



Windows Kernel Security

*A Deep Dive into Two Exploits
Demonstrated at Pwn2Own*

Thomas Imbert
Security researcher, Synacktiv
August 25, 2023



About me

- **Thomas Imbert**
- **@masthoon**

- **Security Engineer at Synacktiv**
 - Offensive security company
 - Pentest, Reverse engineering, Development, Incident response
 - Offices in France and we are hiring!

Agenda

- **Introduction to Pwn2Own contest**
- **Finding and exploiting a vulnerability in Cloud Filter (cldflt.sys)**
- **Advances in Windows Kernel mitigations**
- **Analysis of a second LPE in MSKS Server driver (mskssrv.sys)**



Pwn2Own

- **Ethical hacking contest organized by Zero Day Initiative (ZDI)**
- **Pwn2Own Vancouver 2023 in March**
 - Applications, virtualization, browsers, OS, Automotive: Tesla, ...

Target	Prize	Master of Pwn Points
Ubuntu Desktop	\$30,000	3
Microsoft Windows 11	\$30,000	3
Apple macOS	\$40,000	4

Pwn2Own – Windows entries

- **One entry per target**
- **3 attempts of 10 minutes**
- **Two exploits developed:**
 - Standalone Windows LPE from unprivileged user
 - Windows LPE Add-on after VirtualBox escape

Available Add-on Prizes:

Add-on Prize	Prize	Master of Pwn Points
Escalation of privilege leveraging a Windows kernel vulnerability on the host operating system.	\$50,000	5

Pwn2Own – Results

- **Synacktiv entries:** Windows, macOS, Ubuntu, VirtualBox and Tesla



The graphic is a leaderboard for the Pwn2Own 2023 competition. It features a dark blue background with a hexagonal pattern at the bottom left. On the left side, the text 'MASTER OF PWN' is written vertically in large, bold, yellow and white letters. On the right side, the text 'LEADERBOARD' is written vertically in large, bold, yellow letters. The central table lists the top five teams, their prize money, and their points. Each team's name is preceded by a numbered hexagon (1-5) in yellow or blue. The prize money is in yellow boxes, and the points are in blue boxes.

		PRIZE \$	POINTS
1	Synacktiv	\$530,000	53
2	STAR Labs	\$195,000	19.5
3	Team Viettel	\$115,000	12
4	Qrious Security	\$55,000	5.5
5	AbdulAziz Hariri	\$50,000	5



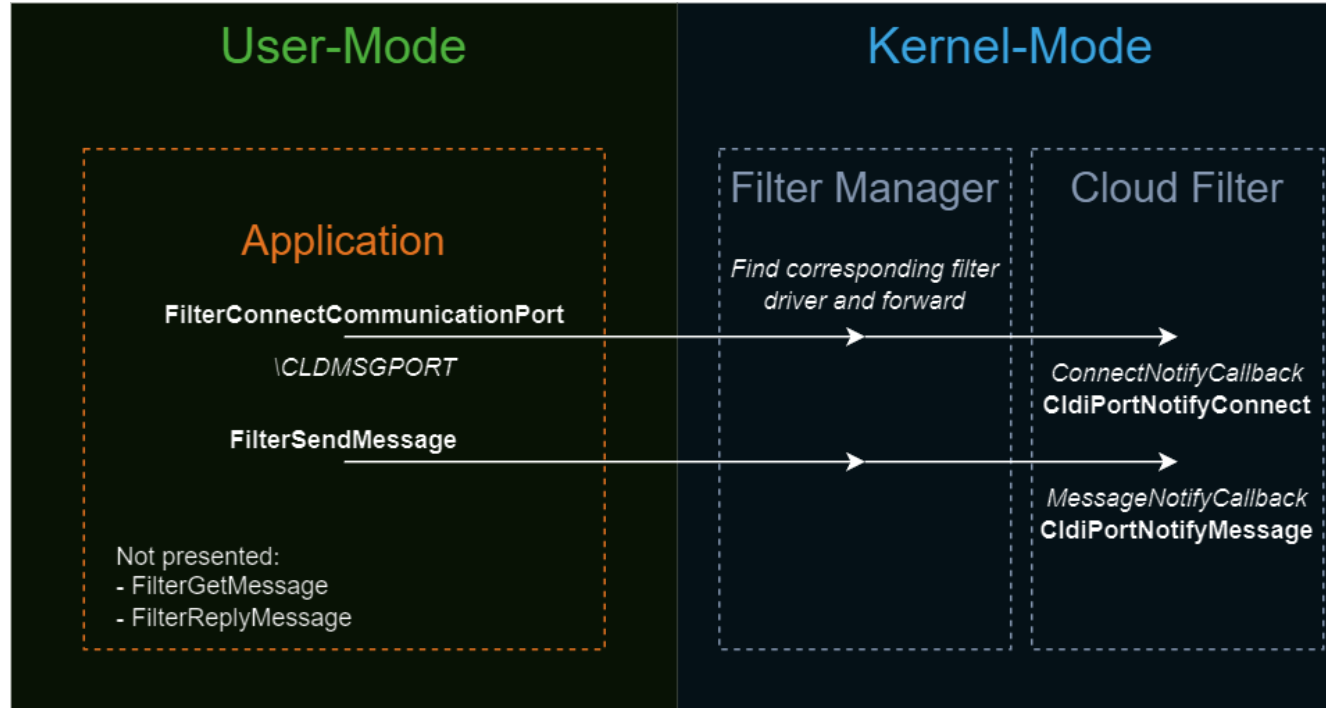
Cloud Filter

Incentives for picking cldflt.sys

- **Pwn2Own requires vulnerability in kernel**
- **3 attempts of 10 minutes: long to trigger / unstable bugs works**
- **Idea: Pick a driver with a different interface than classic IOCTL**
 - Hopefully less reviewed / fuzzed
- **Filter communication port**

