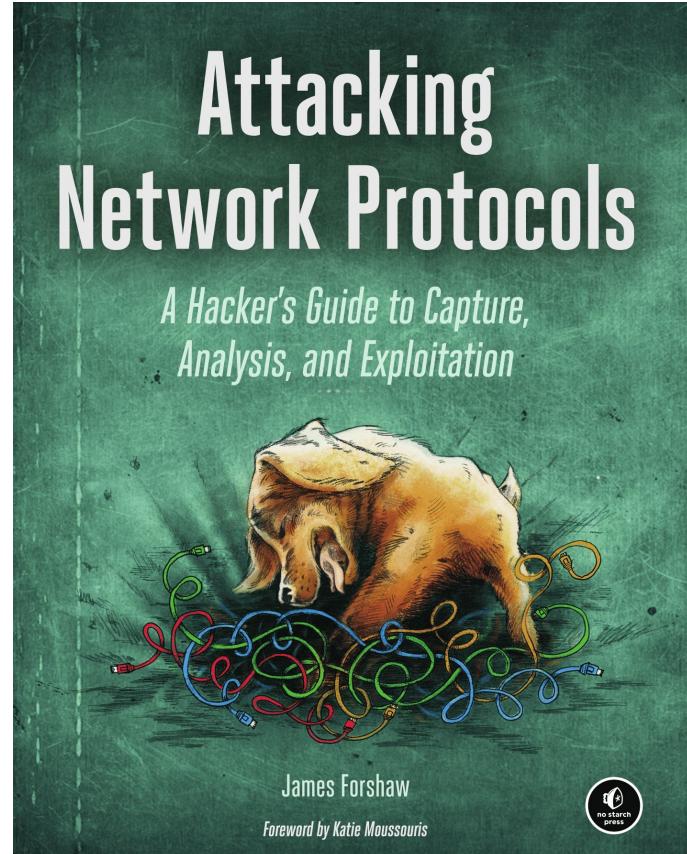


# Windows Internals and Local Attack Surface Analysis using Powershell

James Forshaw @tiraniddo

# Who am I?

- Researcher in Google's Project Zero
- Specialize in Windows
  - Especially local privilege escalation
  - Logical vulnerability specialist
- Author of a book on attacking network protocols
- @tiraniddo on Twitter.

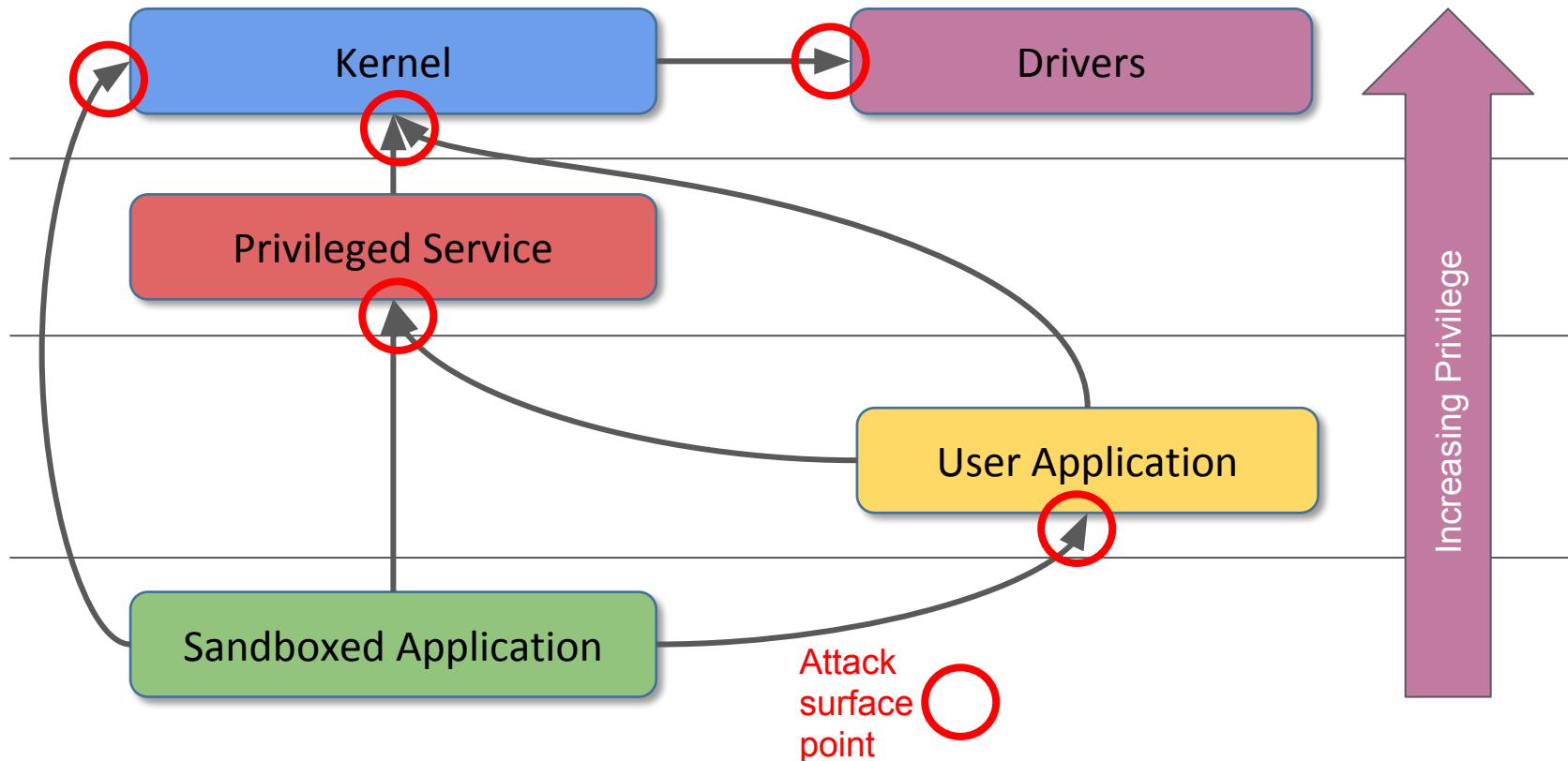


# Workshop Agenda

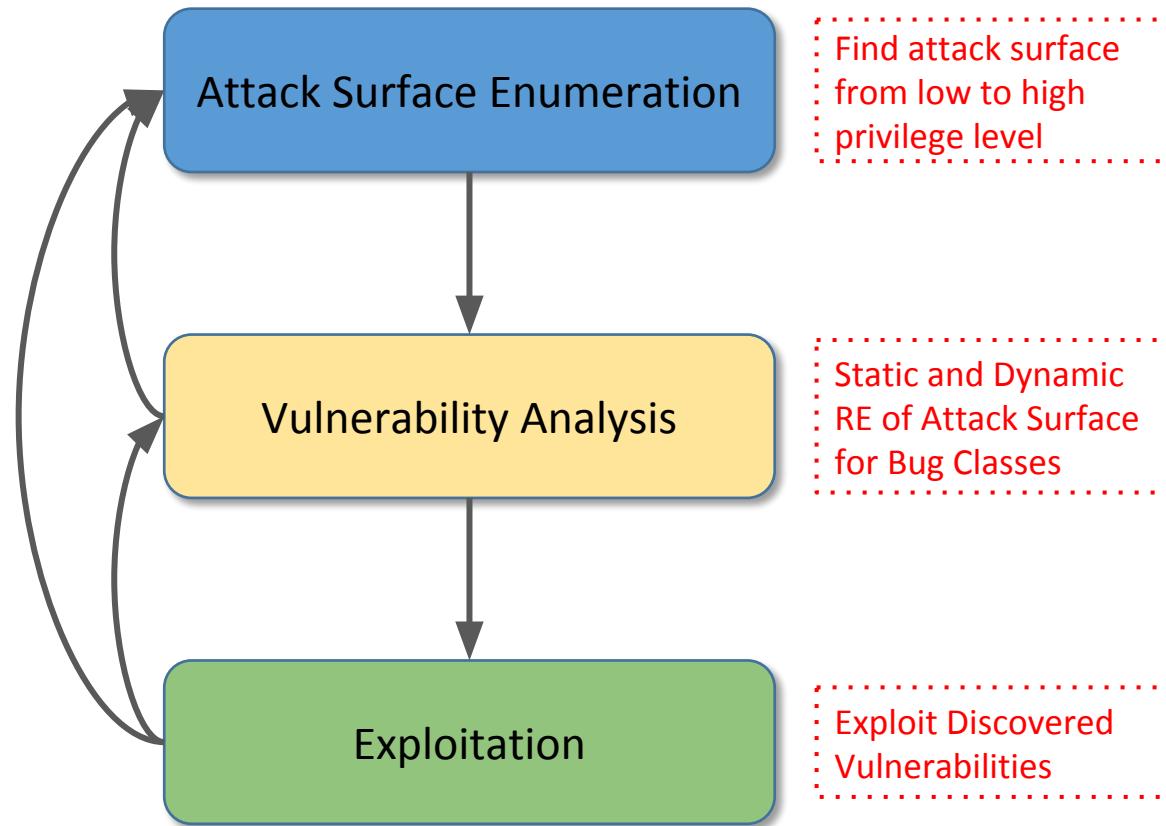
- INTRODUCTION
  - Setting up environment
- PART 1 - Windows Internals
  - Windows Resource Management
  - System Calls and Kernel Interactions
  - Process and Threads
  - Security Reference Monitor - Tokens, Security Descriptors and Access Checking
  - IO Manager
- PART 2 - Local Attack Surface Analysis
  - Enumerating Local Resource Attack Surface
  - RPC
  - DCOM
  - Miscellaneous

# INTRODUCTION

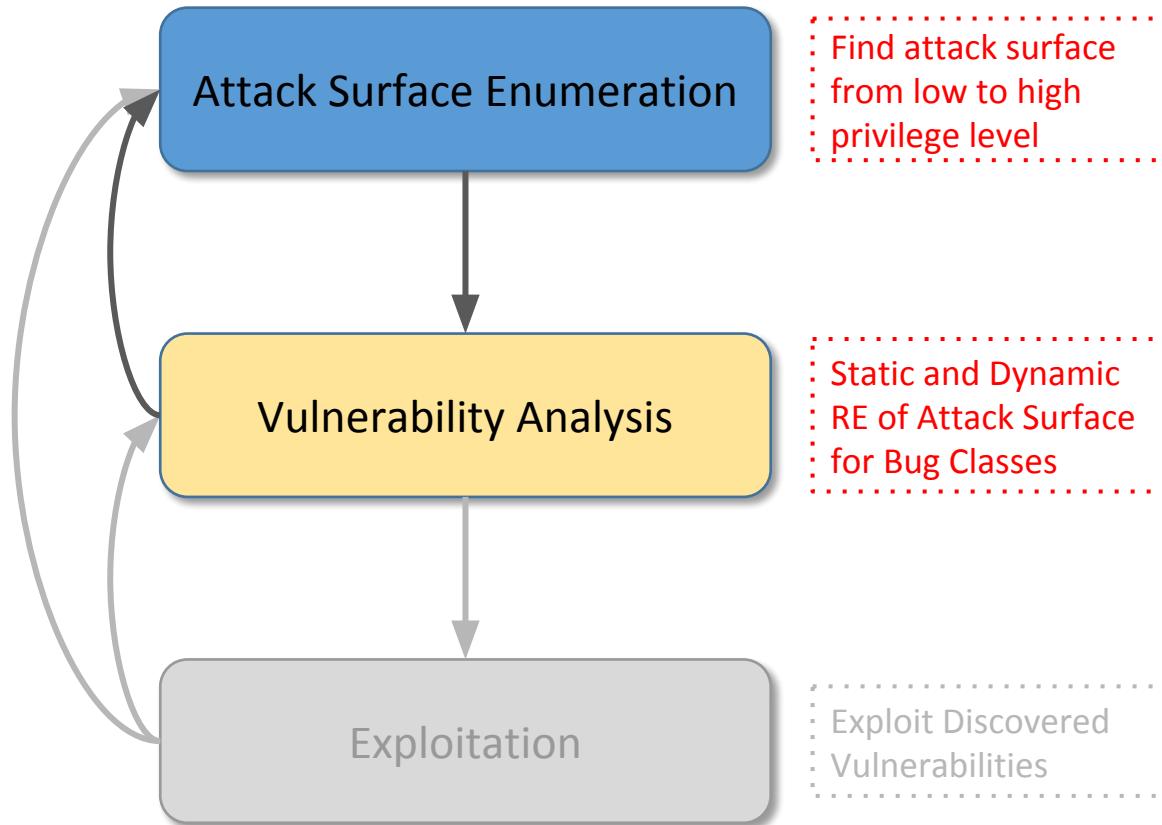
# Windows Local Attack Surface



# My Approach to Finding EOP Vulnerabilities



# My Approach to Finding EOP Vulnerabilities



# EOP Workshop

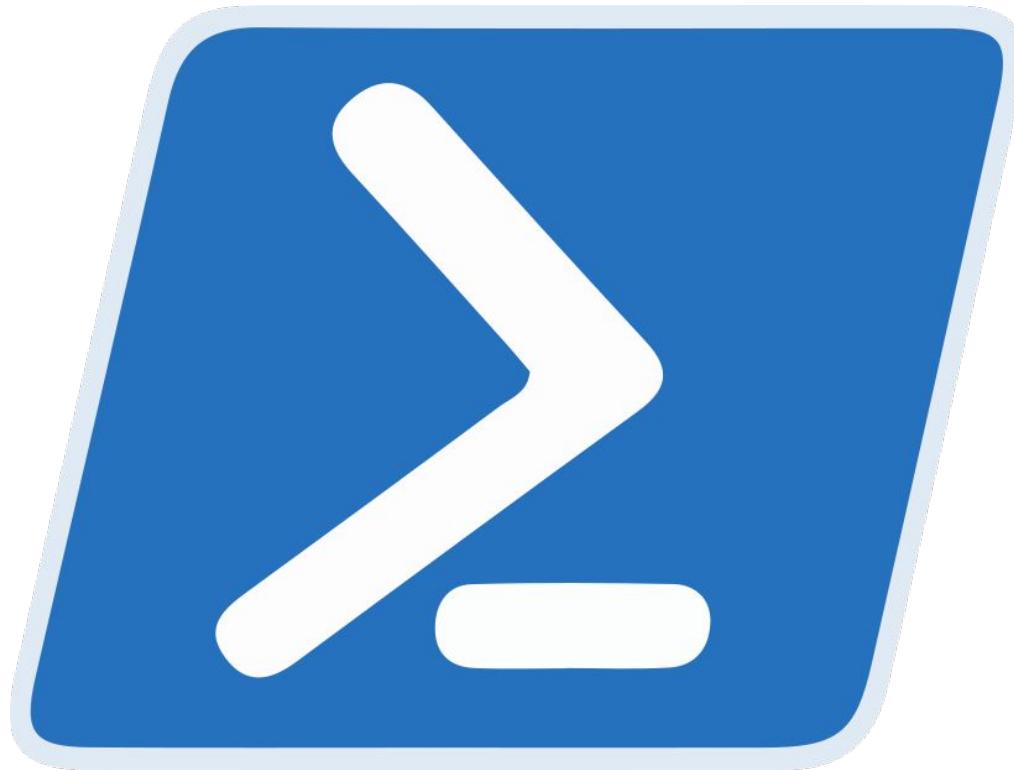
The screenshot shows a GitHub repository page. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below the navigation bar, the repository name 'tyranid / windows-logical-eop-workshop' is displayed, along with buttons for Unwatch (with 9 notifications), Star (with 94 stars), Fork (with 39 forks), and Edit.

The main content area shows the repository's summary: 3 commits, 1 branch, 3 releases, 1 contributor, and a license of GPL-3.0. There are buttons for Branch: master, New pull request, Create new file, Upload files, Find file, and Clone or download.

A list of recent commits is shown:

- James Forshaw Updated driver client. Latest commit 68e67f5 on 5 Sep 2017
- COMServer Initial implementation a year ago
- CopyFile Initial implementation a year ago
- DemoClient Updated driver client. 11 months ago
- DummyExe Initial implementation a year ago

<https://github.com/tyranid/windows-logical-eop-workshop>



# Why PowerShell?

# Pre-requisites

- Workshop based on access to Windows 10 1803
  - VM is preferred but native install is okay
- Powershell 5 with .NET >= 4.6 (all default)
- Download of tools from GITHUB [1]
- IDA Pro or IDA Free (7) edition for some basic reverse engineering
- Debugging Tools for Windows (get from Windows SDK [2])

[1] <https://github.com/tyranid/windows-attacksurface-workshop/releases/tag/BSIDESLV-2018>

[2] <https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk>

# Tools

- Two PowerShell/.NET tools we'll use:
  - NtObjectManager - PowerShell module to access native Windows APIs as well as perform attack surface analysis
  - OleViewDotNet - GUI + PS module to extract COM information and look for interesting objects
- All commands should come with help
- Use the '**Get-Help**' cmdlet for more information

***Both modules are open source***

NtObjectManager -

<https://github.com/google/sandbox-attacksurface-analysis-tools>

OleViewDotNet -

<https://github.com/tyranid/oleviewdotnet>

# Enabling Local Kernel Debugging

- You can debug the kernel from the local user, however you can't break execution (for obvious reasons I hope).
- Very useful to inspect the running kernel and objects.
- Enable debugging as admin with the following command, then reboot:

```
bcdedit /DEBUG ON
```

## ***WARNING:***

Don't enable if you have Bitlocker with a TPM.  
Otherwise you won't be able to reboot without  
your recovery key!!!!

# Public Symbols

- Almost all Microsoft code has public symbols available.
- My PS modules support symbol loading, using DBGHELP
- Best to use DBGHELP which comes with Debugging Tools for Windows as it supports remote symbol servers.

## ***DBGHELP Path:***

```
%ProgramFiles(x86)%\Windows Kits\10\debuggers\x64\dbghelp.dll
```

## ***Symbol Path:***

```
srv*c:\sym*https://msdl.microsoft.com/download/symbols
```

Copy Workshop modules to user's Modules

```
PS> Copy-Item -Recurse Modules\* `"  
"$env:USERPROFILE\Documents\WindowsPowerShell\Modules\"
```

Allow unsigned modules and scripts

```
PS> Set-ExecutionPolicy -Scope CurrentUser `'  
-ExecutionPolicy Bypass
```

Import the modules.

```
PS> Import-Module Workshop
```

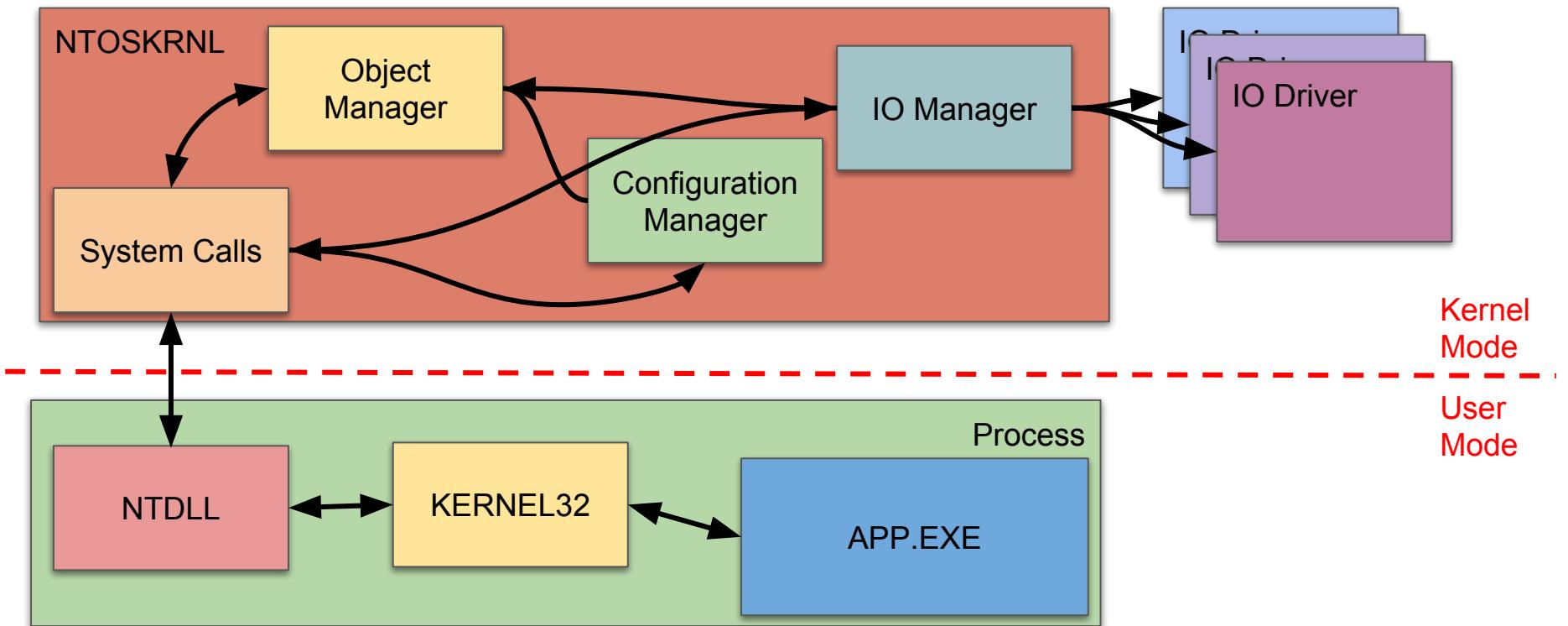
# WORKSHOP 0

## Setting up the Environment

# PART 1

# Windows Internals

# Windows NT Architecture



# Windows Object Manager

WinObj - Sysinternals: www.sysinternals.com

File View Help

Object Directories

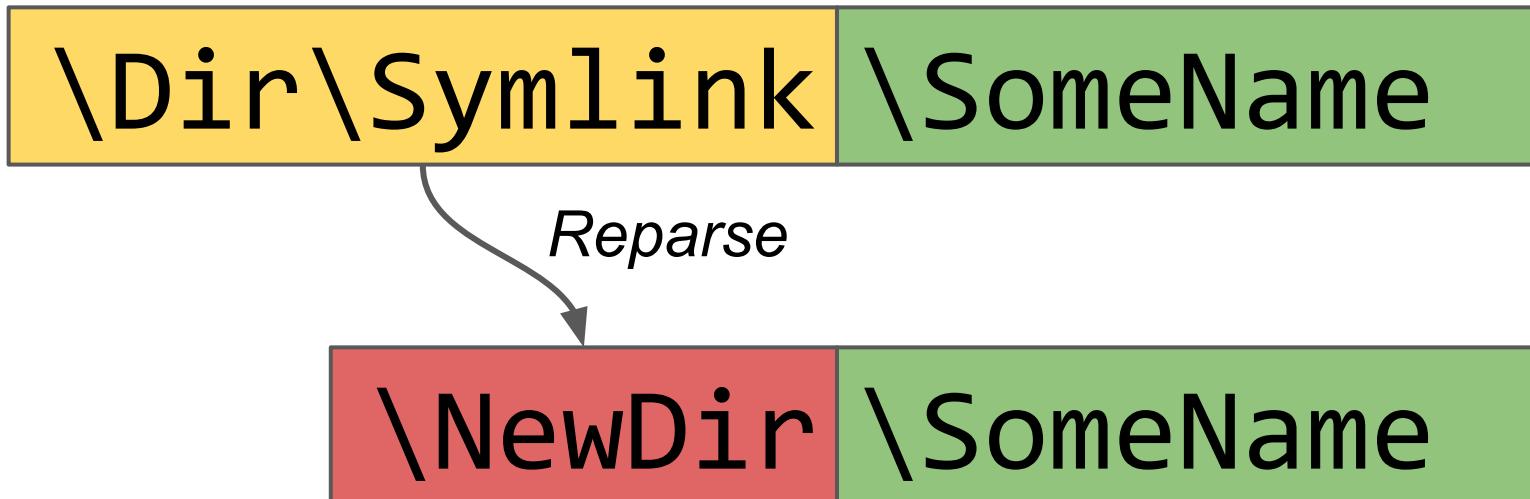
Name	Type	Symlink
Bit9Port	FilterConnectionPort	
CLDMSGPORT	FilterConnectionPort	
clfs	Device	
CsrSbSyncEvent	Event	
Dfs	SymbolicLink	\Device\dfsclient
DosDevices	SymbolicLink	\??
EFSInitEvent	Event	
Fat	Device	
FatCdrom	Device	
LanmanServerAnnounceEvent	Event	
LsaPerformance	Section	
MicrosoftMalwareProtectionAsyncPortWD	FilterConnectionPort	
MicrosoftMalwareProtectionControlPortWD	FilterConnectionPort	
MicrosoftMalwareProtectionPortWD	FilterConnectionPort	
MicrosoftMalwareProtectionRemoteloPortWD	FilterConnectionPort	
MicrosoftMalwareProtectionVeryLowloPortWD	FilterConnectionPort	
NETLOGON_SERVICE_STARTED	Event	
Ntfs	Device	

Named Objects

18

# Object Manager Paths and Symbolic Links

- Object Manager path separator is backslash.
  - Forward slash is just another character.
- Object manager supports symbolic links to “reparse” to another path.



# Important Object Directories

<i>Path</i>	<i>Description</i>
\Device	Default location for kernel driver Device Objects
\GLOBAL??	System location for symbolic links to devices
\BaseNamedObjects	System location for named resources
\Sessions\X	Directory for the login session X
\Session\0\DosDevices	Directory for the “DOS Devices” for logged in users.
\??	“Fake” prefix which refers to per-user Dos Devices.
\RPC Control	Default location of ALPC ports for RPC services

# Example Object Types

Name	Description	Require Name
Directory	An object manager namespace directory (not a file directory)	No
SymbolicLink	An object manager symbolic link	No
Device	A device object created by a driver	No
File	A file object (never seen in the object manager namespace)	Yes
Process	A process object	No
Token	An access token object	No
ALPC Port	An Advanced Local Procedure Call port.	Yes
Key	A registry key	Yes

List recursively the object manager namespace.

```
PS> ls -Recurse NtObject:\
```

Find all named “Device” objects

```
PS> ls -Recurse NtObject:\ | ? TypeName -eq Device
```

Find all “SymbolicLink” objects

```
PS> ls -Recurse NtObject:\ | ? IsSymbolicLink
```

Get all registered NT types.

```
PS> Get-NtType
```

Get the object \Device\NamedPipe

```
PS> $obj = Get-Item NtObject:\Device\NamedPipe
```

# WORKSHOP 1

## Inspecting Object Manager and Types

# Object Manager System Calls

- System calls typically named with pattern *Nt{Operation}{Type}*
- Typically at least 4 system calls for each object type
  - **Create** - Create a new kernel object (e.g. NtCreateFile)
  - **Open** - Open an existing object (e.g. NtOpenEvent)
  - **QueryInformation** - Query information from an open object (e.g. NtQueryInformationKey)
  - **SetInformation** - Set information on an object, (e.g. NtSetInformationJob)
- *Nt* prefix used for user mode, *Zw* prefix used for kernel mode
- *NtObjectManager* exposes these system calls through the following cmdlets:
  - New-Nt{Type} - Exposes the creation system call
  - Get-Nt{Type} - Exposes the open system call
- Cmdlets return instances of a wrapper object

# Example “Open” System Call

```
NTSTATUS NtOpenFile(  
    OUT PHANDLE FileHandle,  
    IN ACCESS_MASK DesiredAccess,  
    IN POBJECT_ATTRIBUTES ObjectAttributes,  
    OUT PIO_STATUS_BLOCK IoStatusBlock,  
    IN ULONG ShareAccess,  
    IN ULONG OpenOptions  
) ;
```

Error status.  
Success  $\geq 0$   
Error  $< 0$

Returns a HANDLE  
to the object

Access desired on  
the open object

Specify the  
“attributes” of the  
object to open

Type specific  
parameters

# Object Attributes

```
struct OBJECT_ATTRIBUTES {
    ULONG Length;
    HANDLE RootDirectory;
    PUNICODE_STRING ObjectName;
    ULONG Attributes;
    PVOID SecurityDescriptor;
    PVOID SecurityQualityOfService;
};
```

Combined to lookup name

Attributes flags for object operation

Specify security descriptor for new object

Used for impersonation

# Attribute Flags

<b>Name</b>	<b>Description</b>
OBJ_INHERIT	Mark new handle as inheritable
OBJ_PERMANENT	Makes new object permanent (needs SeCreatePermanentPrivilege)
OBJ_EXCLUSIVE	Makes new object exclusive, no one else can open the object.
OBJ_CASE_INSENSITIVE	Lookup object name case insensitive (normally ignored)
OBJ_OPENIF	Open the object if it already exists when using a “Create” system call
OBJ_OPENLINK	Open the link object rather than the target (only for Registry Keys)
OBJ_KERNEL_HANDLE	Open the new handle as kernel only (only usable from kernel)
OBJ_FORCE_ACCESS_CHECK	Force an access check to be made when calling from kernel mode
OBJ_IGNORE_IMPERSONATED_DEVICEMAP	Ignore impersonated device map
OBJ_DONT_REPARSE	Fail if reparse encountered during open operation

# Object Manager Name Invalid Characters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	-
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Create the directory ABC in the user's device map directory

```
PS> $dir = New-NtDirectory -Path "\??\ABC"
```

Open the existing ABC directory

```
PS> $dir = Get-NtDirectory -Path "\??\ABC"
```

Open an object by path without knowing the type.

```
PS> $obj = Get-NtObject -Path "\??\ABC"
```

Create a directory with a name using special chars and NUL

```
PS> New-NtDirectory -Path "\??\`0`r`n`t_ABC"
```

Open an event with name ABC relative to an opened directory object.

```
PS> Get-NtEvent -Path "ABC" -Root $dir
```

Get known name and error message for a status code.

```
PS> Get-NtStatus 0xC0000022
```

# Object Lifetime

- No explicit object delete operation in object manager.
  - Object lasts as long as there are references/handles to the object
  - Once the object is destroyed its name is also removed
- In PowerShell you must capture open object references otherwise GC will eventually close the handle.
- Possible to persist named objects even when all references closed using *NtMakePermanentObject* or *OBJ\_PERMAMENT*
  - Needs *SeCreatePermanentPrivilege*
- Possible to “delete” permanent objects using *NtMakeTemporaryObject*.
  - Only needs DELETE permission on the opened object.

