



Discovering and exploiting
a kernel pool overflow on
modern Windows 10

Sthack 2021

~:\$ whoami



- **Fabien Perigaud**
- **@0xf4b on Twitter**

- **Working for Synacktiv**
 - Offensive security
 - 90 ninjas: pentest, reverse engineering, development
 - We are hiring!

- **Reverse engineering team technical leader**
 - 30 reversers
 - Reverse, vulnerability research and exploitation, low level dev

Agenda



- **This short introduction**
- **Pwn2Own Vancouver 2021**
- **Windows 10 Kernel Attack Surface**
- **Vulnerability Discovery**
- **Exploitation: “expecting shell root”**
- **Results and Conclusion**

Pwn2Own Vancouver 2021





- **“Hacking contest” by ZDI**
 - Prove exploitation of devices or software in widespread use
- **Usually takes place twice a year**
 - Vancouver: virtualization, browsers, OS, Tesla, ...
 - Tokyo: smartphones, “smart devices”, routers, printers, ...
 - Miami 2020: ICS/SCADA
- **Pwn all the things!**
 - Payload should prove arbitrary code execution
 - Remote/elevated shell, blinking leds on a router, image display on a printer...

Pwn2Own - Rules



- **Each category has its own rules...**
 - ... but rules can be adapted if kindly asked :)
 - For example, enabling a non-default service...
 - ... as long as this configuration matches reality!
- **5 minutes time slot, up to 3 attempts**
 - Exploit can be time consuming (race condition, huge allocations, ...)
 - Stability can be perfectible :)

Pwn2Own – Rules (2)



- **For each target, only the first pwn is considered a WIN**
 - In practice, every successful pwn is a WIN...
 - ... unless a vulnerability collision occurs...
 - ... between contestants, or **with the vendor!**

- **Order of the contestants is drawn at random**
 - Small papers in a hat :)
 - In the end, Synacktiv contestants usually get the last slot

Pwn2Own – Rewards



- **Each pwned target is rewarded, depending on the estimated difficulty**
 - Cash prize (way lower than the 0-day market, but still interesting :))
 - Master of Pwn points → additional cash prize for the “Master of Pwn”
- **Add-on bonus**
 - For some targets
 - LPE, sandbox escape, ...
 - Usually a few more points and dollars

Pwn2Own Vancouver 2021



■ **Targets**

- Desktop browsers (Chrome, Safari, Firefox, Edge)
- Enterprise applications (Office, Reader, Zoom, Teams)
- Server (RDP, Exchange, SharePoint)
- Automotive (Tesla Model 3)
- Local Privilege Escalation (Windows, Ubuntu)

■ **LPEs: “easier” targets!**



- **Focus on the Windows 10 LPE**
- **Interesting execution context**
 - Unprivileged user...
 - ... but no sandboxing!
 - Medium integrity level
- **Vulnerability must be in the kernel**

Windows 10 Kernel Attack Surface





■ Ntoskrnl

- Windows kernel image
- Interrupts, memory management, kernel objects (processes, threads, files, registry, ...), syscalls and more
- Very interesting target, might be reachable from the hardest sandbox level
- Drawback: huge focus from security researchers

■ Win32k

- Huge graphic subsystem, own syscall table
- Old code base, many vulnerabilities
- Also reachable from some sandbox contexts
- Drawback: also a huge focus from security researchers



■ Drivers

- PE loaded in Kernel-land
- “.sys” file on the disk
- Usually linked to a service

■ Userland access

- Driver create a Device object “XXX”
- Userland opens the device through “\\?\GLOBALROOT\Device\XXX”

Kernel Attack Surface - Devices



- **List all the devices present on a default Windows 10**
 - WinDBG to the rescue!
 - !object \Device → 159 devices

```
0: kd> !object \Device
```

```
Object: fffff988952031060 Type: (ffffaf07a6c71900) Directory
```

```
ObjectHeader: fffff988952031030 (new version)
```

```
HandleCount: 2 PointerCount: 65717
```

```
Directory Object: fffff988952041e60 Name: Device
```

Hash	Address	Type	Name
----	-----	----	----
00	ffffaf07adaafcb0	Device	00000030
	ffffaf07ad94a050	Device	NDMP2
	ffffaf07a82c8360	Device	NTPNP_PCI0002
01	ffffaf07ad9d2050	Device	NDMP3

```
...
```



■ Dumb/dirty way to check access rights

- Try to open device with R/W access
- Fast to write with Python ctypes

```
for dev in devices:  
    file_handle = windll.kernel32.CreateFileA("\\\\?\\GLOBALROOT\\Device\\  
%s" % dev, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING,  
FILE_ATTRIBUTE_NORMAL, 0)  
    if file_handle != INVALID_HANDLE_VALUE:  
        print("[*] OK %s" % dev)
```

Check devices access (2)

```
C:\Users\unpriv\Desktop>check_devices.py
```



```
...  
[*] OK Afd  
[*] OK ahcache  
[*] OK Beep  
[*] OK CNG  
[*] OK gpuenergydrv  
[*] OK KsecDD  
[*] OK LanmanDatagramReceiver  
[*] OK Mailslot  
[*] OK Mup  
[*] OK NamedPipe  
[...]  
[*] OK Netbios  
[...]  
[*] OK Null  
[*] OK PEAuth  
[*] OK RdpBus  
[*] OK Spaceport  
[*] OK Tdx  
[*] OK USBFDO-0  
[*] OK USBPDO-0  
[*] OK VBoxGuest  
[*] OK WindowsTrustedRT  
[*] OK WINDRVR6
```














Pick a victim



Afd	ahcache	Beep	CNG	gpuenergydrv
KsecDD	LanmanDatagramReceiver	Mailslot	Mup	NamedPipe
Netbios	Null	PEAuth	RdpBus	Spaceport
Tdx	USBFDO-0	USBPDO-0	VBoxGuest	
WinDRV6	WindowsTrustedRT			

















Pick a victim



 Ard	 ahcache	 Beep	 C:\G	gpuenergydrv
 KsecDD	LanmanDatagramReceiver	 Mailslot	Mup	 NamedPipe
 Netbios	 Null	PEAuth	 RdpBus	Spaceport
 Tdx	USBFDO-0	USBPDO-0	 VBoxGuest	
WinDRV6	WindowsTrustedRT			