Filter Communication Port

- Communication mechanism between User and Kernel mode
- Port identified by a name, used by Filesystem Filter drivers



*Internally, it uses
IOCTL and FSCTL to the
Filter Manager
(fltMgr.sys)*

Incentives for picking cldflt.sys (2)

■ Cloud Filter port

- One of the few interfaces reachable from unprivileged user
- ~20 different messages (excluding FSCTL)
- Asynchronous handling
- Complex implementation

■ Quick manual review found a couple of race conditions

- Not exploitable or very hard to trigger

Cloud Filter (cldflt.sys)

- **File System Minifilter driver**
- **Windows component: Cloud Files API**
 - Manage *placeholders* and *hydrate* them on access
 - Forward file access callbacks to user-mode sync provider process
 - Sync provider process synchronize files with remote cloud storage
- **Utilized by OneDrive**

Fuzzing – First steps

- Manual review takes too long especially for Pwn2Own
- First, write a simple sync provider:

- Cloud Filter API documentation
- Windows CloudMirror sample
- James Forshaw, cldflt CVE PoC

CfRegisterSyncRoot function (cfapi.h)

Article • 04/04/2023

[Feedback](#)

Performs a one time sync root registration, allowing a sync provider to claim an entire directory tree structure, rooted at *SyncRootPath*, as their own to manage.

Syntax

C++

[Copy](#)

```
HRESULT CfRegisterSyncRoot(  
    [in] LPCWSTR          SyncRootPath,  
    [in] const CF_SYNC_REGISTRATION *Registration,  
    [in] const CF_SYNC_POLICIES *Policies,  
    [in] CF_REGISTER_FLAGS RegisterFlags  
);
```

Fuzzing – Writing the sync provider harness

- Add a few files and directories placeholders (*CfCreatePlaceholders*)
- Implement basic support for callbacks (with random errors and mutations)

```
CF_CALLBACK_REGISTRATION Callback[14] = {  
    CF_CALLBACK_TYPE_FETCH_DATA, FakeCallbackFetchData,  
    CF_CALLBACK_TYPE_VALIDATE_DATA, FakeCallbackValidateData,  
    CF_CALLBACK_TYPE_CANCEL_FETCH_DATA, FakeCallbackCancelData,  
    CF_CALLBACK_TYPE_FETCH_PLACEHOLDERS, FakeCallbackFetchPlaceholder,  
    CF_CALLBACK_TYPE_CANCEL_FETCH_PLACEHOLDERS, FakeCallbackCancelFetchPlaceholder,  
    CF_CALLBACK_TYPE_NOTIFY_FILE_OPEN_COMPLETION, FakeCallbackOpenCompletion,  
    CF_CALLBACK_TYPE_NOTIFY_FILE_CLOSE_COMPLETION, FakeCallbackCloseCompletion,  
    CF_CALLBACK_TYPE_NOTIFY_DEHYDRATE, FakeCallbackDehydrate,  
    CF_CALLBACK_TYPE_NOTIFY_DEHYDRATE_COMPLETION, FakeCallbackDehydrateCompletion,  
    CF_CALLBACK_TYPE_NOTIFY_DELETE, FakeCallbackDelete,  
    CF_CALLBACK_TYPE_NOTIFY_DELETE_COMPLETION, FakeCallbackDeleteCompletion,  
    CF_CALLBACK_TYPE_NOTIFY_RENAME, FakeCallbackRename,  
    CF_CALLBACK_TYPE_NOTIFY_RENAME_COMPLETION, FakeCallbackRenameCompletion,  
    CF_CALLBACK_TYPE_NONE, NULL  
};
```

```
 HsmFltPreACQUIRE_FOR_SECTION_SYNCHRONIZATION  
 HsmFltPreCLEANUP  
 HsmFltPreCREATE  
 HsmFltPreDIRECTORY_CONTROL  
 HsmFltPreFILE_SYSTEM_CONTROL  
 HsmFltPreLOCK_CONTROL  
 HsmFltPreMDL_READ  
 HsmFltPreNETWORK_QUERY_OPEN  
 HsmFltPrePREPARE_MDL_WRITE  
 HsmFltPreQUERY_ALLOCATED_RANGES  
 HsmFltPreQUERY_OPEN  
 HsmFltPreREAD  
 HsmFltPreSET_INFORMATION  
 HsmFltPreWRITE
```