Create a permanent event calls ABC in the user's device map directory

```
PS> New-NtEvent \??\ABC -ObjectAttributes Permanent
```

Make an object permanent.

```
PS> $obj.MakePermanent()
```

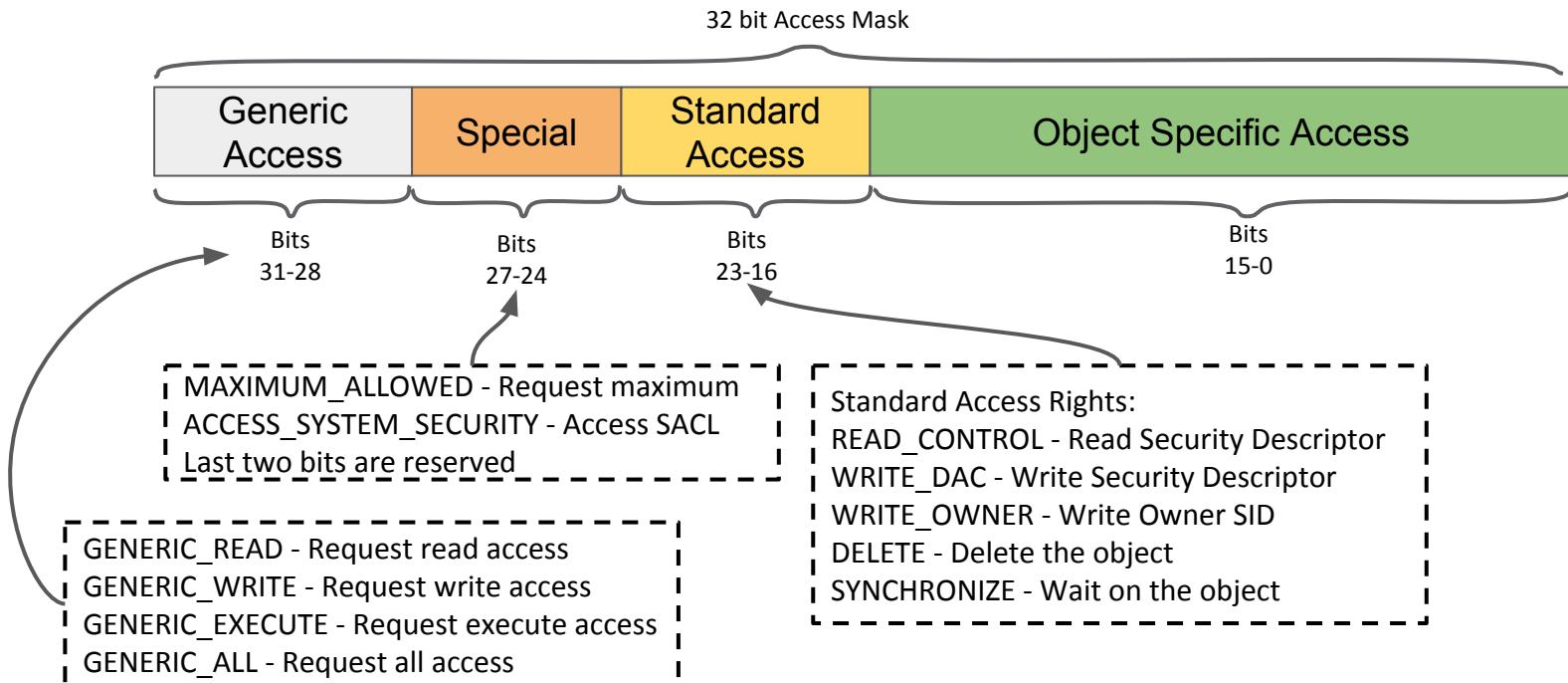
“Delete” object and close the opened object handle.

```
PS> $obj.MakeTemporary(); $obj.Close()
```

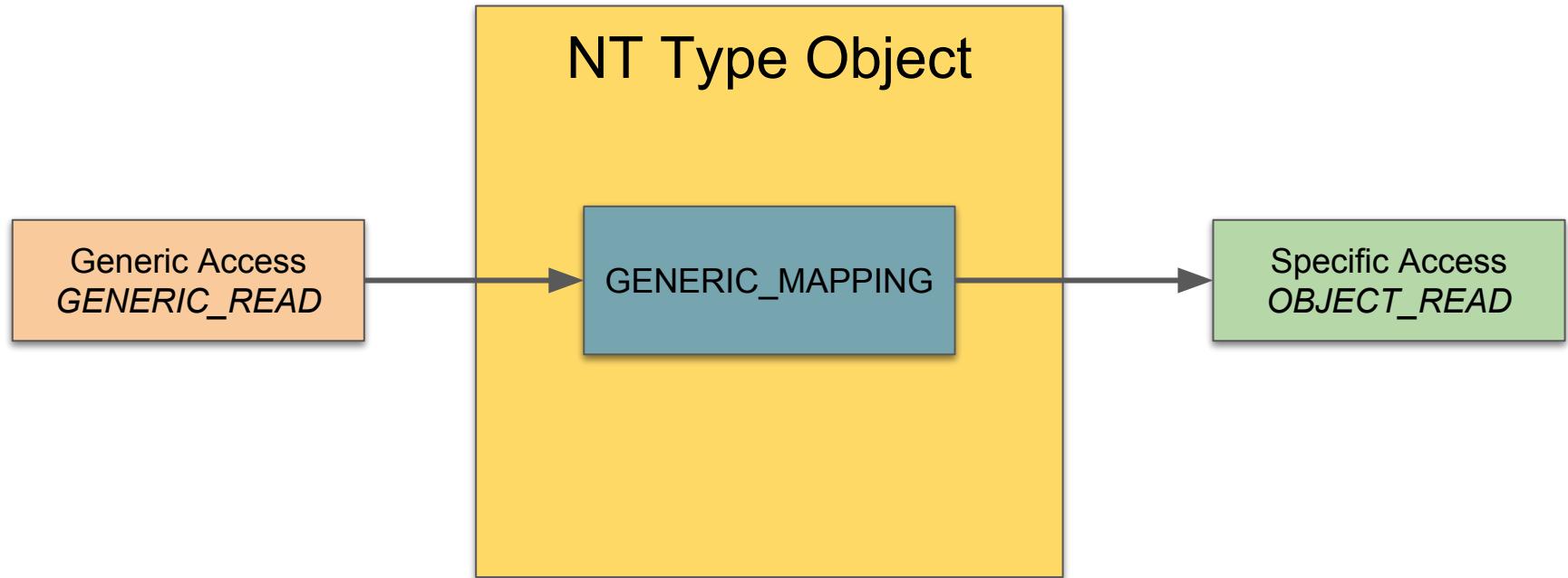
Use an object then close it immediately after.

```
PS> Use-NtObject($obj = Get-NtEvent "\??\ABC") {  
    # Do something with $obj  
}
```

# Access Masks



# Mapping Generic Access Rights



# Specific Access Masks

<i>Process Access</i>	<i>Access Mask</i>	<i>File Access</i>
Terminate (X)	0x00000001	ReadData (R)
CreateThread (W)	0x00000002	WriteData (W)
SetSessionId (?)	0x00000004	AppendData (W)
VmOperation (W)	0x00000008	ReadEa (R)
VmRead (R)	0x00000010	WriteEa (W)
VmWrite (W)	0x00000020	Execute (X)
DupHandle (W)	0x00000040	DeleteChild (?)
CreateProcess (W)	0x00000080	ReadAttributes (R)

Get process access rights as raw access mask

```
PS> Get-NtAccessMask -ProcessAccess Terminate
```

Convert a raw access mask to a File access mask

```
PS> Get-NtAccessMask 0xFF -ToSpecificAccess File
```

Show Key type GenericRead, GenericWrite and GenericExecute Access

```
PS> Get-NtType Key | select GenericRead, `  
GenericWrite, GenericExecute
```

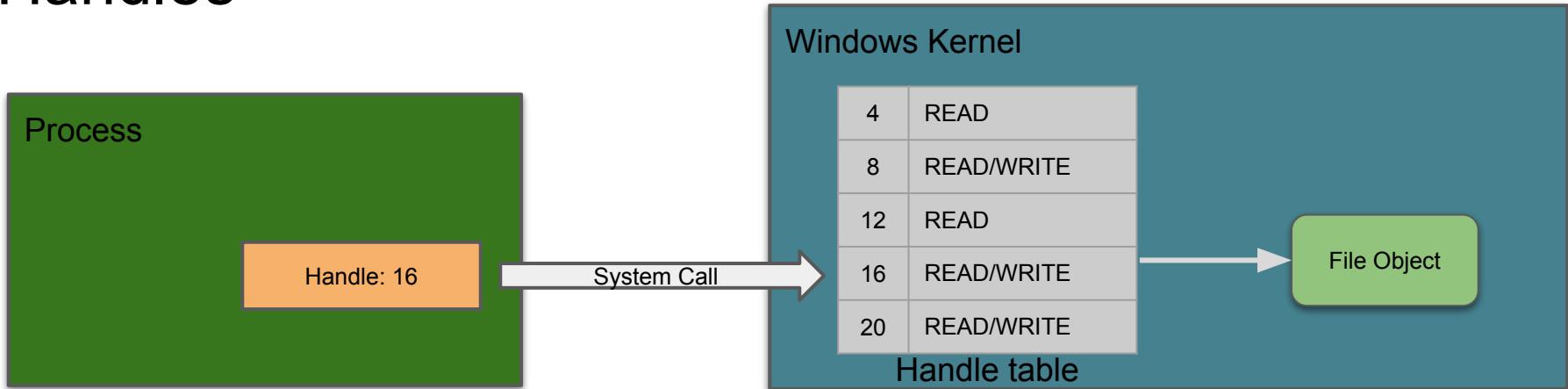
Map GenericRead to specific access rights for a File

```
PS> Get-NtAccessMask -FileAccess GenericRead `  
-MapGenericRights -ToSpecificAccess File
```

Get granted access on an opened object.

```
PS> $obj.GrantedAccess
```

# Handles



```
NTSTATUS ObReferenceObjectByHandle(  
    _In_     HANDLE  
    _In_     ACCESS_MASK  
    _In_opt_ POBJECT_TYPE  
    _In_     KPROCESSOR_MODE  
    _Out_    PVOID  
    _Out_opt_ POBJECT_HANDLE_INFORMATION HandleInformation  
);
```

Handle from user mode.  
Required Access  
Returns Kernel Object Pointer  
Kernel only handles are global and have the top bit set. Can't be used from a user context.

Handle,  
DesiredAccess,  
ObjectType,  
AccessMode,  
\*Object,

# Handle Duplication

```
NTSTATUS NtDuplicateObject(
```

```
    HANDLE     SourceProcessHandle,
```

```
    HANDLE     SourceHandle,
```

```
    HANDLE     TargetProcessHandle,
```

```
    PHANDLE    TargetHandle,
```

```
    ACCESS_MASK DesiredAccess,
```

```
    ULONG      HandleAttributes,
```

```
    ULONG      Options
```

```
);
```

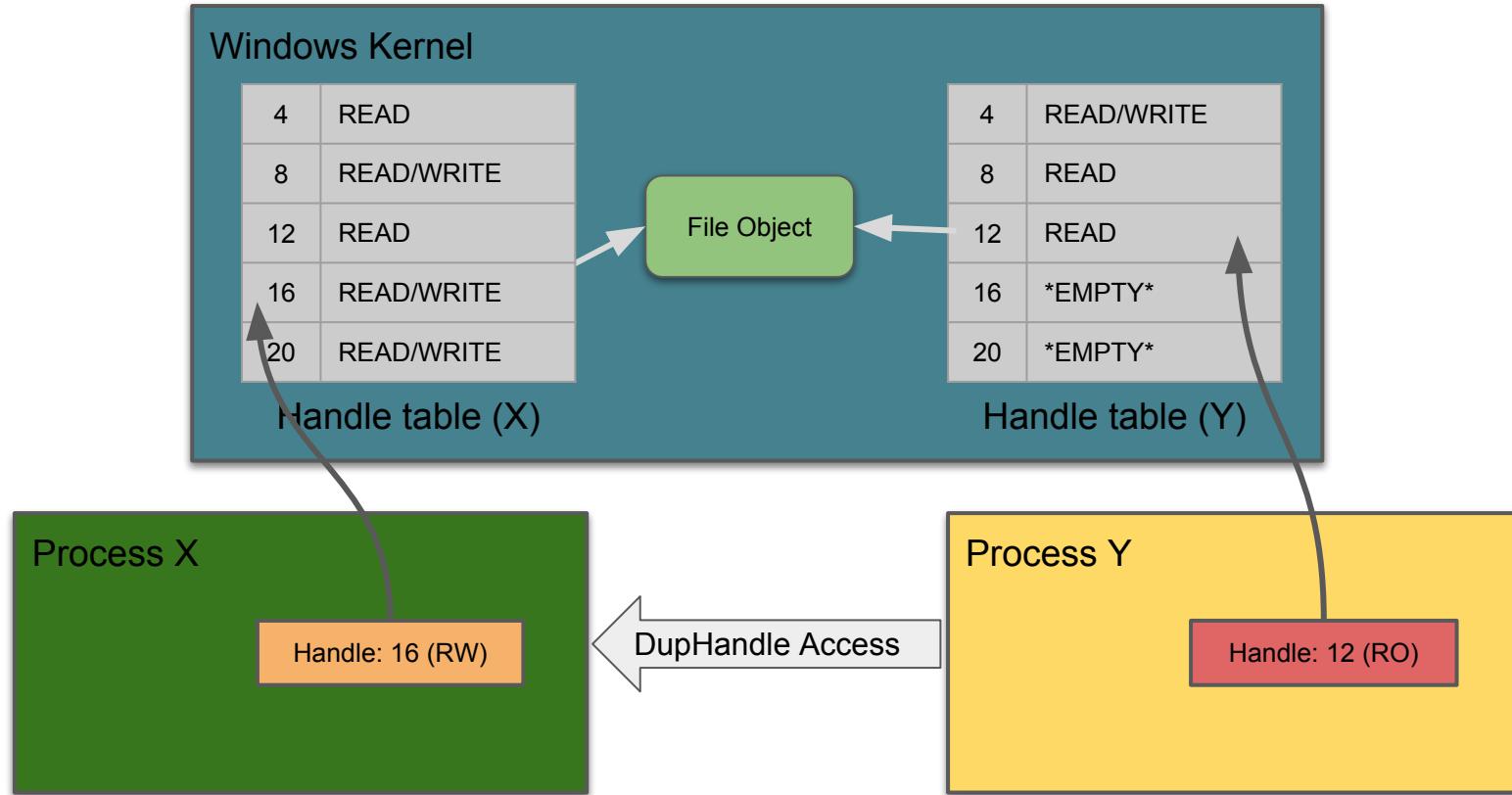
Source and target  
process handles need  
DupHandle access

Special values:  
-1 - Source Process  
-2 - Calling Thread

Can drop  
access with no  
access check

DUPLICATE\_SAME\_ATTRIBUTES  
DUPLICATE\_SAME\_ACCESS  
DUPLICATE\_CLOSE\_SOURCE

# Handle Duplication



Get list of handles to “Key” types.

```
PS> Get-NtHandle -ObjectTypes "Key"
```

Get list of handles open in the current process.

```
PS> Get-NtHandle -ProcessId $pid
```

Get list of handles with object name “Blah” from PID 16 and duplicate the handle

```
PS> $hs = Get-NtHandle -pid 16 | ? Name -eq "Blah"  
PS> $obj = $hs[0].GetObject()
```

Resolve NT object address and print as hex

```
PS> Resolve-NtObjectAddress $obj  
PS> "0x{0:X}" -f $obj.Address
```

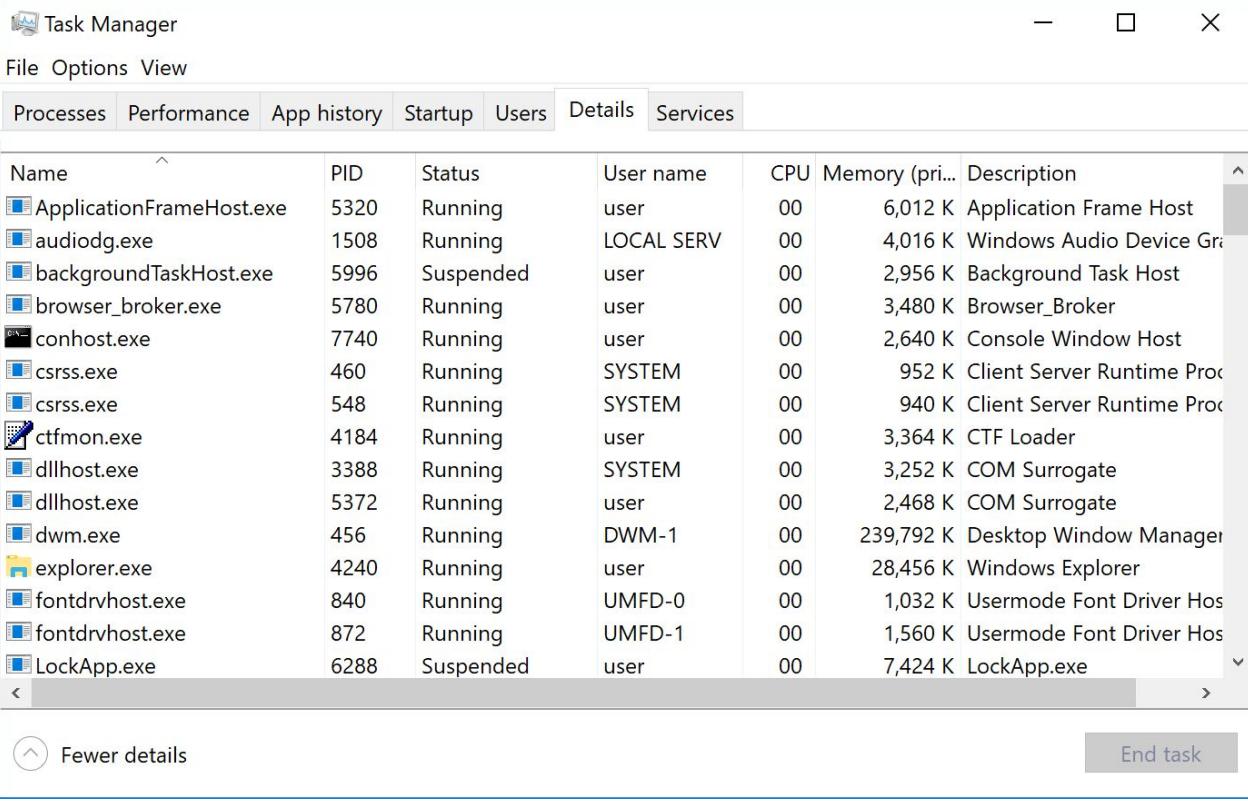
# Kernel Object Structures

Type Name	Kernel Object
Directory	nt!_OBJECT_DIRECTORY
SymbolicLink	nt!_OBJECT_SYMBOLIC_LINK
Device	nt!_DEVICE_OBJECT
File	nt!_FILE_OBJECT
Process	nt!_EPROCESS
Token	nt!_TOKEN
ALPC Port	nt!_ALPC_PORT
Key	nt!_CM_KEY_BODY

# WORKSHOP 2

## Accessing Objects

# Processes and Threads



The screenshot shows the Windows Task Manager interface with the "Processes" tab selected. The window title is "Task Manager". The menu bar includes "File", "Options", and "View". Below the menu is a tab bar with "Processes" (selected), "Performance", "App history", "Startup", "Users", "Details", and "Services". The main area is a table listing processes:

Name	PID	Status	User name	CPU	Memory (pri...)	Description
ApplicationFrameHost.exe	5320	Running	user	00	6,012 K	Application Frame Host
audiogd.exe	1508	Running	LOCAL SERV	00	4,016 K	Windows Audio Device Gr...
backgroundTaskHost.exe	5996	Suspended	user	00	2,956 K	Background Task Host
browser_broker.exe	5780	Running	user	00	3,480 K	Browser_Broker
conhost.exe	7740	Running	user	00	2,640 K	Console Window Host
csrss.exe	460	Running	SYSTEM	00	952 K	Client Server Runtime Proc...
csrss.exe	548	Running	SYSTEM	00	940 K	Client Server Runtime Proc...
ctfmon.exe	4184	Running	user	00	3,364 K	CTF Loader
dllhost.exe	3388	Running	SYSTEM	00	3,252 K	COM Surrogate
dllhost.exe	5372	Running	user	00	2,468 K	COM Surrogate
dwm.exe	456	Running	DWM-1	00	239,792 K	Desktop Window Manager
explorer.exe	4240	Running	user	00	28,456 K	Windows Explorer
fontdrvhost.exe	840	Running	UMFD-0	00	1,032 K	Usermode Font Driver Hos...
fontdrvhost.exe	872	Running	UMFD-1	00	1,560 K	Usermode Font Driver Hos...
LockApp.exe	6288	Suspended	user	00	7,424 K	LockApp.exe

At the bottom left is a "Fewer details" button, and at the bottom right is an "End task" button.

# Process and Thread Enumeration

- Two ways of enumerating processes and threads
  - *NtGetNextProcess* and *NtGetNextThread* enumerates objects directly.
  - *NtQuerySystemInformation* with *SystemProcessInformation* info class to get PIDs/TIDs without opening the objects.
- *NtGetNext\** useful if you want to open the objects directly. Enumerating threads needs an open handle to the process.
- System information approach will show all processes including ones you can't open for security reasons.
  - Process and thread access is independent.
  - Can open a threads inside a process you wouldn't normally be able to access

# Interesting Process and Thread Access

<b><i>Process Access</i></b>	<b><i>Description</i></b>
CreateThread	Create a new thread in the process
VmOperation	Allocate arbitrary memory in the process
VmRead	Read arbitrary memory in the process.
VmWrite	Write arbitrary memory in the process.
DupHandle	Duplicate handles in and out of the process.
CreateProcess	Create a child process using this process.
<b><i>Thread Access</i></b>	<b><i>Description</i></b>
GetContext	Get the current thread's register context
SetContext	Set the current thread's register context
SetThreadToken	Apply an impersonation token to the thread.

Get all processes as objects.

```
PS> Get-NtProcess
```

Open process 1234 with VMREAD access.

```
PS> Get-NtProcess -ProcessId 1234 -Access VmRead
```

Get all process information (no elevated privileges required).

```
PS> Get-NtProcess -InfoOnly
```

Get all threads as objects.

```
PS> Get-NtThread
```

Get all threads as objects using the system information.

```
PS> Get-NtThread -FromSystem
```

Get all thread information (no elevated privileges required).

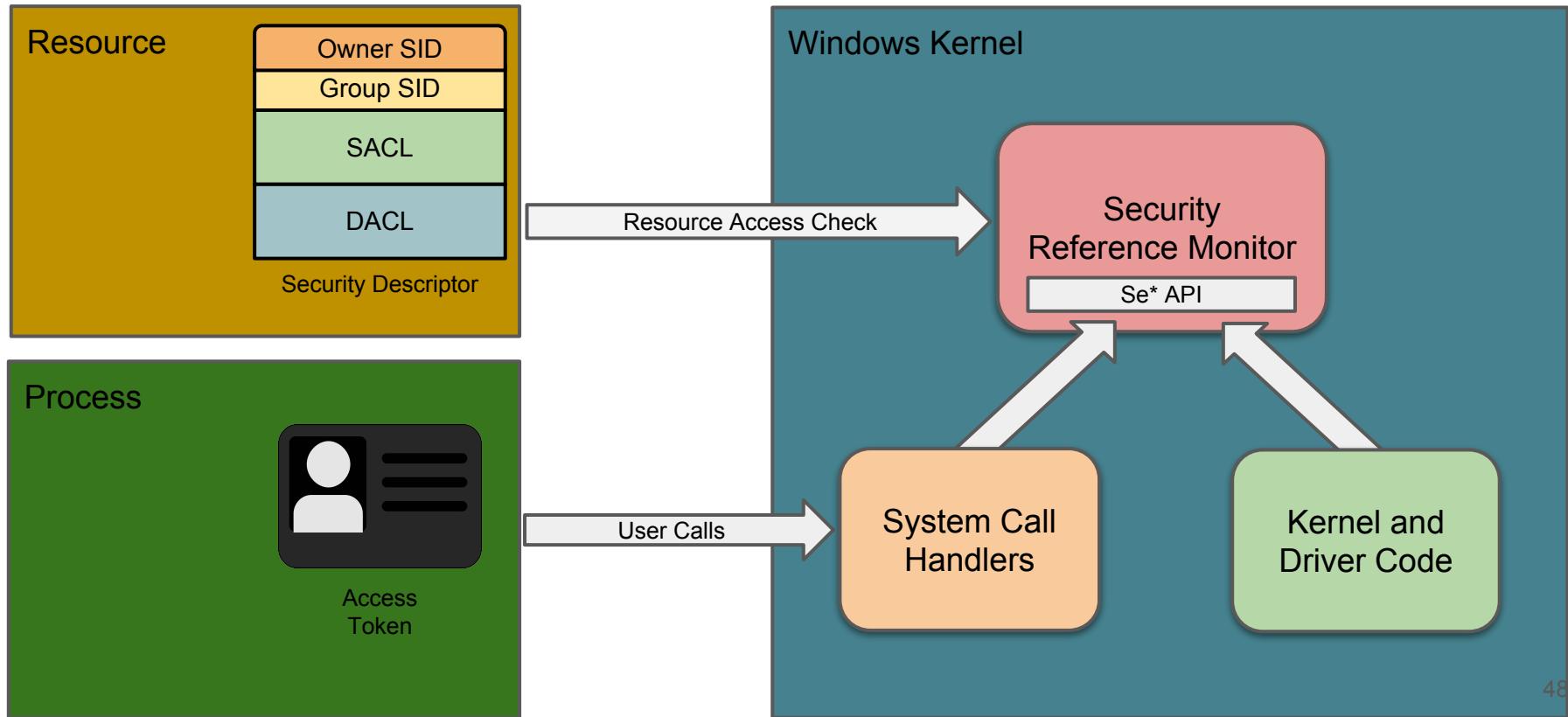
```
PS> Get-NtThread -InfoOnly
```

# WORKSHOP 3

## Inspecting Processes and Threads

# Windows Security Model

# Windows Security Components



# Security Identifiers

- A Security Identifier (SID) is how Windows represents a user or group (think of it like an expanded UID/GID from Unix)

S-1-5-RID1-RID2-...-RIDN

The diagram shows the SID structure: S-1-5-RID1-RID2-...-RIDN. Arrows point from labels to specific parts: 'Version (always 1)' points to the 'S' and the first dash; 'Authority' points to the '1'; and a bracket labeled 'Relative Identifiers' covers the '5' and the sequence of RIDs.

