controlRecord.deserialize(in);
    m_controlRecordShadow.deserialize(in);
    m_baseLogRecord.deserialize(in);
    m baseLogRecordShadow.deserialize(in);
    m_truncateRecord.deserialize(in);
    m truncateRecordShadow.deserialize(in);
bool CCLFSFormat::mutate(istream & in)
bool CCLFSFormat::serialize(istream & in)
```



```
CCLFSDocument::CCLFSDocument(const string filename)
    :m template filename(filename)
    ,m template stream(filename, ios::in | ios::binary)
    /* number: 0 */m engine.registerFilter(make unique<CCommonErrorBypass>());
    /* number: 1 */m engine.registerFilter(make unique<CPOC XXX 1>());
    /* number: 2 */m engine.registerFilter(make unique<CPOC XXX 2>());
    /* number: 3 */m engine.registerFilter(make unique<CPOC XXX 3>());
    /* number: 4 */m engine.registerFilter(make unique<CPOC XXX 4>());
    /* number: 5 */m engine.registerFilter(make unique<CPOC XXX 5>());
void CCLFSDocument::mutate()
    m clfs format.mutate();
    m engine.triggerFilter(3, m orginal clfs format, m clfs format);
}
```



```
bool CPOCFilterEngine::triggerFilter(size t filterIndex,
CCLFSFormat& originalFormat, CCLFSFormat& format)
    bool b_triggered = false;
    for (size t i = 0; i < m filters.size(); i++)</pre>
        if (i == filterIndex)
            m filters[i]->infect(originalFormat, format);
            b_triggered = true;
        else
            m filters[i]->immune(originalFormat, format);
    return b_triggered;
```



Q&A

Thank you!