Pick a victim



 Ard	 ahcache	 Beep	 CnG	gpuenergydrv
 KsecDD	 LanmanDatagramReceiver	 Mailslot	Mup	 NamedPipe
 Netbios	 Null	 PEAuth	 RdpBus	Spaceport
 Tdx	 USBFDO-0	 USBFDO-0	 VBoxGuest	
WinDRV6	WindowsTrustedRT			

Pick a victim



Arb	ahcache	Beep	CirG	gpuenergydrv
KsecDD	LanmanDatagramReceiver	Mailslot	Mup	NamedPipe
Netbios	Null	PEAuth	RdpBus	Spaceport
Tdx	USBPDO-0	USBPDO-0	VBoxGuest	
WinDRV6	WindowsTrustedRT			



Find the corresponding driver



■ WinDBG again

```
0: kd> dt nt!_DEVICE_OBJECT fffffaf07a83320a0
+0x000 Type           : 0n3
+0x002 Size           : 0xc58
+0x004 ReferenceCount : 0n0
+0x008 DriverObject   : 0xffffaf07`a8323cf0 _DRIVER_OBJECT
[...]
0: kd> !object 0xffffaf07`a8323cf0
Object: fffffaf07a8323cf0 Type: (ffffaf07a6cca380) Driver
ObjectHeader: fffffaf07a8323cc0 (new version)
HandleCount: 0 PointerCount: 5
Directory Object: fffff98895213220 Name: spaceport
```

Find the corresponding driver



■ Corresponding service in the registry

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\spaceport

Name	Type	Data
(Default)	REG_SZ	(value not set)
DisplayName	REG_SZ	@spaceport.inf,%Spaceport_ServiceDesc%;Storage...
ErrorControl	REG_DWORD	0x00000003 (3)
Group	REG_SZ	System Bus Extender
ImagePath	REG_EXPAND_SZ	System32\drivers\spaceport.sys
Owners	REG_MULTI_SZ	spaceport.inf
Start	REG_DWORD	0x00000000 (0)
Tag	REG_DWORD	0x00000008 (8)
Type	REG_DWORD	0x00000001 (1)

Vulnerability Discovery





- **Driver Object can have 28 defined Major Functions**
 - Open / Close
 - Read / Write
 - IOCTL
 - Etc.
- **Usually defined in the DriverEntry function**
 - IOCTLs are usually the first thing to look at ...
 - ... but others are also less analyzed!

Driver interaction – Major Functions



```
DriverObject->MajorFunction[0] = SpSuccess;
DriverObject->MajorFunction[2] = SpSuccess;
DriverObject->MajorFunction[3] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_READ
DriverObject->MajorFunction[4] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_WRITE
DriverObject->MajorFunction[9] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_FLUSH_BUFFERS
DriverObject->MajorFunction[13] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_FILE_SYSTEM_CONTROL
DriverObject->MajorFunction[14] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_DEVICE_CONTROL
DriverObject->MajorFunction[15] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_INTERNAL_DEVICE_CONTROL
DriverObject->MajorFunction[16] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_SHUTDOWN
DriverObject->MajorFunction[18] = SpSuccess;
DriverObject->MajorFunction[22] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_POWER
DriverObject->MajorFunction[23] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_SYSTEM_CONTROL
DriverObject->MajorFunction[27] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_PNP
*(&SpControlDispatchTable + 14) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlDeviceControl;
*(&SpControlDispatchTable + 15) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlScsi;
*(&SpControlDispatchTable + 16) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlShutdown;
*(&SpControlDispatchTable + 27) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlPnp;
```

Driver interaction – Major Functions



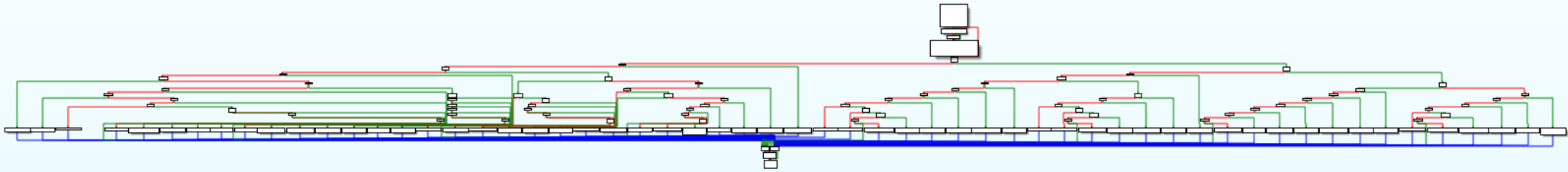
```
DriverObject->MajorFunction[0] = SpSuccess;
DriverObject->MajorFunction[2] = SpSuccess;
DriverObject->MajorFunction[3] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_READ
DriverObject->MajorFunction[4] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_WRITE
DriverObject->MajorFunction[9] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_FLUSH_BUFFERS
DriverObject->MajorFunction[13] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_FILE_SYSTEM_CONTROL
DriverObject->MajorFunction[14] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_DEVICE_CONTROL
DriverObject->MajorFunction[15] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_INTERNAL_DEVICE_CONTROL
DriverObject->MajorFunction[16] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_SHUTDOWN
DriverObject->MajorFunction[18] = SpSuccess;
DriverObject->MajorFunction[22] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_POWER
DriverObject->MajorFunction[23] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_SYSTEM_CONTROL
DriverObject->MajorFunction[27] = (PDRIVER_DISPATCH)SpDispatch;// IRP_MJ_PNP
*(&SpControlDispatchTable + 14) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlDeviceControl;
*(&SpControlDispatchTable + 15) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlScsi;
*(&SpControlDispatchTable + 16) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlShutdown;
*(&SpControlDispatchTable + 27) = (int (__stdcall * near *)(struct _DEVICE_OBJECT *, struct _IRP *))SpControlPnp;
```