World/Everyone	S-1-1-0
Creator Owner	S-1-3-0
Local SYSTEM	S-1-5-18
Authenticated Users	S-1-5-11
Anonymous	S-1-5-7

Get the Everyone group SID from a SDDL form.

```
PS> Get-NtSid S-1-1-0
```

Get the Everyone group SID from a known SDDL form.

```
PS> Get-NtSid WD
```

Get SID from current username.

```
PS> Get-NtSid -Name ([Environment]::UserName)
```

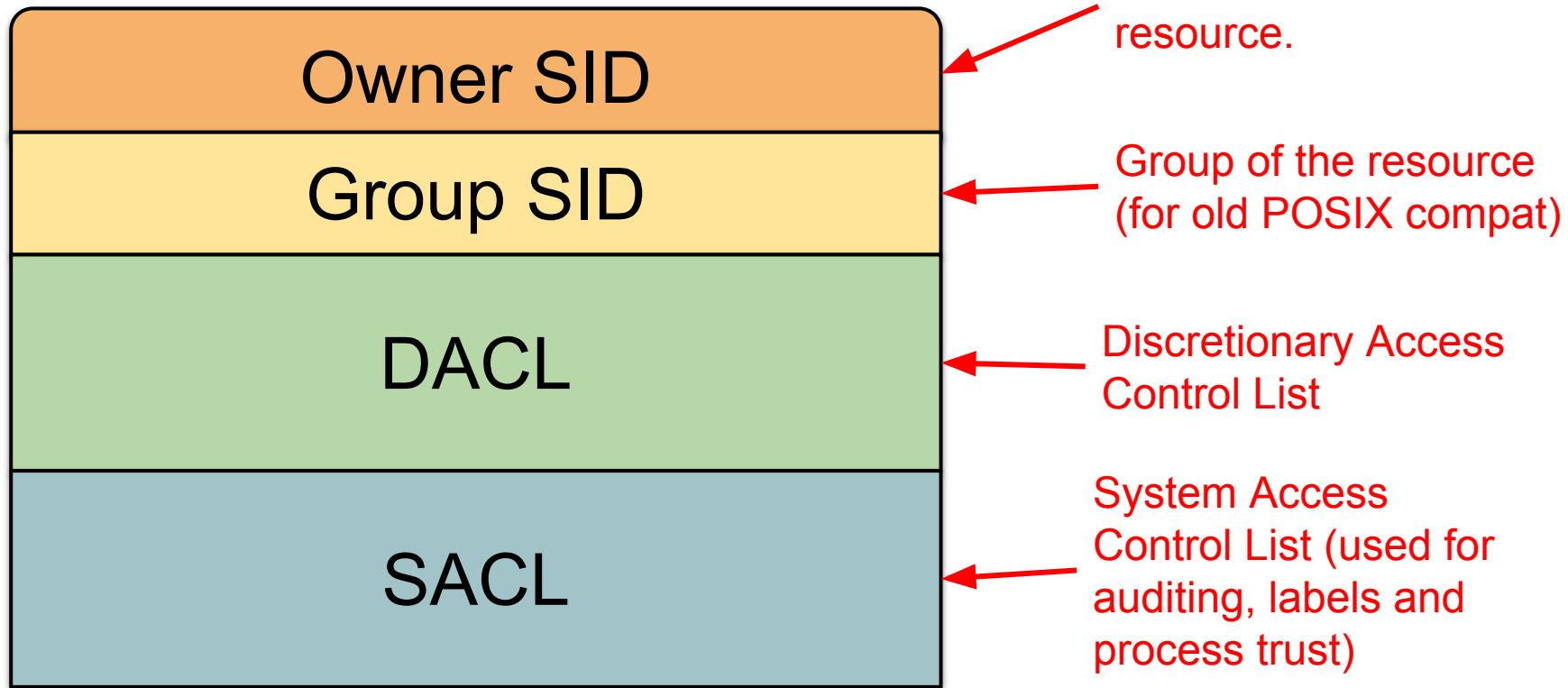
Get service SID for the service “WebClient”

```
PS> Get-NtSid -ServiceName "WebClient"
```

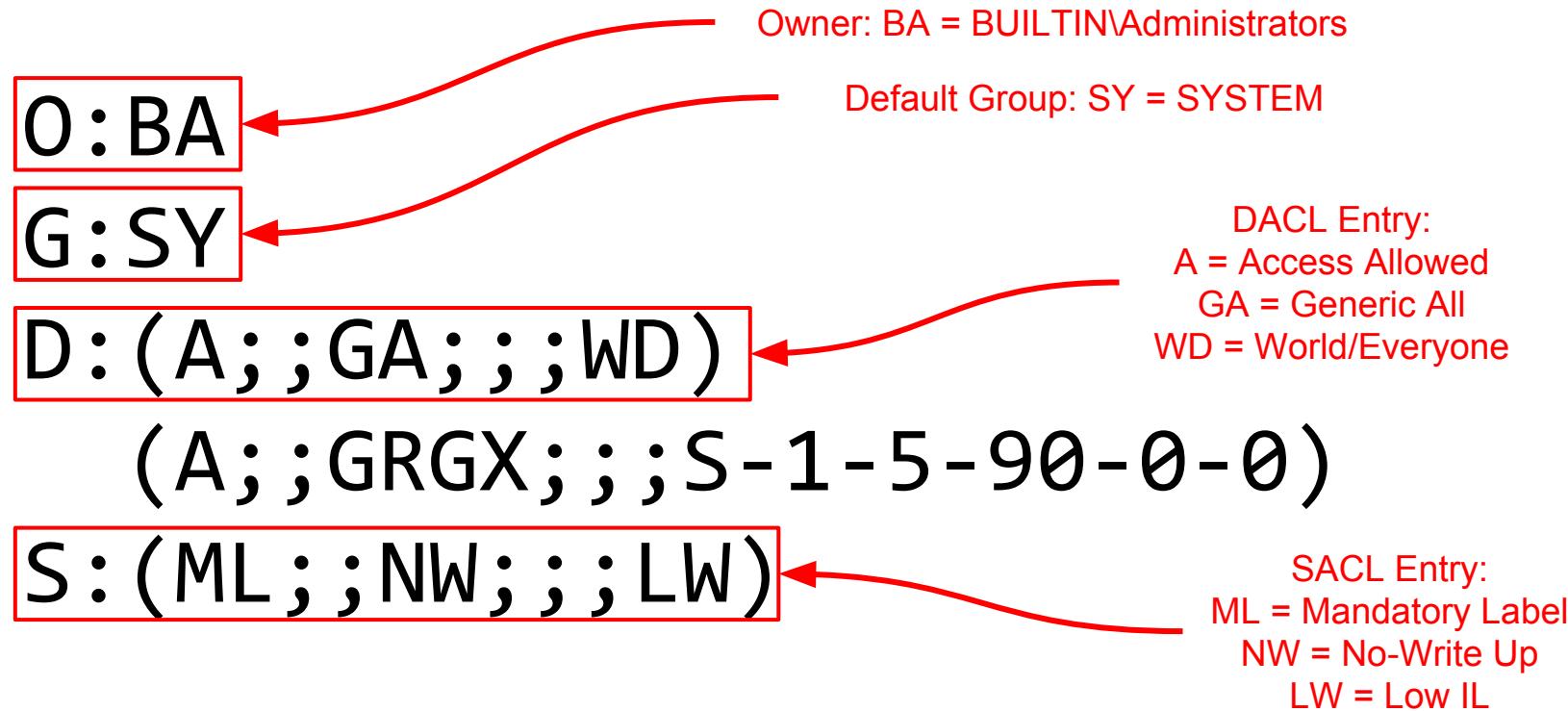
Get a well known SID for the Builtin Administrators

```
PS> Get-NtSid -KnownSid BuiltinAdministrators
```

# Security Descriptor



# Security Descriptor Definition Language (SDDL)



# Access Required for Security Descriptors

<i>Access Granted</i>	<i>Read</i>	<i>Write</i>
ReadControl	Owner Group Dacl Label	
WriteDacl		Dacl
WriteOwner		Owner Group Label
AccessSystemSecurity	Sacl (not including Label)	Sacl (not including Label)

Get a .NET style ACL through the drive provider (only use for back-compat)

```
PS> Get-Acl NtObject:\Dir\ABC
```

Get the security descriptor from an opened object (needs ReadControl access)

```
PS> $sd = Get-NtSecurityDescriptor $obj
```

Get the security descriptor from a path.

```
PS> Get-NtSecurityDescriptor "\Dir\ABC"
```

Set the DACL on the security descriptor of an object

```
PS> Set-NtSecurityDescriptor $obj $sd Dacl
```

View the security descriptor of an object in a GUI

```
PS> Show-NtSecurityDescriptor $obj
```

# Security Descriptors and Inheritance

- New resources by default will inherit Security Descriptor for parent container
  - For example the object directory/file directory/registry key.
- Most resource creation calls can specify explicit Security Descriptor using `OBJECT_ATTRIBUTES`
- If no inheritable ACEs, uses default DACL.
  - Even for Files, which is an odd behaviour.
- Special ACEs
  - OWNER RIGHTS - Limits/Grants Owner Access
  - CREATOR OWNER - SID replaced during inheritance with current owner SID
  - SELF - Replaced by the SID specified in `AccessCheckByType`
- Default Owner and Primary Group assigned from fields in the token unless explicitly specified.
- Mandatory label assigned if IL < Medium unless inherited.

# Default Security Descriptor

The screenshot shows the Windows Security descriptor editor for a user account. The tabs at the top are Main Details, Groups, Privileges, Default Dacl (selected), Misc, Operations, and Security. The Default Owner is listed as DESKTOP-KHV61TH\user, and the Primary Group is DESKTOP-KHV61TH\None. The Default DACL table shows three entries:

Group	Access	Flags	Type
DESKTOP-KHV61TH\user	GenericAll	None	Allowed
NT AUTHORITY\SYSTEM	GenericAll	None	Allowed
NT AUTHORITY\LogonSessionId_0_159719	GenericExecute, GenericRead	None	Allowed

Red arrows point from the text labels to the corresponding sections in the interface.

Default Owner

Default Group

Default DACL

```
< >
```

Generate a security descriptor based on a token.

```
PS> $sd = New-NtSecurityDescriptor -Token $token
```

Add an access allowed entry to an existing SD.

```
PS> Add-NtSecurityDescriptorDaclAce $sd -Sid  
"S-1-1-0" -AccessMask 0xFF
```

Generate a security descriptor based on an SDDL string.

```
PS> $sd = New-NtSecurityDescriptor  
-Sddl "O:BAG:BAD:(A;;GA;;WD)"
```

# WORKSHOP 4

## Accessing Security Descriptors

# Access Token

The screenshot shows a PowerShell window titled "powershell.exe:9364 - User DESKTOP-KHV61TH\user - To..." displaying token details. The "Main Details" tab is selected. The token information is organized into two main sections: "Token" and "Source".

**Token:**

- User: DESKTOP-KHV61TH\user
- User SID: S-1-5-21-2696641275-1856431857-3807057343-1000
- Token Type: Primary
- Impersonation Level: N/A
- Token ID: 0000000-012E30F1
- Authentication ID: 0000000-00824B43
- Origin Login ID: 0000000-000003E7
- Modified ID: 0000000-012E4E5E
- Integrity Level: High
- Session ID: 3
- Elevation Type: Full
- Is Elevated: True

**Source:**

- Name: User32
- Id: 0000000-00824B18

Annotations with red arrows point from specific fields to labels:

- User SID: The user's SID
- Token Type and Impersonation Level: Token type and impersonation level
- Token ID: Unique Token ID
- Authentication ID: Authentication Session ID
- Integrity Level: Mandatory Label
- Session ID: Terminal Session ID

# Token Groups and Privileges

Main Details		Groups	Privileges	Default Dacl	Misc	Operations	Security
Name			Flags				
BUILTIN\Administrators			Mandatory, Enabled, Owner				
BUILTIN\Users			Mandatory, Enabled				
CONSOLE LOGON			Mandatory, Enabled				
DESKTOP-KHV61TH\None			Mandatory, Enabled				
DESKTOP-KHV61TH\user			None				
Everyone			Mandatory, Enabled				
LOCAL			Mandatory, Enabled				
NT AUTHORITY\Authenticated Users			Mandatory, Enabled				
NT AUTHORITY\INTERACTIVE			Mandatory, Enabled				
NT AUTHORITY\Local account			Mandatory, Enabled				
NT AUTHORITY\Local account and member of Administrators group			Mandatory, Enabled				
NT AUTHORITY\LogonSessionId_0_8538852			Mandatory, Enabled, Logon				
NT AUTHORITY\NTLM Authentication			Mandatory, Enabled				
NT AUTHORITY\This Organization			Mandatory, Enabled				

Main Details		Groups	Privileges	Default Dacl	Misc	Operations	Security
Name			Flags				
SeBackupPrivilege			Disabled				
SeChangeNotifyPrivilege			Default Enabled				
SeCreateGlobalPrivilege			Default Enabled				
SeCreatePageFilePrivilege			Disabled				
SeCreateSymbolicLinkPrivilege			Disabled				
SeDebugPrivilege			Enabled				
SeDelegateSessionUserImpersonatePrivilege			Disabled				
SeImpersonatePrivilege			Default Enabled				
SeIncreaseBasePriorityPrivilege			Disabled				
SeIncreaseQuotaPrivilege			Disabled				
SeIncreaseWorkingSetPrivilege			Disabled				
SeLoadDriverPrivilege			Disabled				
SeManageVolumePrivilege			Disabled				
SeProfileSingleProcessPrivilege			Disabled				
SeRemoteShutdownPrivilege			Disabled				
SeRestorePrivilege			Disabled				
SeSecurityPrivilege			Disabled				
SeShutdownPrivilege			Disabled				
SeSystemEnvironmentPrivilege			Disabled				
SeSystemProfilePrivilege			Disabled				
SeSystemTimePrivilege			Disabled				
SeTakeOwnershipPrivilege			Disabled				
SeTimeZonePrivilege			Disabled				
SeUndockPrivilege			Disabled				

# Token Privileges and Security Bypasses

- Token privileges can be enabled or disabled.

<b><i>Privilege Name</i></b>	<b><i>Security Bypass</i></b>
SeDebugPrivilege	Bypasses access checking on Processes and Threads
SeBackupPrivilege	Bypasses read access check on Files and Keys
SeRestorePrivilege	Bypasses write access check on Files and Keys
SeSecurityPrivilege	Grants access to the SACL
SeTakeOwnershipPrivilege	Set arbitrary owner SIDs in a security descriptor
SeAssignPrimaryTokenPrivilege	Bypass process token assignment check
SeImpersonatePrivilege	Bypass impersonation token assignment check

# Impersonation Security Level

🔒 User DESKTOP-KHV61TH\user - TokenId 00000000-0088...

Main Details Groups Privileges Default Dacl Misc Operations Security

Token

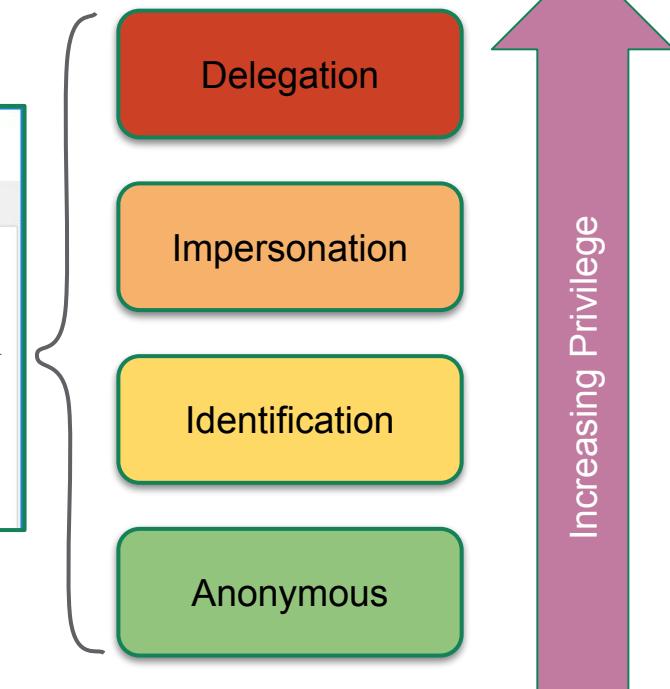
User: DESKTOP-KHV61TH\user

User SID: S-1-5-21-2696641275-1856431857-3807057343-1000

Token Type: Impersonation

Impersonation Level: Delegation

The screenshot shows a window with user information. The 'Impersonation Level' field is highlighted with an orange box. An arrow points from this field to a vertical stack of four colored boxes: orange (Delegation), yellow (Impersonation), green (Identification), and light green (Anonymous). To the right of this stack is a large purple arrow pointing upwards, labeled 'Increasing Privilege' vertically.



# Token Duplication

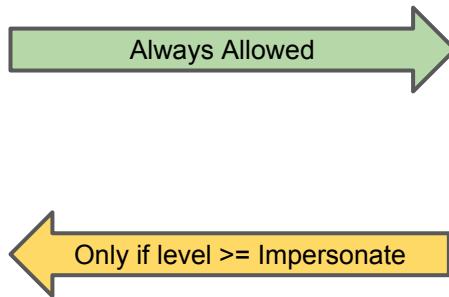
Can duplicate token's between types if have *Duplicate* access on handle.  
Can specify token type and impersonation security level.

powershell.exe:6968 - User DESKTOP-KHV61TH\user - To... — □ X

Main Details Groups Privileges Default Dacl Misc Operations Security

Token	User:	DESKTOP-KHV61TH\user
User SID:	S-1-5-21-2696641275-1856431857-3807057343-1000	
Token Type:	Primary	
Impersonation Level:	N/A	
Token ID:	00000000-008801F7	
Authentication ID:	00000000-00824B60	
Origin Login ID:	00000000-000003E7	
Modified ID:	00000000-0087E3C9	
Integrity Level:	Medium	Set Integrity Level
Session ID:	3	
Elevation Type:	Limited	Linked Token
Is Elevated:	False	
Source		
Name:	User32	
Id:	00000000-00824B18	

Primary



User DESKTOP-KHV61TH\user - TokenId 00000000-0088... — □ X

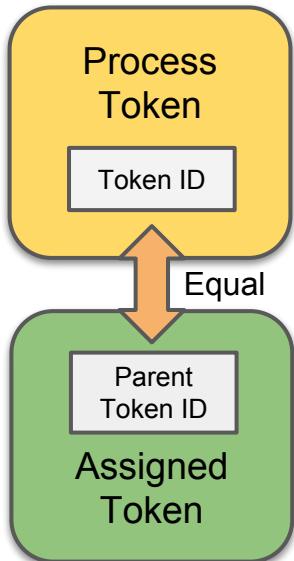
Main Details Groups Privileges Default Dacl Misc Operations Security

Token	User:	DESKTOP-KHV61TH\user
User SID:	S-1-5-21-2696641275-1856431857-3807057343-1000	
Token Type:	Impersonation	
Impersonation Level:	Delegation	
Token ID:	00000000-008FF34	
Authentication ID:	00000000-00824B60	
Origin Login ID:	00000000-000003E7	
Modified ID:	00000000-0087E3C9	
Integrity Level:	Medium	Set Integrity Level
Session ID:	3	
Elevation Type:	Limited	Linked Token
Is Elevated:	False	
Source		
Name:	User32	
Id:	00000000-00824B18	

Impersonation

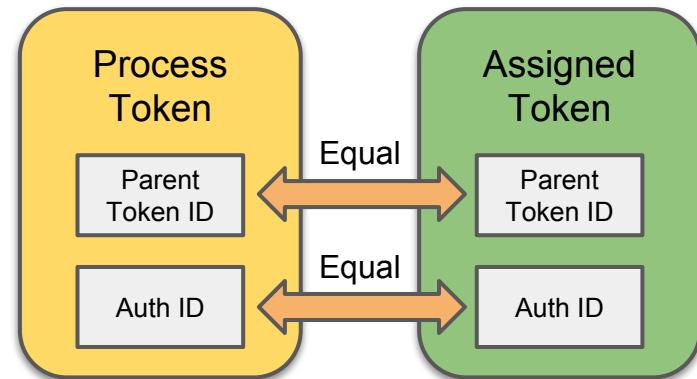
# Assigning a Token to a Process

Parent Token



OR

Sibling Token



NtFilterToken

NtDuplicateToken  
NtCreateLowBoxToken

# Setting an Impersonation Token

- Impersonation assigns a token to a thread, replaced the token used in access checks for the majority of system calls

## *Direct Setting*

`SetThreadToken()`  
`ImpersonateLoggedOnUser()`  
`NtSetInformationThread(...)`

## *Indirect Setting*

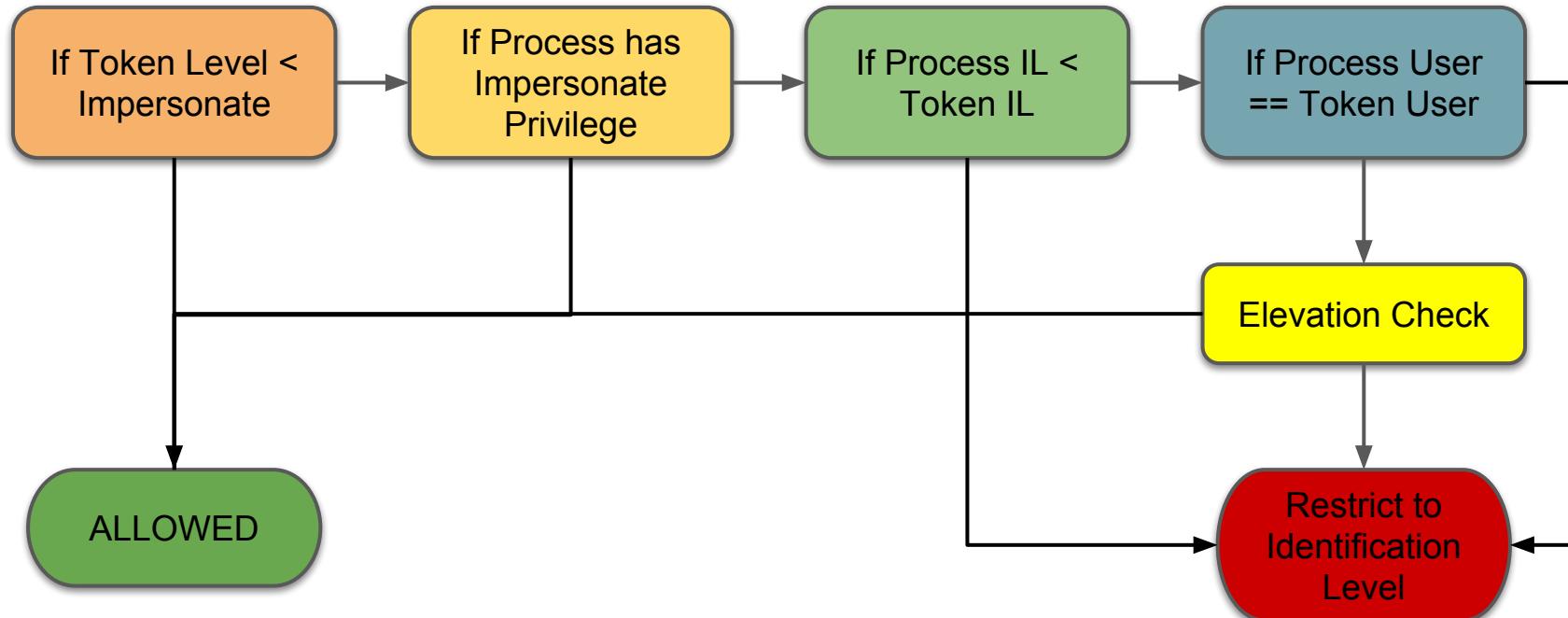
`ImpersonateNamedPipeClient()`  
`RpcImpersonateClient()`  
`ColImpersonateClient()`

## *Kernel Setting*

`PsiImpersonateClient()`  
`SelImpersonateClient/Ex()`

# Impersonation Token Security

PsImpersonateClient(...) ► SeTokenCanImpersonate(...)



