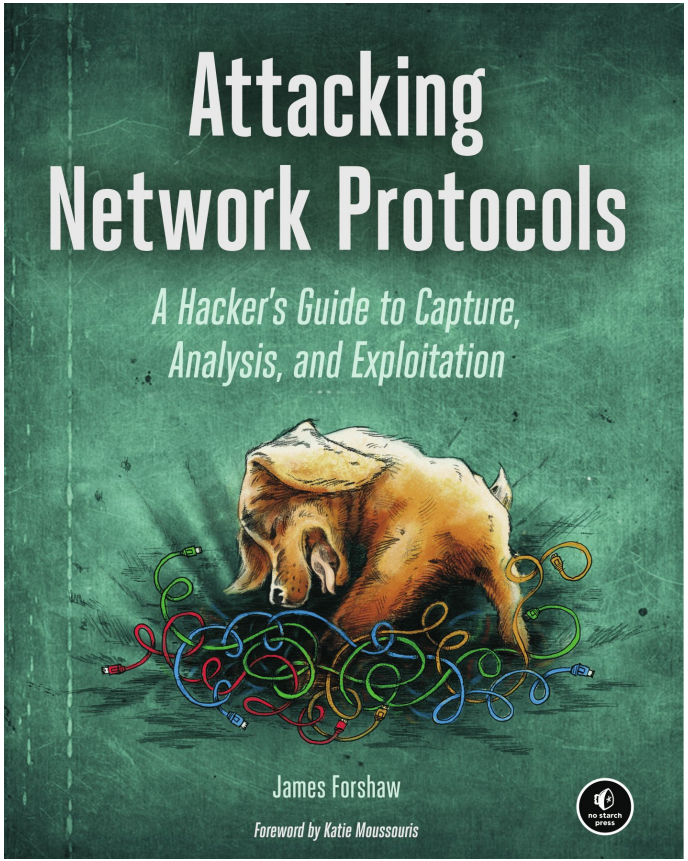
Windows Internals and Local Attack Surface Analysis using Powershell

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.

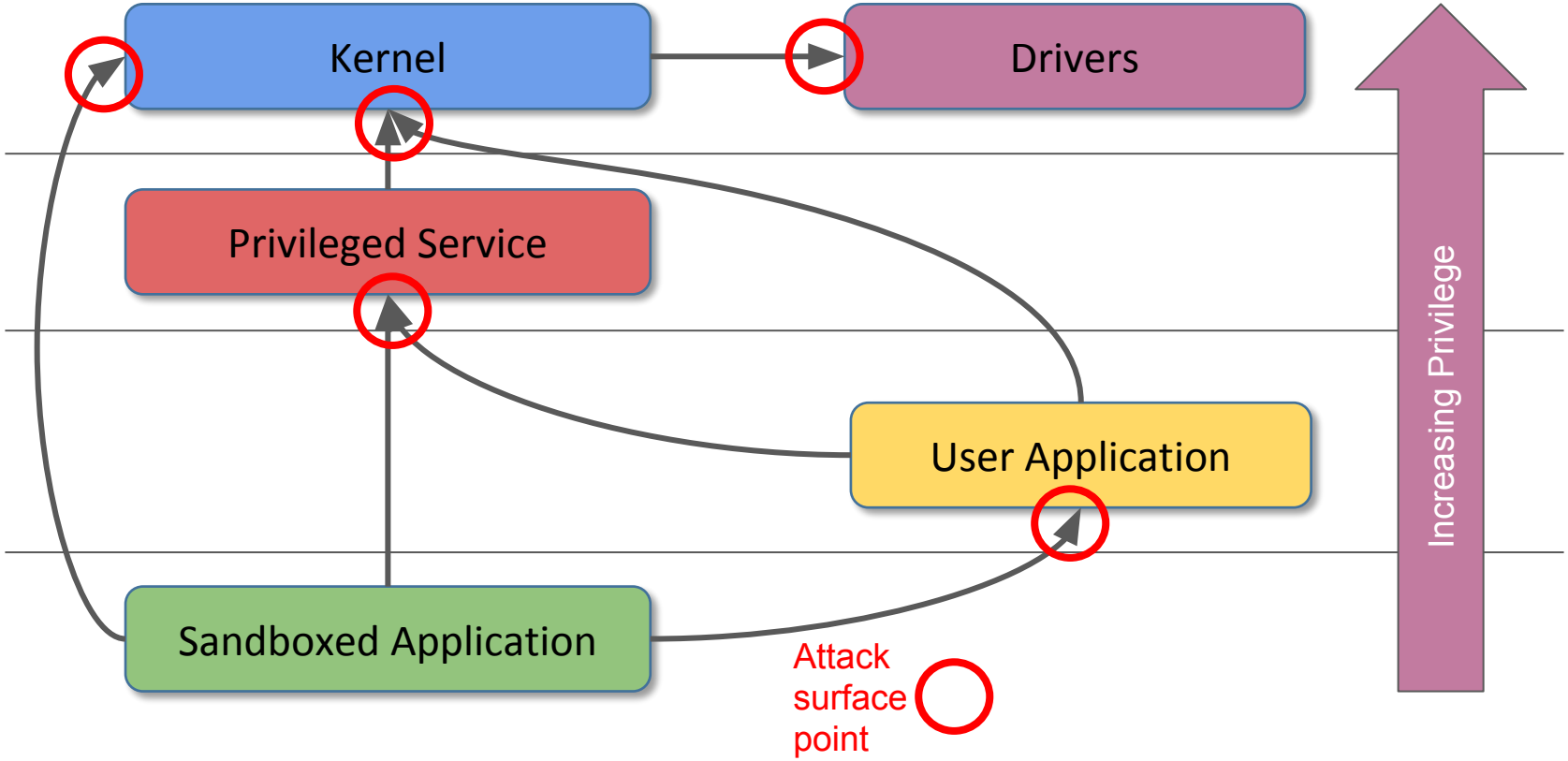


Workshop Agenda

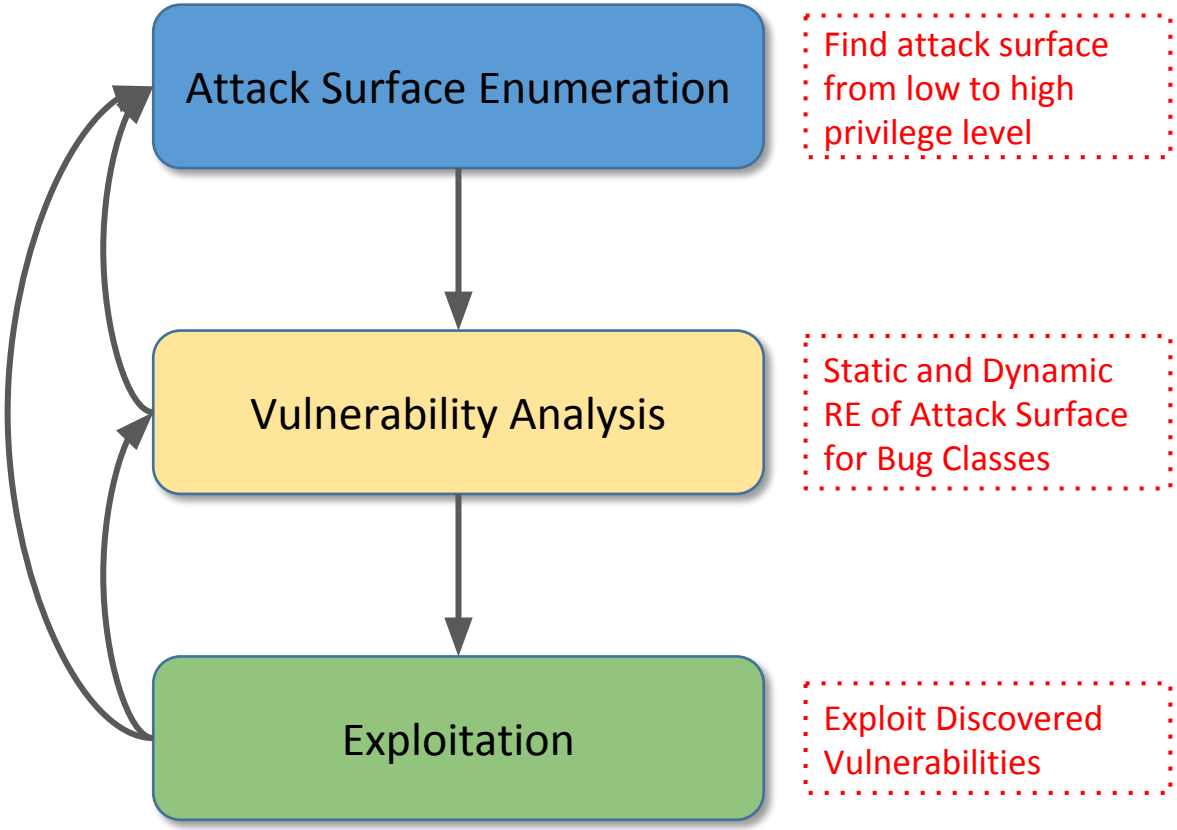
- INTRODUCTION
  - Setting up environment
- PART 1 - Windows Internals
  - Windows Resource Management
  - System Calls and Kernel Interactions
  - Process and Threads
  - Security Reference Monitor - Tokens, Security Descriptors and Access Checking
  - IO Manager
- PART 2 - Local Attack Surface Analysis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