SpControlDeviceControl

27



- **58 handled IOCTLs**



- **Manual review**

- Focus on “SpIoctl[Create|Set].*”
- Ignore when privileges are checked (“SpAccessCheck.*”)



- **Many checks are missing**
 - When the research was performed :)
- **Several vulnerabilities have been found!**
 - Constraints to be reachable
 - Might require creation of objects from a privileged context
 - Might be OK in real life, not for the contest
- **One really interesting vulnerability**



- No permission check
- Buffer overflow 101
 - Controlled memcpy size
 - Controlled content

```
v8->sp_list.Flink = &v7->sp_list;
v7->sp_list.Blink = &v8->sp_list;
p_sp_work->sp_list.Blink = &p_sp_work->sp_list;
p_sp_work->sp_list.Flink = &p_sp_work->sp_list;
p_sp_work->work_info->error_code = user_buffer->error_code;
p_sp_work->work_info->content_size = user_buffer->content_size;
memmove(
    (char *)p_sp_work->work_info + (unsigned int)p_sp_work->work_info->content_offset,
    (char *)user_buffer + (unsigned int)user_buffer->content_offset,
    (unsigned int)user_buffer->content_size);
KeSetEvent((PRKEVENT)&p_sp_work->set_event, 0, 0);
```

SploitSetControlWork



- No permission check
- Buffer overflow 101
 - Controlled memcpy size
 - Controlled content

```
v8->sp_list.Flink = &v7->sp_list;
v7->sp_list.Blink = &v8->sp_list;
p_sp_work->sp_list.Blink = &p_sp_work->sp_list;
p_sp_work->sp_list.Flink = &p_sp_work->sp_list;
p_sp_work->work_info->error_code = user_buffer->error_code;
p_sp_work->work_info->content_size = user_buffer->content_size;
memmove(
    (char *)p_sp_work->work_info + (unsigned int)p_sp_work->work_info->content_offset,
    (char *)user_buffer + (unsigned int)user_buffer->content_offset,
    (unsigned int)user_buffer->content_size);
KeSetEvent((PRKEVENT)&p_sp_work->set_event, 0, 0);
```

Dest:
pool (heap)
buffer

SploitSetControlWork



- No permission check
- Buffer overflow 101
 - Controlled memcpy size
 - Controlled content

```
v8->sp_list.Flink = &v7->sp_list;
v7->sp_list.Blink = &v8->sp_list;
p_sp_work->sp_list.Blink = &p_sp_work->sp_list;
p_sp_work->sp_list.Flink = &p_sp_work->sp_list;
p_sp_work->work_info->error_code = user_buffer->error_code;
p_sp_work->work_info->content_size = user_buffer->content_size;
memmove(
    (char *)p_sp_work->work_info + (unsigned int)p_sp_work->work_info->content_offset,
    (char *)user_buffer + (unsigned int)user_buffer->content_offset,
    (unsigned int)user_buffer->content_size);
KeSetEvent((PRKEVENT)&p_sp_work->set_event, 0, 0);
```

Source:
user controlled
buffer

SploitSetControlWork



- No permission check
- Buffer overflow 101
 - Controlled memcpy size
 - Controlled content

```
v8->sp_list.Flink = &v7->sp_list;
v7->sp_list.Blink = &v8->sp_list;
p_sp_work->sp_list.Blink = &p_sp_work->sp_list;
p_sp_work->sp_list.Flink = &p_sp_work->sp_list;
p_sp_work->work_info->error_code = user_buffer->error_code;
p_sp_work->work_info->content_size = user_buffer->content_size;
memmove(
    (char *)p_sp_work->work_info + (unsigned int)p_sp_work->work_info->content_offset,
    (char *)user_buffer + (unsigned int)user_buffer->content_offset,
    (unsigned int)user_buffer->content_size);
KeSetEvent(((PRKEVENT)&p_sp_work->set_event, 0, 0);
```

Size:
from user
controlled
buffer

SpioctlSetControlWork – Reaching the bug



- **SpioctlSetControlWork looks for a SP_WORK_INFO in a doubly-linked list (“LIST1”)**
 - Identified by a provided ID
- **“LIST1” is populated by another IOCTL: SpioctlGetControlWork**
 - Gets an entry from another doubly-linked list (“LIST2”)
 - Put it in “LIST1” and return its ID
- **“LIST2” is populated by SP_CONTROL_WORK::Run**
 - Reachable from several IOCTLs
 - SpioctlAttachSpaceRemote is a good candidate

Workflow



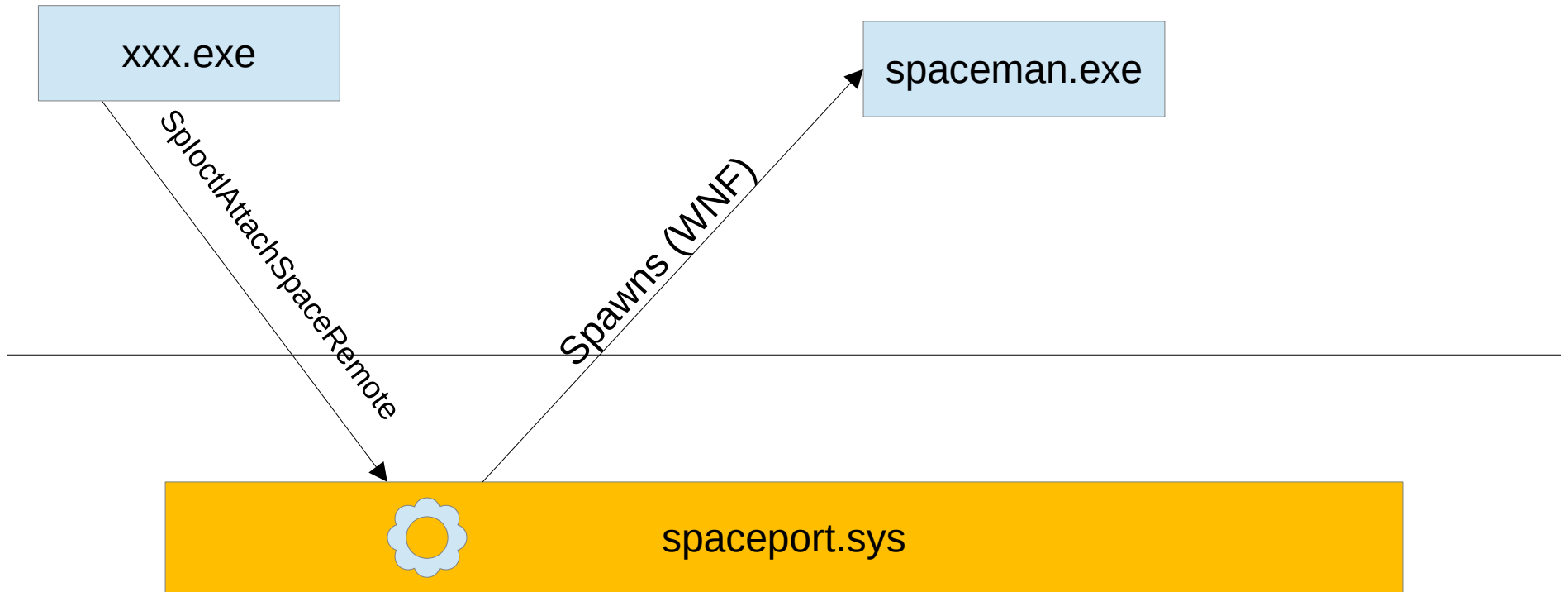
- **If we call the IOCTLS sequentially**
 - Stuck when calling SpioctlGetControlWork → we never get an ID
- **What's happening?**