Get the current process's token.

```
PS> $token = Get-NtToken
```

Gets the primary token from a process object.

```
PS> $token = Get-NtToken -Primary -Process $proc
```

Enable a privilege on the current process token.

```
PS> Set-NtTokenPrivilege SeDebugPrivilege
```

Duplicate a token as an impersonation token.

```
PS> Copy-NtToken Impersonation -Token $token
```

View the token in a GUI

```
PS> Show-NtToken $token
```

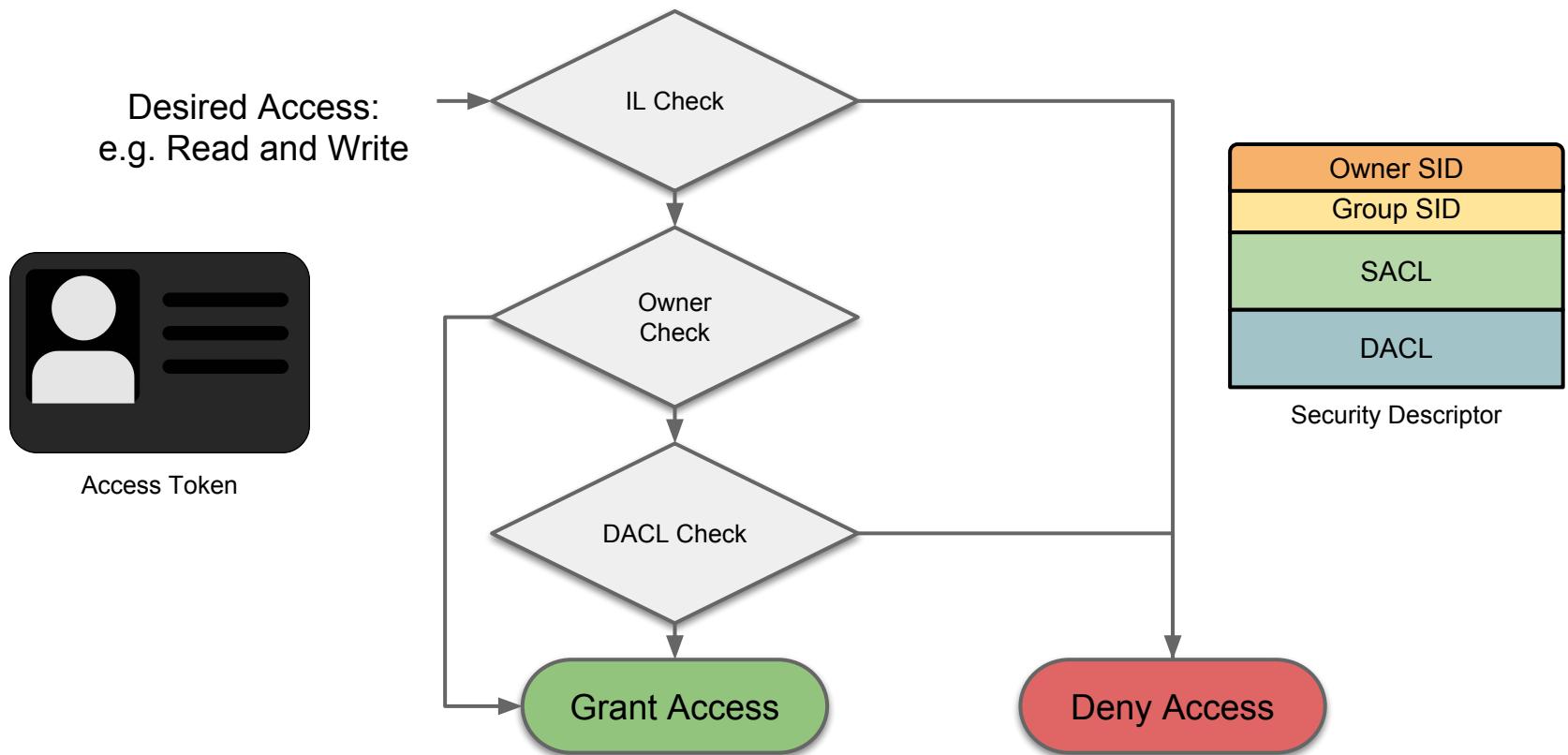
Impersonate a token and run a function.

```
PS> Invoke-NtToken $token { ... Do something }
```

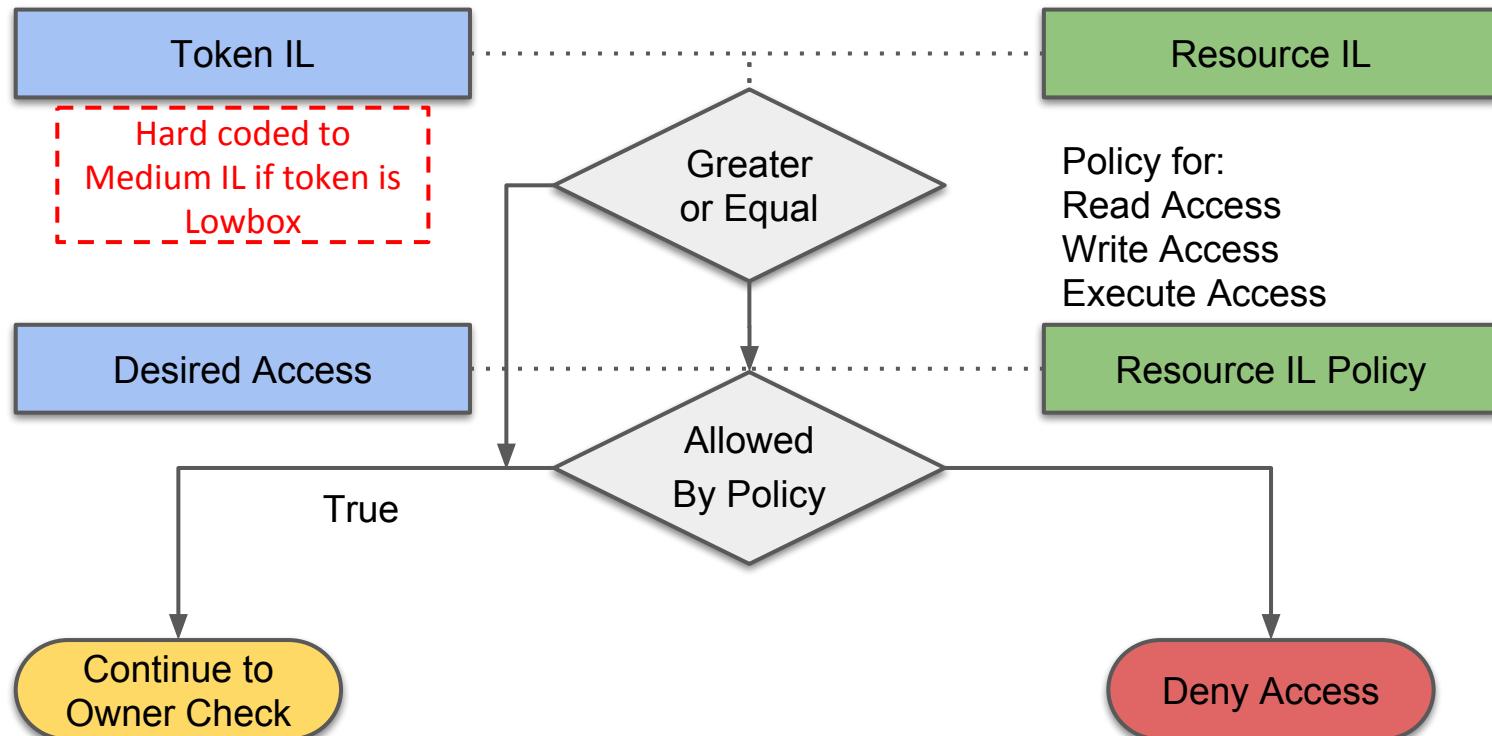
# WORKSHOP 5

## Access Tokens

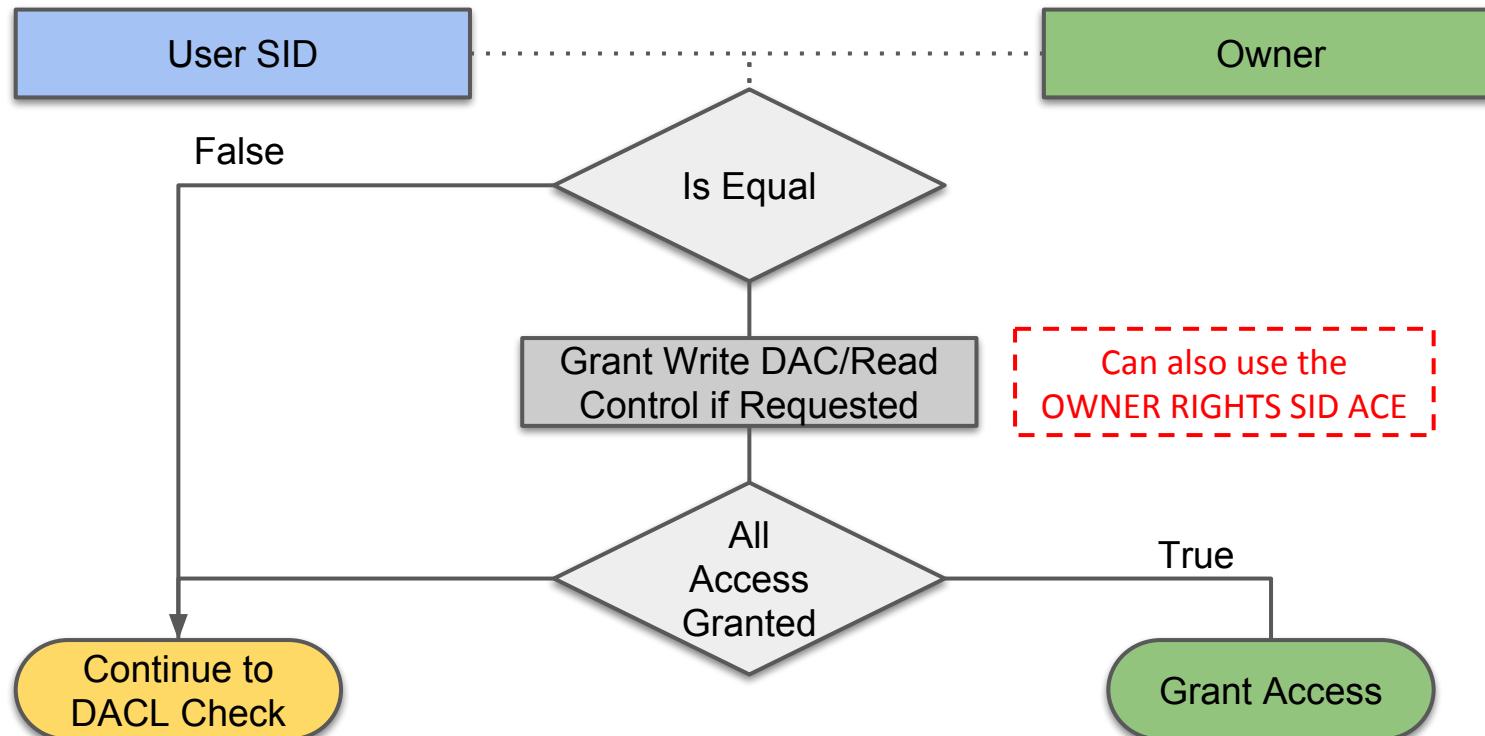
# Security Access Check (SeAccessCheck)



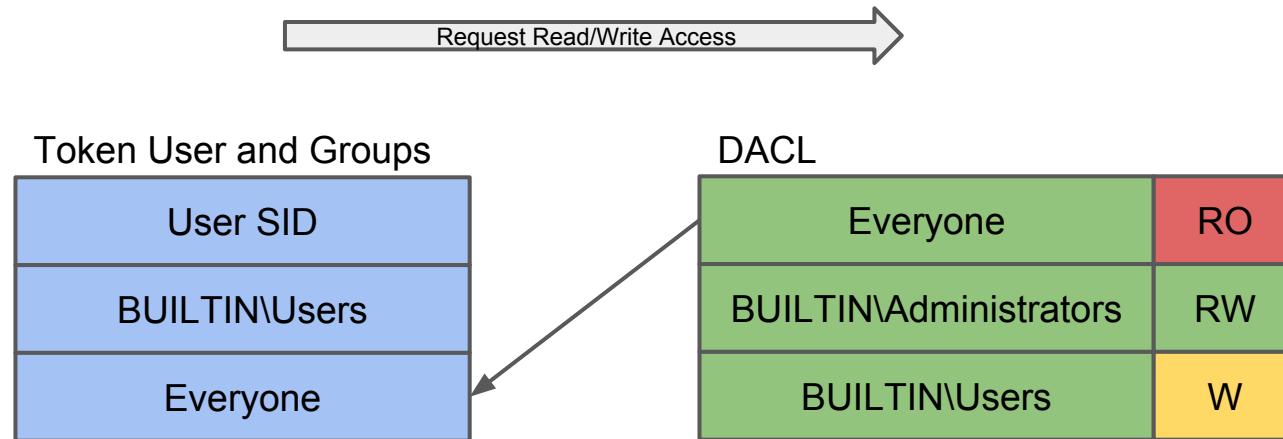
# Mandatory Integrity Level Check



# Owner Check

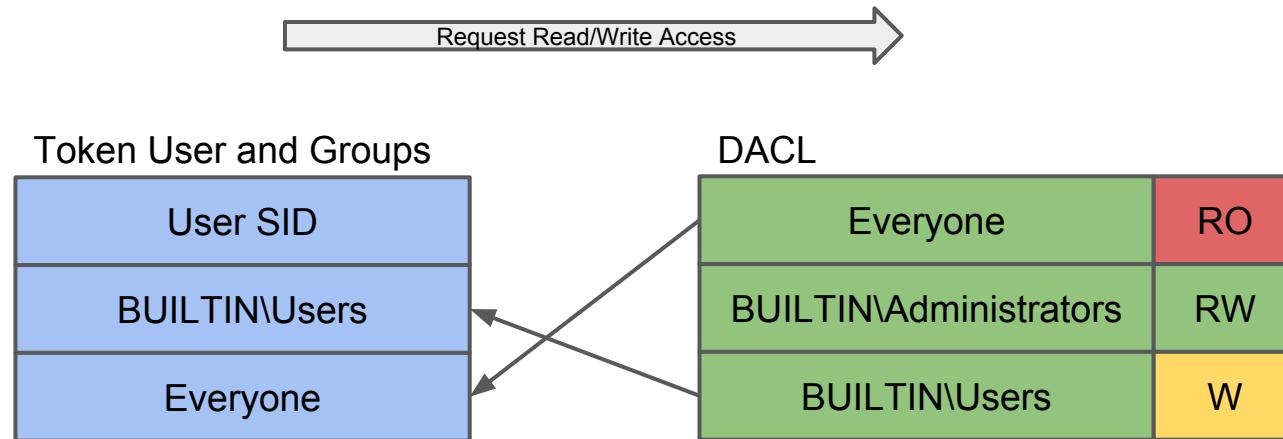


# Kernel DACL Check



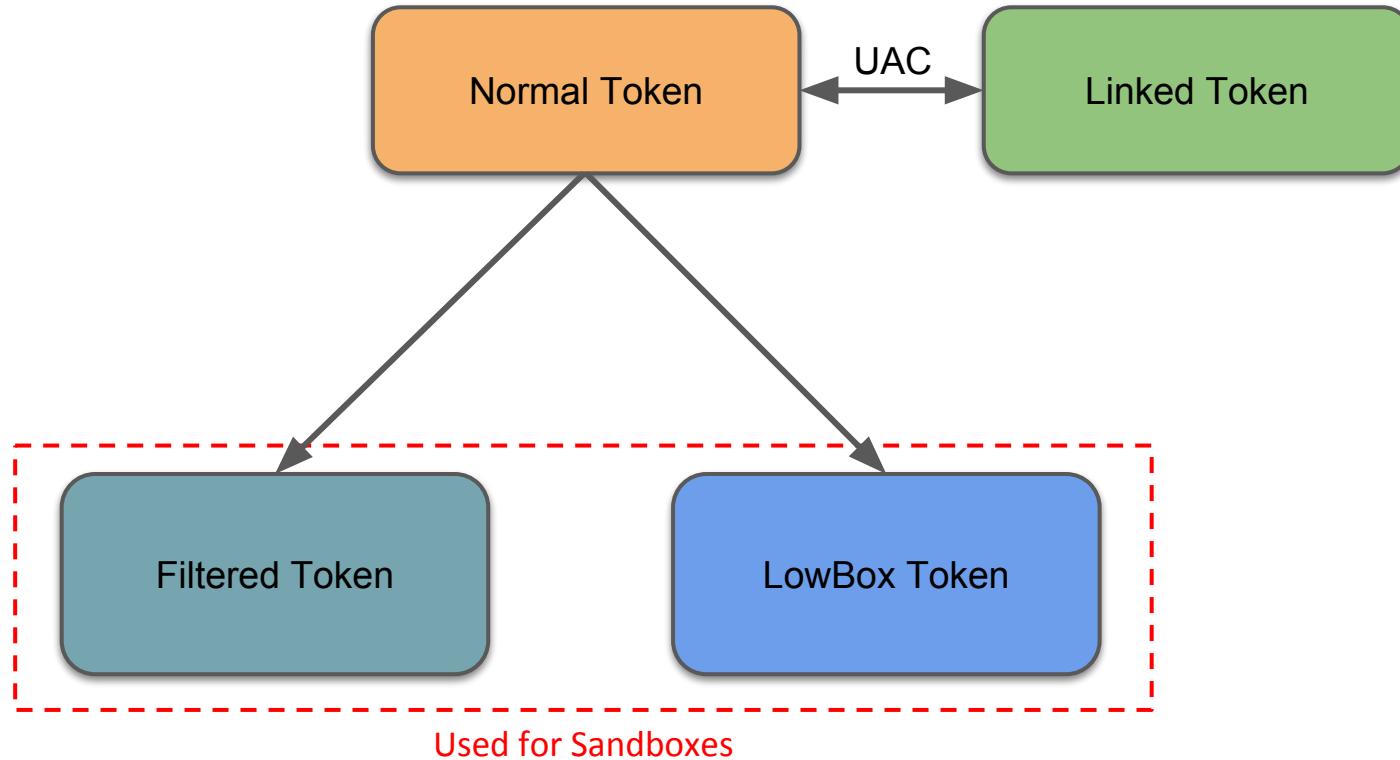
Current Granted Access: Read Only

# Kernel DACL Check



Final Granted Access: Read/Write

# Sandbox Tokens



# Restricted Token

User DESKTOP-KHV61TH\user - TokenId 00000000-0127...

Main Details	Groups	Privileges	Restricted SIDs	Default Dacl	Misc	Operations	Security
Name			Flags				
BUILTIN\Administrators			UseForDenyOnly				
BUILTIN\Users			Mandatory, Enabled				
CONSOLE LOGON			UseForDenyOnly				
DESKTOP-KHV61TH\None			UseForDenyOnly				
DESKTOP-KHV61TH\user			None				
Everyone			Mandatory, Enabled				
LOCAL			UseForDenyOnly				
NT AUTHORITY\Authenticated Users			UseForDenyOnly				
NT AUTHORITY\INTERACTIVE			Mandatory, Enabled				
NT AUTHORITY\Local account			UseForDenyOnly				
NT AUTHORITY\Local account and member of Administrators group			UseForDenyOnly				
NT AUTHORITY\LogonSessionId_0_8538852			Mandatory, Enabled, Logo				
NT AUTHORITY\NTLM Authentication			UseForDenyOnly				
NT AUTHORITY\This Organization			UseForDenyOnly				

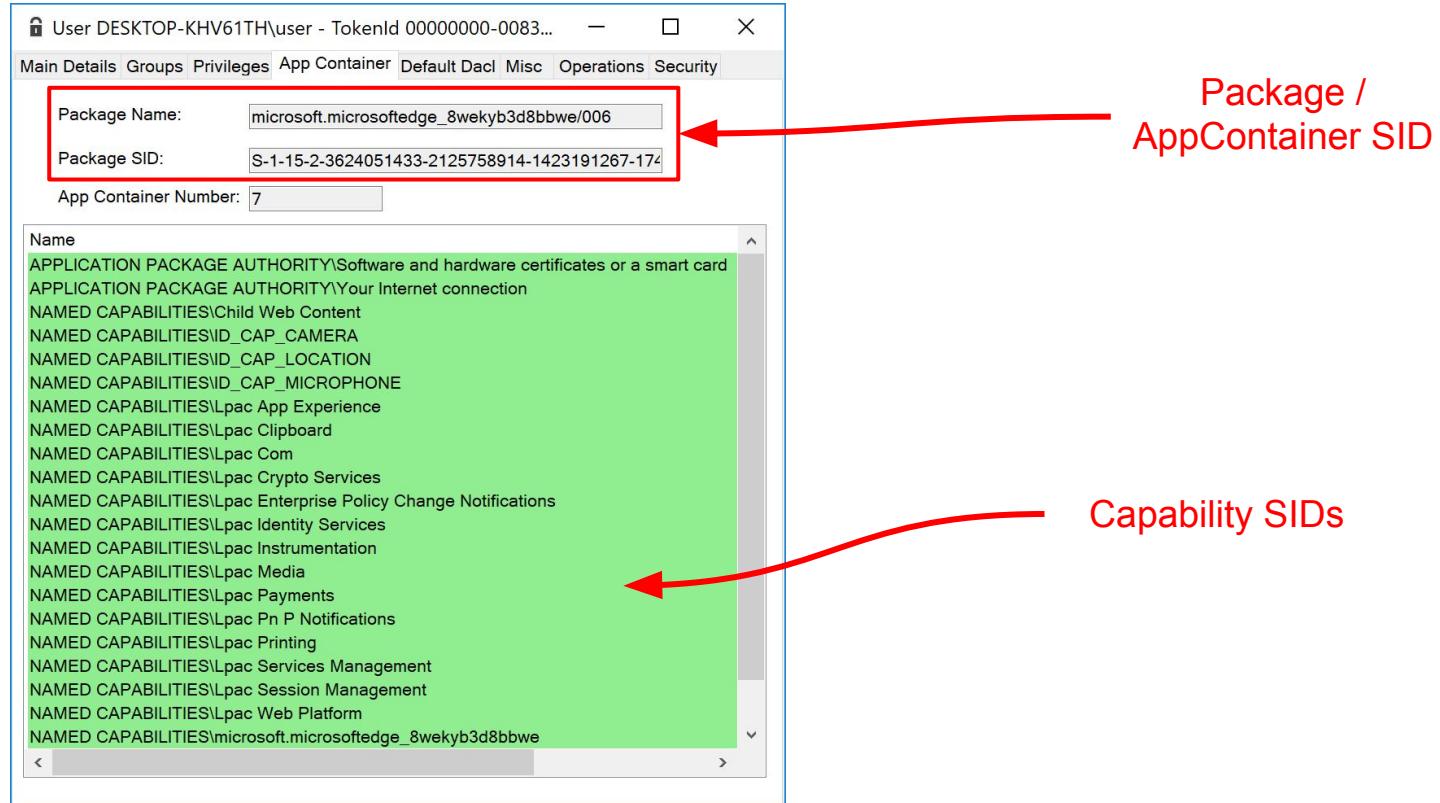
Changing Groups  
to Deny Only

User DESKTOP-KHV61TH\user - TokenId 00000000-0127...

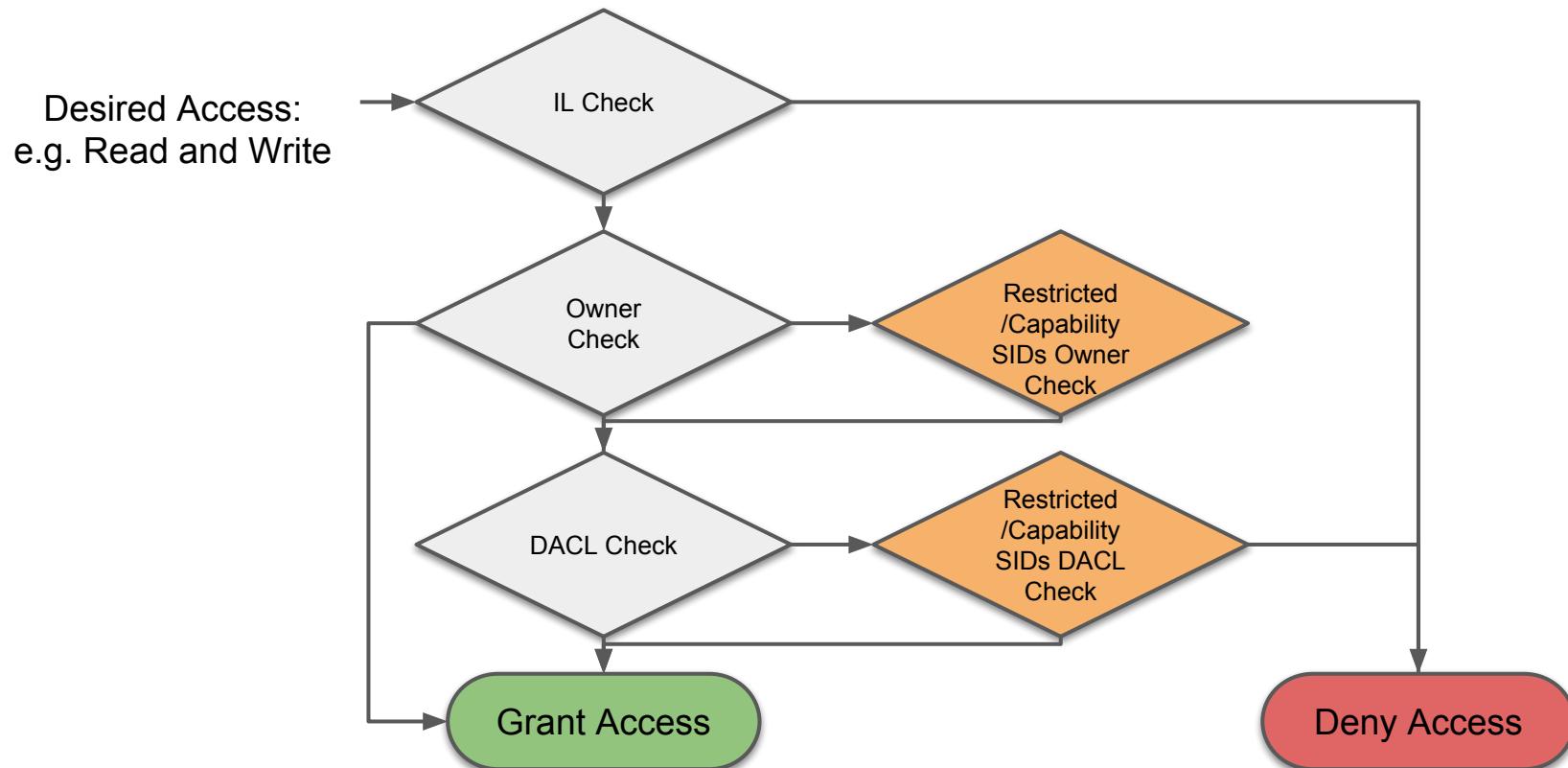
Main Details	Groups	Privileges	Restricted SIDs	Default Dacl	Misc	Operations	Security
Name			Flags				
BUILTIN\Users			Mandatory, Enabled				
Everyone			Mandatory, Enabled				
NT AUTHORITY\LogonSessionId_0_8538852			Mandatory, Enabled				
NT AUTHORITY\RESTRIC			Mandatory, Enabled				

Adding Restricted  
SIDs

# LowBox Token



# Restricted/Lowbox Token Access Check



Get a list of process objects with sandbox tokens.

```
PS> $ps = Get-NtProcess -FilterScript `  
    {$_.IsSandboxToken}
```

Create a LowBox token from the current token.

```
PS> Get-NtToken -LowBox -PackageSid "PackageName"
```

Create a filtered token with two restricted SIDs

```
PS> Get-NtToken -Filtered -RestrictedSids WD,BA
```

Get the maximum granted access for an object for the current token.

```
PS> Get-NtGrantedAccess -Object $obj
```

Get the maximum granted access for an object using an opened token.

```
PS> Get-NtGrantedAccess -Object $obj -Token $token
```

# WORKSHOP 6

## Access Checking

# Registry Hives

- Configuration Manager is responsible for the Registry.
- Registry accessed through special root key in the object manager namespace.
- A registry hive is a loaded registry file, which is loaded into an attachment point using *NtLoadKey(Ex)* system calls.
- Four main attachment points in Windows 10 1803:
  - \Registry\Machine - Attachment point for system hives
  - \Registry\A - Attachment point for user loaded hives
  - \Registry\WC - Attachment point for containers, also used for desktop bridge
  - \Registry\User - Attachment point for user hives.

# Win32 to Native Registry Mapping

<b><i>Win32 Path</i></b>	<b><i>Native Path</i></b>	<b><i>Description</i></b>
HKEY_LOCAL_MACHINE	\Registry\Machine	System registry attachment point
HKEY_USERS	\Registry\User	User hive attachment points
HKEY_CURRENT_USER	\Registry\User\{SID}	User specific hive
HKEY_CLASSES_ROOT	Simulated hive	Merge of machine software hive and user's software hive.
HKEY_PERFORMANCE_DATA	Simulated	Performance information
HKEY_PERFORMANCE_TEXT	Simulated	Performance information

Open the machine software hive.

```
PS> $k = Get-NtKey -Path \Registry\Machine\Software
```

Query values from an open key.

```
PS> $k.QueryValues()
```

Open the user's software hive using a Win32 style path

```
PS> $k = Get-NtKey -Win32Path HKCU\Software
```

Recurse child keys and execute a visitor pattern.

```
PS> Get-NtKeyChild $k -MaxDepth 4 -Recurse `  
-Visitor { Write-Host "$($_.FullPath)" }
```

Load a per-user registry hive.

```
PS> Add-NtKey "\?\?\c:\temp\reg.hiv" -KeyPath `  
\Registry\A\ABC -LoadFlags AppKey
```

# WORKSHOP 7

## Accessing Registry Keys

# Device Objects

- Created by loaded kernel drivers using *IoCreateDevice* kernel API.