INTRODUCTION

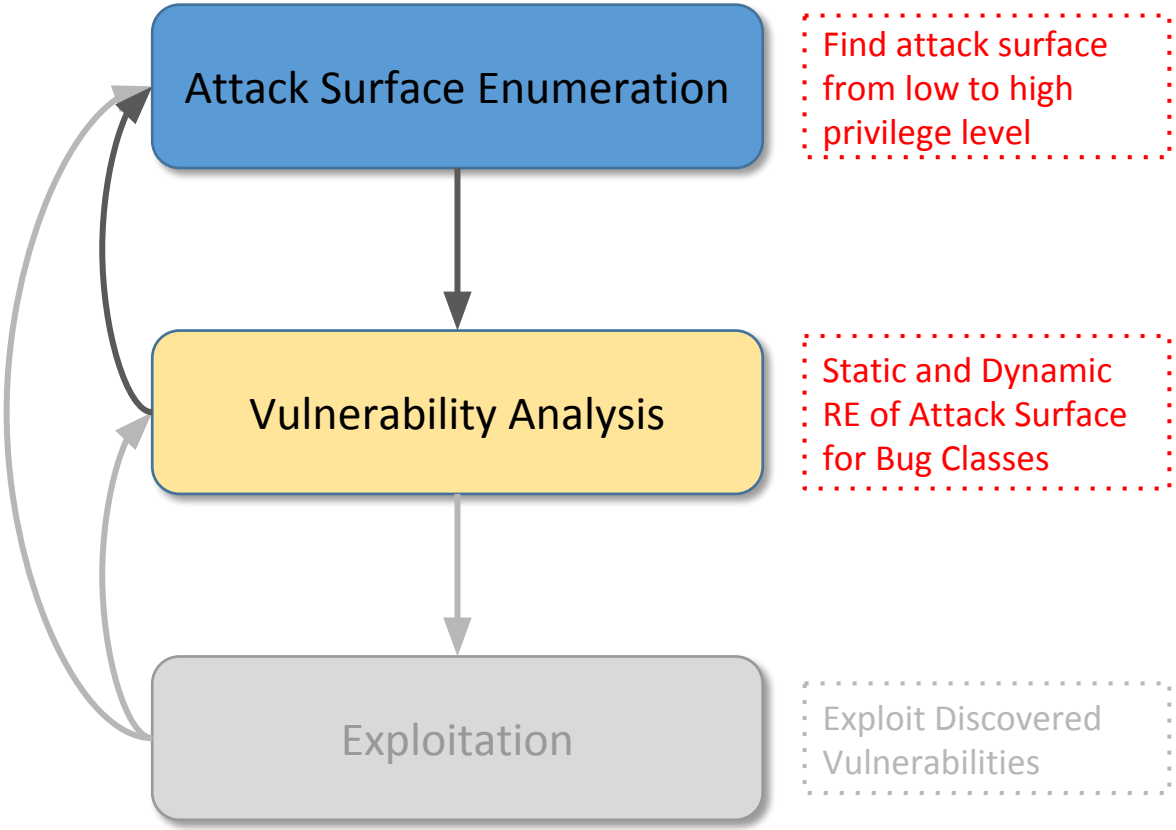
Windows Local Attack Surface



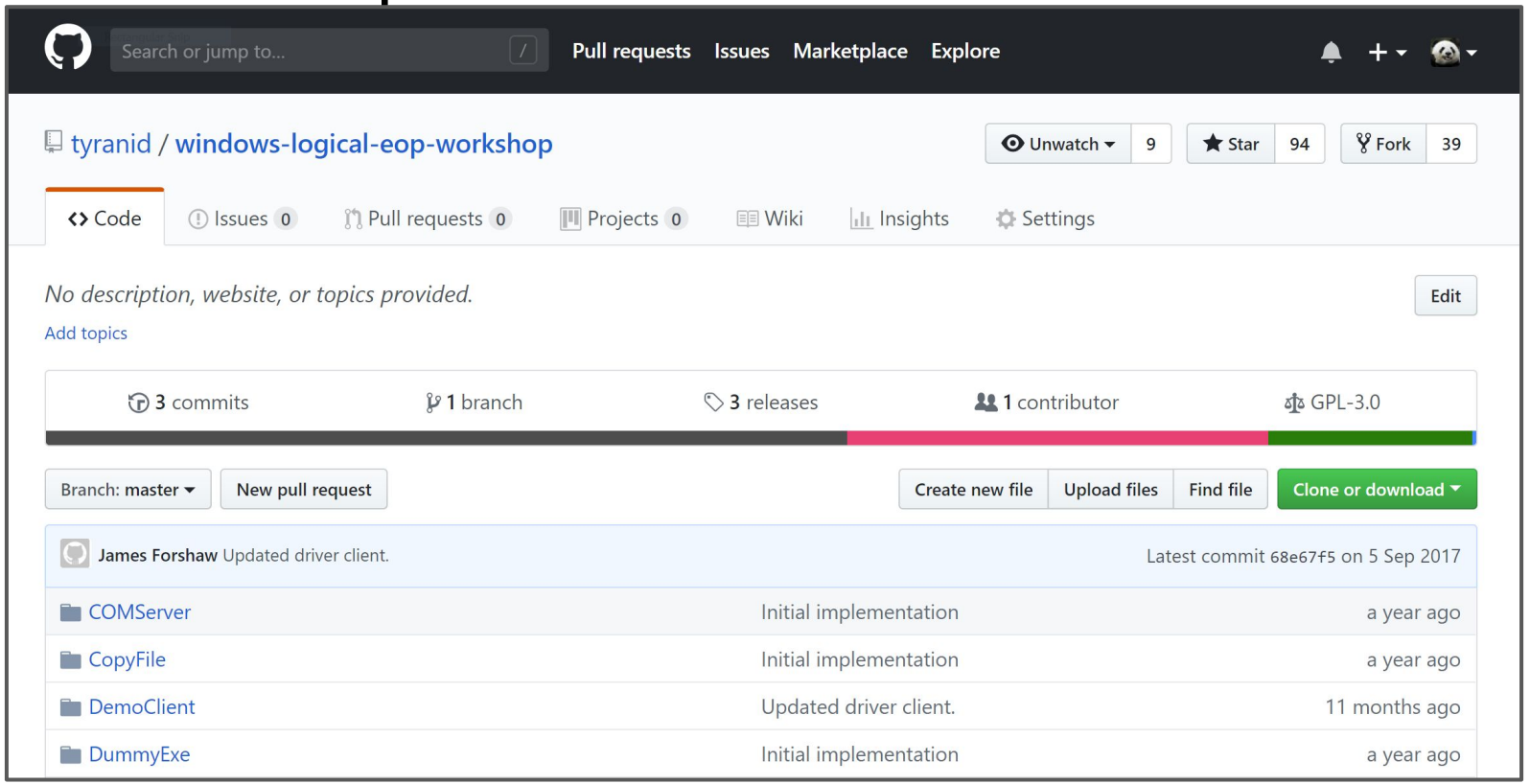
My Approach to Finding EOP Vulnerabilities



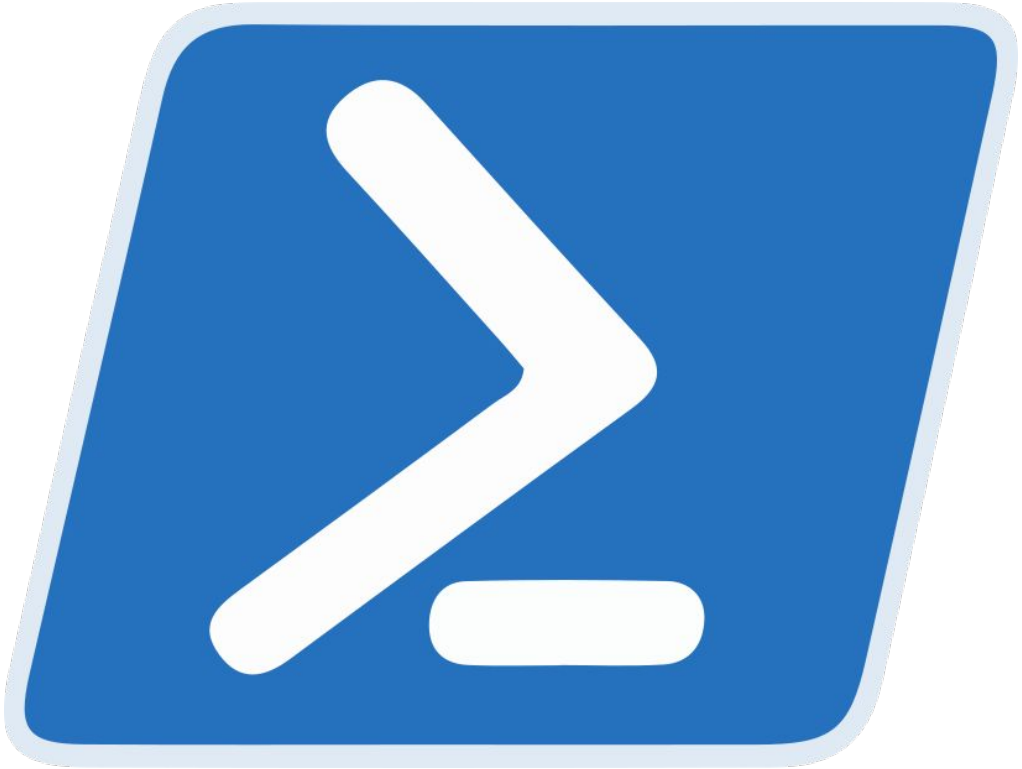
My Approach to Finding EOP Vulnerabilities



EOP Workshop



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.