Usual workflow

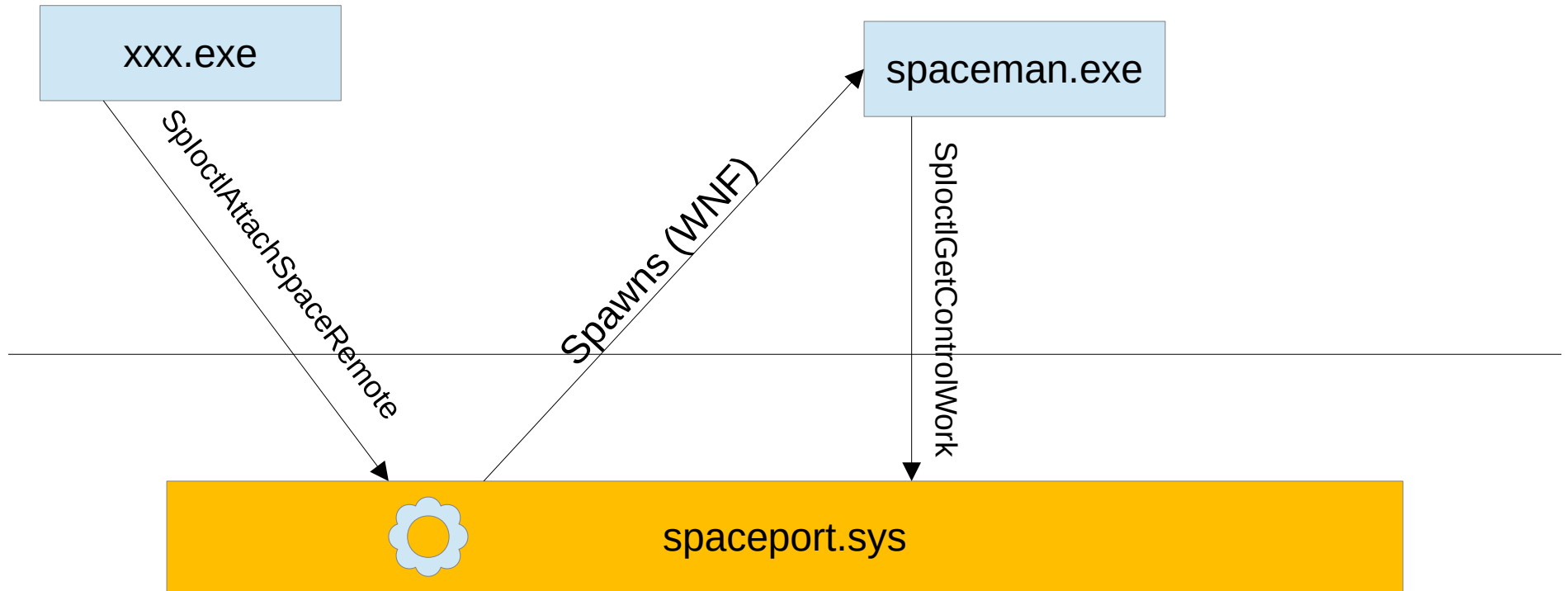
35



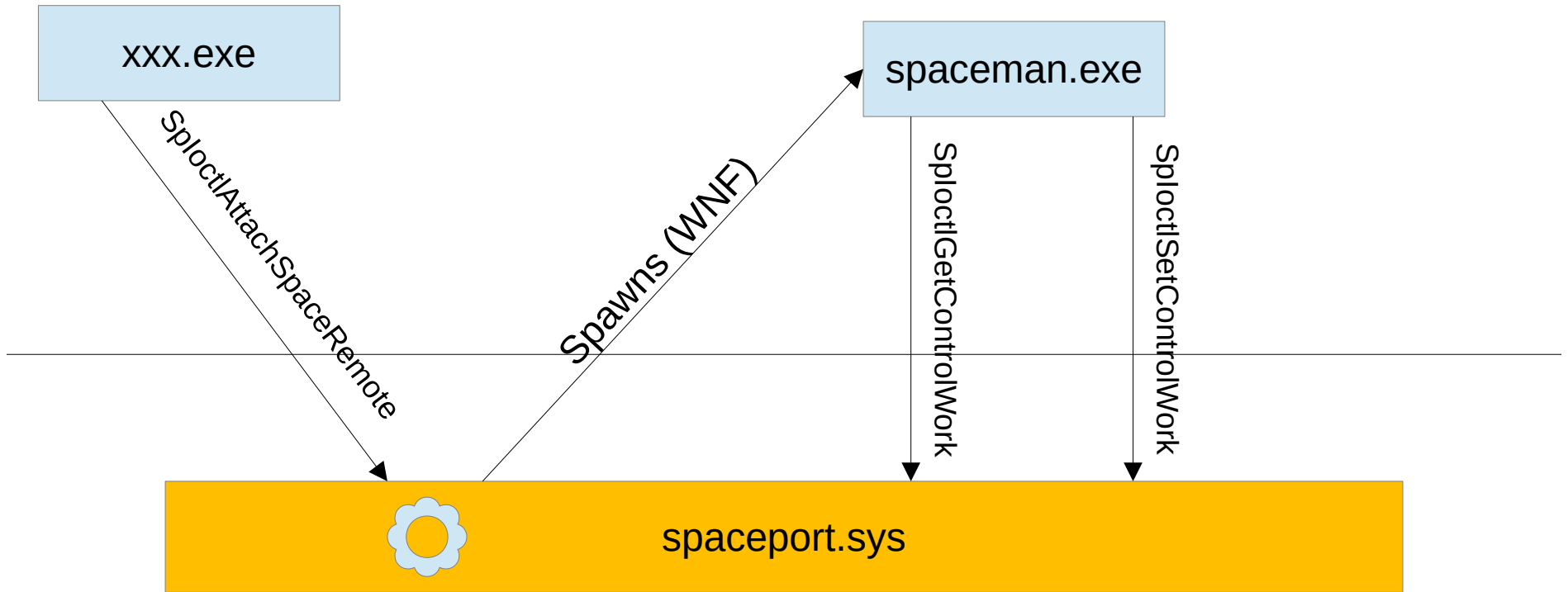
Usual workflow



Usual workflow



Usual workflow





- **If we call the IOCTLs sequentially, we get raced by spaceman.exe**
 - No more ID to be retrieved by SpIoctlGetControlWork
- **However...**
 - ... meet asynchronous DeviceIoControl!
 - Call SpIoctlGetControlWork before SpIoctlAttachSpaceRemote
 - List is empty, driver puts the request on hold
 - IRP is queued, and dequeued when SP_CONTROL_WORK::Run is executed
 - When spaceman.exe is executed, it has been raced by us :)

PoC



- **Thread #1 → call SpioctlGetControlWork**
 - Async, wait for result
- **Thread #2 → call SpioctlAttachRemoteSpace**
 - Blocked until someone issues a SpioctlSetControlWork
- **Thread #1 → ID retrieved, call SpioctlSetControlWork with bogus length**

PoC - BSOD



```
rax=ffffaf07b1584ed0 rbx=0000000000000000 rcx=ffffaf07f39a9112
rdx=fffffffffa81d190 rsi=0000000000000000 rdi=0000000000000000
rip=fffff80465dfa343 rsp=ffffc106ae2686d8 rbp=fffff80465e32048
  r8=0000000042424242 r9=ffffaf07a83321f0 r10=0000000000000000
r11=ffffaf07ee1c62a2 r12=0000000000000000 r13=0000000000000000
r14=0000000000000000 r15=0000000000000000
iopl=0          nv up ei ng nz na po nc
```

```
spaceport!memcpy+0x203:
fffff804`65dfa343 0f104411f0      movups  xmm0,xmmword ptr [rcx+rdx-10h]
ds:ffffaf07`ee1c6292=????????????????????????????????
```

```
ffffc106`ae2686d8 fffff804`65df6f19 : [...] : spaceport!memcpy+0x203
ffffc106`ae2686e0 fffff804`65e4c12b : [...] : spaceport!SP_CONTROL_WORK::Set+0xd9
ffffc106`ae268740 fffff804`65e41d7e : [...] : spaceport!SpIoctlSetControlWork+0x5b
ffffc106`ae268780 fffff804`65df3f50 : [...] : spaceport!SpControlDeviceControl+0x2ee