The screenshot shows two PowerShell sessions in a window titled "Windows PowerShell".

The top session shows the output of the command `Get-Item NtObject:\Device\NamedPipe`. It displays a table with one row:

Name
-----
NamedPipe

A red box highlights the "Name" column. A red arrow points from the text "Registered object is type Device" to the "Device" entry in the table.

The bottom session shows the output of the command `$f = Get-NtFile \Device\NamedPipe`, followed by `$f.NtType`. It displays a table with one row:

Name
-----
File

A red box highlights the "Name" column. A red arrow points from the text "Opened object is type File" to the "File" entry in the table.

```
PS C:\> Get-Item NtObject:\Device\NamedPipe
Name
-----
NamedPipe

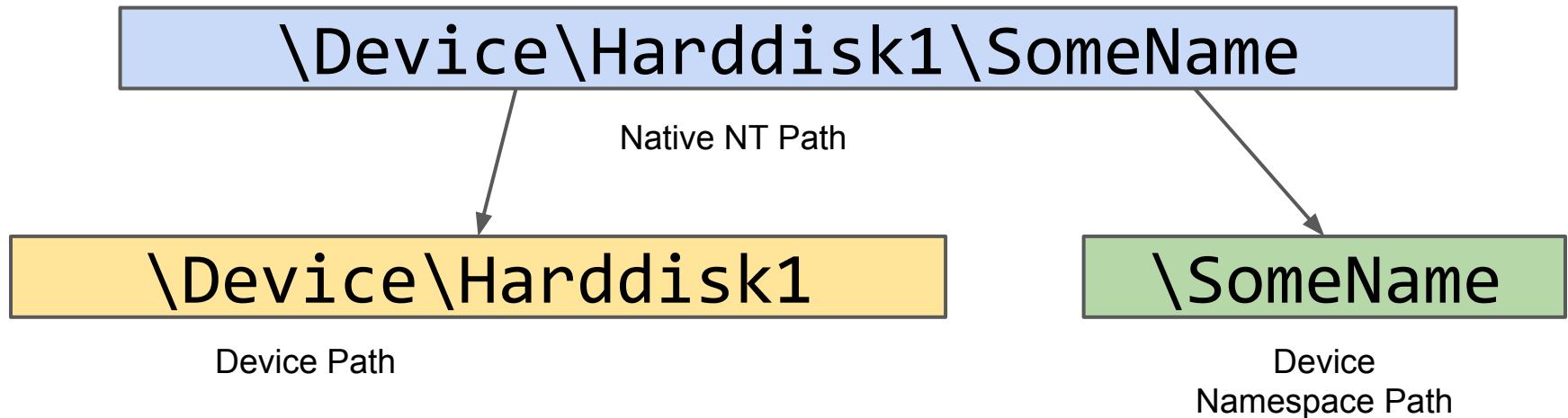
PS C:\> $f = Get-NtFile \Device\NamedPipe
PS C:\> $f.NtType
Name
-----
File
```

# Opening a Device Name

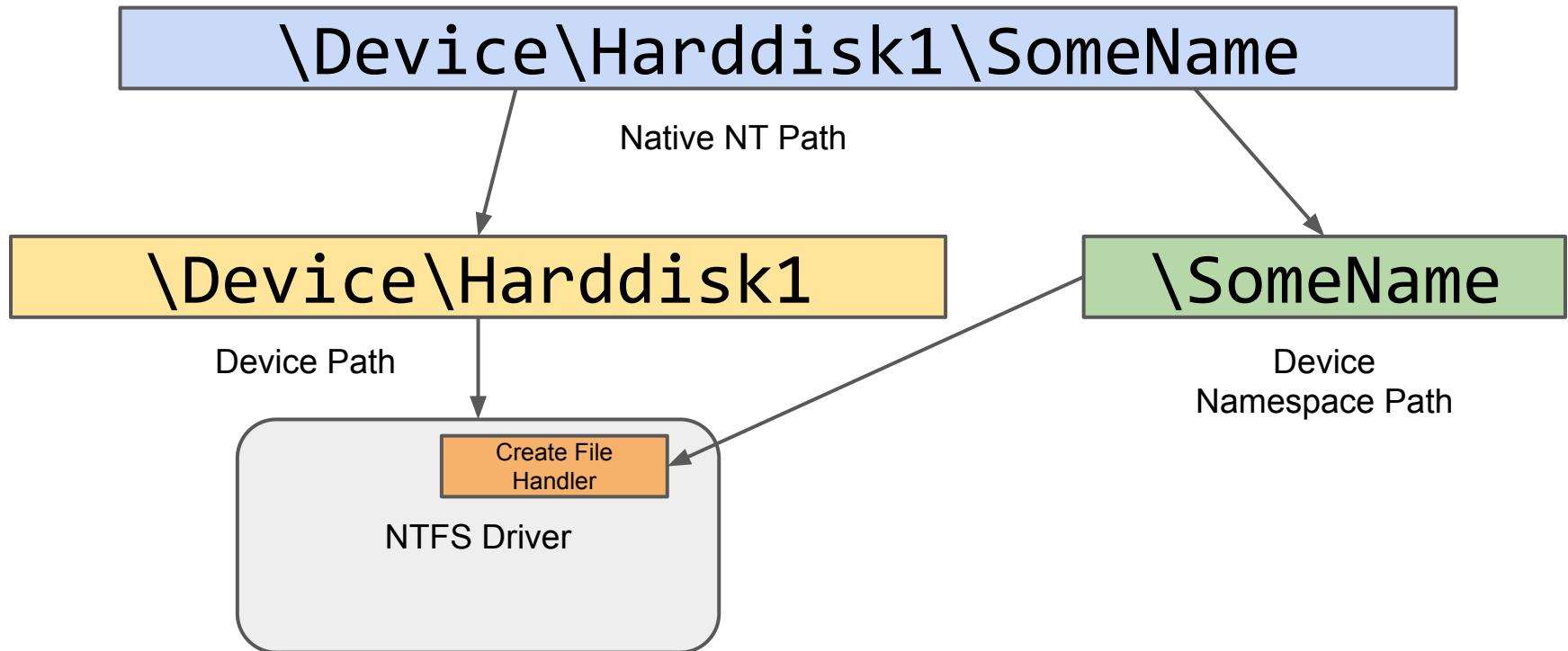
\Device\Harddisk1\SomeName

Native NT Path

# Opening a Device Name



# Opening a Device Name



# Device and File Security

- Kernel by default only secures the Device Object, not the file path.
- Can specify FILE\_DEVICE\_SECURE\_OPEN to IoCreateDevice to enforce Device SD on the file path.

```
Windows PowerShell
PS C:\> $dev = Get-NtFile \Device\NamedPipe
PS C:\> $dev.SecurityDescriptor.ToSddl()
0:BAG:SYD:(A;;0x1201bf;;;WD)(A;;FA;;;SY)(A;;FA;;;BA)(A;;0x12
00a9;;;RC)S:AI(ML;;NW;;;LW)
PS C:\>
PS C:\> $file = Get-NtFile \Device\NamedPipe\atsvc
PS C:\> $file.SecurityDescriptor.ToSddl()
0:SYG:SYD:(A;;0x12019b;;;WD)(A;;0x12019b;;;AN)(A;;FA;;;S-1-5
-80-4125092361-1567024937-842823819-2091237918-836075745)(A;
;RC;;;OW)
```

Security descriptors differ

# Win32 Path Support

Path	Description
some\path	Relative path to current directory
c:\some\path	Absolute directory
c:some\path	Drive relative path
\.\c:\some\path	Device path, canonicalized
\?\c:\some\path	Device path, non-canonicalized
\server\share\path	UNC path to share on server

# Canonicalization

- Type of Win32 path affects canonicalization behaviour

Path	Result of Canonicalization
c:\path\../badgers	c:\badgers
c:\..\d:/badgers	c:\d:\badgers
\.\c:\path\../badgers	c:\badgers
\.\c:\..\d:/badgers	\.\d:\badgers (WTF!)
\?\c:\path\../badgers	\?\c:\path\../badgers

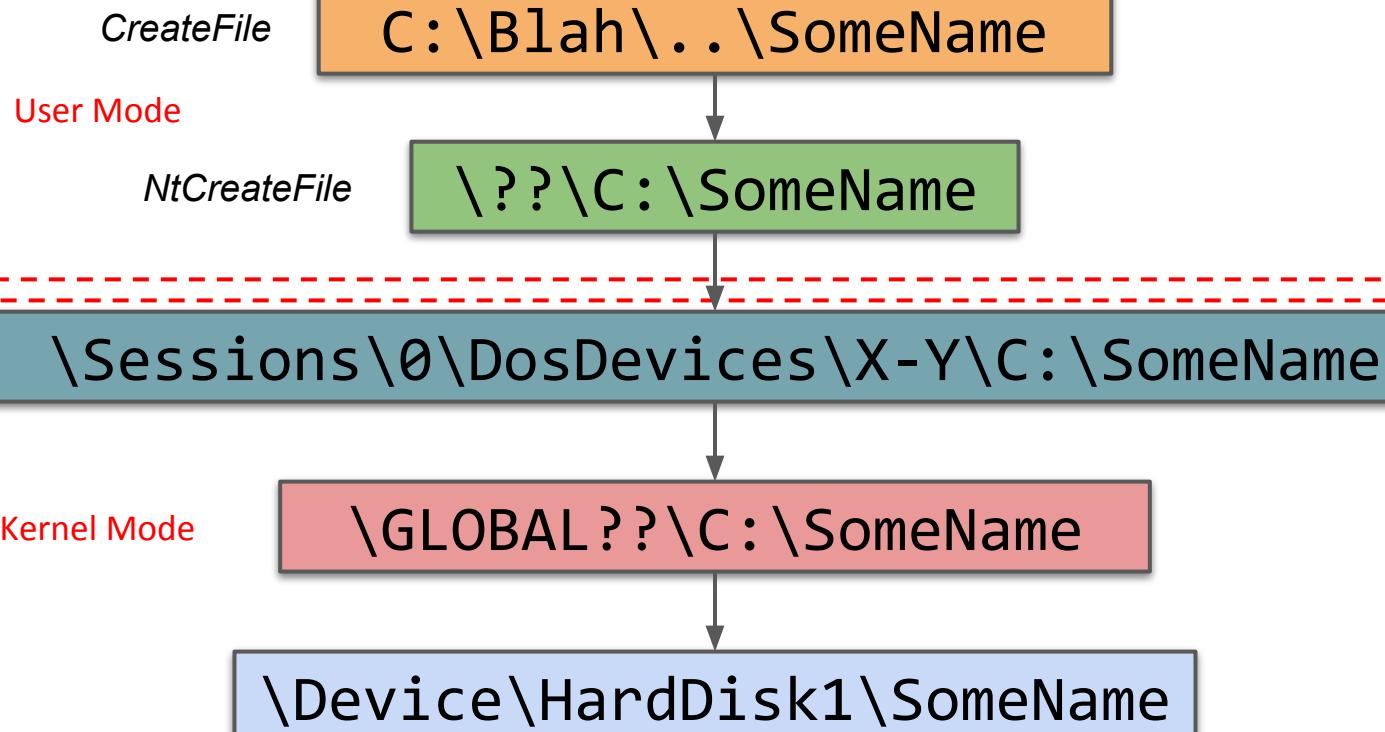
# NTFS Invalid Characters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	-
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

# NTFS File Sharing and Locking

- File system calls take a sharing mode flag, determines whether a second open will succeed.
- Has three flags:
  - FILE\_SHARE\_READ Subsequent opens can be for read access
  - FILE\_SHARE\_WRITE Subsequent opens can be for write access
  - FILE\_SHARE\_DELETE Subsequent opens can be for delete access
- Not specifying any share flags means file can't be reopened
- Some access such as *ReadAttributes* or *WriteEa* are allowed regardless of the share mode.

# Opening a File From User Mode



# File Options and Disposition

```
PS> New-NtFile -Path "\??\c:\blah" `  
-Options <OPTIONS> -Disposition <DISPOSITION>
```

**DirectoryFile** : Open/Creates a directory

**NonDirectoryFile** : Open a non-directory file?

**OpenForBackupIntent**: Combined with

SeBackupPrivilege or SeRestorePrivilege to bypass  
access checks

**SynchronousIoNonAlert** - Synchronous file access.

**Create**: Create file if it doesn't exist.

**Open**: Open file if it exists.

**OpenIf**: Open file if it exists, create if not.

**Overwrite**: Overwrite file if it exists

**OverwriteIf**: Overwrite file if it exists, create if not.

**Supersede**: Replace file.

Open the windows folder.

```
PS> $f = Get-NtFile -Path "\??\c:\windows" `  
-ShareMode Read
```

Open the windows folder using a Win32 style path

```
PS> $f = Get-NtFile -Win32Path "c:\windows"
```

Resolve a Win32 path to an NT file

```
PS> Get-NtFilePath -Path "...\\..\\ABC"
```

Resolve a Win32 path to an NT file

```
PS> Get-NtFilePathType -Path "X:Y"
```

Create a directory.

```
PS> New-NtFile "\??\c:\dir" -Options NonDirectoryFile
```

# Driver Device Creation

```
NTSTATUS DriverEntry(  
    PDRIVER_OBJECT DO, ...){  
  
    IoCreateDevice(DriverObject,  
                  FILE_DEVICE_TYPE_DISK  
                  FILE_DEVICE_SECURE_OPEN);  
  
    DO->MajorFunction[IRP_MJ_CREATE] = CreateFunction;  
    DO->MajorFunction[IRP_MJ_CLOSE] = CloseFunction;    IRP Handlers  
    DO->MajorFunction[IRP_MJ_CLEANUP] = CleanupFunction;  
    DO->MajorFunction[IRP_MJ_FILE_SYSTEM_CONTROL] = FSControlFunction;  
    DO->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IoControlFunction;  
  
    return status;  
}
```

Device Characteristics:

SECURE\_OPEN = 0x00100

ALLOW\_APPCONTAINER\_TRAVERSAL = 0x20000

# IO Request Packet (IRP) Handlers

```
NTSTATUS DeviceIoControlFunction(PDEVICE_OBJECT, PIRP Irp) {
    PIO_STACK_LOCATION io_stack = IoGetCurrentIrpStackLocation(Irp);

    ULONG ctl_code      = io_stack->Parameters.DeviceIoControl.IoControlCode;
    PVOID buffer        = Irp->AssociatedIrp.SystemBuffer;
    ULONG input_length  = io_stack->Parameters.DeviceIoControl.InputBufferLength;
    ULONG output_length = io_stack->Parameters.DeviceIoControl.OutputBufferLength;

    // Process IOCTL.
}
```

# IO Control Codes

- Windows supports typical IOCTL pattern through NtDeviceIoControlFile and NtFsControlFile system calls.
- Control Code encodes what permissions the device handle needs to call and includes memory handling flags.



`FILE_ANY_ACCESS = 0`  
`FILE_READ_ACCESS = 1`  
`FILE_WRITE_ACCESS = 2`

`METHOD_BUFFERED = 0`  
`METHOD_IN_DIRECT = 1`  
`METHOD_OUT_DIRECT = 2`  
`METHOD_NEITHER = 3`

Get the device type and characteristics for an open file.

```
PS> $f.DeviceType; $f.Characteristics
```

Get an IO control code from a number.

```
PS> Get-NtIoControlCode 0x98330
```

Get an IO control code from a number and try and lookup name.

```
PS> Get-NtIoControlCode 0x98330 -LookupName
```

Send an FS control code, 3 bytes in with max 10 bytes out

```
PS> $f.FsControl($code, [byte[]]@(1, 2, 3), 10)
```

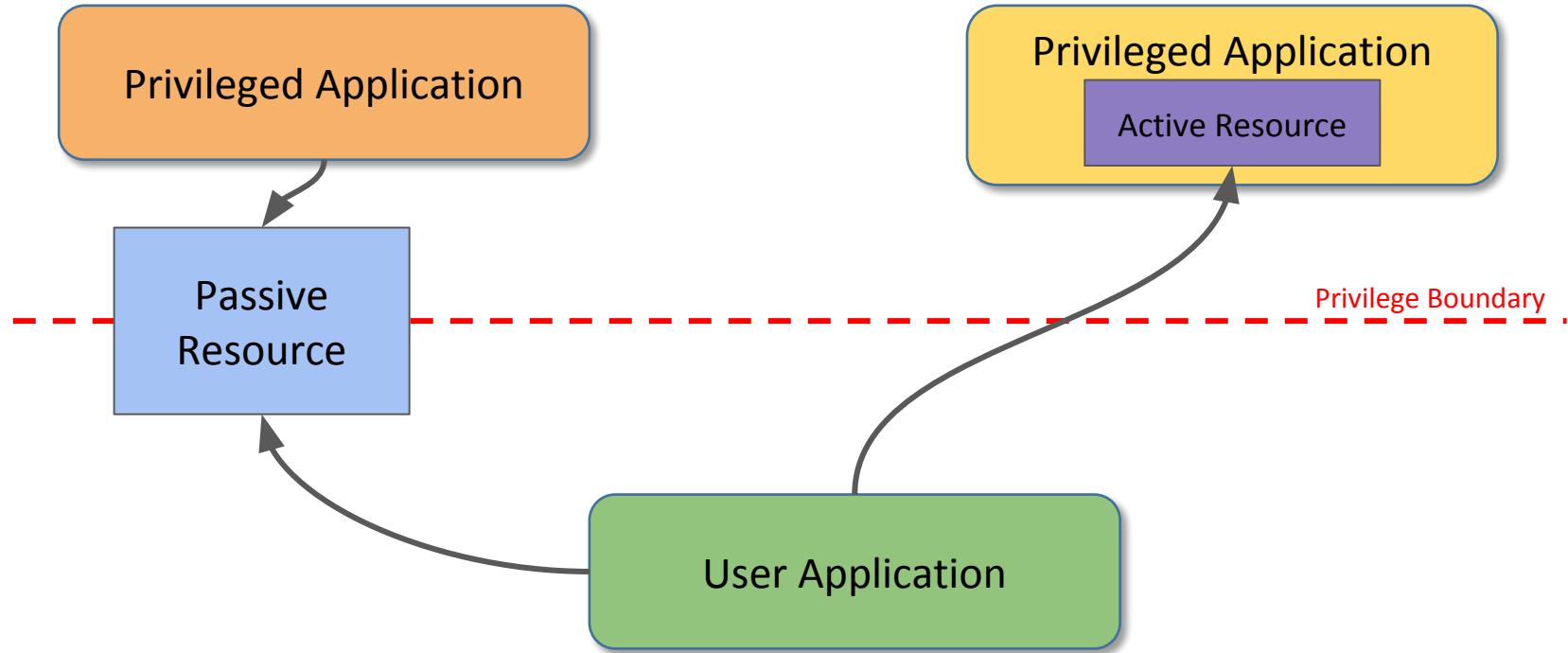
# WORKSHOP 8

## Analysing Windows IO

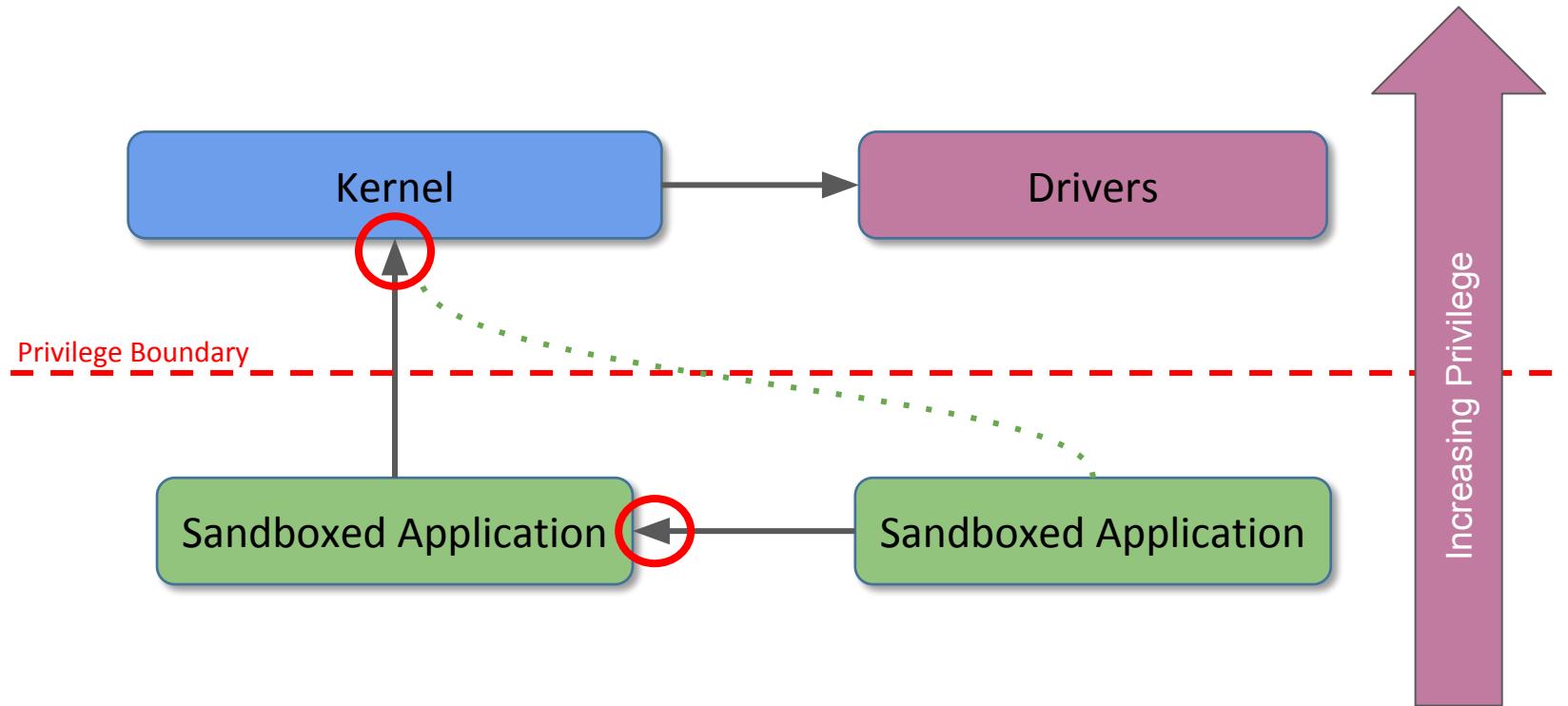
# PART 2

# Attack Surface Analysis

# Passive vs Active Attack Surface



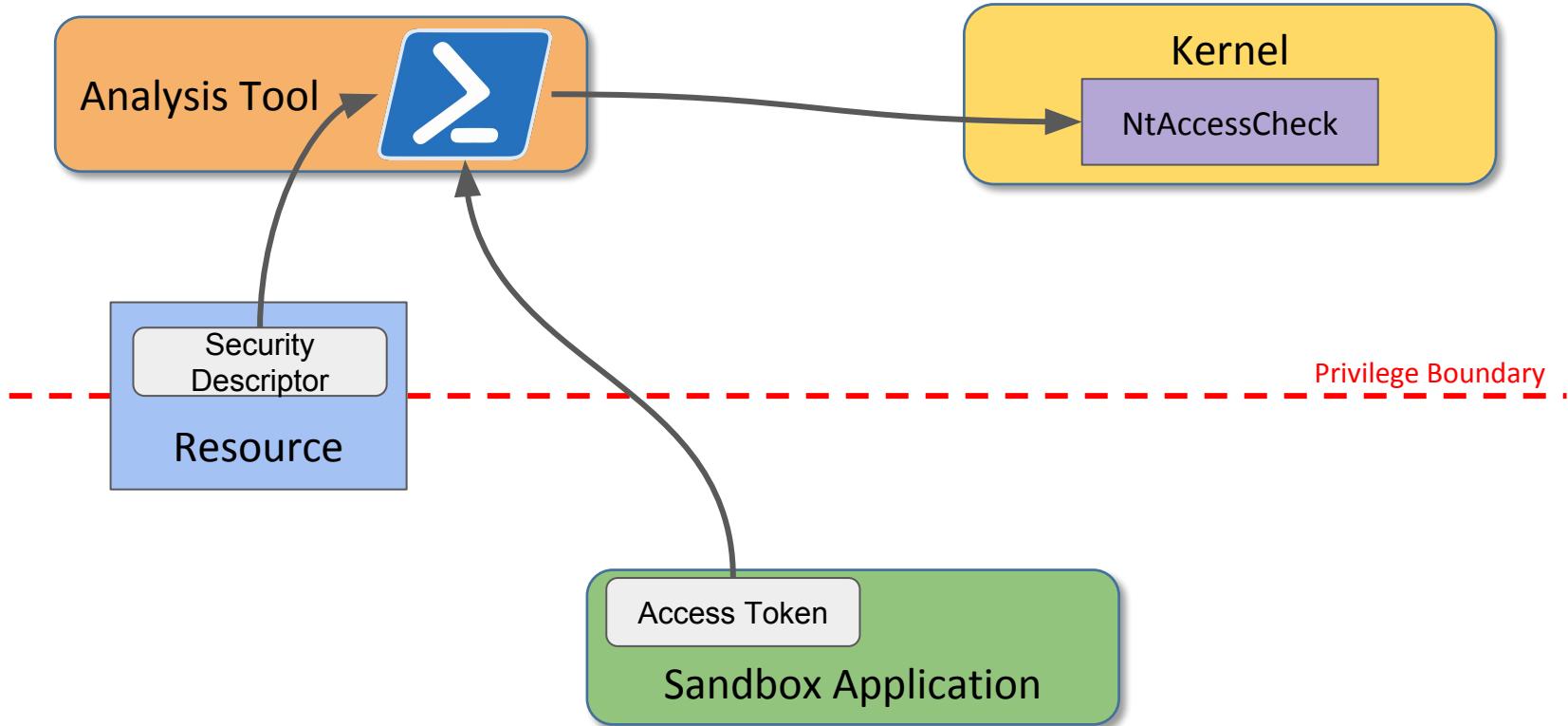
# Don't Always Think of Going Up



# Probing Accessible Resources

- Good idea to determine levels of attack surface is to probe what resources you can access from your desired privilege level.
- Primarily interested in WRITE, but in some cases (such as processes) READ is also important.
- This could include:
  - Files
  - Registry Keys
  - Processes and Threads
  - Sections/File Mappings
  - Kernel Driver Device Objects
  - Named Pipes
  - Services

# Analysis Checking Implementation



# Sandbox Attack Surface Analysis Cmdlets

Tool Name	Description
Get-AccessibleFile	Enumerate accessible files or named pipes
Get-AccessibleProcess	Enumerate accessible processes and/or threads
Get-AccessibleDevice	Enumerate accessible device objects
Get-AccessibleKey	Enumerate accessible registry keys
Get-AccessibleObject	Enumerate accessible names kernel resources
Get-AccessibleService	Enumerate accessible services.