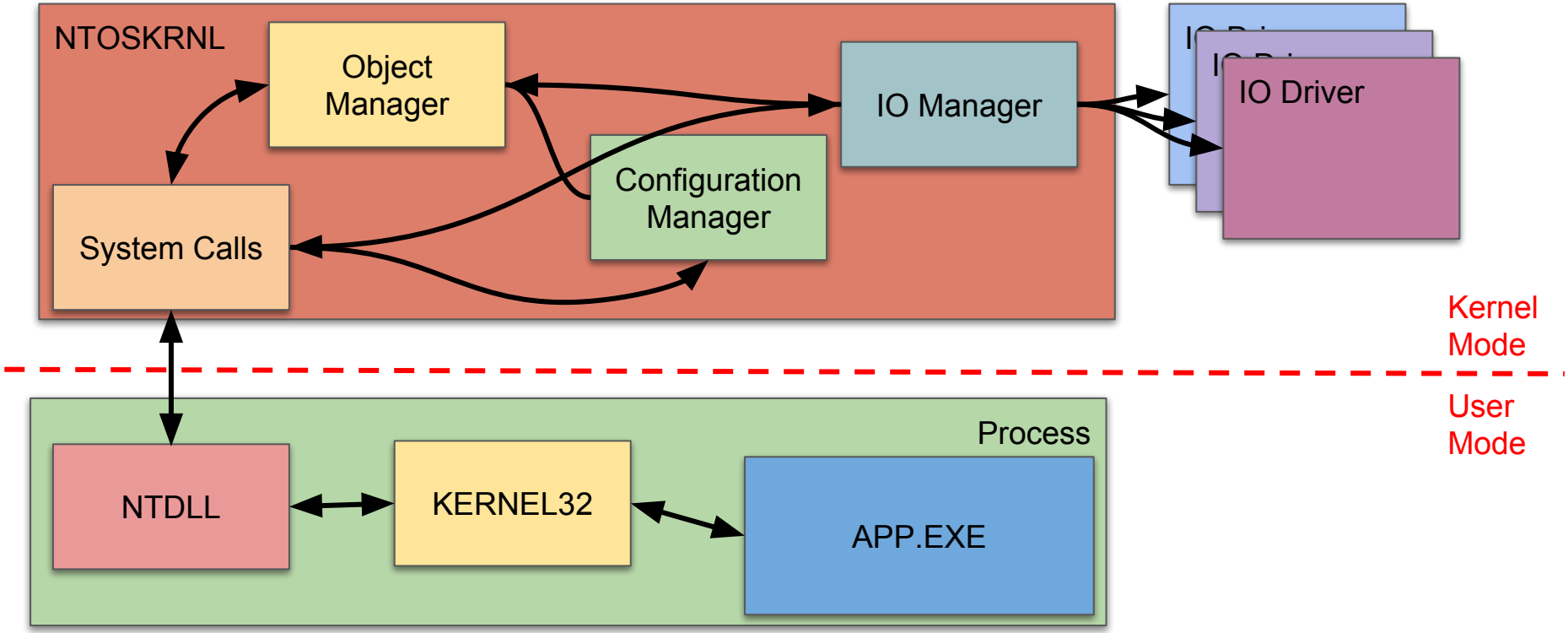
```
PS> Import-Module Workshop
```

WORKSHOP 0

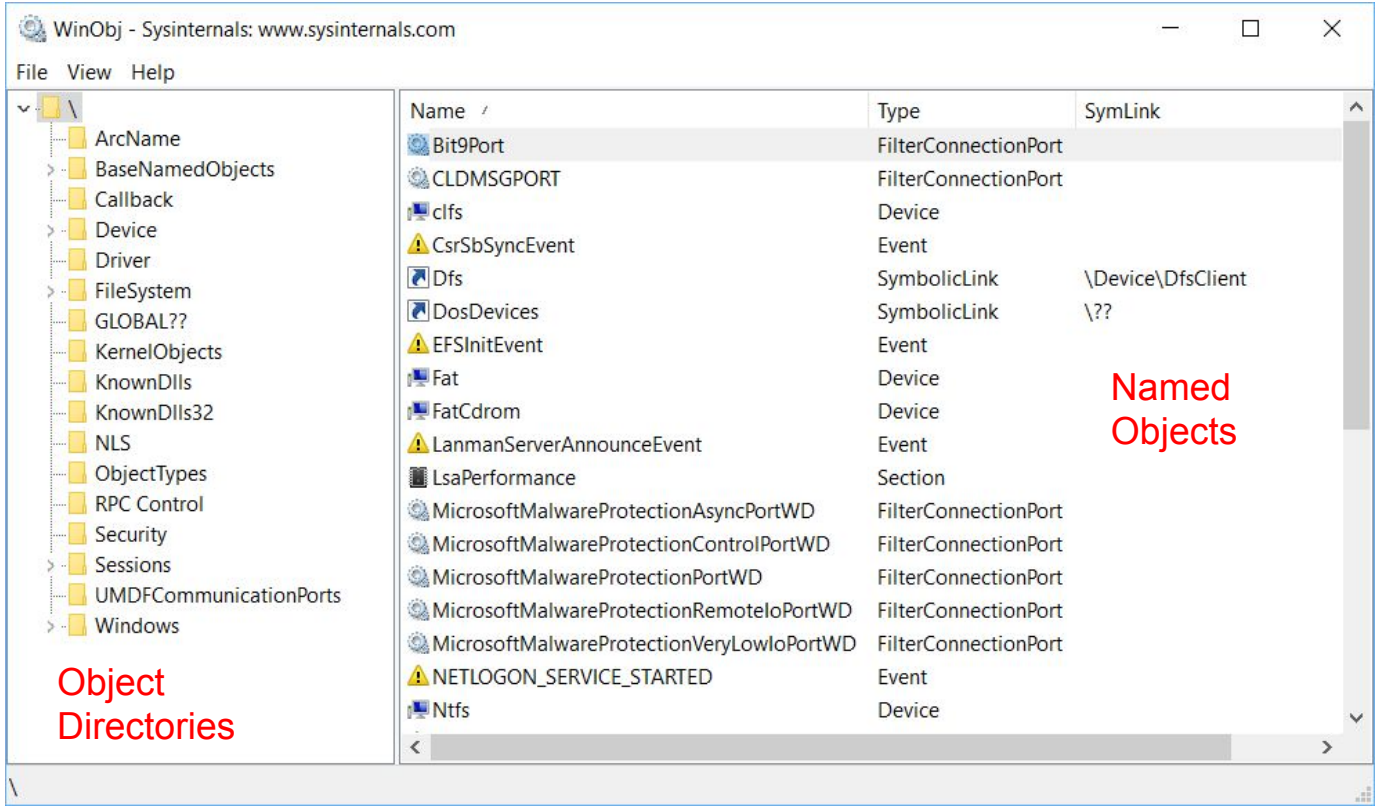
Setting up the Environment

PART 1
Windows Internals

Windows NT Architecture

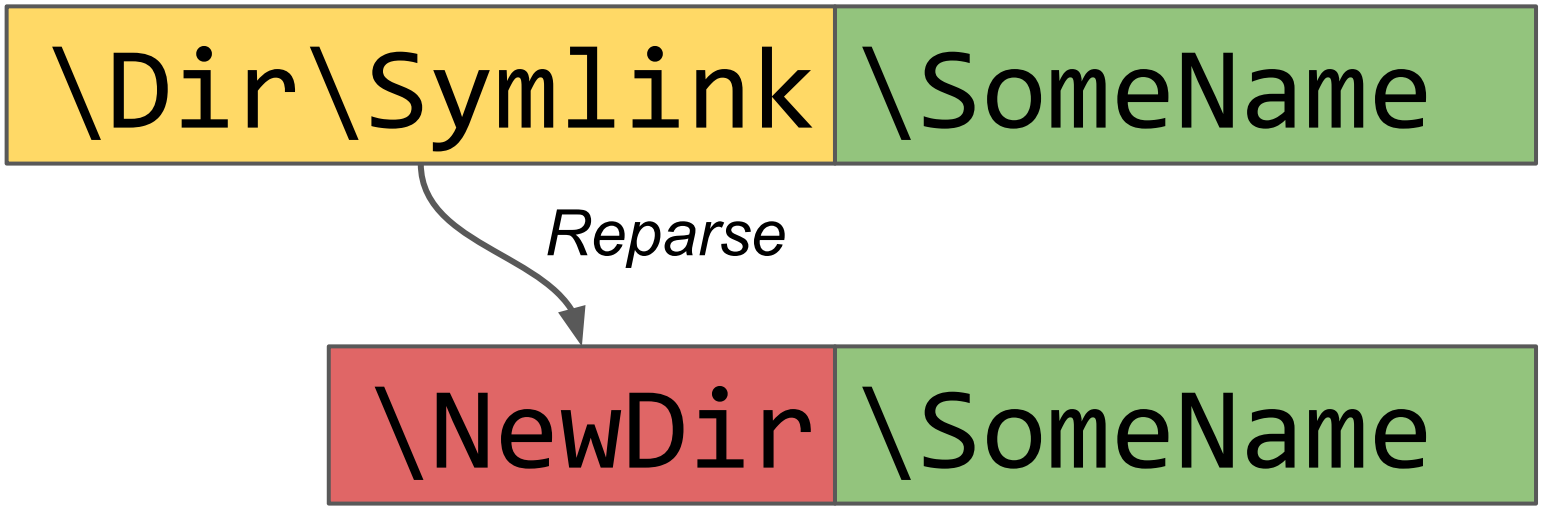


Windows Object Manager



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

Path	*Description*
\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
```

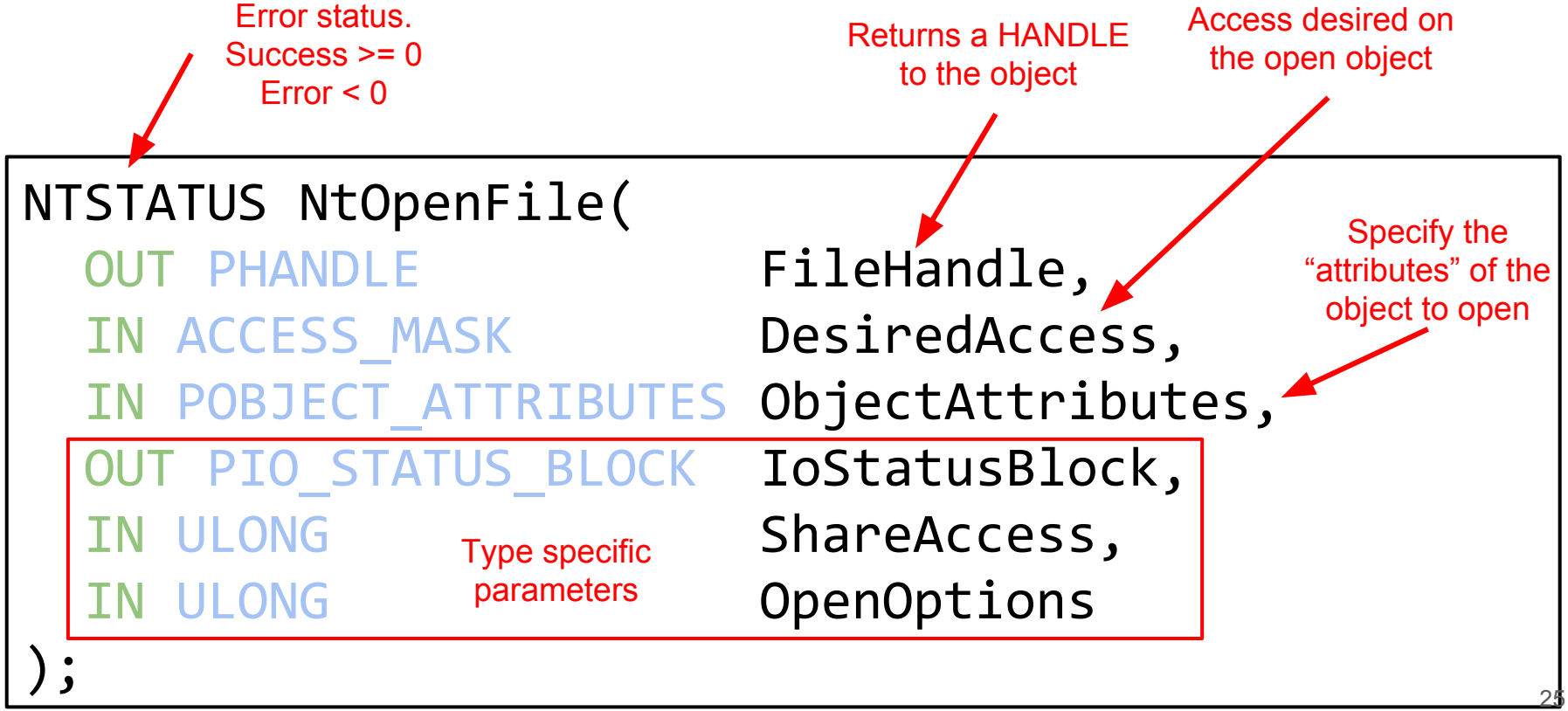
WORKSHOP 1

Inspecting Object Manager and Types

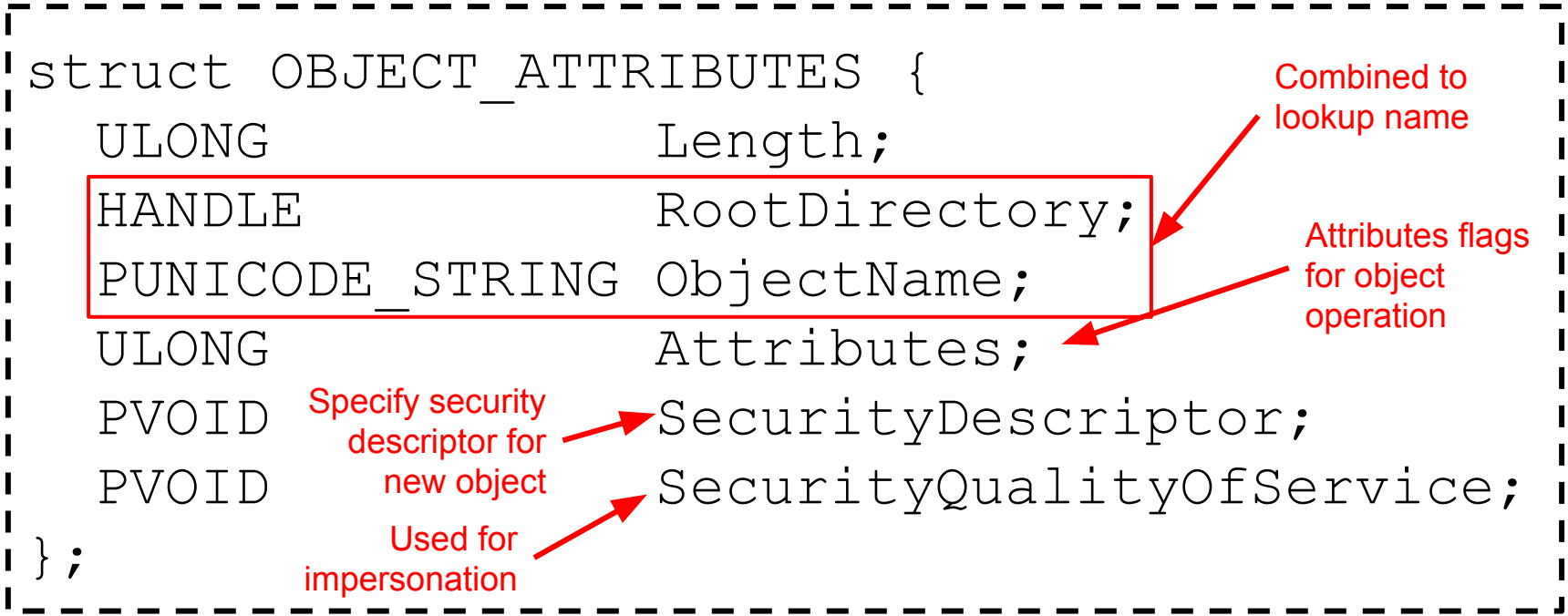
Object Manager System Calls

- 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



Object Attributes



Attribute Flags

Name	***Description***
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 *NtMakeTermporaryObject*.
  - 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()
```