```

Exploitation: “expecting shell root”



Vulnerability primitive



■ Pool overflow

- The pool is the Windows Kernel heap

■ Target allocation

- SP_WORK_INFO allocation is made in the NonPagedPoolNx
- Size is 0x160 bytes
- Lies in the LFH (Low Fragmentation Heap)

■ Overflow constraints

- None :)
- We control content and size

Mitigations



■ **kASLR**

- Not a problem, Medium integrity level
- Various kernel APIs to get objects and modules addresses

■ **DEP / SMEP / CFG**

- Kernel code execution is hard
- Data-only exploitation ftw!

■ **SMAP?**

- Only in a few contexts, not in ours :)

Exploitation strategy



- **Data only**
- **We want to run an elevated cmd.exe**

- **Target: process Token!**
 - Swap token with a privileged one? (System)
 - Or enable powerful privileges!

- **Let's turn our pool overflow into something interesting!**

La French Tech – SSTIC 2020

46



SYNACKTIV
DIGITAL SECURITY

SSTIC 2020

Scoop the Windows 10 Pool!

05 Juin 2020

Paul Fariello (@paulfariello)
Corentin Bayet (@OnlyTheDuck)



■ **Aligned Chunk Confusion**

- New generic pool overflow exploitation method
- Abuses the CacheAligned bit in the POOL_HEADER
- Read their paper for details!

■ **Requirements**

- Shape the pool to control the chunk after the vulnerable one



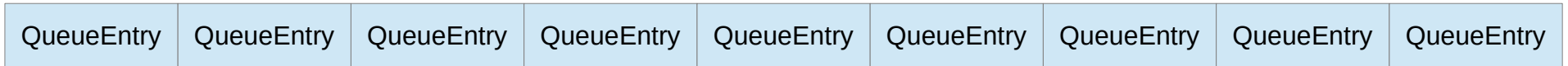
- **Spray a bunch of 0x160 bytes allocations in NonPagedPoolNx**
 - We can use pipe objects
 - PipeQueueEntry is in the NonPagedPoolNx, and we can control its size

QueueEntry	QueueEntry	QueueEntry	QueueEntry	QueueEntry	QueueEntry	QueueEntry	QueueEntry	QueueEntry
------------	------------	------------	------------	------------	------------	------------	------------	------------