Cmdlets take a number of common arguments:

- ProcessIds PID : Specify a list of PIDs to impersonate when doing the access check
- Recurse : Recursively enumerate names resources
- AccessRights ACCESS: Comma separated list of access rights to check for
- AllowPartialAccess : Allow partial access rights to match

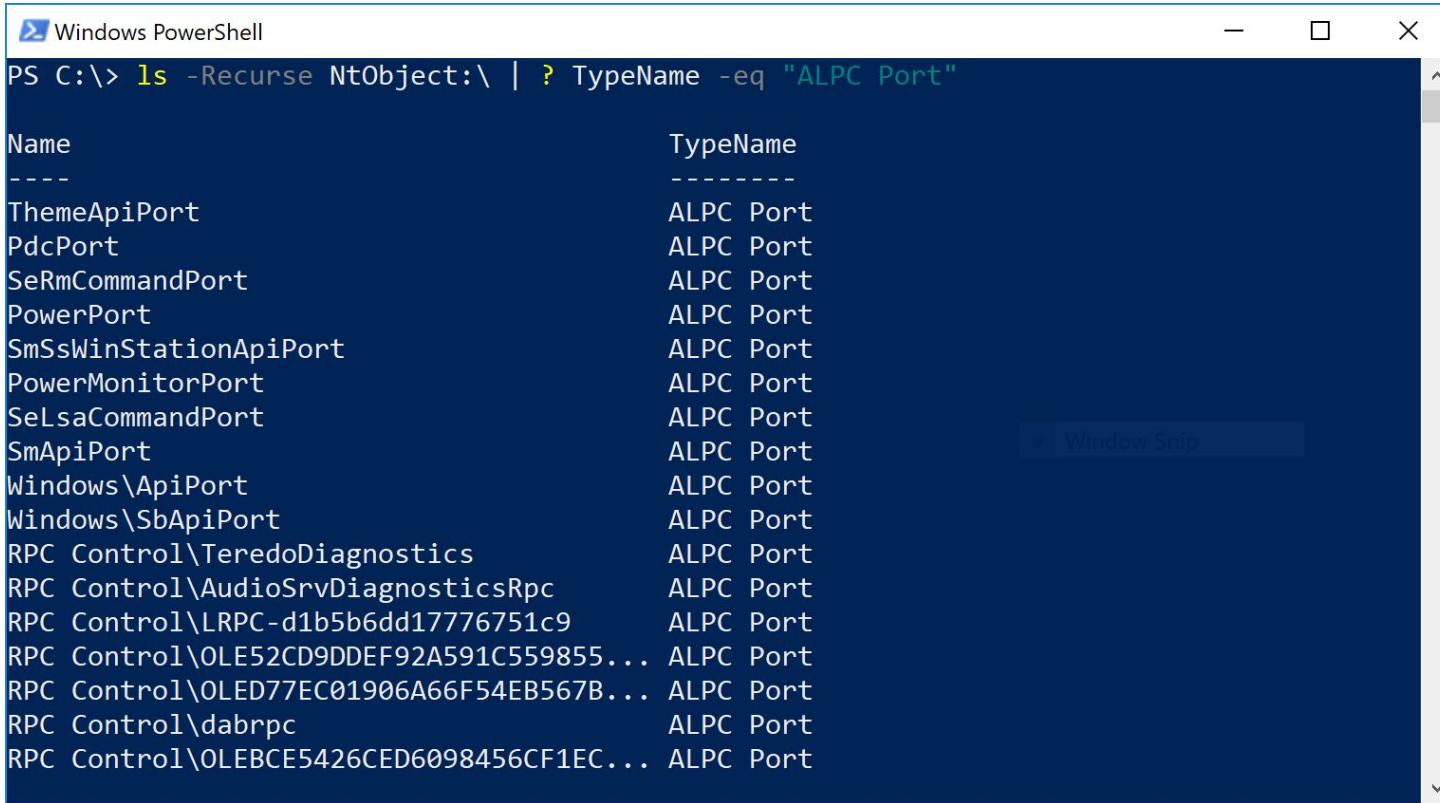
# WORKSHOP 9

## Finding and Analysing Resource Access

# RPC Services

- The most common technique on Windows to provide privilege separation between components.
- Used in many common services:
  - Local Security Subsystem (LSASS)
  - AppInfo service (UAC)
  - Secondary Logon service (seclogon)
- Many RPC services are undocumented and contain complex functionality
- Local RPC usually exposed over ALPC
- RPC can also be accessible over Named Pipes

# ALPC Ports



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command run is `PS C:\> ls -Recurse NtObject:\ | ? TypeName -eq "ALPC Port"`. The output is a table showing the names and types of various ALPC ports:

Name	TypeName
ThemeApiPort	ALPC Port
PdcPort	ALPC Port
SeRmCommandPort	ALPC Port
PowerPort	ALPC Port
SmSsWinStationApiPort	ALPC Port
PowerMonitorPort	ALPC Port
SeLsaCommandPort	ALPC Port
SmApiPort	ALPC Port
Windows\ApiPort	ALPC Port
Windows\SbApiPort	ALPC Port
RPC Control\TeredoDiagnostics	ALPC Port
RPC Control\AudioSrvDiagnosticsRpc	ALPC Port
RPC Control\LRPC-d1b5b6dd17776751c9	ALPC Port
RPC Control\OLE52CD9DDEF92A591C559855...	ALPC Port
RPC Control\OLED77EC01906A66F54EB567B...	ALPC Port
RPC Control\dabrpcl	ALPC Port
RPC Control\OLEBCE5426CED6098456CF1EC...	ALPC Port

Get all accessible ALPC ports based on Process ID 1234

```
PS> Get-AccessibleAlpcPort -ProcessIds 1234
```

Get all objects for ALPC server ports.

```
PS> Get-NtAlpcServer
```

Get the object for an ALPC server port.

```
PS> Get-NtAlpcServer "\RPC Server\PortName"
```

Get list of all names ALPC port handles.

```
PS> Get-NtHandle -ObjectTypes "ALPC Port" | `  
? Name -ne "" | select ProcessId, Object, Name
```

# Network Data Representation (NDR)

- Server defined interface using an IDL file. Compiler converts to a server Stub build with NDR which handles marshaling of parameters and structures
- Client must have a corresponding Proxy built from the same IDL interface definition otherwise there's likely to be a mismatch.
- Each interface has a defined unique ID (UUID) and version.

```
[  
    uuid (201ef99a-7fa0-444c-9399-19ba84f12a1a),  
    version(1.0),  
]  
interface LaunchAdminProcess  
{  
    long RAiLaunchAdminProcess([in][unique][string] wchar_t* ExecutablePath);  
}
```

# Registering an Interface

```
RPC_SERVER_INTERFACE RpcServerInterface = {  
    {{0x4870536E, ...}, // UUID  
    {{0x8A885D04, ...}, // Syntax GUID  
    // ...  
};  
  
RpcServerRegisterIfEx(  
    (RPC_IF_HANDLE)&RpcServerInterface,  
    NULL, NULL,  
    Flags, // ← Registration flags, some security related:  
    RPC_C_LISTEN_MAX_CALLS_DEFAULT,  
    SecurityCallback // ← Optional security callback.  
);
```

Server registration structure (generally auto-generated)

Registration flags, some security related:  
RPC\_IF\_ALLOW\_CALLBACKS\_WITH\_NO\_AUTH  
RPC\_IF\_ALLOW\_LOCAL\_ONLY  
RPC\_IF\_ALLOW\_SECURE\_ONLY  
RPC\_IF\_SEC\_NO\_CACHE

# RPC Endpoints

- Configured on server using *RpcServerUseProtseqEp*
- Configured on client using *RpcStringBindingCompose*

```
RPC_STATUS RPC_ENTRY RpcServerUseProtseqEp(  
    unsigned char *Protseq,  
    unsigned int MaxCalls,  
    unsigned char *Endpoint,  
    void *SecurityDescriptor);
```

```
RPC_STATUS RPC_ENTRY RpcStringBindingCompose(  
    TCHAR *ObjUuid,  
    TCHAR *ProtSeq,  
    TCHAR *NetworkAddr,  
    TCHAR *EndPoint,  
    TCHAR *Options,  
    TCHAR **StringBinding);
```

Protocol  
sequence

Optional  
Endpoint  
Name

# Protocol Sequences

<i>Protocol Sequence</i>	<i>Optional Endpoint Name</i>	<i>Description</i>
ncalrpc	NAME	Local RPC (ALPC)
ncacn_np	\pipe\NAME	Windows Named Pipe
ncacn_ip_tcp	(port number)	TCP/IP
ncacn_ip_udp	(port number)	UDP/IP
ncacn_http	(port number)	HTTP

Note all endpoints and protocol sequences are multiplexed in a single process.

# RPC Security

## Connect Time Security (Local)

The screenshot shows the Windows Security dialog for a local connection. It displays the following information:

- Owner:** DESKTOP-KHV61TH\user
- Group:** DESKTOP-KHV61TH\None
- Integrity:** N/A
- DACL** (selected tab):
  - ACL Entries:** Shows three entries: "Allowed Everyone" (highlighted in blue), "Allowed NT AUTHORITY\ANONYMOUS LOGON" (highlighted in green), and "Allowed APPLICATION PACKAGE AUTHORITY\ALL APPLICATION".
  - Specific Access:** Shows checkboxes for various permissions:

Name	Access Mask
Connect	0x00000001
Delete	0x00010000
Read Control	0x00020000
Write Dac	0x00040000
Write Owner	0x00080000
Synchronize	0x00100000
- Edit** button at the bottom.

## Runtime Security

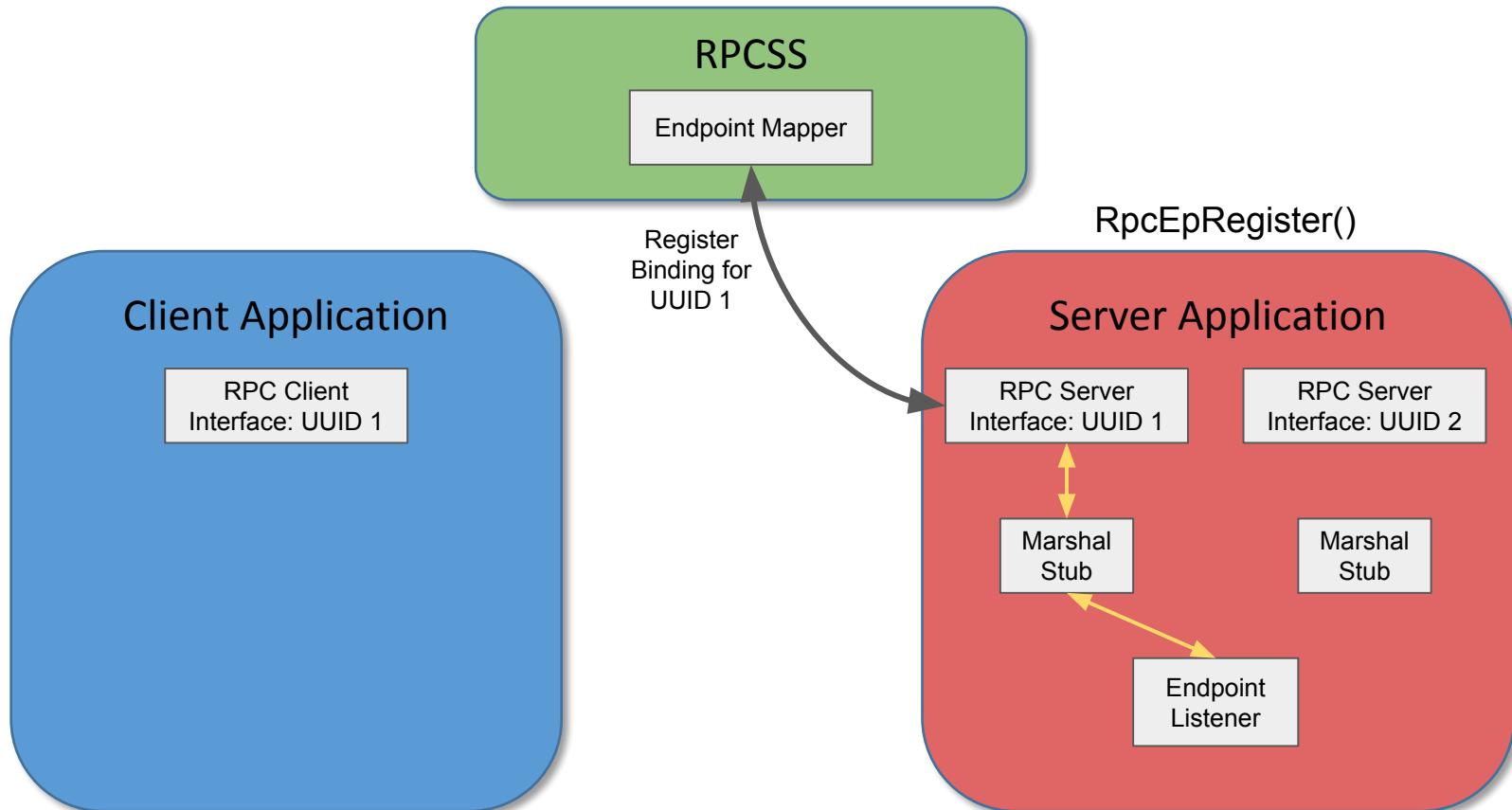
```
RPC_STATUS RPC_ENTRY RpcServerRegisterIf3(
    _In_     RPC_IF_HANDLE     IfSpec,
    _In_opt_ UUID              *MgrTypeUuid,
    _In_opt_ RPC_MGR_EPV       *MgrEpv,
    _In_     unsigned int      Flags,
    _In_     unsigned int      MaxCalls,
    _In_     unsigned int      MaxRpcSize,
    _In_opt_ RPC_IF_CALLBACK_FN *IfCallbackFn,
    _In_opt_ void              *SecurityDescriptor
);
```

Security callback, run code  
to verify the client

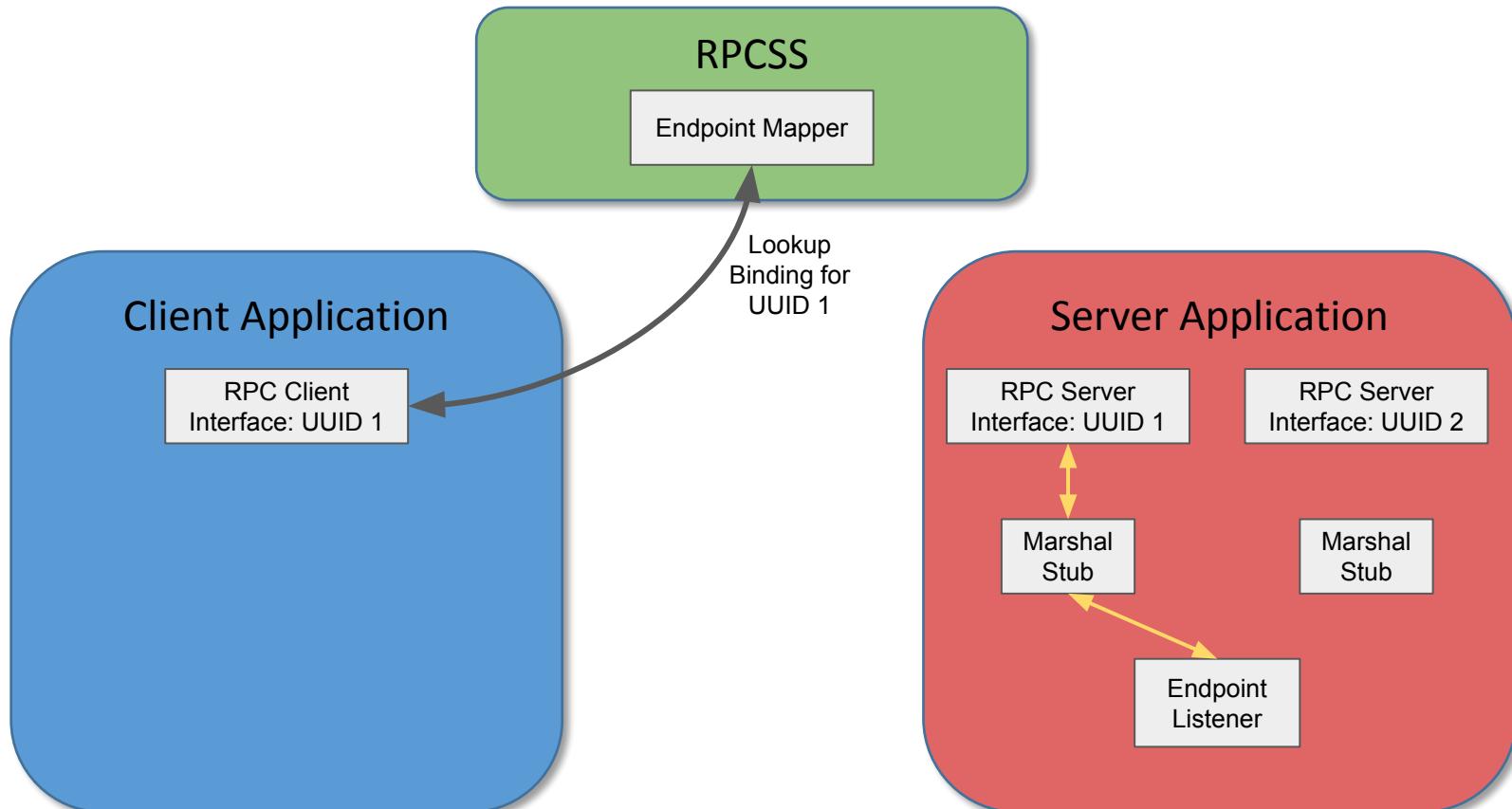
Static Security Descriptor

As all endpoints of multiplexed, you can pick  
the one with the lowest connect time security

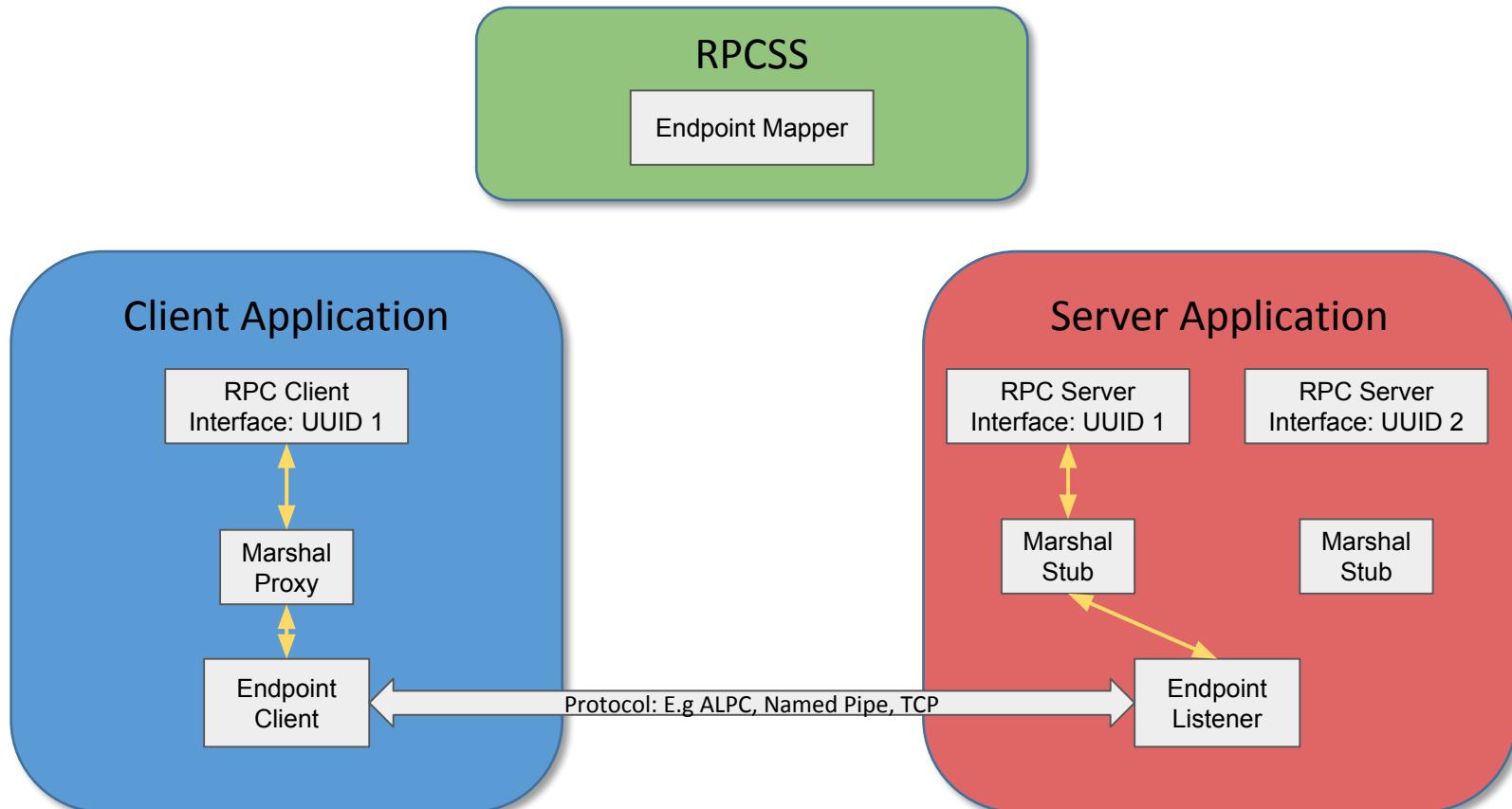
# RPC Endpoint Mapping



# RPC Endpoint Mapping



# RPC Endpoint Mapping



Sets the global symbol resolver to use WinDBG's DBGHELP

```
PS> Set-GlobalSymbolResolver "c:\dbg\dbghelp.dll"
```

Carve out the RPC services in RPCSS.DLL

```
PS> Get-RpcServer "c:\windows\system32\rpcss.dll"
```

Enumerate all DLLs in system32 and extract RPC servers.

```
PS> ls "c:\windows\system32\*.dll" | Get-RpcServer
```

Get all registered endpoints.

```
PS> Get-RpcEndpoint
```

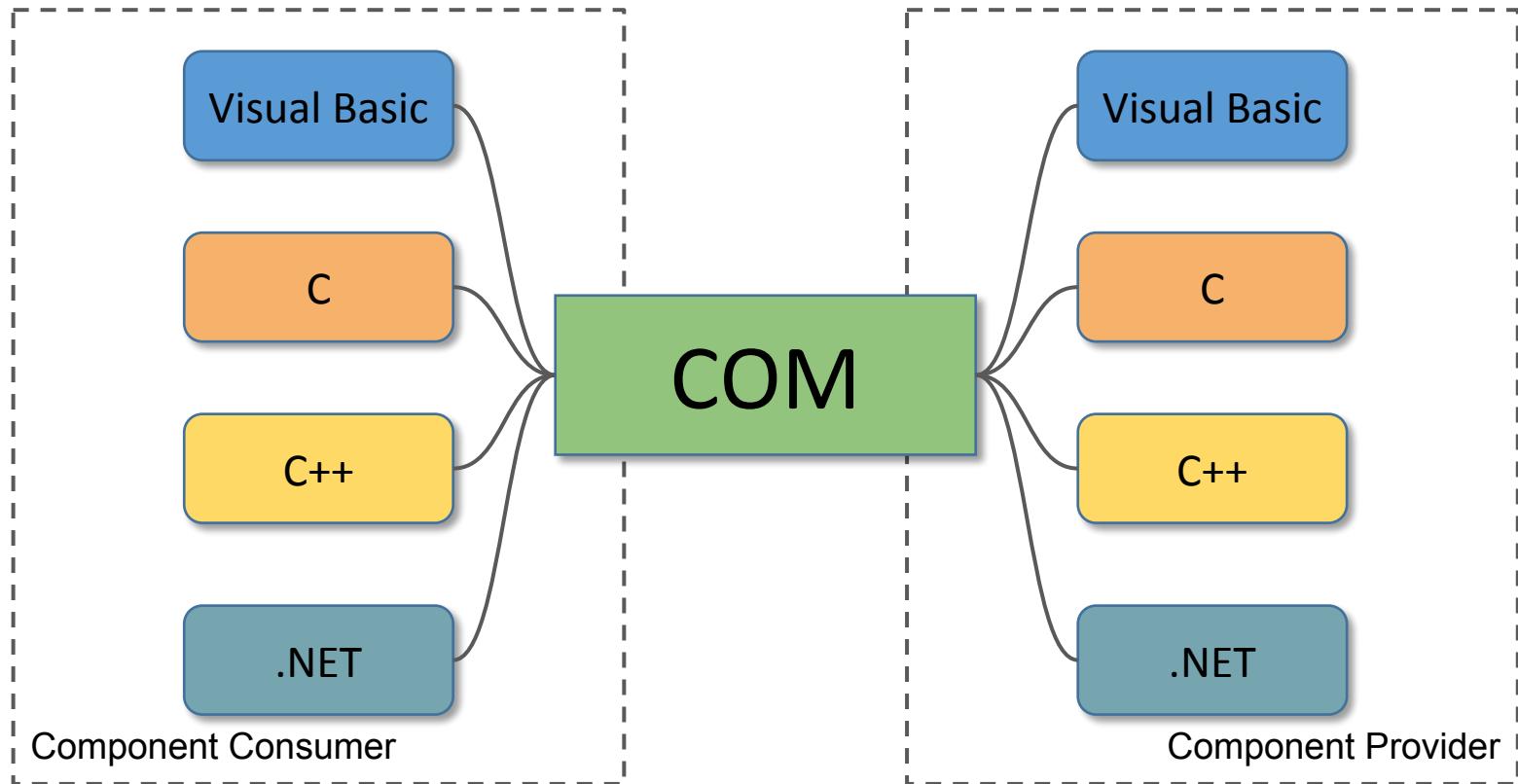
Format an RPC endpoint as text.

```
PS> $rpc_server.FormatAsText()
```

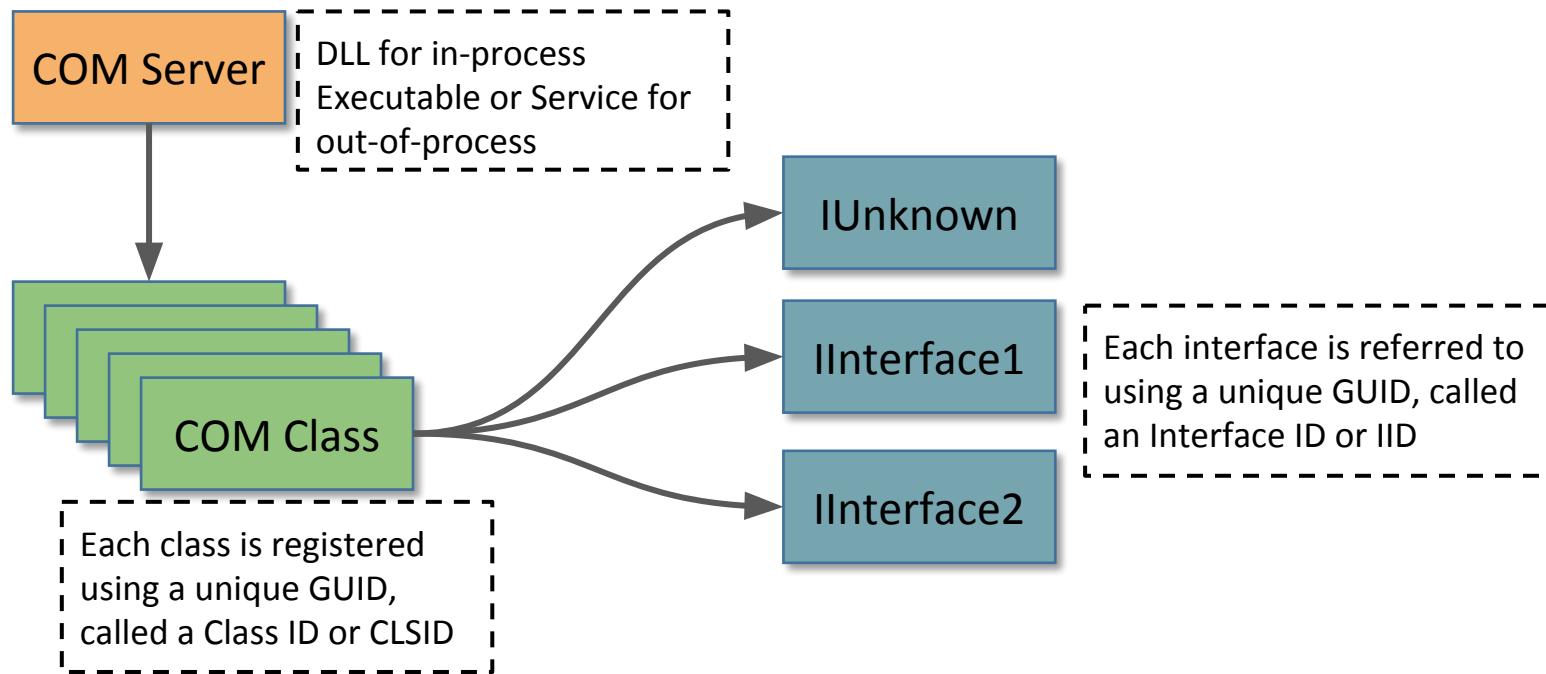
# WORKSHOP 10

## Finding and Analysing RPC Interfaces

# COM is Middleware



# COM Services



# IUnknown, the Root of all COM Evil

Standard  
COM error  
code.  
>= 0 is  
Success

```
DEFINE_GUID(IID_IUnknown,
    "00000000-0000-0000-C000-000000000046") ;

struct IUnknown {
    HRESULT QueryInterface(GUID& iid, void** ppv) ;
    LONG AddRef() ;
    LONG Release() ;
};
```

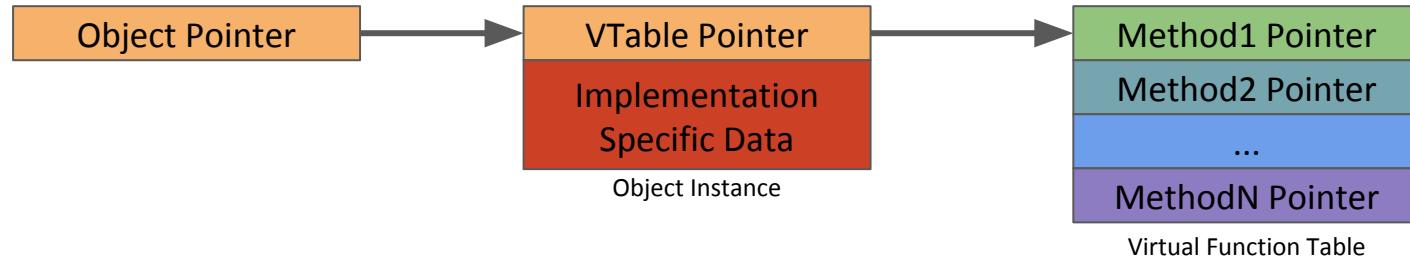
Interfaces defined using a 128 bit Globally Unique ID.