User-Mode callbacks called in response to kernel-mode callbacks of cldflt

■ Write a client process to execute file system operations

- Open
- Read
- Write
- Delete
- Rename
- List
- ...

```
AReadFile(base, L"\\PlacemeDir\\TEST2");  
AListDirectory(base);  
AReadFile(base, L"\\Placeme");  
AMoveFile(base, L"\\Placeme", L"\\Placeme2");  
AWriteFile(base, L"\\Placeme");  
AReadFile(base, L"\\Placeme2");  
ADeleteRawFile(base, L"\\Placeme2");
```

Generated operations

- **Attach a kernel debugger or configure kernel crash dump storage**
- **Configure Special Pool with verifier.exe to improve memory corruption detection**
 - Special Pool on ntoskrnl.exe, fltMgr.sys and cldflt.sys
 - Crash on Use-After-Free and Out-Of-Bounds access

■ Got an interesting crash after a minute

```
nt!IoCancelIrp+0x2b:  
ffffff802`713919cb c6434401          mov     byte ptr [rbx+44h],1 // 0x5c003a0043003244=??
```

```
PROCESS_NAME: explorer.exe
```

```
STACK_TEXT:  
nt!IoCancelIrp+0x2b  
FLTMGR!FltCancelIo+0x20  
cldflt!CldiStreamStartCountdownTimer+0x143  
cldflt!CldiStreamRestartCountdownTimer+0x66  
cldflt!CldStreamQueryProgress+0x141a  
cldflt!CldiPortProcessQueryProgress+0x2b7  
cldflt!CldiPortProcessFilterControl+0x4d  
cldflt!CldiPortNotifyMessage+0x824  
FLTMGR!FltpFilterMessage+0xda  
...  
nt!NtDeviceIoControlFile+0x56  
nt!KiSystemServiceCopyEnd+0x28  
ntdll!NtDeviceIoControlFile+0x14  
FLTLib!FilterpDeviceIoControl+0x136  
FLTLib!FilterSendMessage+0x31  
cldapi!CfQueryProgress+0x3e7
```

*The IRP object pointer
(IoCancelIrp) seems to be
corrupted with a wide string.*

*For curious reader: Special Pool
did not catch the UAF because
of the minute delay.*

- **Each file operation is associated with a I/O request packet (IRP) in kernel**
- **The Filter Manager creates an IRP control object which saves the IRP and the callback data (*FLT_CALLBACK_DATA*) for each filesystem operation**

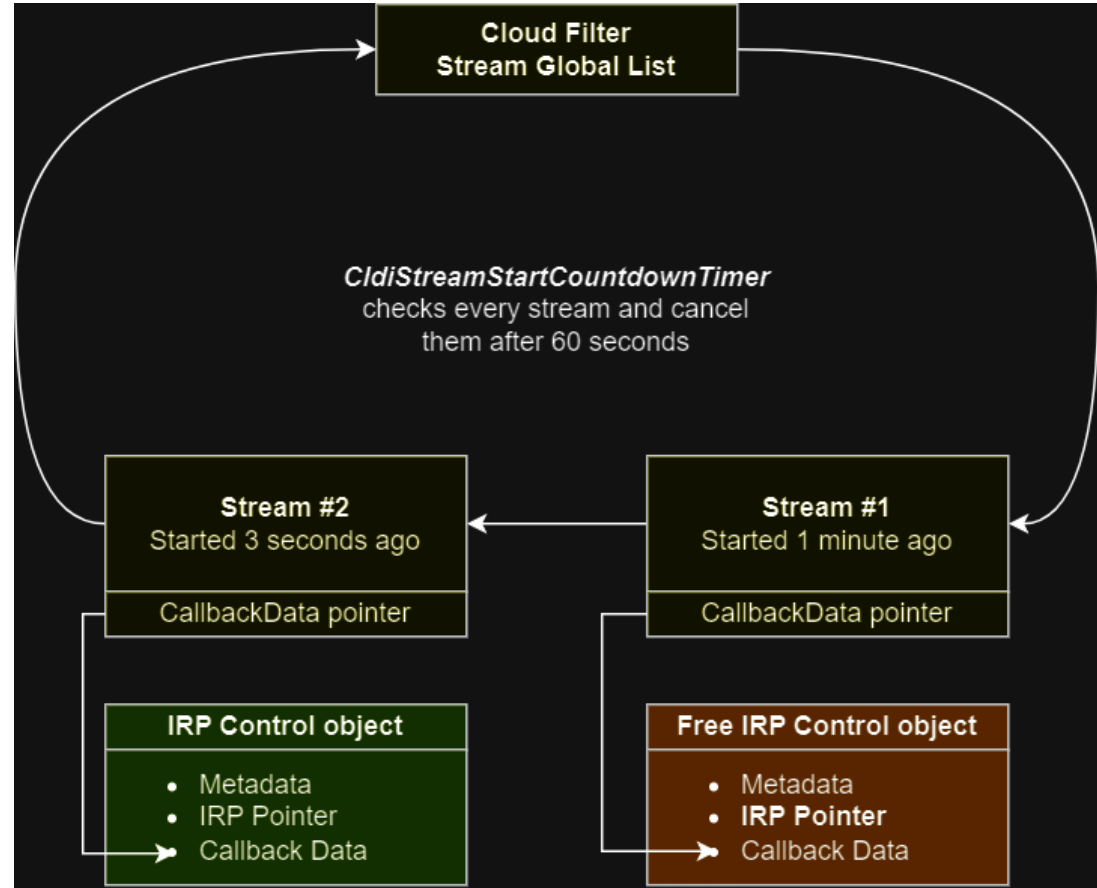
Vulnerability in Cloud Filter (2)

- The Cloud Filter stores this `FLT_CALLBACK_DATA` on pending I/O (ex: user-mode callback not answering) in a global list.
- After a minute, if the request is still pending in the global list, the IRP is cancelled using *IoCancelIrp*.
- This prevents a sync provider to block file I/O indefinitely.