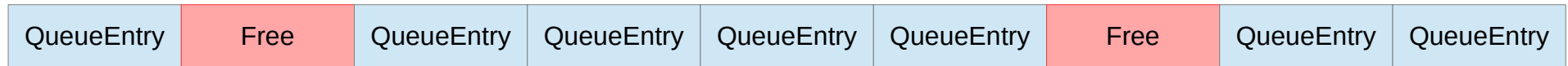
Pool massaging



- **Spray a bunch of 0x160 bytes allocations in NonPagedPoolNx**
 - We can use pipe objects
 - PipeQueueEntry is in the NonPagedPoolNx, and we can control its size



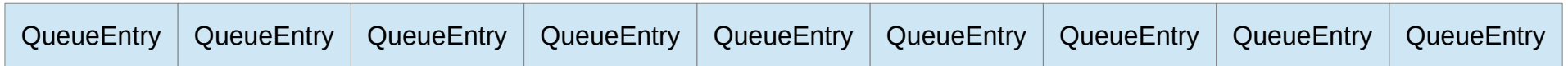
- **Free a few ones to create holes**



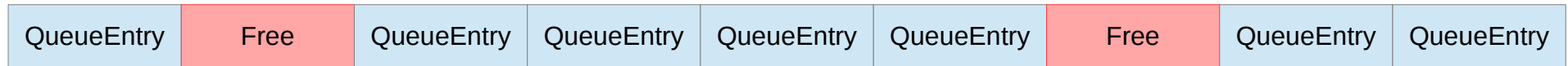
Pool massaging



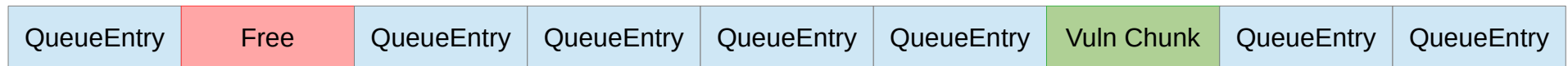
- **Spray a bunch of 0x160 bytes allocations in NonPagedPoolNx**
 - We can use pipe objects
 - PipeQueueEntry is in the NonPagedPoolNx, and we can control its size



- **Free a few ones to create holes**



- **Our vuln chunk should lie in one of these holes**

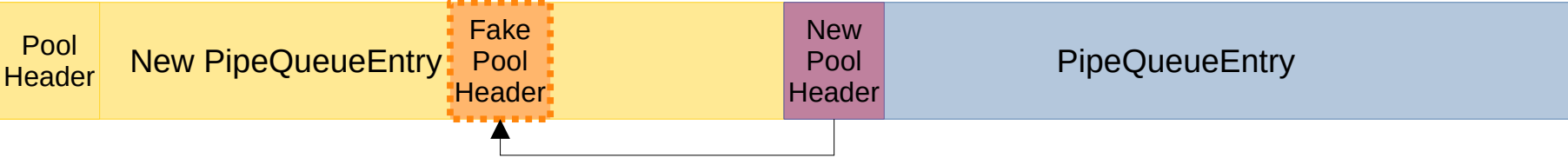
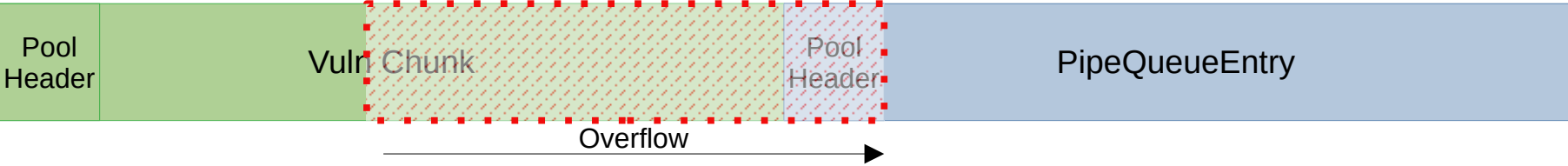
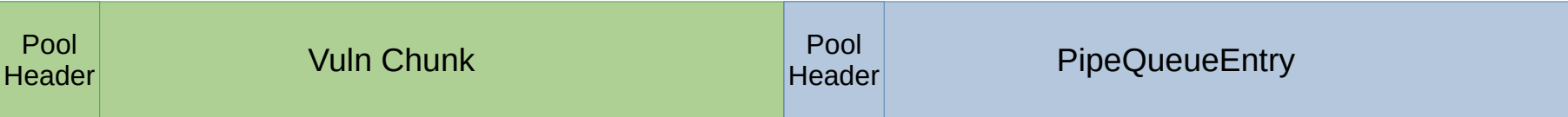


Creating an overlapping chunk



- **We can now trigger the overflow to change the next chunk (called victim) POOL_HEADER**
 - poolType set to 4 → CacheAligned
 - previousSize set to 0x100
- **When freeing the victim, the allocator will look for a second POOL_HEADER 0x100 bytes before the chunk**
 - This creates a fake chunk of 0x260 bytes (0x160+0x100)
- **Caveat: after exploitation, our vuln chunk is freed**
 - We reuse it with a controlled PipeQueueEntry!