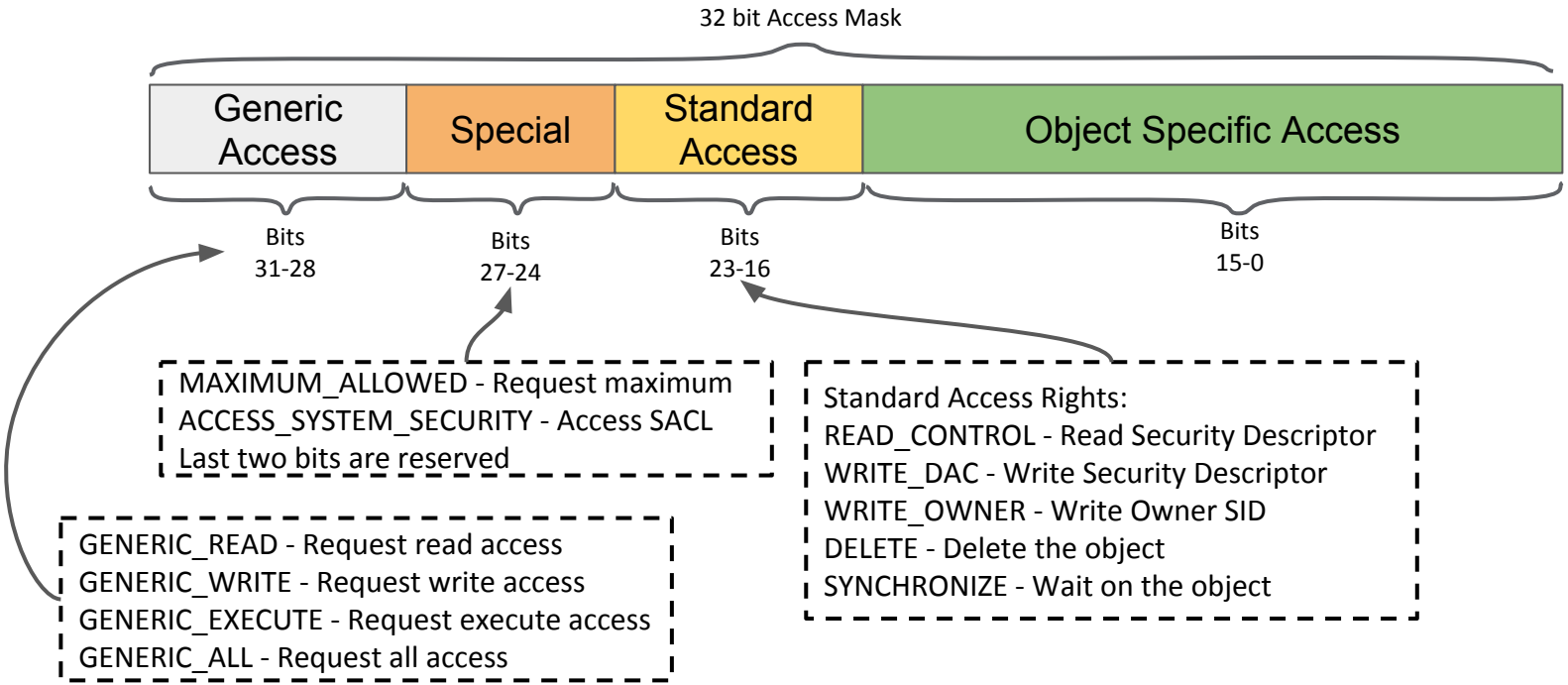
"Delete" object and close the opened object handle.

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

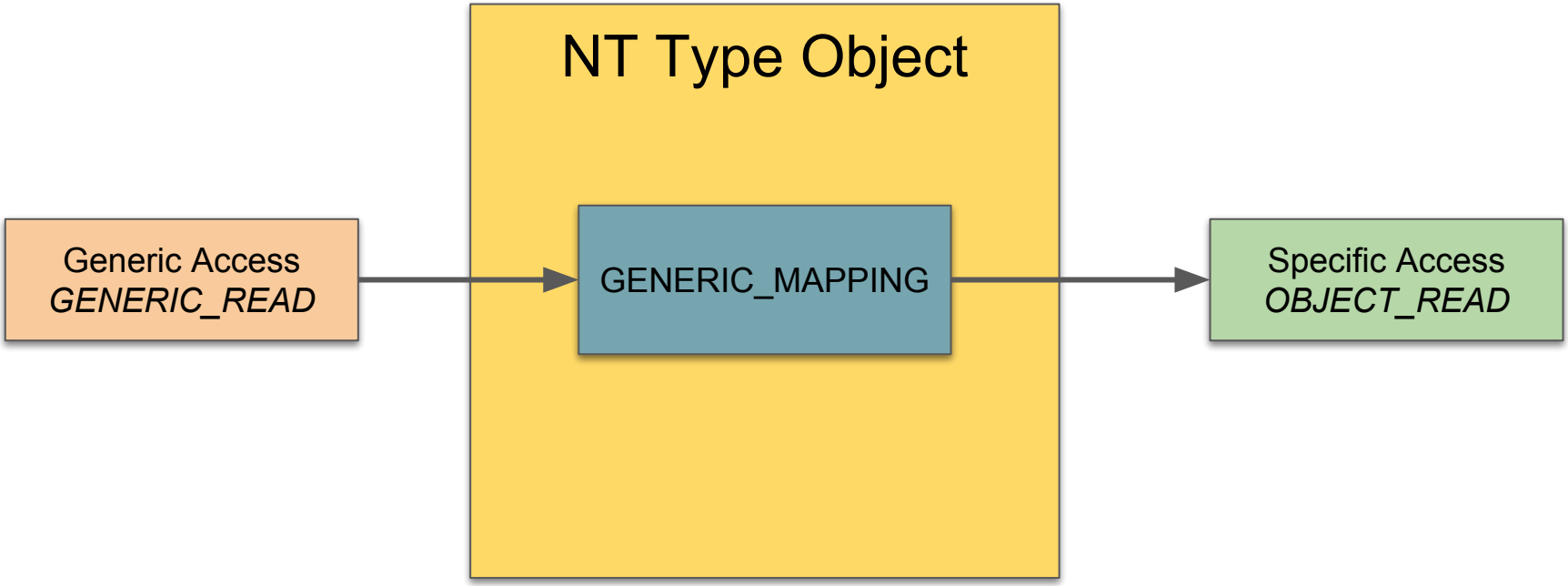
Use an object then close it immediately after.

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

Access Masks



Mapping Generic Access Rights



Specific Access Masks

Process Access	*Access Mask*	*File Access*
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
```

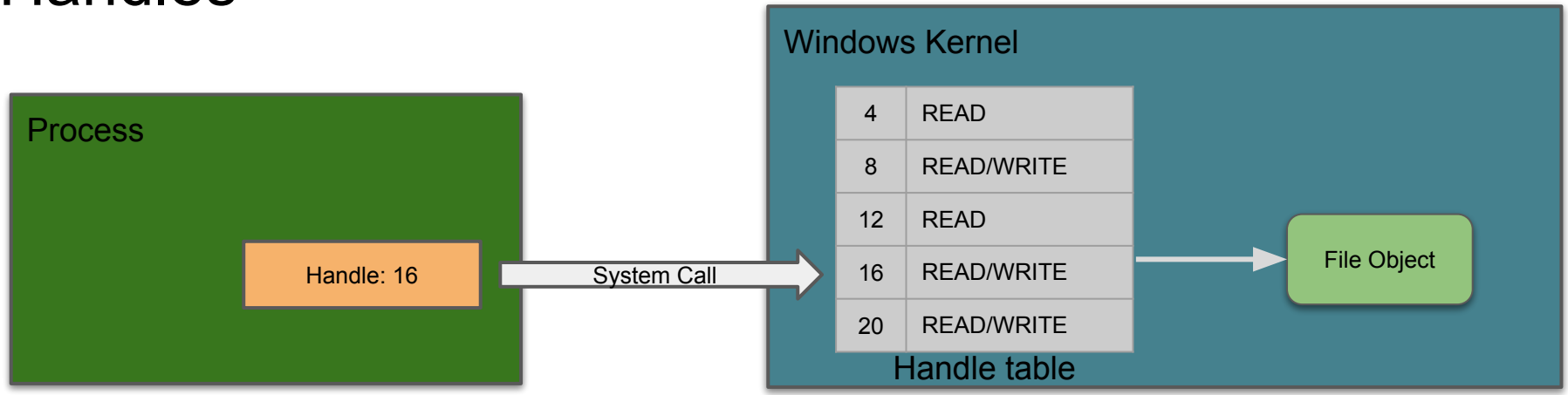
Map GenericRead to specific access rights for a File

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

Get granted access on an opened object.

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

Handles




```
NTSTATUS ObReferenceObjectByHandle(
  _In_      HANDLE                     Handle,
  _In_      ACCESS_MASK                DesiredAccess,
  _In_opt_  POBJECT_TYPE               ObjectType,
  _In_      KPROCESSOR_MODE            AccessMode,
  _Out_     PVOID                      *Object,
  _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 Duplication

```
NTSTATUS NtDuplicateObject(
  HANDLE      SourceProcessHandle,
  HANDLE      SourceHandle,
  HANDLE      TargetProcessHandle,
  PHANDLE     TargetHandle,
  ACCESS_MASK DesiredAccess,
  ULONG       HandleAttributes,
  ULONG       Options
);
```

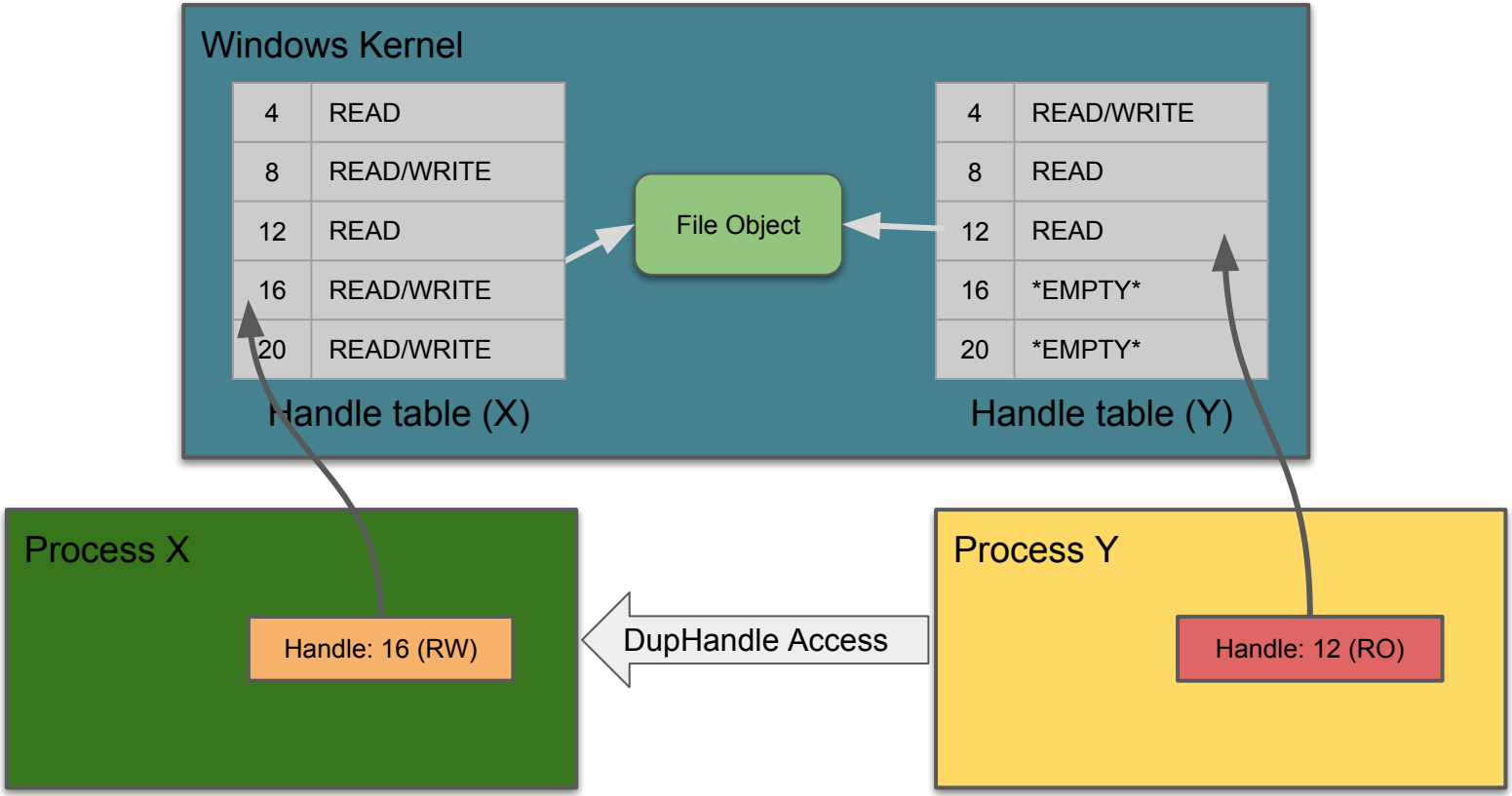
Source and target process handles need DupHandle access