Vulnerability in Cloud Filter (3)

■ Vulnerability

- The Cloud Filter Global List **is not cleared** properly if the sync provider die **but** the IRP control object **is freed**.



Use-After-Free Vulnerability

- **The vulnerability is an Use-After-Free of the IRP Control Object allocated by FltMgr.sys in *FltpAllocateIrpCtrlInternal* on the NonPagedNx pool.**
- **To trigger the vulnerability, three processes are required:**
 - Sync provider never answering the file deletion (no *ACK_DELETE*)
 - Client process deleting a placeholder file
 - A scheduler starting both processes and killing them
 - Waiting 1 minute and calling *CfQueryProgress* to trigger the timer check

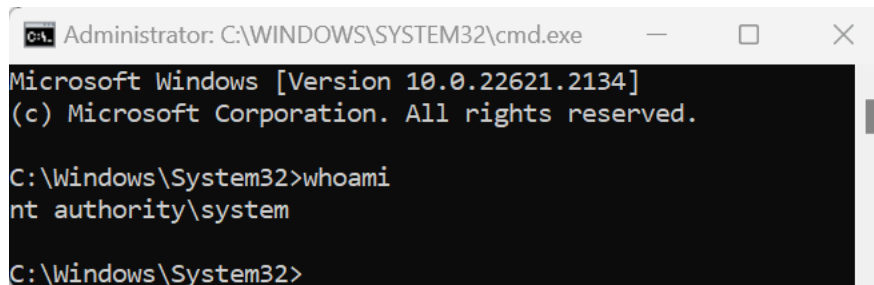
- **If the UAF allocation content is controlled:**

- Arbitrary IRP pointer passed to *IoCancelIrp*
 - **Arbitrary function call primitive**

```
BOOLEAN IoCancelIrp(PIRP Irp)
{ // Simplified
  KIRQL Irql = KeAcquireQueuedSpinLock(LockQueueIoCancelLock);
  Irp->Cancel = 1; // Arbitrary write byte with value 1
  PVOID CancelRoutinePtr = Irp->CancelRoutine;
  if ( CancelRoutinePtr )
  {
    CancelRoutinePtr(Irp->Tail.Overlay.CurrentStackLocation->DeviceObject, Irp); // Arbitrary function call
    return 1;
  }
}
```

Windows kernel exploitation 101

- **Goal:** Elevate our privileges to SYSTEM and run a command prompt
- **Data only technique:** Corrupt the current process TOKEN privileges
- **Require:** Arbitrary write and information leak of the TOKEN address



```
Administrator: C:\WINDOWS\SYSTEM32\cmd.exe
Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>whoami
nt authority\system

C:\Windows\System32>
```

- **No infoleak required:**
- **By design Windows API** *NtQuerySystemInformation* with *SystemExtendedHandleInformation* **discloses the TOKEN address.**
 - Note: Driver base address may be disclosed by *SystemModuleInformation* class.
 - It does not work in sandboxes.

■ Supervisor Mode Execution Protection (SMEP) and NX:

- Enabled, function pointer must point to executable kernel memory

■ Supervisor Mode Access Protection (SMAP):

- Not enabled in most situations, possible to forge fake objects in user-mode memory

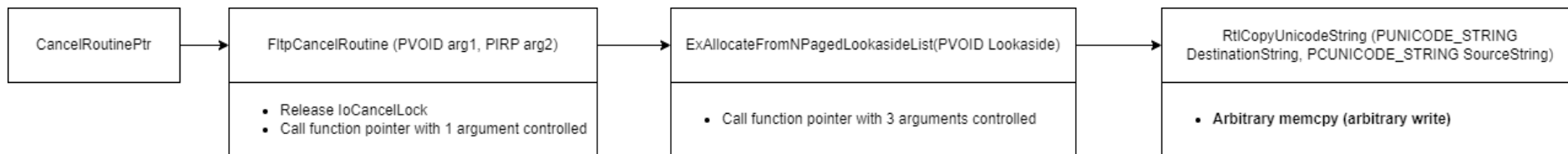
■ Reuse allocation content:

- Spray *NonPagedNx* using Named Pipe data entries (size 0x588)