Used to reference count the object.

```
Interface2* intf2 ;
if (intf1->QueryInterface(IID_Interface2,
    (void**)&intf2) >= 0) {
    // Success, we can call methods.
    intf2->Release() ;
}
```

Cast is a GIANT code smell!

# Common ABI



```
struct ObjectVTable {
    void (*SetInt)(struct Object* This, int i);
    int   (*GetInt)(struct Object* This);
};

struct Object {
    struct ObjectVTable* Vtbl;
    // Implementation specific data follows.
};

struct Object* obj;
obj->Vtbl->SetInt(obj, 1234);
```

Object pointer is first.  
Arguments passed left to right

VTable Pointer at start

# Class Factories

- Component classes can't be directly 'newed' so COM defines a factory interface, *IClassFactory*

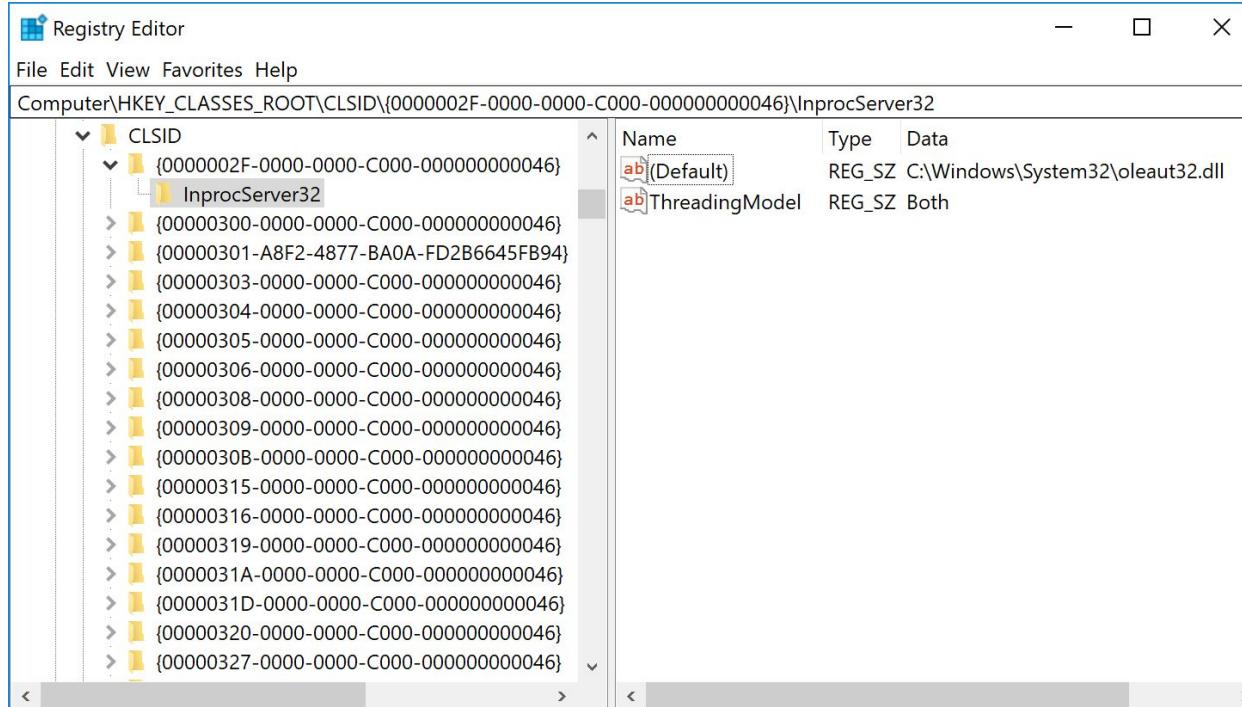
```
DEFINE_GUID(IID_ClassFactory,
    "00000001-0000-0000-C000-00000000046");

struct IClassFactory : public IUnknown {
    HRESULT CreateInstance(
        IUnknown *pUnkOuter,
        REFIID riid,
        void **ppvObject);

    HRESULT LockServer(BOOL fLock);
};
```

# Class Registration

- Each Class Factory has a unique GUID, the Class ID (CLSID)



# Creating Class Factories and Instances

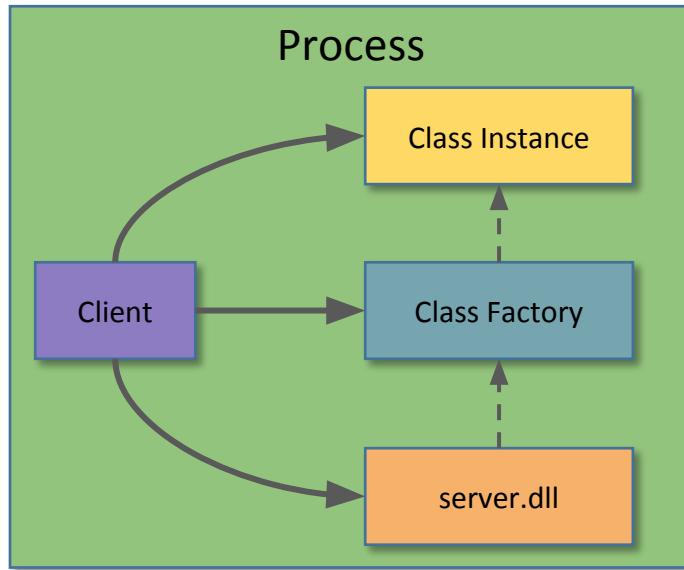
```
HRESULT CoGetClassObject(REFCLSID rclsid,  
                         DWORD dwClsContext,  
                         COSERVERINFO *pServerInfo,  
                         REFIID riid,  
                         LPVOID *ppv);
```

Specifies what type of server to lookup:

- CLSCTX\_INPROC\_SERVER
- CLSCTX\_INPROC\_HANDLER
- CLSCTX\_LOCAL\_SERVER
- CLSCTX\_REMOTE\_SERVER

```
HRESULT CoCreateInstance(REFCLSID rclsid,  
                        IUnknown *punkOuter,  
                        DWORD dwClsCtx,  
                        REFIID riid,  
                        LPVOID *ppv);
```

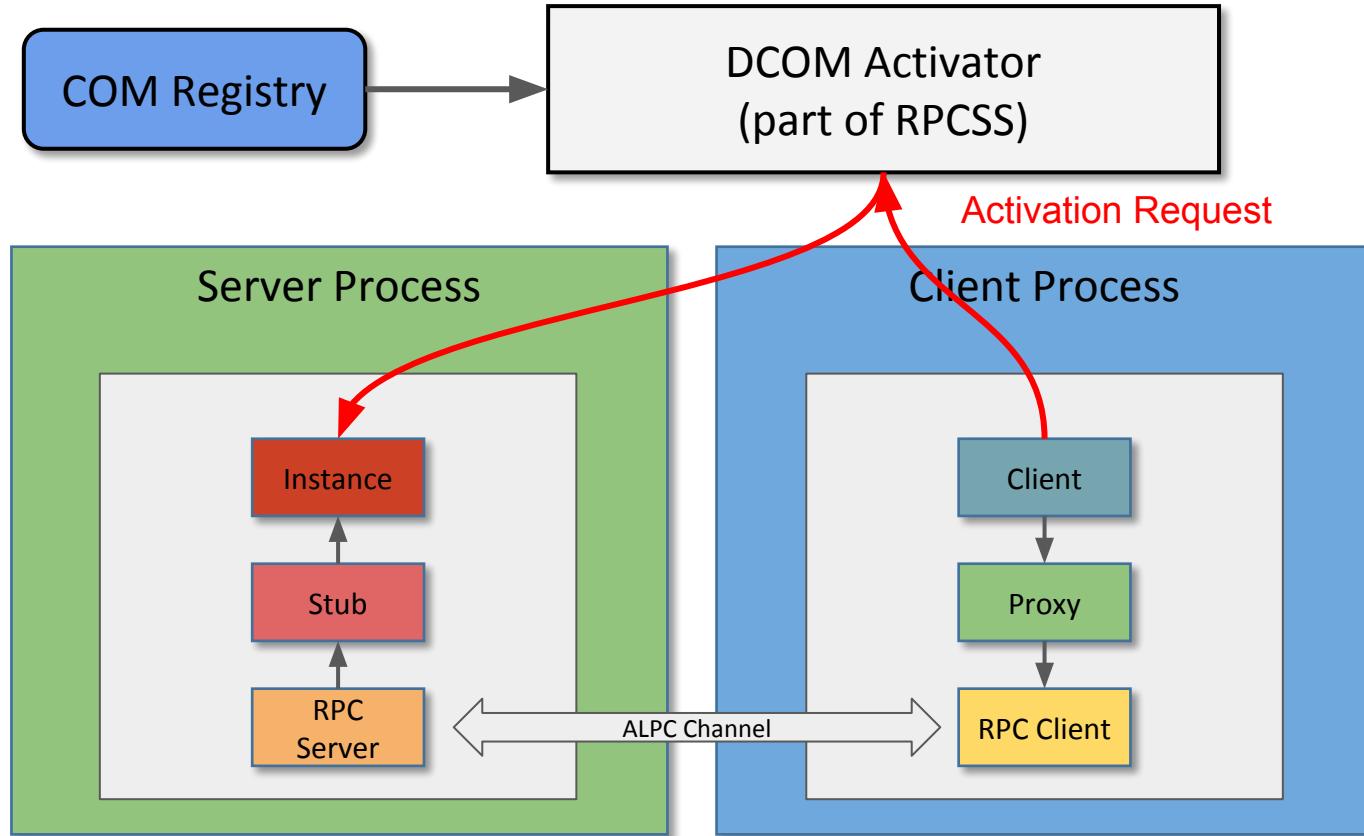
# In-Process Server



- DLL server filename specified in *InProcServer32* key.
- DLL loaded into process and class factory created by calling exported method:

```
HRESULT DllGetClassObject (REFCLSID rclsid,  
                          REFIID riid,  
                          LPVOID *ppv) ;
```

# Local Server



Parse registry information into a database and set as current database.

```
PS> Get-ComDatabase -SetCurrent
```

Get all COM classes from the current database.

```
PS> Get-ComClass
```

Get all COM classes which have a registered local server.

```
PS> Get-ComClass -ServerType LocalServer32
```

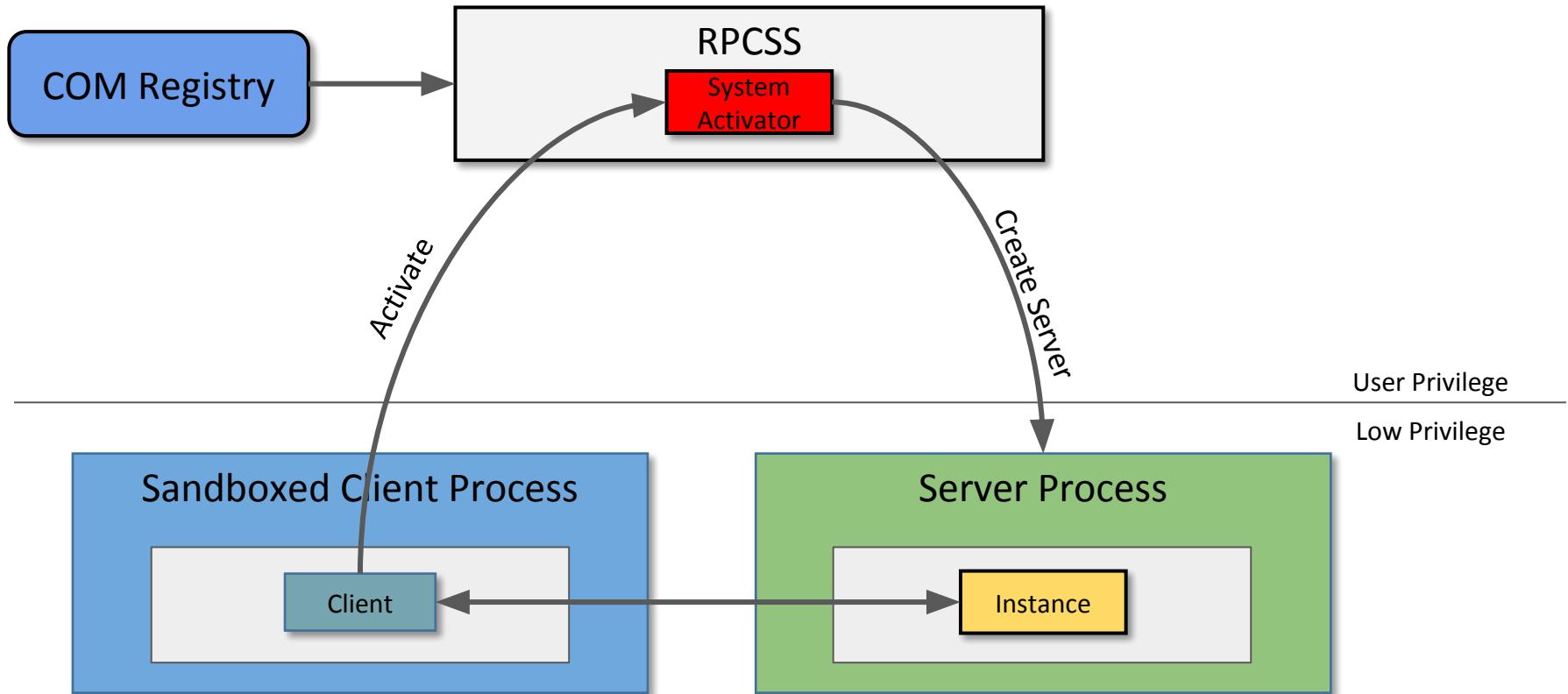
Get all COM classes which have the name Flash in their registration

```
PS> $cls = Get-ComClass | ? Name -Match Flash
```

Enumerate all accessible interfaces for a class.

```
PS> $intfs = Get-ComClassInterface $cls
```

# Local Server Activate as Activator (AAA)



# Activation Via AppID

OleView .NET v1.5 - 64bit

File Registry Object Security Processes Storage Help

**AppReadiness...**

Object Properties

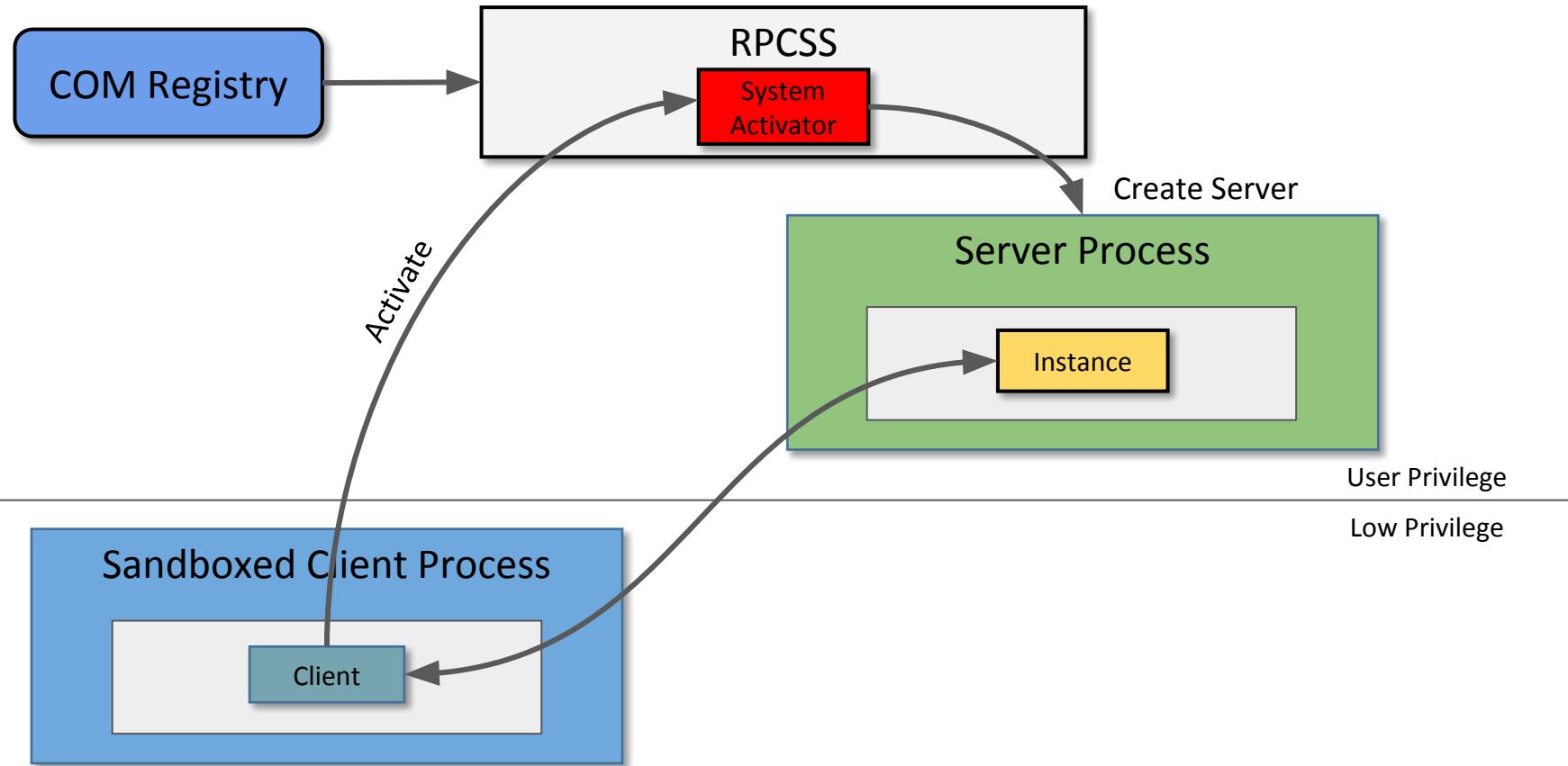
CLSID	Supported Interfaces
AppID	AppReadiness Service
AppID:	88283D7C-46F4-47D5-8FC2-DB0B5CF0CB54
Run As:	N/A
Service:	AppReadiness
Flags:	None
Launch Permission:	O:S-1-5-21-528768928-1231760990-50533070-500G:S-1-5-21-528768928-1231760990-50533070-500
Access Permission:	:8928-1231760990-50533070-500D:(A;;CC;;;BA)(A;;CC;;;IU)(A;;CC;;;SU)(A;;CC;;;SY)S:(ML;;NX;;;LW)
DLL Surrogate:	N/A

Classes are registered with the AppID GUID

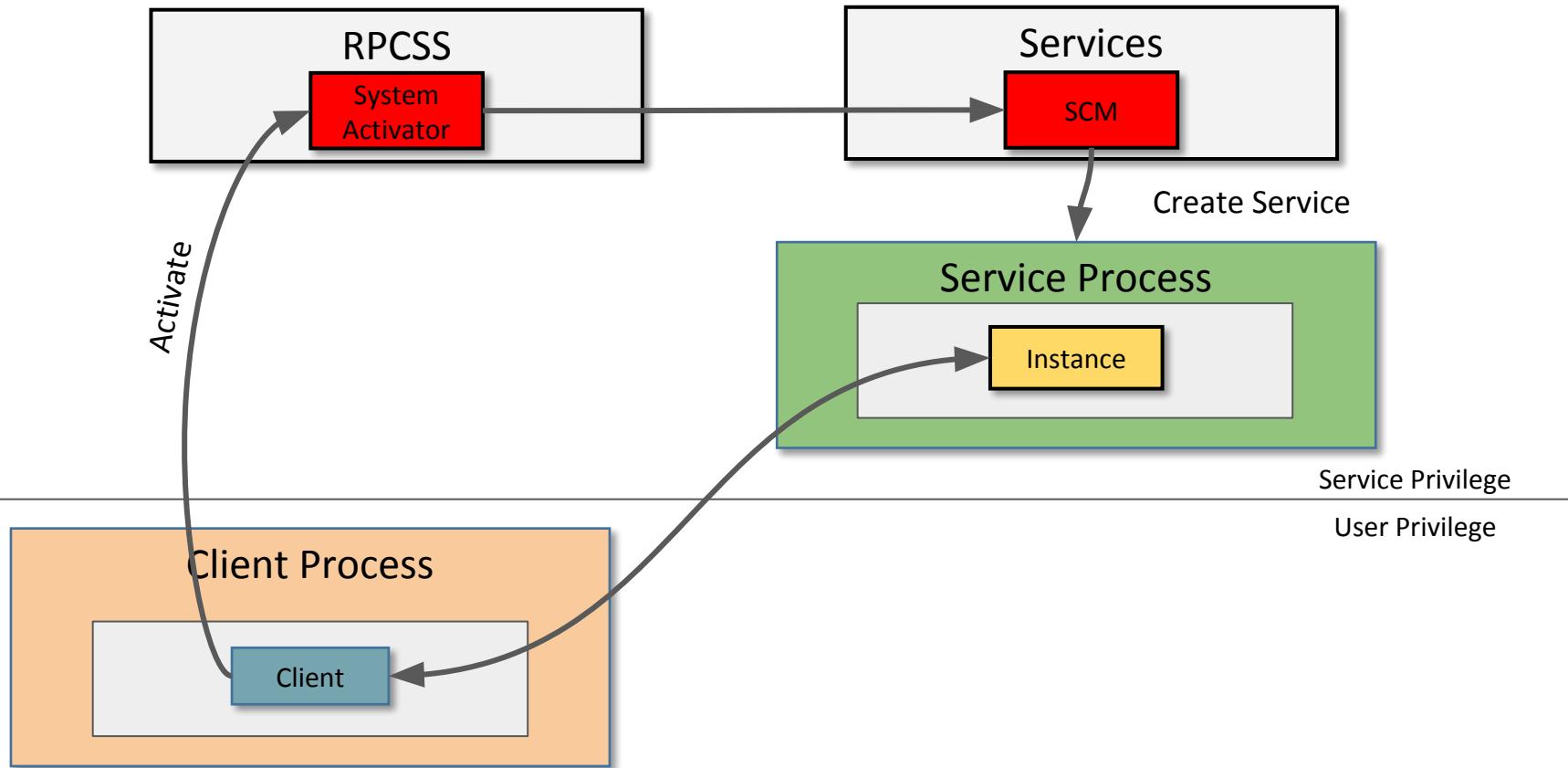
RunAs field or Service name start

Launch and Access Permissions

# Local Server RunAs Interactive User



# Local Server Service Activation



Get all AppID entries.

```
PS> Get-ComAppId
```

Get all COM classes registered to run as an interactive user.

```
PS> Get-ComClass -InteractiveUser
```

Get all COM classes which are registered in the AxInstSv service.

```
PS> Get-ComClass -ServiceName "AxInstSv"
```

Get com classes which run in a service and group by the service name.

```
PS> Get-ComClass -Service | `  
      group {$_.AppIDEntry.ServiceName}
```

Parse COM information from processes and return registered class information.

```
PS> Get-ComRegisteredClass
```

# DCOM Security

OLE Launch Security

Owner: S-1-5-21-528768928-1231760990-50533070-500  
Group: S-1-5-21-528768928-1231760990-50533070-500  
Integrity: Low

DACL SACL

ACL Entries

Type	Account	Access	Flags
Allowed	BUILTIN\Administrators	Execute	None
Allowed	NT AUTHORITY\INTERACTIVE	Execute	None
Allowed	NT AUTHORITY\SERVICE	Execute	None
Allowed	NT AUTHORITY\SYSTEM	Execute	None

Specific Access

Name	Access Mask
<input checked="" type="checkbox"/> Execute	0x00000001
<input type="checkbox"/> Execute Local	0x00000002
<input type="checkbox"/> Execute Remote	0x00000004
<input type="checkbox"/> Activate Local	0x00000008
<input type="checkbox"/> Activate Remote	0x00000010

OLE Access Security

Owner: BUILTIN\Administrators  
Group: BUILTIN\Administrators  
Integrity: N/A

DACL

ACL Entries

Type	Account	Access
Allowed	NT AUTHORITY\SYSTEM	Execute, ExecuteLocal
Allowed	NT AUTHORITY\SELF	GenericAll
Allowed	NT AUTHORITY\INTERACTIVE	Execute, ExecuteLocal

Specific Access

Name	Access Mask
<input checked="" type="checkbox"/> Execute	0x00000001
<input checked="" type="checkbox"/> Execute Local	0x00000002
<input checked="" type="checkbox"/> Execute Remote	0x00000004

Launch = Create a new instance of the server.  
Activate = Create new object on existing server.  
Enforced in RPCSS

Access = Call methods on existing objects.  
Enforced in Server Process  
SELF = Process Token User SID

# Access Rights

Name	Value	Description
Execute	0x00000001	Grants all rights for local and remote (LEGACY)
Execute Local	0x00000002	Grants local execute and launch
Execute Remote	0x00000004	Grants remote execute and launch
Activate Local	0x00000008	Grants local activation
Activate Remote	0x00000010	Grants remote activation

Execute access is a legacy right. If only available access then grants all access remote and local

Integrity level only applies to Launch/Activate.

# Security Through CoInitializeSecurity

```
HRESULT CoInitializeSecurity(  
    PSECURITY_DESCRIPTOR pSecDesc,  
    LONG cAuthSvc,  
    SOLE_AUTHENTICATION_SERVICE * asAuthSvc,  
    void * pReserved1,  
    DWORD dwAuthnLevel,  
    DWORD dwImpLevel,  
    void * pAuthList,  
    DWORD dwCapabilities,  
    void * pReserved3  
) ;
```

Optional SD:  
NULL = No Access Security!

EOAC\_APPID - pSecDesc is a AppID GUID  
EOAC\_ACCESS\_CONTROL - pSecDesc is a  
pointer to an IAccessControl implementation  
EOAC\_NO\_CUSTOM\_MARSHAL - Disables  
custom marshaling in the process.