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
```

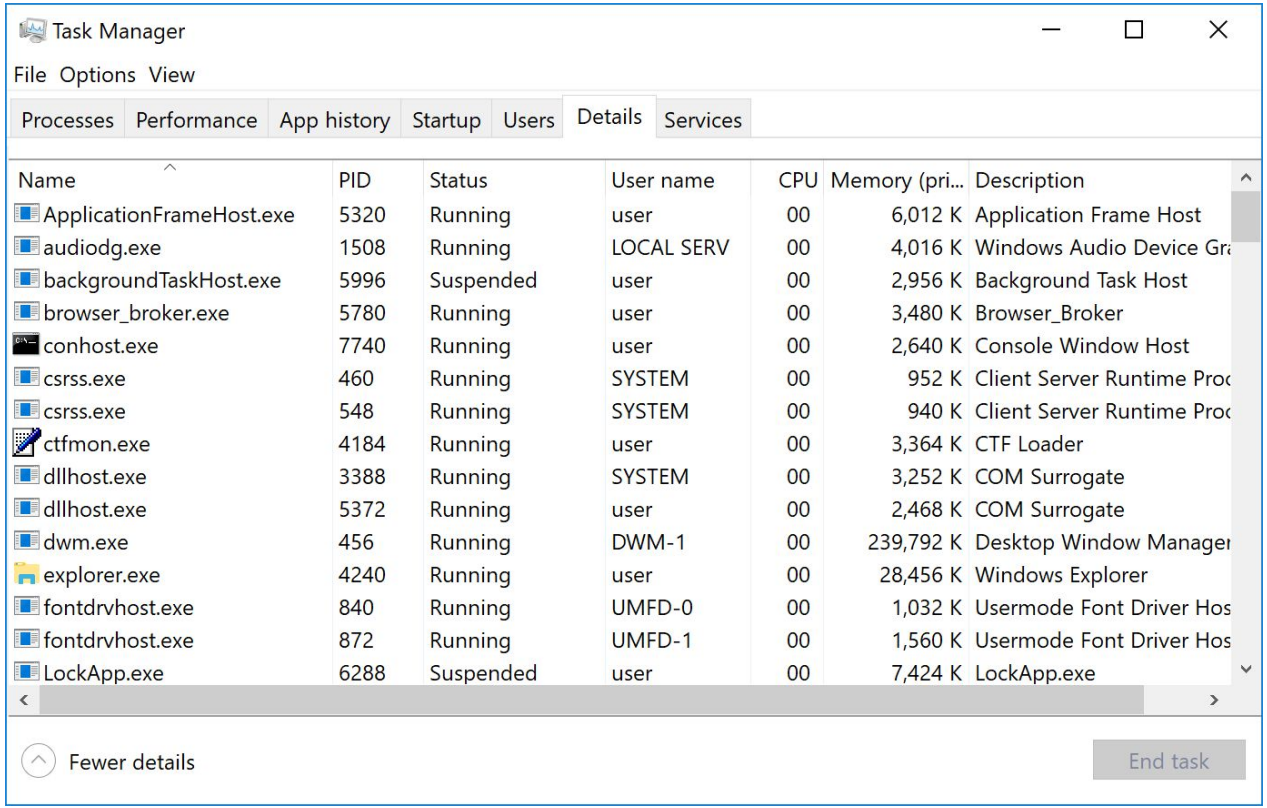
Kernel Object Structures

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



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

Process Access	***Description***
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.
Thread Access	***Description***
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).

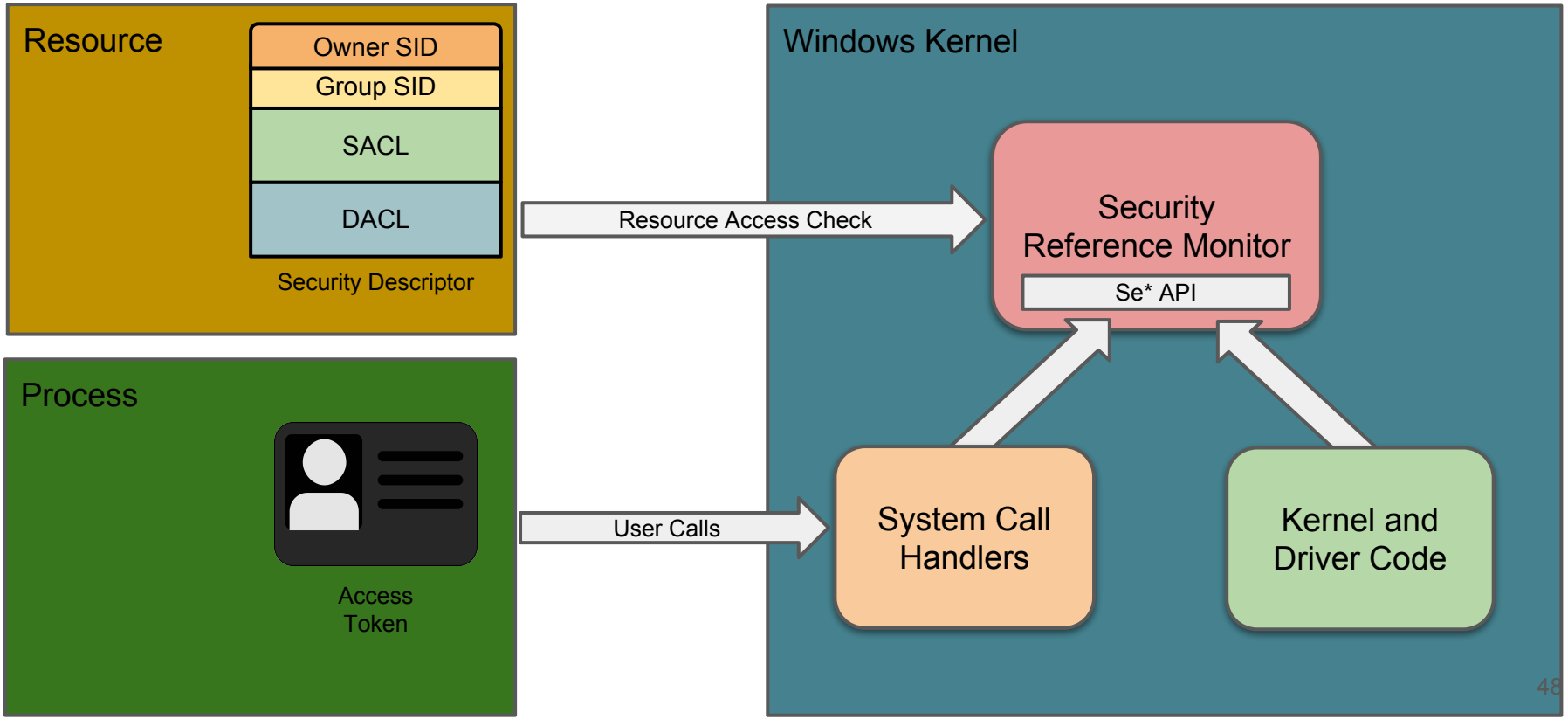
```
PS> Get-NtThread -InfoOnly
```

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 or it like an expanded UID/GID from Unix)

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

More at - https://msdn.microsoft.com/en-us/library/windows/desktop/aa379649(v=vs.85).aspx

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)
```

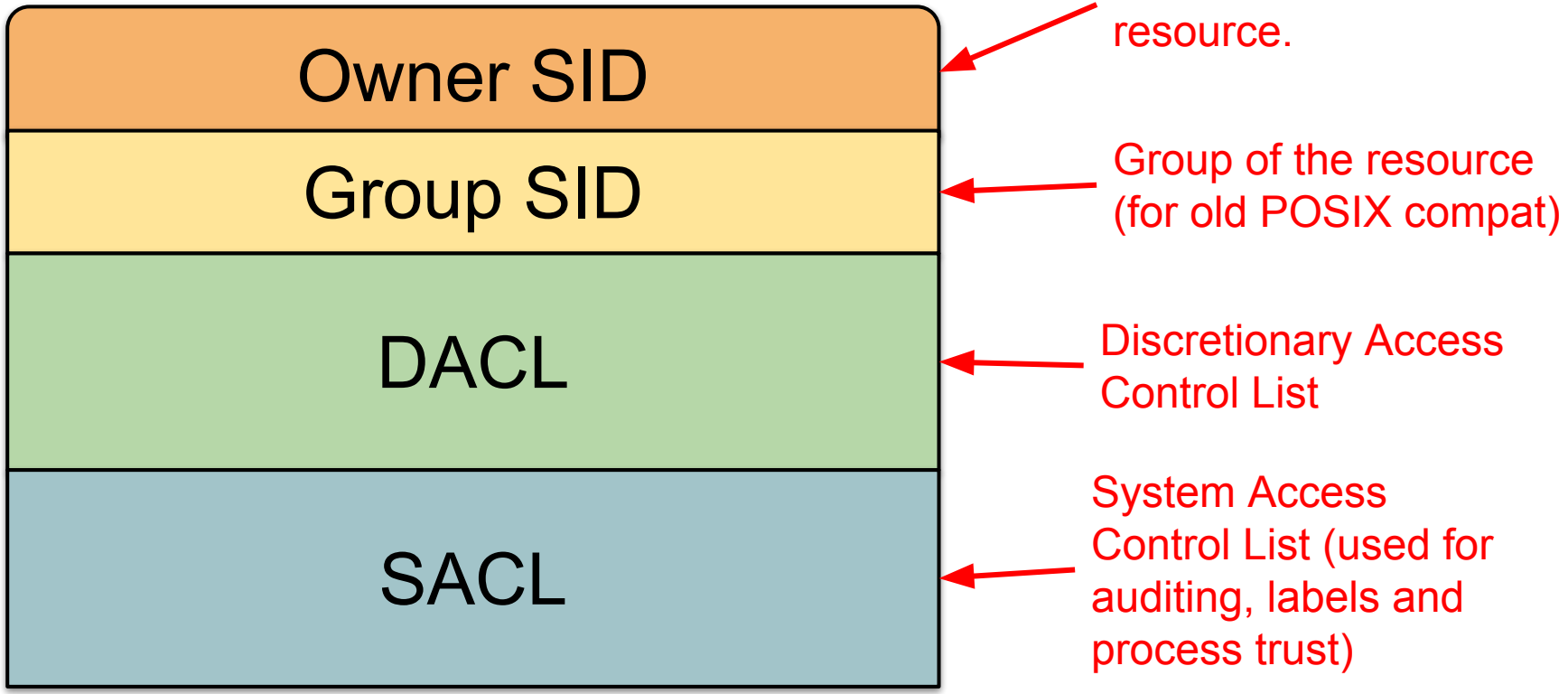
Get service SID for the service “WebClient”

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

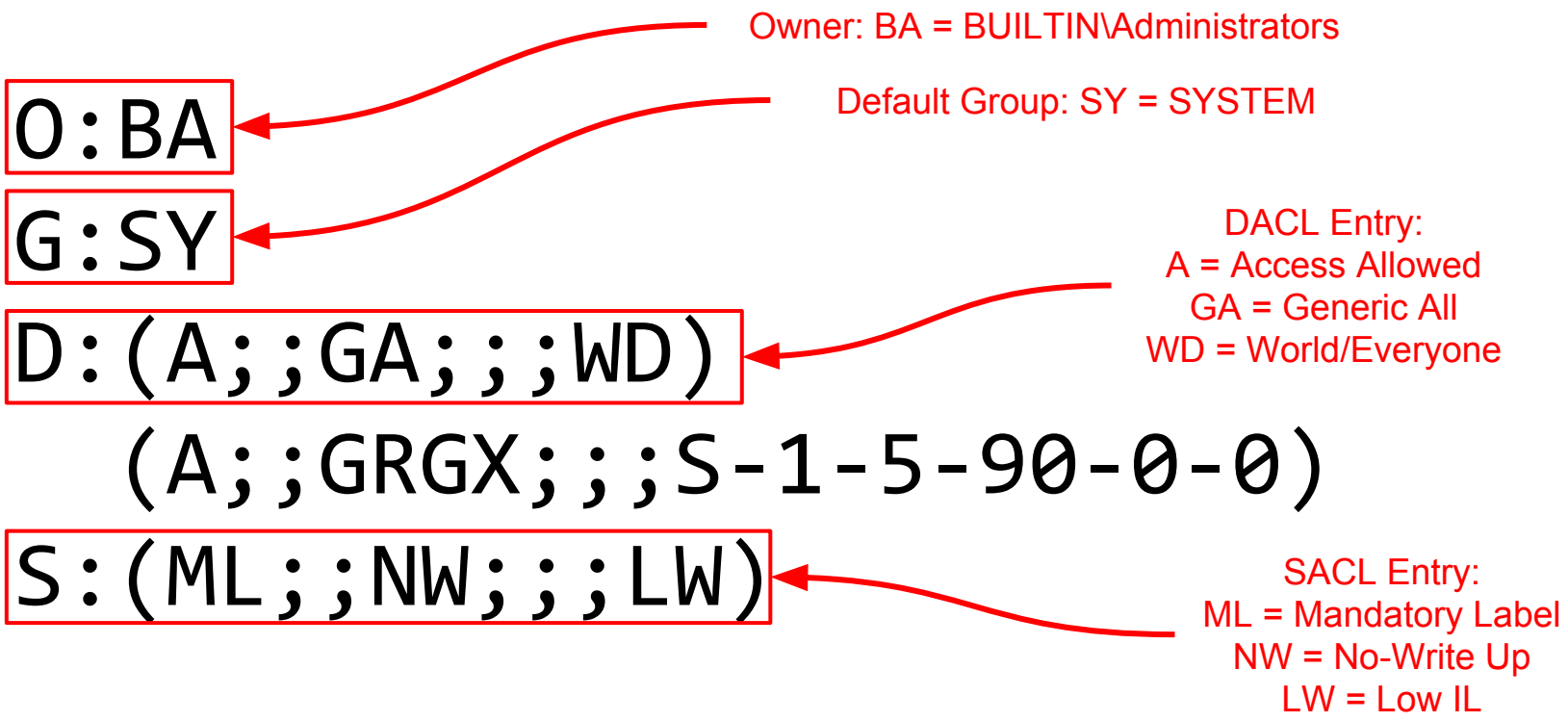
Get a well known SID for the Builtin Administrators