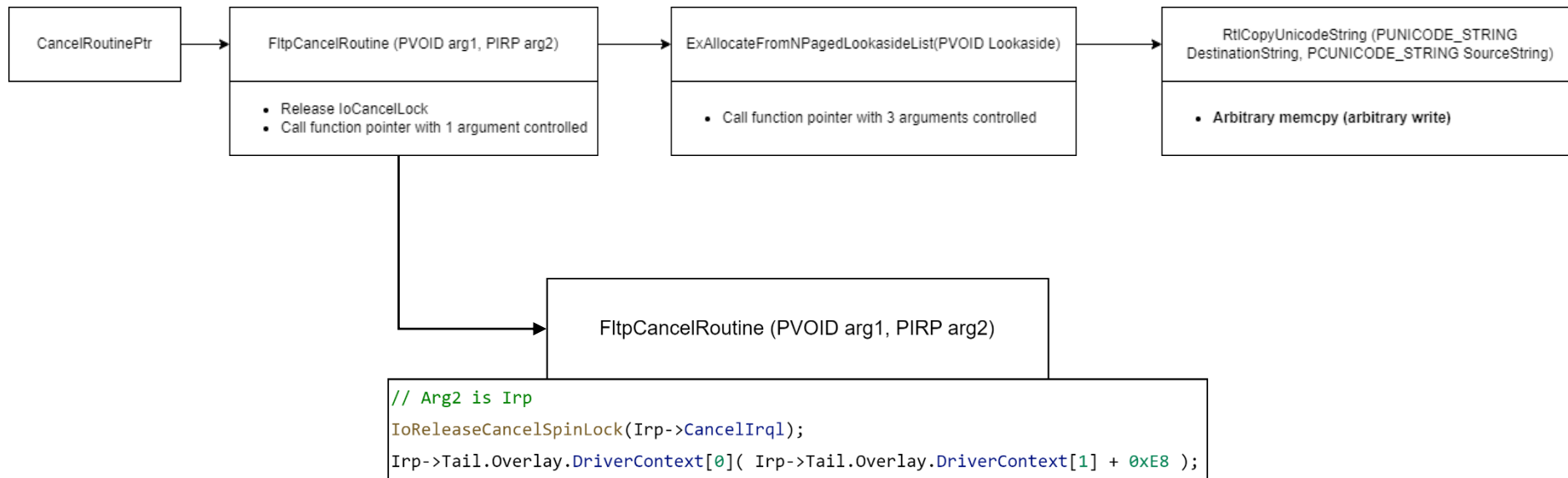
- **Control Flow Guard (CFG) validates indirect branch targets**
- **Arbitrary function call primitive is limited to valid targets**
 - `_guard_dispatch_icall` is called to check against a bitmap
- **Instead of ROP gadgets, jump on function gadgets**
 - First 2 function parameters are controlled
 - Chain allowed functions to control all parameters and achieve write primitive

Windows kernel CFG Chain

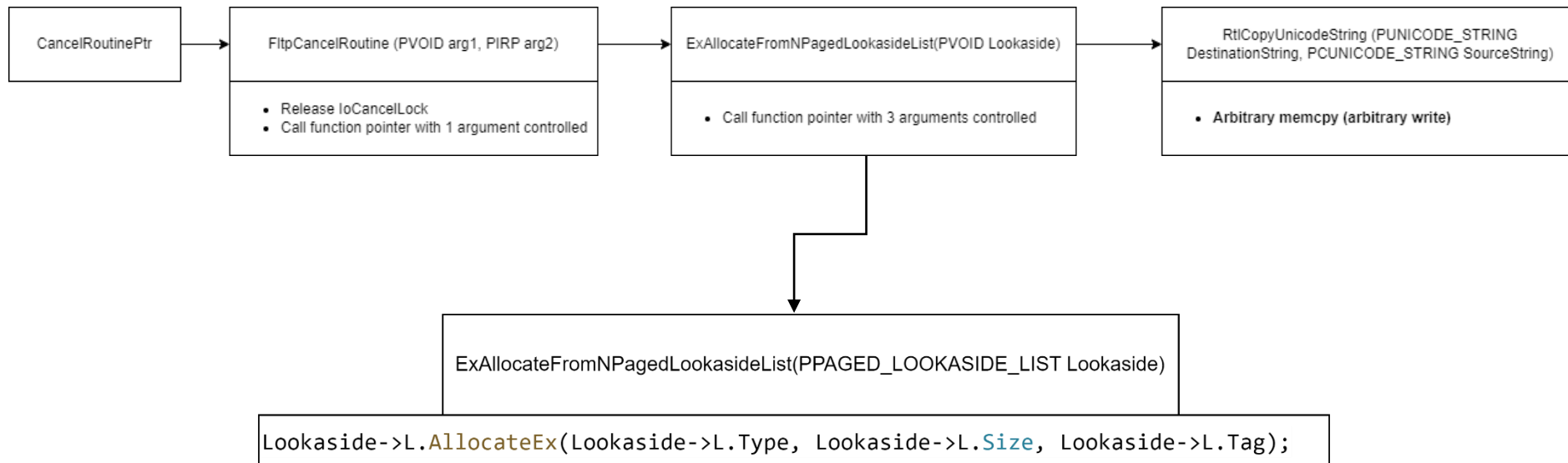
- Assume, the IRP pointer in the UAF object is fully controlled and points to user mode memory
- Chain function gadget:



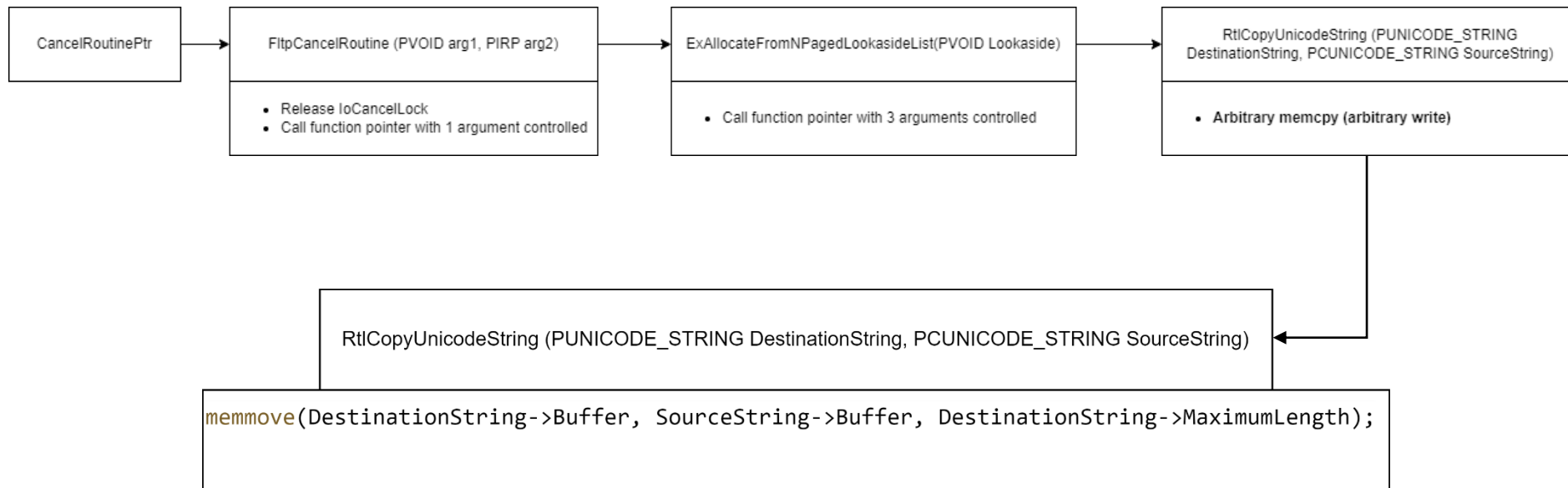
Windows kernel CFG Chain (2)



Windows kernel CFG Chain (3)



Windows kernel CFG Chain (4)



■ **Arbitrary write achieved !**

■ Exploit steps:

- Prepare fake objects in user mode: *IRP*, *PAGED_LOOKASIDE_LIST*, *UNICODE_STRING*
- Trigger the IRP control object free by killing the processes
- Spray Named Pipe data entry to reuse the allocation (fake IRP points to UM)
- Wait one minute and trigger UAF vulnerability
- Verify token privileges are all granted thanks to the arbitrary write (gadgets chain)
- Spawn a SYSTEM shell (*SeDebugPrivilege* used to control *winlogon* process)

- **Not very reliable**
 - But on fresh boot: >90%
- **Exploit takes between 1 and 6 minutes**
 - Issue: IRP control object is not always free, a lookaside list is used for performance
 - The vulnerability is played multiple times (8) to fill the lookaside list
- **Perfect for Pwn2Own** (3 attempts of 10 minutes with reboot)
- **Worked on second try at P2O !** (After a BSOD in the first attempt)

Cloud Filter timeline

- **Manual review: 2 days**
- **Fuzzing: 1 day**
- **Root cause and reproduction: 3 days**
- **Exploit: 2 days**



Windows Kernel Mitigations

Will it work next year?

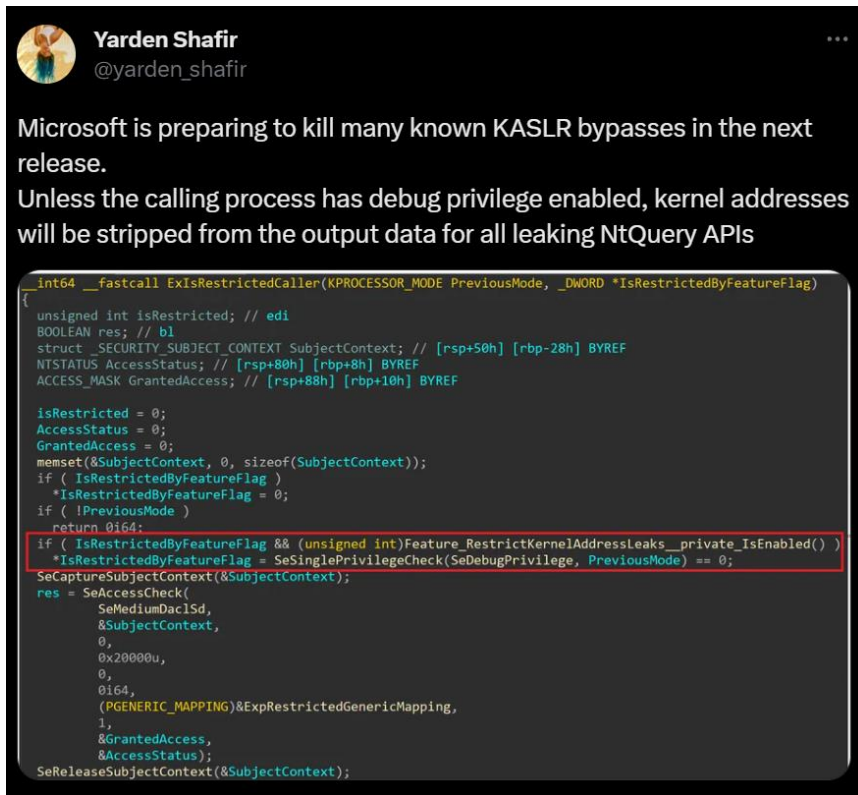
- **Microsoft is actively working on mitigations to kill bug classes**
 - Zeroing most variables and pool allocations prevents uninitialized memory vulnerabilities
 - CastGuard: mitigates C++ type confusions