Creating an overlapping chunk - Graphics

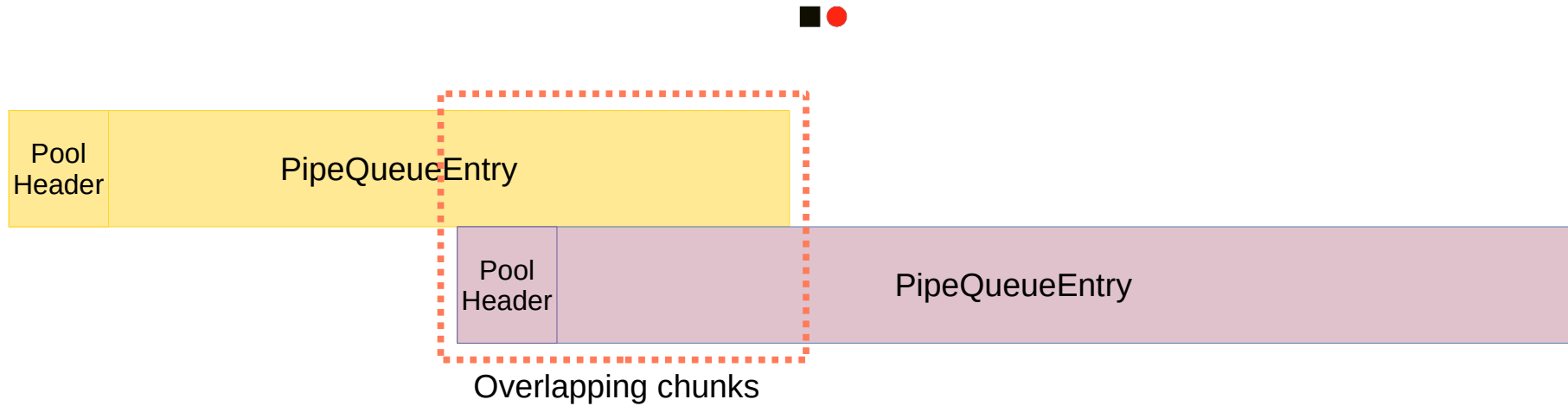


Creating an overlapping chunk – Next steps



- **We chose a size of 0x260 bytes for our fake chunk**
 - LFH is for allocations < 0x200
 - Bigger allocations are handled by the VS (variable size) allocator
 - Lookaside lists can be enabled for faster allocations!
- **When the victim chunk is freed**
 - Fake POOL_HEADER is read
 - Fake chunk is added to the 0x260 bytes lookaside list
- **We can now make a new allocation of 0x260 bytes to reuse the fake chunk!**

Creating an overlapping chunk – Result



- **Leak by reading the first pipe**
 - Gives the PipeQueueEntry structure content

PipeQueueEntry structure



```
struct PipeQueueEntry {  
    LIST_ENTRY list;  
    IRP *IRP;  
    uint64 t security;  
    int isDataInKernel;  
    int remaining_bytes;  
    int DataSize;  
    int field_2C;  
    char data[0];  
};
```

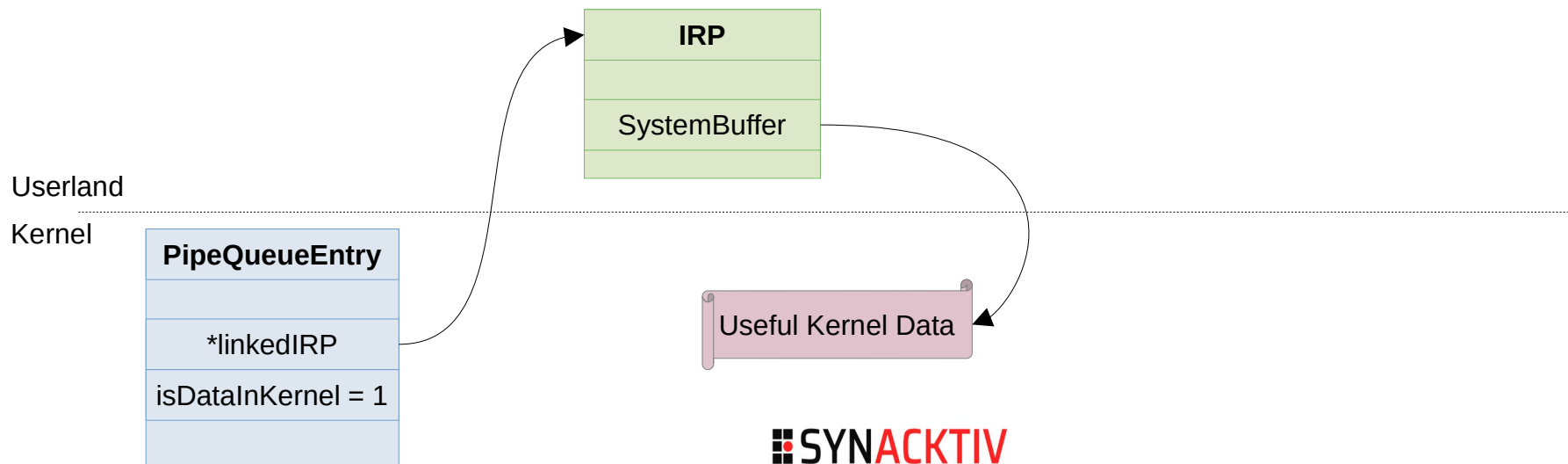
→ List of PipeQueueEntry structures

```
if (PipeQueueEntry->isDataInKernel == 1)  
    data_ptr = (PipeQueueEntry->linkedIRP->SystemBuffer);  
else  
    data_ptr = PipeQueueEntry->data;
```

Arbitrary Read Primitive



- Free the first PipeQueueEntry
- Reuse the chunk
 - We can overwrite the second PipeQueueEntry structure
 - Change the linkedIRP pointer to make it point to userland



Attacking the ProcessBilled pointer



■ POOL_HEADER has a ProcessBilled field

- Obfuscated pointer to an EPROCESS
- If PoolQuota flag is set, EPROCESS→QuotaBlockPtr→value is decremented when allocation is freed

■ Arbitrary decrement primitive

- Requires ability to forge a new obfuscated pointer
- `ProcessBilled == @EPROCESS ^ @chunk ^ ExpPoolQuotaCookie`