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

Security Descriptor



Security Descriptor Definition Language (SDDL)



Access Required for Security Descriptors

Access Granted	*Read*	*Write*
ReadControl	Owner Group Dacl Label	
WriteDac		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
```

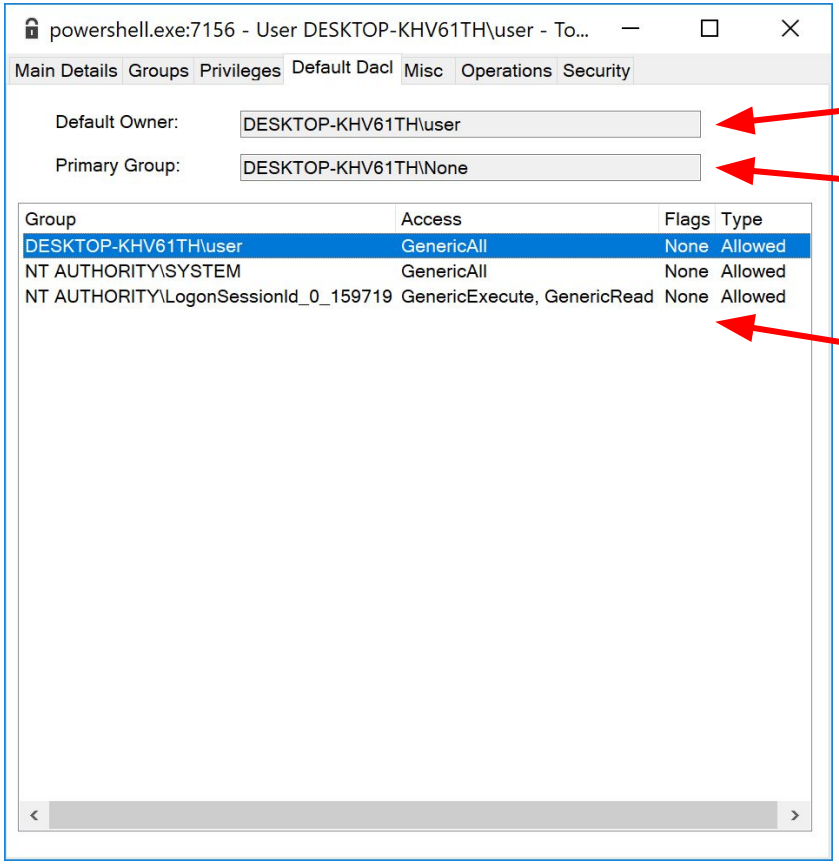
View the security descriptor of an object in a GUI