Show the COM launch security descriptor for a class in a GUI.

```
PS> Show-ComSecurityDescriptor $cls
```

Show the COM access security descriptor for a class in a GUI.

```
PS> Show-ComSecurityDescriptor $cls -ShowAccess
```

Get service hosted classes accessible by the current user.

```
PS> Get-ComClass -Service | Select-ComAccess
```

Get interactive user classes accessible by the token from PID 1234

```
PS> Get-ComClass -InteractiveUser | `  
Select-ComAccess -ProcessId 1234
```

# Interface Proxies and Stubs

The screenshot shows two windows: the Windows Registry Editor and the OleViewDotNet 64bit utility.

**Windows Registry Editor:** The left pane shows a tree view of registry keys under `Computer\HKEY_CLASSES_ROOT\Interface\{475CA8F3-9417-48BC-B9D7-4163A7844C02}`. A red arrow points from the key `ProxyStubClSID32` to the `OleViewDotNet` window.

**OleViewDotNet 64bit:** The right pane displays the properties of the `e1ba88ba-7a83-421a-a05d-71...` entry. The `Proxies` tab is selected, showing a list of proxy interfaces and their corresponding CLSIDs.

Name	IID
IMagneticStripeReaderBankCardDataReceivedEventArgs	2E958823-A31A-4763-882C-23725E39B08E
IPosPrinterStatusUpdatedEventArgs	2EDB87DF-13A6-42BD-BA81-B0E7C3E5A3CD
ICashDrawerStatusUpdatedEventArgs	30AAE98A-0D70-459C-9553-87E124C52488
ITypedEventHandler_2_Windows_CDDevices_CPointOfService_CClaimedPosP...	31424F6F-CFEB-5031-8A95-BEA59B09E584
IAsyncOperationCompletedHandler_1_Windows_CDDevices_CPointOfService_...	32C55F7B-8EE3-555D-998B-78C98AA9627B
IBarcodeScannerStatusUpdatedEventArgs	355D8586-9C43-462B-A91A-816DC97F452D
IJournalPrinterCapabilities	3B5CCC43-E047-4463-BB58-17B5BA1D8056
IAsyncOperation_1_Windows_CDDevices_CPointOfService_CClaimedMagnetic...	41630BD4-F45A-590D-8A4E-F70C9E49AD01
IBarcodeScannerDataReceivedEventArgs	4234A7E2-ED97-467D-AD2B-01E44313A929
IAsyncOperation_1_Windows_CDDevices_CPointOfService_CCashDrawer	45007467-92F2-5BFF-B191-AA5000FEDD9E
<b>IClaimedMagneticStripeReader</b>	<b>475CA8F3-9417-48BC-B9D7-4163A7844C02</b>
IClaimedBarcodeScanner	4A63B49C-8FA4-4332-BB26-945D11D81E0F
ITypedEventHandler_2_Windows_CDDevices_CPointOfService_CClaimedBarc...	4F64E49A-BD8C-549D-970C-A5A250BD27CA
IMagneticStripeReaderCardTypesStatics	528F2C5D-2986-474F-8454-7CCD0592BD5F
IAsyncOperation_1_FIVectorView_1_UINT32	52C56F3C-713A-5162-9E62-362CE7ED53BE
IRceiptOrSlipJob	532199BE-C8C3-4DC2-89E9-5C4A37B34DDC
IMagneticStripeReaderEncryptionAlgorithmsStatics	53B57350-C3DB-4754-9C00-41392374A109
IAsyncOperationCompletedHandler_1_FIVectorView_1_UINT32	55772F29-DA64-5C87-871C-074337A84573
IAsyncOperationCompletedHandler_1_Windows_CDDevices_CPointOfService_...	57836710-F186-5636-891D-F8C5398EA6DF
IPosPrinterCharacterSetIdsStatics	5C709EFF-709A-4FE7-B215-06A748A38B39
IBarcodeScannerReport	5CE4D8B0-A489-4B96-8C4-F0BF8A37753D
IDebugConnectionStation	5D115ECE-DAA0-A1E9-9C9C-59D82A0A4E5

**Text at the bottom:**

Proxy and Stub contain Network  
Data Representation (NDR)  
bytecode to marshal raw  
parameters.

140

Get all interfaces with a registered proxy class

```
PS> Get-ComInterface -Proxy
```

Get the COM class which implements the proxy for a specified IID.

```
PS> Get-ComClass -Iid "00000000-...-000000000046"
```

Parse proxy information for an interface object.

```
PS> Get-ComProxy -Interface $intf
```

Parse proxy information from an IID

```
PS> Get-ComProxy -Iid "00000000-...-000000000046"
```

Format a proxy as text.

```
PS> Format-ComProxy -Proxy $proxy
```

# Late Bound Objects - IDispatch

```
[uuid("00020400-0000-0000-c000-00000000046")]
class IDispatch : public IUnknown {
    HRESULT GetTypeInfo(unsigned int iTInfo,
                        ITypeInfo ** ppTInfo);  
  
    HRESULT GetIDsOfNames(OLECHAR ** rgszNames,
                          unsigned int cNames,
                          DISPID * rgDispId);  
  
    HRESULT Invoke(DISPID dispIdMember,
                  // ...
                  DISPPARAMS * pDispParams,
                  VARIANT * pVarResult, ...);  
};
```

Query for a  
ITypeInfo object.

Convert a name  
to a Dispatch ID

Invoke a method based  
on a dispatch ID.  
Passing an array of  
arbitrary parameters.

# Type Libraries

OleView .NET v1.5 - 64bit

File Registry Object Security Processes Storage Help

TypeLibs

Filter: Mode: Contains Apply

- + 4B21F542-0EB5-4205-A12B-59BF2F2555FE
- + 6271895B-E67F-4DEE-B68B-BF74ACE07753
- + 63390F96-F295-425F-A658-0F0C88E8C3C8
- + 686BA761-D755-4927-929F-94C8F67AF1DF
- + Accessibility
- + AccessibilityCplAdmin 1.0 Type Library
- + Active DS Type Library
- + ActiveMovie control type library
- + AgentWmiLib
- AP Client 1.0 HelpPane Type Library
  - + AP Client 1.0 HelpPane Type Library : Version 1.0
- + AP Client 1.0 Type Library
- + AppIdPolicyEngineApi 1.0 Type Library
- + Assistance Platform Client 1.0 Data Services Type Lib
- + ATL 2.0 Type Library
- + azroles 1.0 Type Library
- + BdeUISrv 1.0 Type Library
- + C68E3F27-AAD0-4DC4-B7E6-B3249770763D
- + CertCli 1.0 Type Library
- + CertEnc 1.0 Type Library
- + CertEnroll 1.0 Type Library

Showing 265 of 265 entries

AP Client 1.0 H...

Type Library

Name: AP Client 1.0 HelpPane Type Library Open

ID: 8CEC5857-07A1-11D9-B15E-000D56BFE6EE

Version: 1.0

Win32 Path: C:\Windows\SysWOW64\HelpPaneProxy.dll

Win64 Path: C:\Windows\System32\HelpPaneProxy.dll

Object Properties

Get all COM type libraries

```
PS> Get-ComTypeLib
```

Get the COM type library which implements an IID proxy.

```
PS> Get-ComTypeLib -Iid "00000000-...0000000046"
```

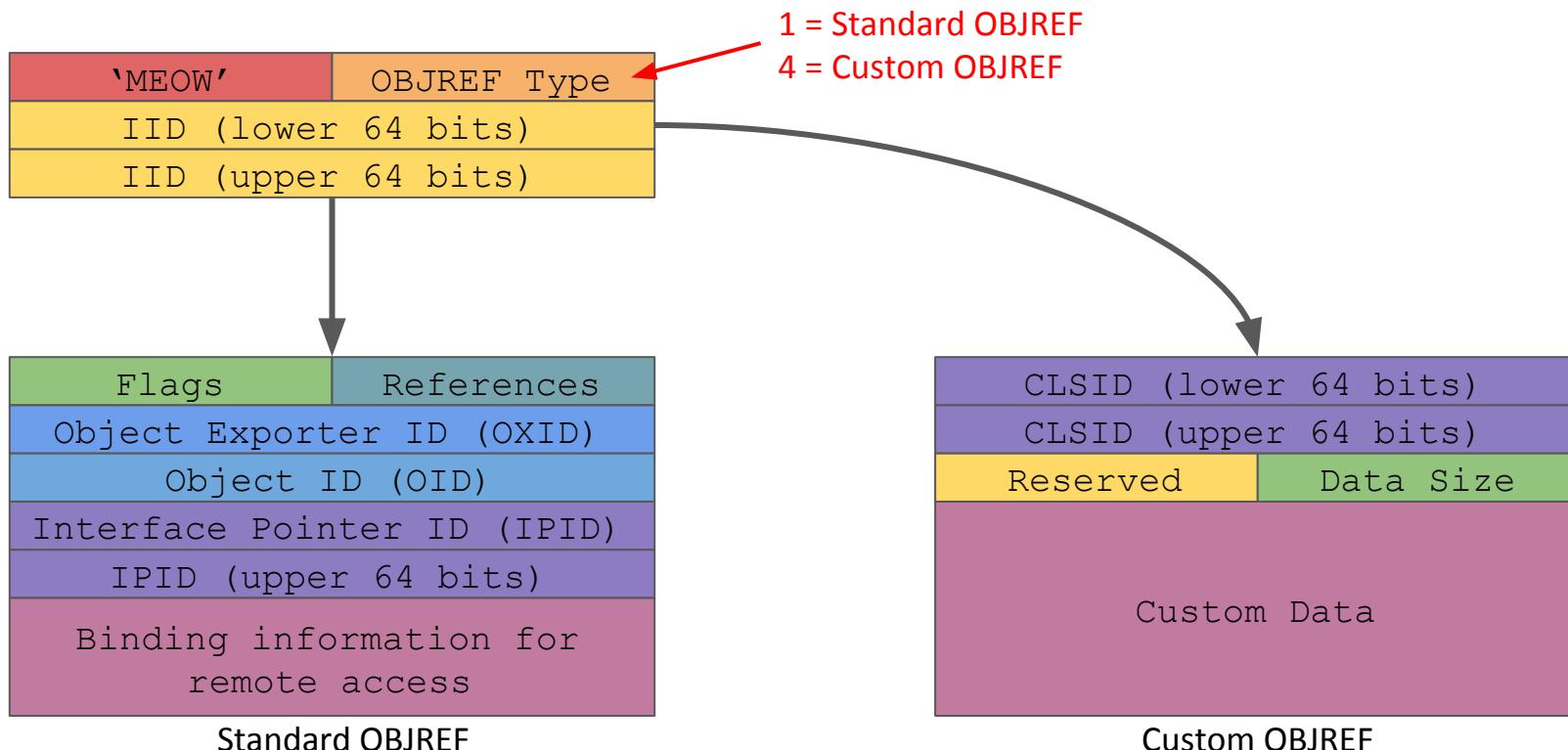
Convert a COM type library to a .NET assembly.

```
PS> $asm = Get-ComTypeLibAssembly -TypeLib $tlb
```

Format a COM type library as text.

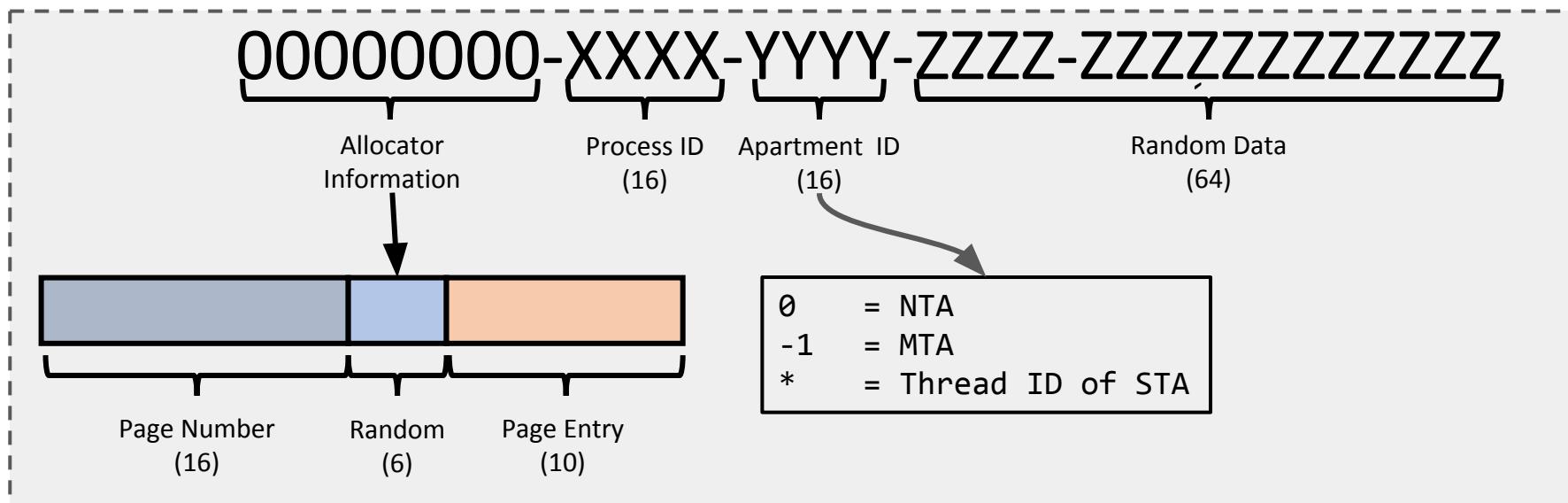
```
PS> Format-ComTypeLib -TypeLib $tlb
```

# COM Marshaling



# IPIPID Structure

- You'd assume that the IPIPID was a random GUID.



# Unpacking the Marshal OBJREF

OleView .NET v1.5 - 64bit

File Registry Object Security Processes Storage Help

Marshal Vie...

OBJREF Type: Standard IID: 00000000-0000-0000-C000-000000000046 IID Name: IUnknown

Standard Flags: 0x00000000 Public Refs: 1

OXID: 0x97E0F4CA52E18871 OID: 0x405D7875517AE97A

IPID: 0000C802-1D48-0000-0885-46B923BF0E9C Apartment: NTA

Process ID: 7496 View Process Name: svchost

String Bindings:

Tower ID	Network Address
Tcp	DESKTOP-KHV61TH
Tcp	192.168.26.129

Security Bindings:

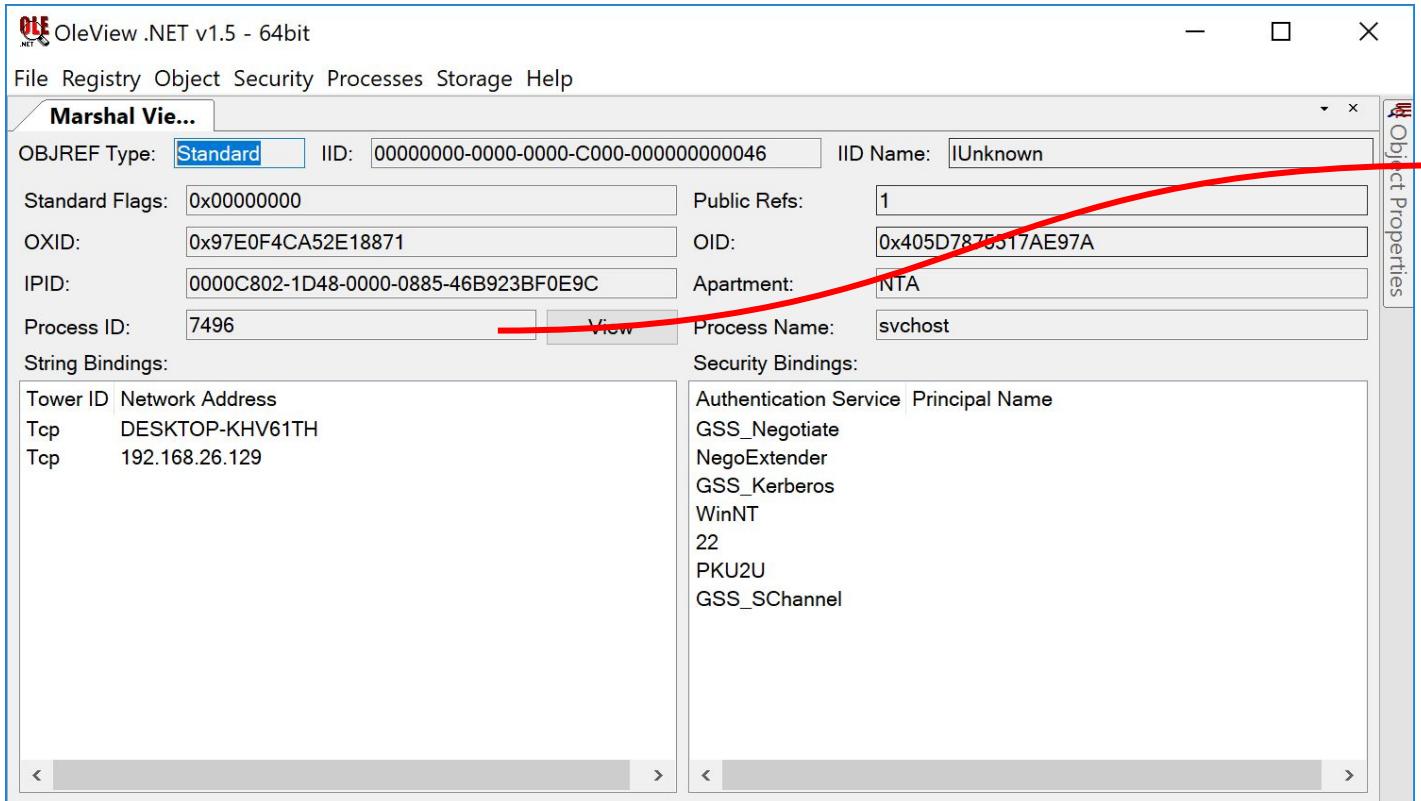
Authentication Service	Principal Name
GSS_Negotiate	
NegoExtender	
GSS_Kerberos	
WinNT	
22	
PKU2U	
GSS_SChannel	

< >

< >

Object Properties

Extract out Process ID



Sets the global symbol resolver to use WinDBG's DBGHELP

```
PS> Set-ComSymbolResolver "c:\windbg\dbghelp.dll"
```

Create a new instance of a COM object.

```
PS> $obj = New-ComObject -Class $cls
```

Marshal all known interfaces for an object as OBJREFs

```
PS> $objrefs = Get-ComClassInterface $cls | `  
Get-ComObjRef $obj
```

Get all known IPIDs for an object and resolve method names (if symbols available).

```
PS> $ipids = Get-ComObjectIpid $obj `  
-ResolveMethodNames
```

Format the COM IPIDs as text.

```
PS> $ipids | Format-ComProxy
```

# WORKSHOP 11

## Find and Analysing COM Services

# Services

- Services are also a securable resource
- Typically look for write privileges to change configuration
  - Everyone should already know about this.
- Instead look for start privileges
  - Increase potential attack surface
  - Some services take arguments during start such as the Mozilla Maintenance Service

```
ServiceController svc = new
    ServiceController ("blah");
// Start a service with arbitrary arguments.
svc.Start (new string[] { "Arg1", "Arg2" });
```

# Service Triggers

- Some services can be started without an explicit Start privilege
- Windows 7 introduced Service Triggers, starts/stops services on certain events:
  - Access to Named Pipe or RPC Endpoints
  - Creation of Firewall Access Rules
  - Joining of a Domain
  - Custom Event Tracing for Windows event
  - Adding a Hardware Device
- ETW is one of the most common and easiest to execute

# ETW Event Trigger

- For example the WebClient service has the following trigger:

```
| WebClient Granted Access: QueryConfig, QueryStatus, EnumerateDependents, Interrogate,  
| UserDefinedControl, ReadControl  
| Trigger: 0 - Type: Custom - Action: Start  
| Subtype: [ETW UUID] {22b6d684-fa63-4578-87c9-effcbe6643c7}
```

- Use the following C++ code to start the WebClient service

```
const GUID _MS_Windows_WebClntLookupServiceTrigger_Provider =  
    { 0x22B6D684, 0xFA63, 0x4578,  
        { 0x87, 0xC9, 0xEF, 0xFC, 0xBE, 0x66, 0x43, 0xC7 } };  
REGHANDLE Handle;  
EventRegister(&_MS_Windows_WebClntLookupServiceTrigger_Provider,  
    nullptr, nullptr, &Handle)  
EVENT_DESCRIPTOR desc;  
EventDescCreate(&desc, 1, 0, 0, 4, 0, 0, 0);  
EventWrite(Handle, &desc, 0, nullptr) == ERROR_SUCCESS;  
EventUnregister(Handle);  
}
```

Get all services which can be started by the current user.

```
PS> Get-AccesibleService -AccessRights Start
```

Get all services which have triggers set.

```
PS> Get-RunningService -IncludeNonActive |  
? { $_.Triggers.Count -gt 0 }
```

Start a service by name.

```
PS> Start-Service ServiceName
```

# Scheduled Tasks

The screenshot shows the Windows Task Scheduler interface. The left pane displays a tree view of task libraries, with the 'Task Scheduler Library' expanded to show 'Microsoft' and 'Windows' sections. The 'Windows' section contains numerous tasks like '.NET Framework', 'Active Direct...', 'AppID', etc. The main pane lists scheduled tasks with columns for Name, Status, Triggers, Next Run Time, and Last Run Time. One task, 'appuriverifierinstall', is selected and highlighted in blue. The right pane, titled 'Actions', provides options for managing tasks, including creating basic or custom tasks, importing tasks, displaying running tasks, enabling history, creating new folders, deleting folders, viewing details, refreshing the list, and getting help. It also includes buttons for running or ending the selected task.

Name	Status	Triggers	Next Run Time	Last Run Time
appuriverifie...	Ready		8/3/2018 6:50:35 PM	
appuriverifie...	Ready	Custom Trigger	7/23/2018 12:39:42 PM	
CleanupTem...	Ready		8/1/2018 3:05:36 PM	
DsSvcCleanup	Ready		8/3/2018 6:50:35 PM	

**General** **Triggers** **Actions** **Conditions** **Settings** **History (disabled)**

**Name:** appuriverifierinstall  
**Location:** \Microsoft\Windows\ApplicationData  
**Author:** Microsoft Corporation  
**Description:** Verifies AppUriHandler host registrations.

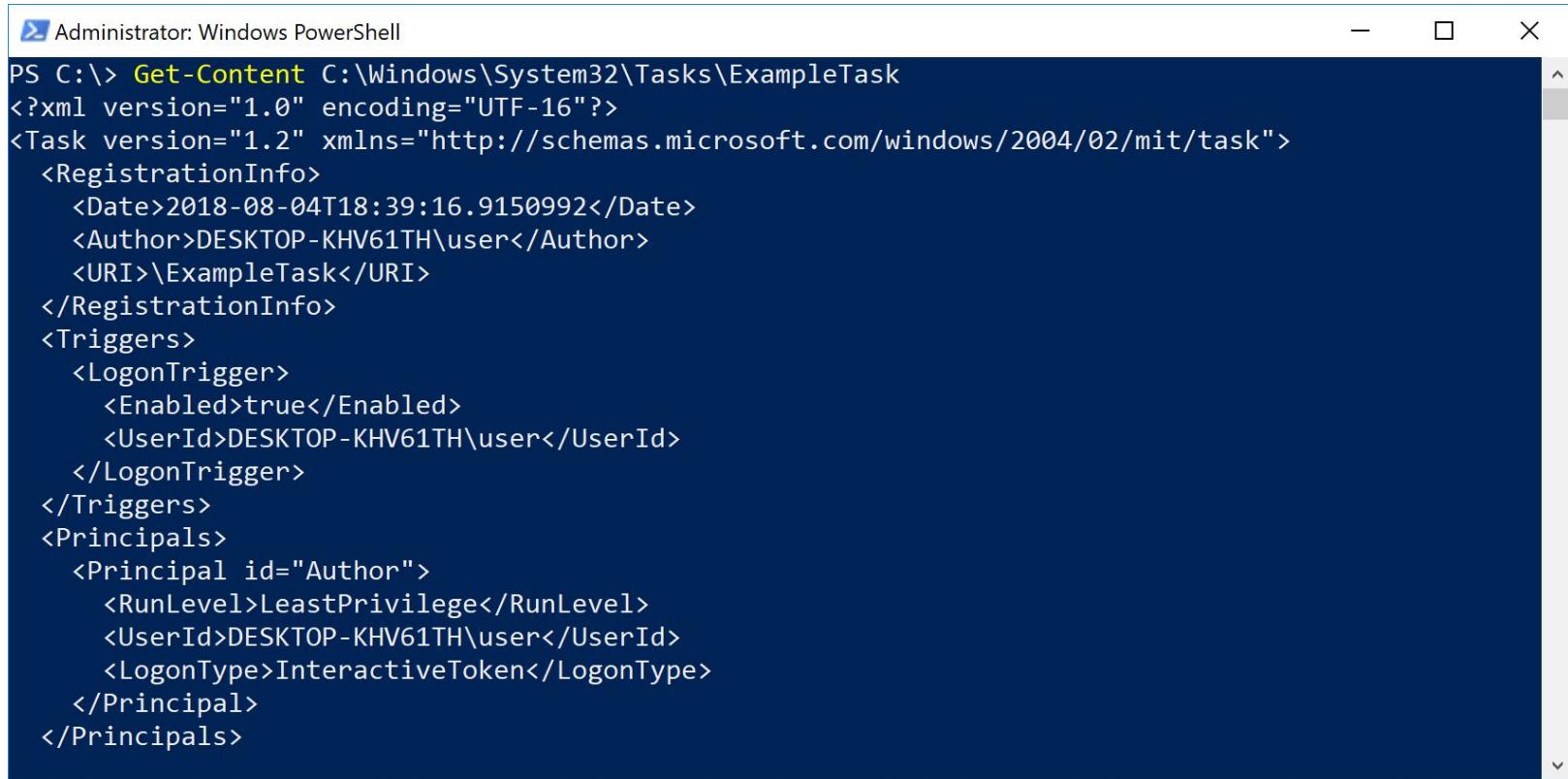
**Actions**

- Create Basic Task...
- Create Task...
- Import Task...
- Display All Running Tas
- Enable All Tasks History
- New Folder...
- Delete Folder
- View
- Refresh
- Help

**Selected Item**

- Run
- End

# Task Files



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command run is "Get-Content C:\Windows\System32\Tasks\ExampleTask". The output displays the XML configuration for a task named "ExampleTask". The XML includes registration information (Date: 2018-08-04T18:39:16.9150992, Author: DESKTOP-KHV61TH\user, URI: \ExampleTask), triggers (LogonTrigger enabled for user DESKTOP-KHV61TH\user), and principals (Author principal with RunLevel LeastPrivilege, User Id DESKTOP-KHV61TH\user, LogonType InteractiveToken).

```
PS C:\> Get-Content C:\Windows\System32\Tasks\ExampleTask
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2018-08-04T18:39:16.9150992</Date>
    <Author>DESKTOP-KHV61TH\user</Author>
    <URI>\ExampleTask</URI>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <UserId>DESKTOP-KHV61TH\user</UserId>
    </LogonTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <RunLevel>LeastPrivilege</RunLevel>
      <UserId>DESKTOP-KHV61TH\user</UserId>
      <LogonType>InteractiveToken</LogonType>
    </Principal>
  </Principals>
</Task>
```

# Task Security

- Task's have an associated security descriptor used to determine what a user can do with the task.
  - Read access allows a user to read the registration
  - Write access allows a user to modify the registration
  - Execute access allows a user to start and stop a task
  - Delete access allows a user to delete a task
- Security descriptor associated with the file on disk
  - Since Windows 10 Anniversary also stored in Registry, but file still useful proxy
- We can use Get-AccessibleFile to determine what tasks a user can see and manipulate.

# WORKSHOP 12

## Misc Attack Surface