Finding ExpPoolQuotaCookie

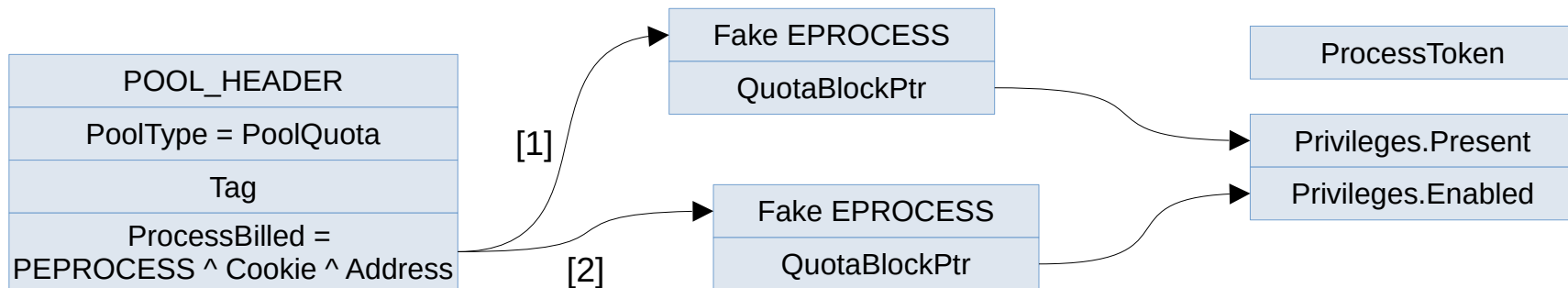


- **From Medium Integrity Level, kernel APIs can be used to get a kernel address from a handle**
 - NtQuerySystemInformation(SystemHandleInformation)
 - We have our EPROCESS address!
- **In the previous leak**
 - We got our PipeQueueEntry address through the doubly-linked list
 - We got its POOL_HEADER ProcessBilled
- **ExpPoolQuotaCookie can be computed**

Arbitrary decrement to privilege escalation



- A process Token contains its privileges
- Strategy: decrement “Enabled” and “Present” fields to enable SeDebugPrivilege
 - We build a fake EPROCESS which QuotaBlockPtr points to the target field
- 2 more reuses needed to change the ProcessBilled twice



SeDebugPrivileges

60



- **Allow debugging every process on the system**
- **Strategy**
 - Open winlogon.exe
 - Inject shellcode
 - Spawn a SYSTEM cmd.exe
- **Quick demo!**

Results and Conclusion



Pwn2Own results

62



- **Exploit worked at first attempt \o/**
- **Debriefing with ZDI**
 - Bug is unknown to ZDI \o/ \o/
- **Debriefing with Microsoft**
 - ... bug is already known to Microsoft ...
 - They proved it by showing the bug report
- **Partial Win :(**

Timeline



- ?? ?? ???? - Vulnerability reported to Microsoft by “vbty” (according to Microsoft advisory)
- 8 April 2021 – Vulnerability exploited during P2O
- 13 July 2021 – Vulnerability fixed by Microsoft (CVE-2021-33751)

Exploitability

The following table provides an [exploitability assessment](#) for this vulnerability at the time of original publication.

Publicly Disclosed	Exploited	Exploitability Assessment
No	No	Exploitation Less Likely

Fix



```
if ( user_buffer->content_size <= (unsigned int)work_info->content_size )
{
    Flink = (SP_WORK *)p_sp_work->sp_list.Flink;
    if ( (SP_WORK *)p_sp_work->sp_list.Flink->Blink != p_sp_work
        || (Blink = (SP_WORK *)p_sp_work->sp_list.Blink, (SP_WORK *)Blink->sp_list.Flink != p_sp_work) )
    {
        __fastfail(3u);
    }
    Blink->sp_list.Flink = &Flink->sp_list;
    Flink->sp_list.Blink = &Blink->sp_list;
    p_sp_work->sp_list.Blink = &p_sp_work->sp_list;
    p_sp_work->sp_list.Flink = &p_sp_work->sp_list;
    p_sp_work->work_info->error_code = user_buffer->error_code;
    p_sp_work->work_info->content_size = user_buffer->content_size;
    memmove(
        (char *)p_sp_work->work_info + (unsigned int)p_sp_work->work_info->content_offset,
        (char *)user_buffer + (unsigned int)user_buffer->content_offset,
        (unsigned int)user_buffer->content_size);
}
```


Free Oday?



```
if ( user_buffer->content_size <= (unsigned int)work_info->content_size )
{
    Flink = (SP_WORK *)p_sp_work->sp_list.Flink;
    if ( (SP_WORK *)p_sp_work->sp_list.Flink->Blink != p_sp_work
        || (Blink = (SP_WORK *)p_sp_work->sp_list.Blink, (SP_WORK *)Blink->sp_list.Flink != p_sp_work) )
    {
        __fastfail(3u);
    }
    Blink->sp_list.Flink = &Flink->sp_list;
    Flink->sp_list.Blink = &Blink->sp_list;
    p_sp_work->sp_list.Blink = &p_sp_work->sp_list;
    p_sp_work->sp_list.Flink = &p_sp_work->sp_list;
    p_sp_work->work_info->error_code = user_buffer->error_code;
    p_sp_work->work_info->content_size = user_buffer->content_size;
    memmove(
        (char *)p_sp_work->work_info + (unsigned int)p_sp_work->work_info->content_offset,
        (char *)user_buffer + (unsigned int)user_buffer->content_offset,
        (unsigned int)user_buffer->content_size);
}
```

Additional fix...



```
_int64 __fastcall SpIoctlSetControlWork(struct _IRP *a1)
{
    __IO_STACK_LOCATION *CurrentStackLocation; // rdi
    unsigned int InputSize; // eax
    int v4; // ebx
    unsigned __int8 *SystemBuffer; // rdx
    unsigned int v6; // ecx
    const char *v7; // r9
    char v9; // [rsp+40h] [rbp+8h] BYREF

    CurrentStackLocation = a1->Tail.Overlay.CurrentStackLocation;
    v9 = 0;
    if ( WPP_GLOBAL_Control != (PDEVICE_OBJECT)&WPP_GLOBAL_Control
        && (HIDWORD(WPP_GLOBAL_Control->Timer) & 1) != 0
        && BYTE1(WPP_GLOBAL_Control->Timer) >= 3u )
    {
        WPP_SF_(WPP_GLOBAL_Control->AttachedDevice, 142i64, &WPP_38b3f3bf976437d2c40d1c1825cf09f4_Traceguids);
    }
    if ( (int)RtlCheckTokenMembership(0i64, SeExports->SeLocalSystemSid, &v9) >= 0 && v9 )
    {
```

Final words



- **A generic pool overflow exploitation method exists!**
 - ... and works on real cases!
 - Thanks to the new kernel pool from 19H1

- **Try Pwn2Own!**
 - Some targets do not require so much effort
 - Attack surface is quite huge!



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

<https://twitter.com/synacktiv>

Nos publications sur : <https://synacktiv.com>