- **Also, Microsoft mitigates a few powerful exploit primitives:**
 - Thread *PreviousMode* overwrite
 - KASLR by design infoleak

KASLR infoleak future removal

- Thanks to Yarden and the security community, tracking changes on Insider builds:

*This mitigation would break the previous exploit.
The bug would then require a powerful infoleak of kernel driver bases to build the function gadgets chain.*



Yarden Shafir
@yarden_shafir

Microsoft is preparing to kill many known KASLR bypasses in the next release.
Unless the calling process has debug privilege enabled, kernel addresses will be stripped from the output data for all leaking NtQuery APIs

```
int64 __fastcall ExIsRestrictedCaller(KPROCESSOR_MODE PreviousMode, _DWORD *IsRestrictedByFeatureFlag)
{
    unsigned int isRestricted; // edi
    BOOLEAN res; // bl
    struct _SECURITY_SUBJECT_CONTEXT SubjectContext; // [rsp+50h] [rbp-28h] BYREF
    NTSTATUS AccessStatus; // [rsp+80h] [rbp+8h] BYREF
    ACCESS_MASK GrantedAccess; // [rsp+88h] [rbp+10h] BYREF

    isRestricted = 0;
    AccessStatus = 0;
    GrantedAccess = 0;
    memset(&SubjectContext, 0, sizeof(SubjectContext));
    if ( IsRestrictedByFeatureFlag )
        *IsRestrictedByFeatureFlag = 0;
    if ( !PreviousMode )
        return 0i64;
    if ( IsRestrictedByFeatureFlag && (unsigned int)Feature_RestrictKernelAddressLeaks_private_IsEnabled() )
        *IsRestrictedByFeatureFlag = SeSinglePrivilegeCheck(SeDebugPrivilege, PreviousMode) == 0;
    SeCaptureSubjectContext(&SubjectContext);
    res = SeAccessCheck(
        SeMediumDaclSd,
        &SubjectContext,
        0,
        0x20000u,
        0,
        0i64,
        (PGENERIC_MAPPING)&ExpRestrictedGenericMapping,
        1,
        &GrantedAccess,
        &AccessStatus);
    SeReleaseSubjectContext(&SubjectContext);
}
```

■ SMAP:

- Enabled in functions (mostly interrupts) when the kernel is sure to never access user-mode memory
- The UAF access runs at IRQL DISPATCH_LEVEL which is not supposed to access user-mode memory, but SMAP is not enabled and allow for easier exploit

■ CFG:

- Current kernel implementation quite limited, lack granularity and allow a lot of function targets (while user-mode has a more fine grained solution: XFG)

Both mitigations are hard or impossible to expand due to compatibility issues with 3rd party drivers.

■ **Control Flow Enforcement Technology (CET):**

- Protect control-flow hijacking using a shadow stack and control transfer instructions
- Not designed to prevent function gadgets chain

■ **Hypervisor Protected Code Integrity (HVCI):**

- Protect kernel integrity, prevent executing custom code in kernel
- Not designed to detect data only attack on the token

Will it work next year? (2)

- **Microsoft seems to focus on high impact scenarios**
 - Restricting attack surface of browser and application sandboxes
 - Reviewing and fuzzing remote services, hypervisor and kernel (ntoskrnl)
- **Local privilege escalation from unprivileged user**
 - Probably not high priority
 - KASLR infoleak hardening proves some efforts
- **Opinion: LPE will still be feasible with 1 kernel bug but it will require more work (or better bugs)**



MSKSSRV

Looking for another bug to exploit

- **Objective: Find a stable bug to chain with VirtualBox exploit**
- **Reviewed random drivers for a quick win**
 - Pick a random driver in *System32\drivers* in IDA
 - Look for simple IOCTL issues
- **Found a couple of bugs in non-default drivers**
 - Optional features
 - Impossible to load without admin access