```
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



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
```

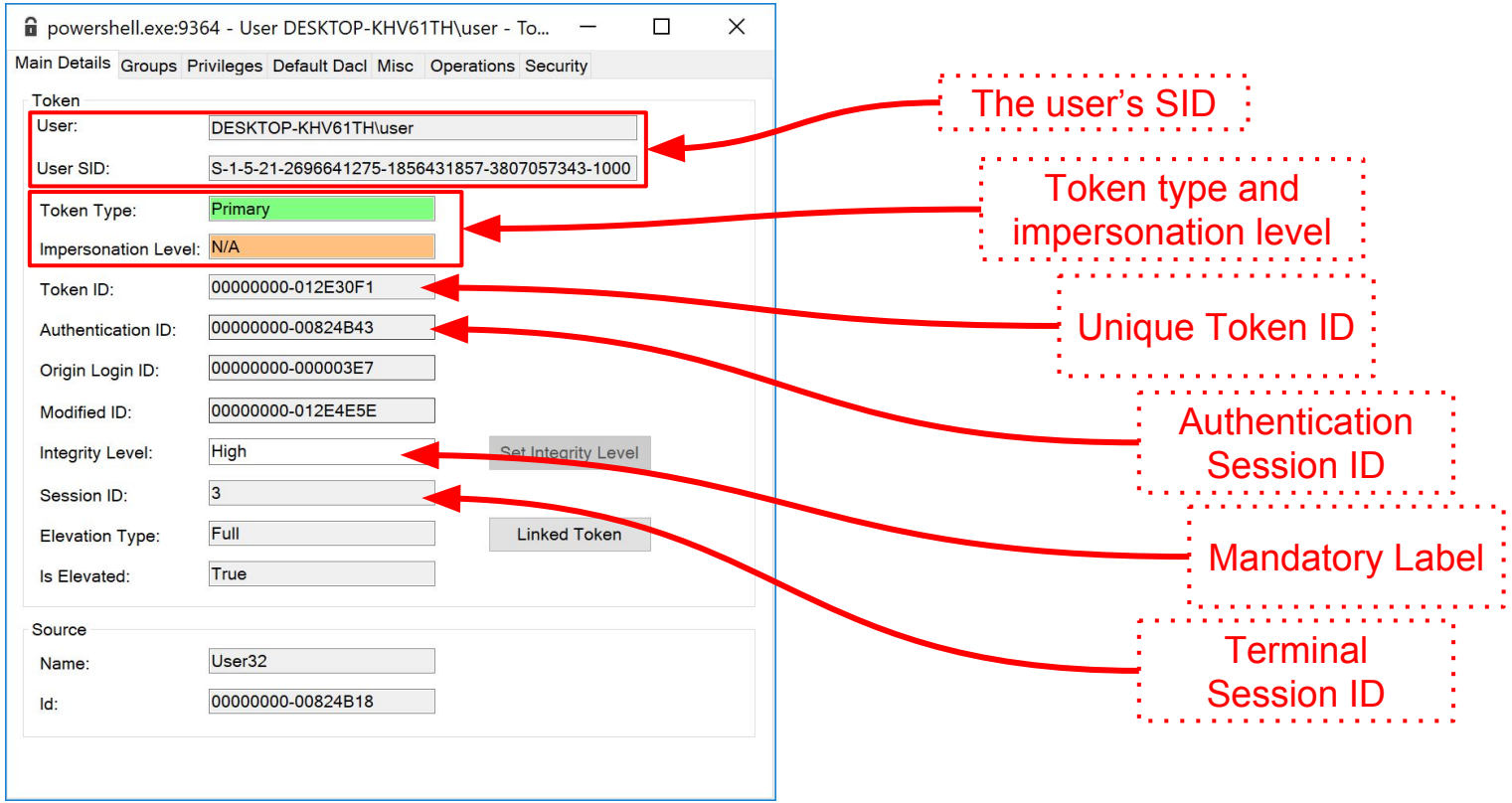
Generate a security descriptor based on an SDDL string.

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

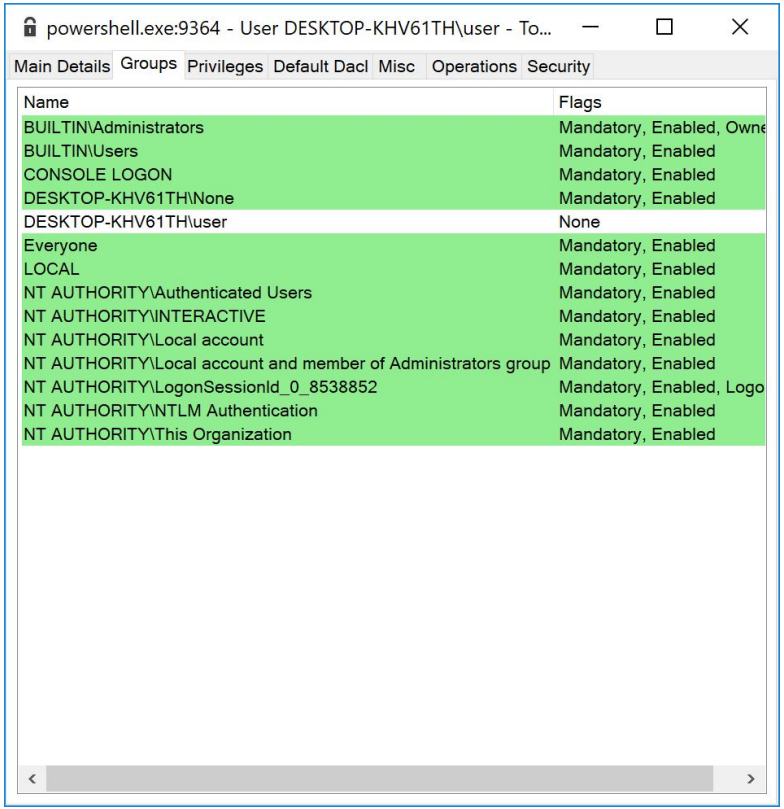
WORKSHOP 4

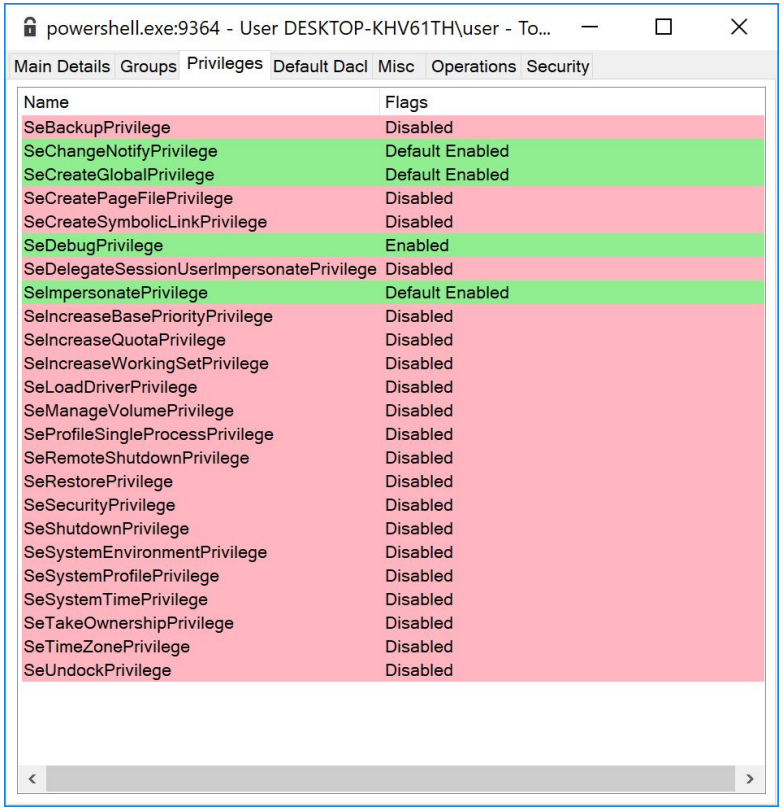
Accessing Security Descriptors

Access Token



Token Groups and Privileges



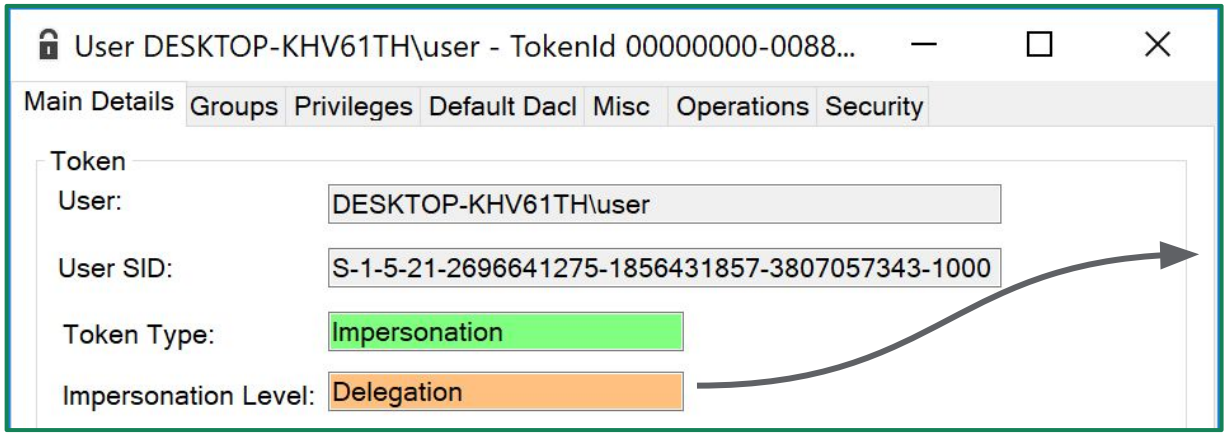


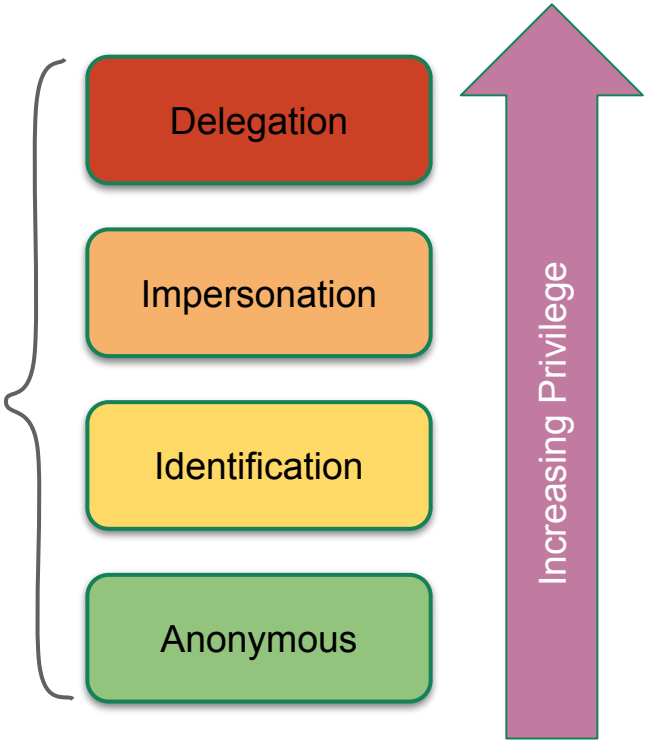
Token Privileges and Security Bypasses

- Token privileges can be enabled or disabled.

Privilege Name	***Security Bypass***
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

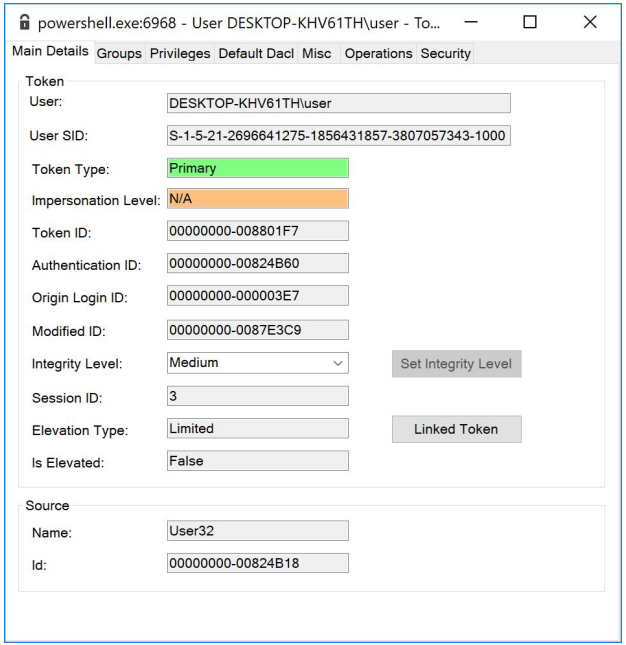
Impersonation Security Level





Token Duplication

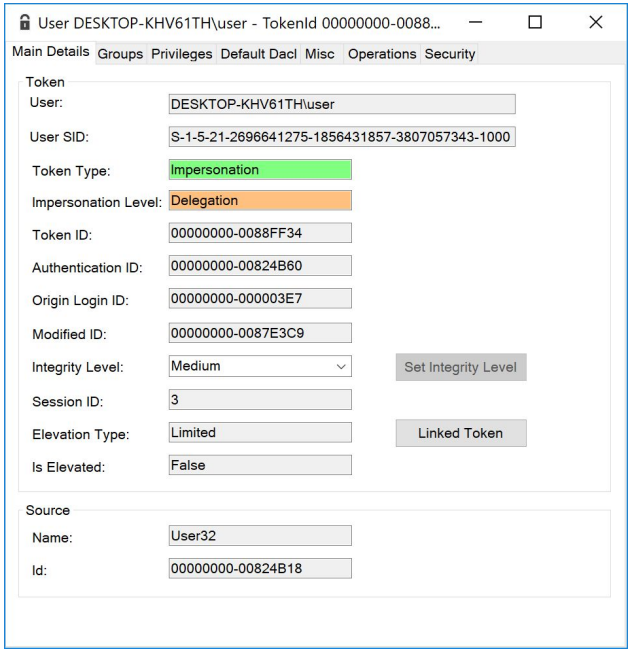
Can duplicate token's between types if have *Duplicate* access on handle.
Can specify token type and impersonation security level.



Primary

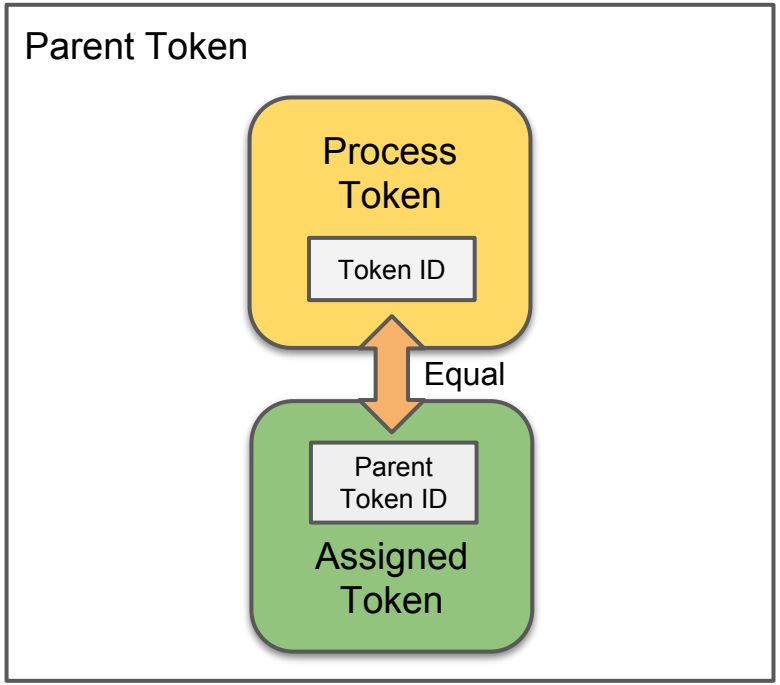
Always Allowed

Only if level >= Impersonate



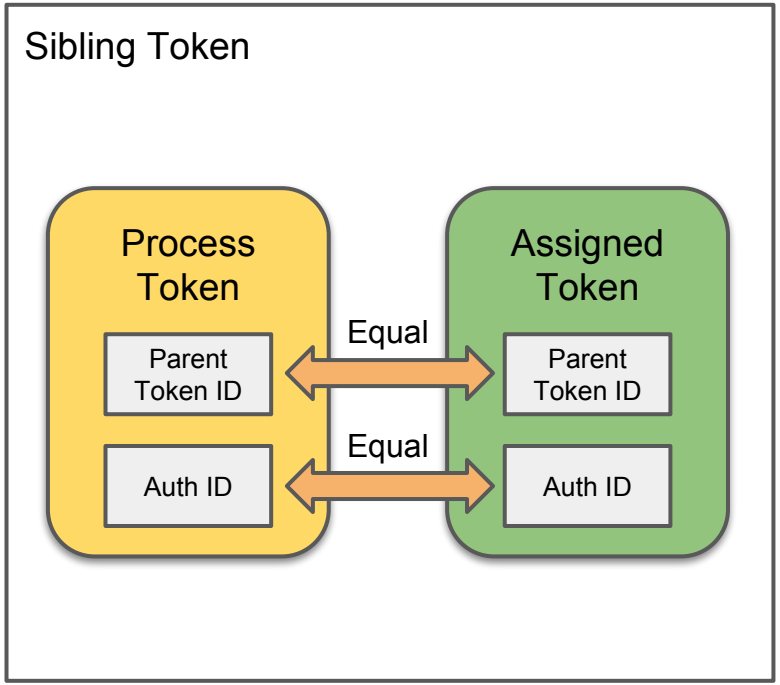