MSKSSRV – Introduction

- **Part of Microsoft Streaming Service component**
- **Allow two processes to share content using Tx/Rx streams**
 - IOCTL interface
 - Streams are basically shared memory (*Section* object or UM mapping)
- **Driver automatically loaded on demand when opened**
 - No admin access required

```
DEFINE_GUIDSTRUCT("3C0D501A-140B-11D1-B40F-00A0C9223196", KSNAMEServer);  
#define KSNAMEServer DEFINE_GUIDNAMED(KSNAMEServer)  
  
HANDLE DeviceHandle;  
KsOpenDefaultDevice(KSNAMEServer, GENERIC_READ | GENERIC_WRITE, &DeviceHandle);
```

■ Found this function reachable with user-mode inputs:

```
1 __int64 __fastcall FsAllocAndLockMdl(void *InputField, ULONG InputField2, _MDL **pMdl)
2 {
3     unsigned int status; // edi
4     _MDL *Mdl; // rax
5     _MDL *outMdl; // rbx
6
7     status = 0;
8     if ( InputField && InputField2 && pMdl )
9     {
10         Mdl = IoAllocateMdl(InputField, InputField2, 0, 0, 0i64);
11         outMdl = Mdl;
12         if ( Mdl )
13         {
14             MmProbeAndLockPages(Mdl, 0, IoWriteAccess);
15             *pMdl = outMdl;
16         }
17         else
18         {
19             return 0xC000009A;
20         }
21     }
22     else
23     {
24         return 0xC000000D;
25     }
26     return status;
27 }
```

MSKSSRV – Quick win (2)

■ MmProbeAndLockPages invalid access mode

```
void MmProbeAndLockPages(  
    [in, out] PMDL      MemoryDescriptorList,  
    [in]      KPROCESSOR_MODE AccessMode,  
    [in]      LOCK_OPERATION Operation  
);
```

[in] AccessMode

The access mode in which to probe the arguments, either **KernelMode** (0) or **UserMode** (1).

■ AccessMode parameter sets to **KernelMode** access mode

```
10  Mdl = IoAllocateMdl(InputField, InputField2, 0, 0, 0i64);  
11  outMdl = Mdl;  
12  if ( Mdl )  
13  {  
14      MmProbeAndLockPages(Mdl, 0, IoWriteAccess);
```

- **Windows Kernel must validate pointers coming from user-mode**
 - Using probe functions, the pointers and size are validated
 - On Windows, the address is validated to be less than *0x7FFFFFFFF0000*
 - Example API: *ProbeForRead* / *ProbeForWrite*

```
// MmProbeAndLockPages calls MiProbeAndLockPrepare which validates the address if AccessMode is set to UserMode
if ( AccessMode == UserMode )
{
    if ( !MdlLength
        || (MdlEndAddress = MdlLength + MdlVirtualAddress - 1, MdlEndAddress < MdlVirtualAddress)
        || MdlEndAddress > 0x7FFFFFFFF0000 )
    {
        return 0xC0000005i64;
    }
}
```

- **Driver IOCTL accepts kernel mode pointer in MDL creation**

- IOCTL 0x2F0408 – *FSRendezvousServer::PublishTx*

- **Memory Descriptor List (MDL)**

- MDL is a kernel object describing one or more virtual memory ranges
 - MDL stores the physical addresses corresponding to the virtual ranges
 - MDL are used for I/O operations and DMA to map and lock memory

- **Driver IOCTL accepts kernel mode pointer in MDL creation**
 - IOCTL 0x2F0408 – *FSRendezvousServer::PublishTx*

- **Resulting MDL can be mapped to user-mode**
 - IOCTL 0x2F0410 – *FSRendezvousServer::ConsumeTx*
 - Arbitrary kernel virtual memory may be mapped to user-mode with read and write access

- **Arbitrary kernel read and write achieved !**

MSKSSRV – Exploit

- Same goal: corrupt the Token privileges
- Steps:
 - Again, leverage by design KASLR infoleak to disclose Token address
 - Map the kernel page containing the Token to user-mode using the vulnerability
 - Overwrite the privileges bitfield to gain *SeDebugPrivilege*
 - Spawn a SYSTEM shell

```
uint8_t* UserModeMapping = MapToUserMode(GetTokenAddress());  
memset(UserModeMapping + OFFSET_OF_TOKEN_PRIVILEGES, 0xFF, 0x10);  
HANDLE winlogon = OpenWinlogonProcess();  
SpawnProcessWithParentHandle(L"C:\\Windows\\System32\\cmd.exe", winlogon);
```

- **100% stable bug**
 - Missing probe are powerful bugs
 - Especially this one since it gives read and write access
- **Exploit takes less than 1 second**
- **Timeline: 1 day for manual review and exploit**
- **Worked on second try at P2O !** (Microsoft Defender blocked the first attempt)



DEMO

Conclusion

- **Pwn2Own is a great opportunity to challenge yourself**
 - Very fun and ethical contest
 - Large attack surface
 - Lots of Windows kernel exploitation documentation available online
 - Vulnerabilities have been fixed in June: CVE-2023-29360 & CVE-2023-29361
 - Try it!
- **Windows Kernel Security is slowly improving but still attackable with limited means**



<https://www.linkedin.com/company/synacktiv>



<https://twitter.com/synacktiv>



<https://synacktiv.com>

In the top-left corner, there are several overlapping red parallelograms of various sizes and orientations.

THANK YOU!