Impersonation

Assigning a Token to a Process



NtFilterToken

OR



NtDuplicateToken
NtCreateLowBoxToken

Setting an Impersonation Token

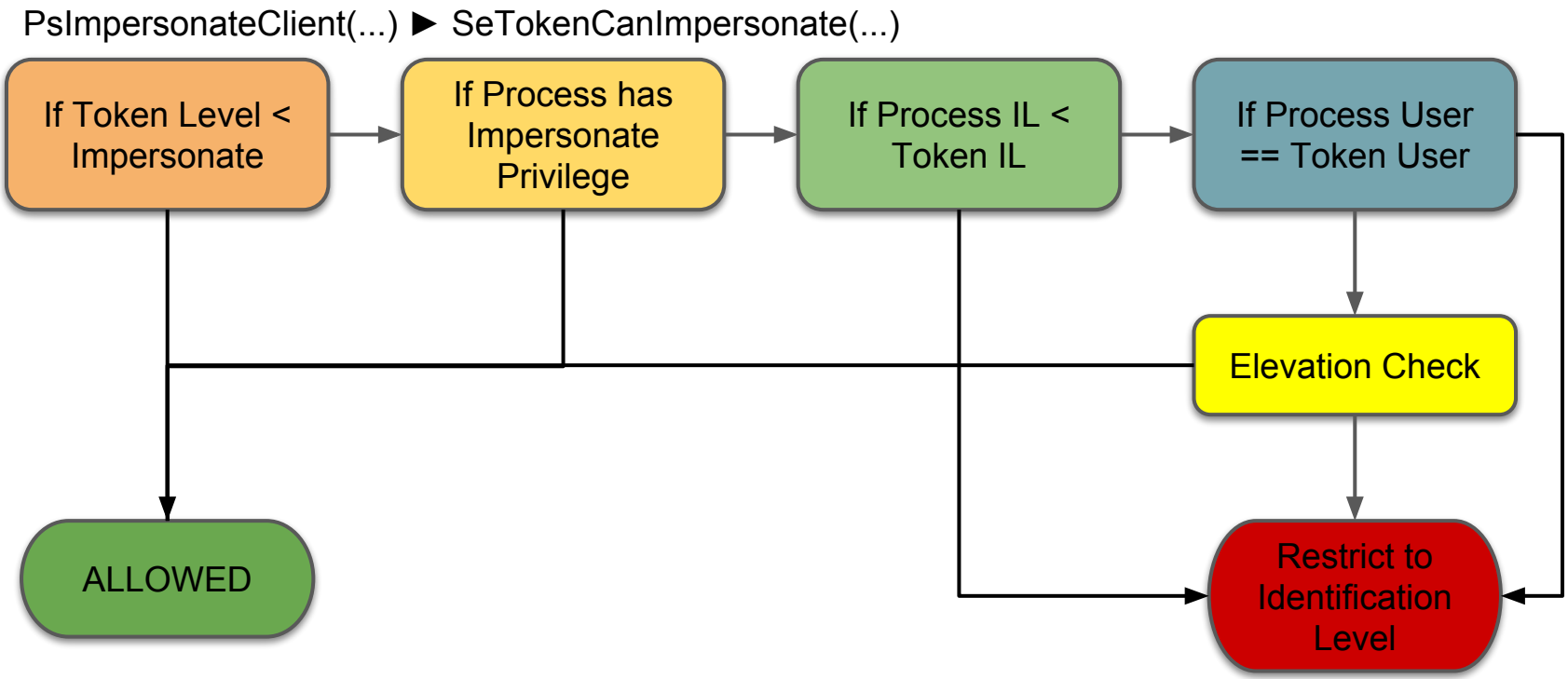
- Impersonation assigns a token to a thread, replaced the token used in access checks for the majority of system calls

Direct Setting
SetThreadToken()
ImpersonateLoggedOnUser()
NtSetInformationThread(...)

Indirect Setting
ImpersonateNamedPipeClient()
RpcImpersonateClient()
CoImpersonateClient()

Kernel Setting
PsImpersonateClient()
SeImpersonateClient/Ex()

Impersonation Token Security



Get the current process's token.

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

Get's 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.

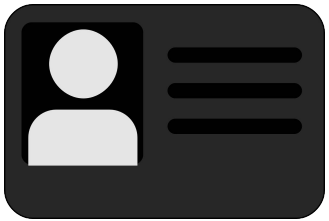
```
PS> Invoke-NtToken $token { ... Do something }
```

WORKSHOP 5

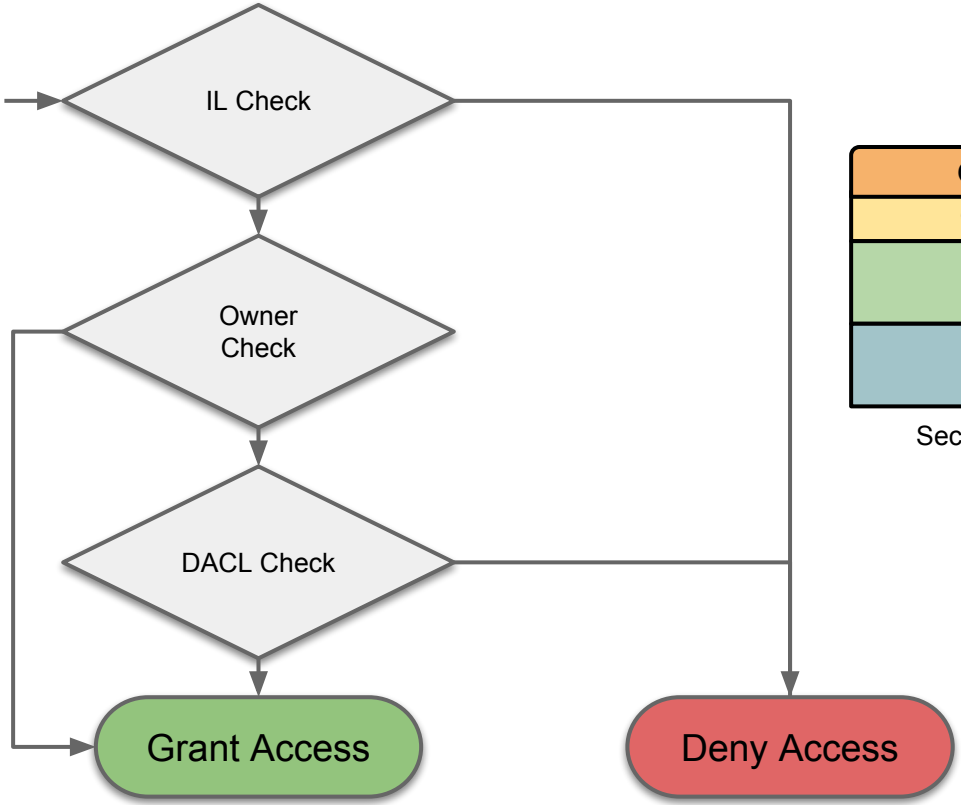
Access Tokens

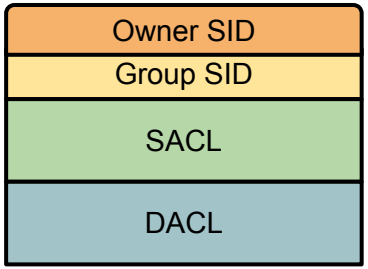
Security Access Check (SeAccessCheck)

Desired Access:
e.g. Read and Write



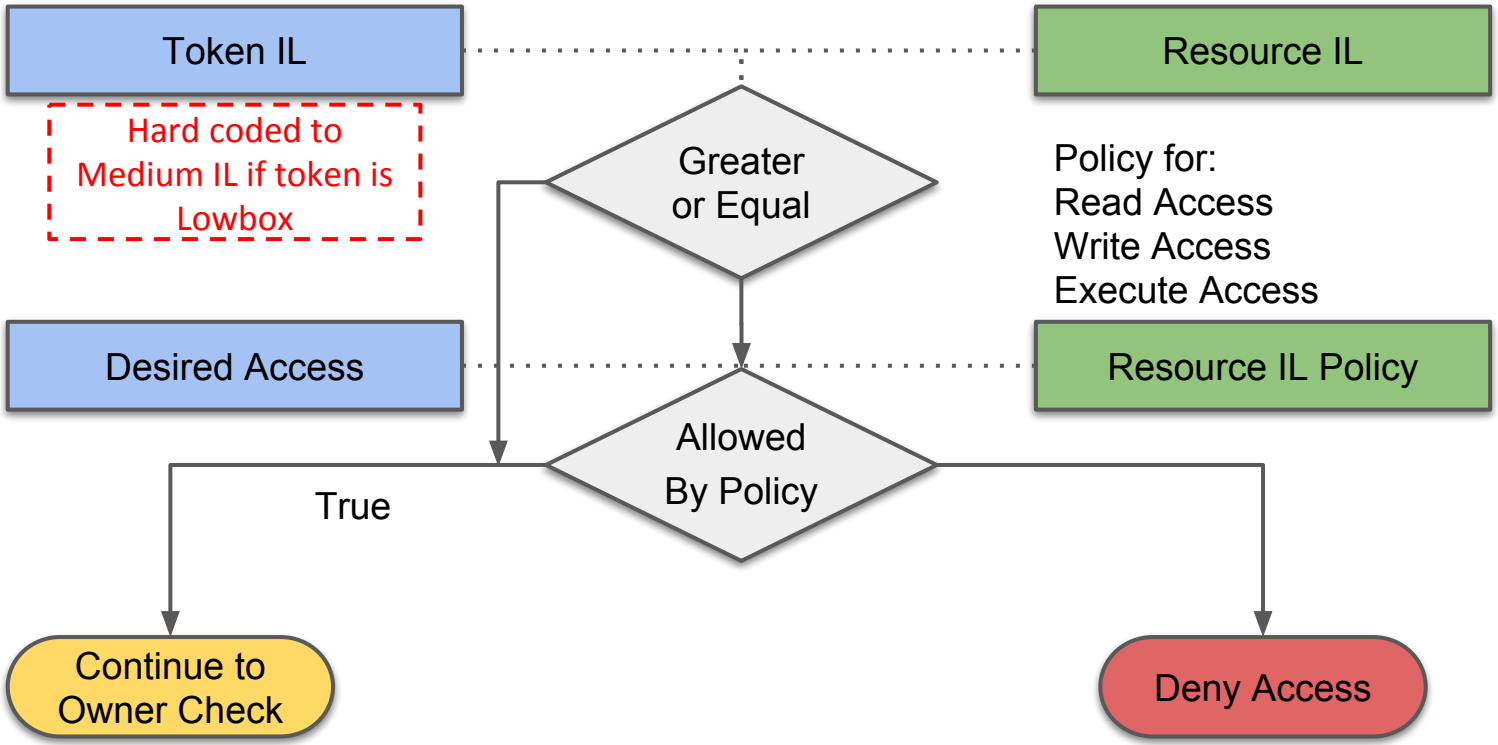
Access Token



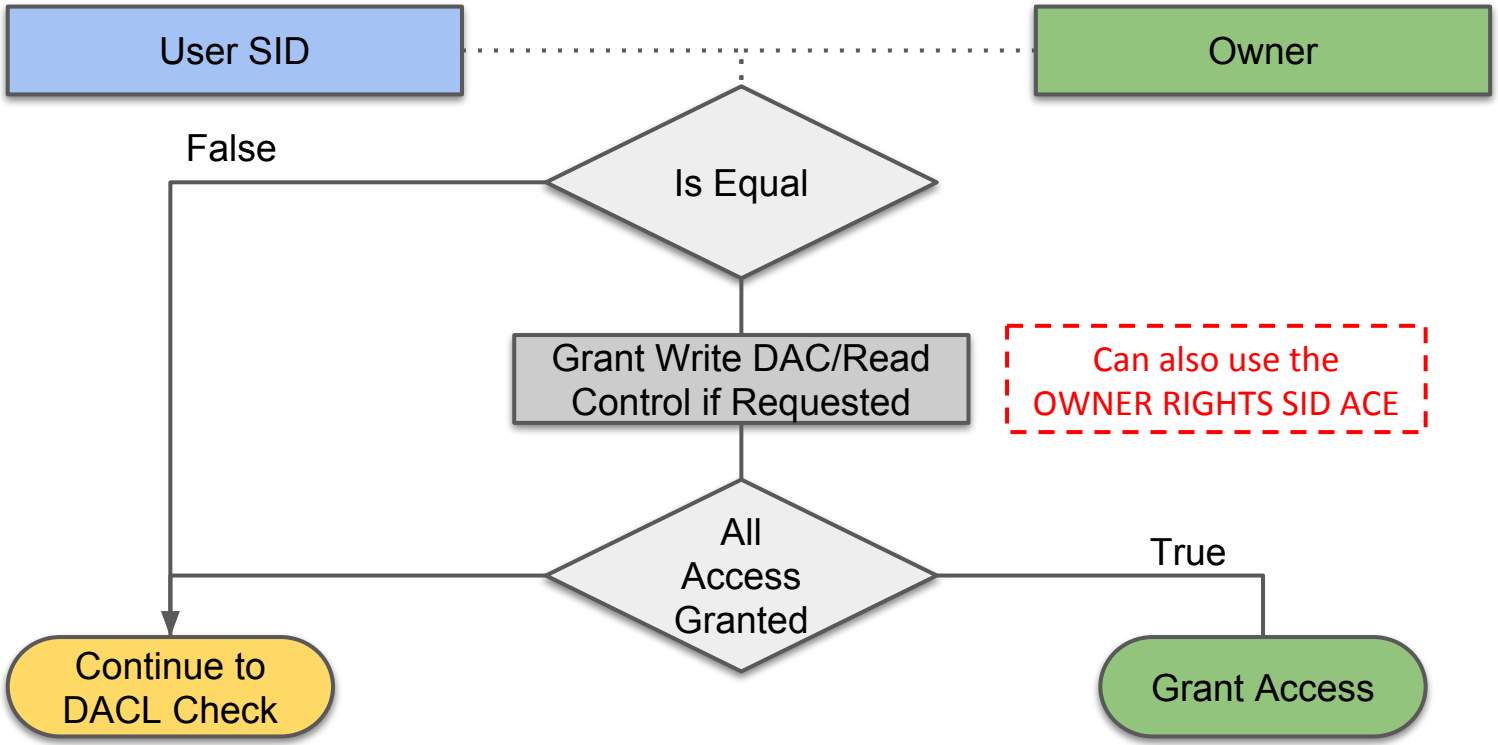


Security Descriptor

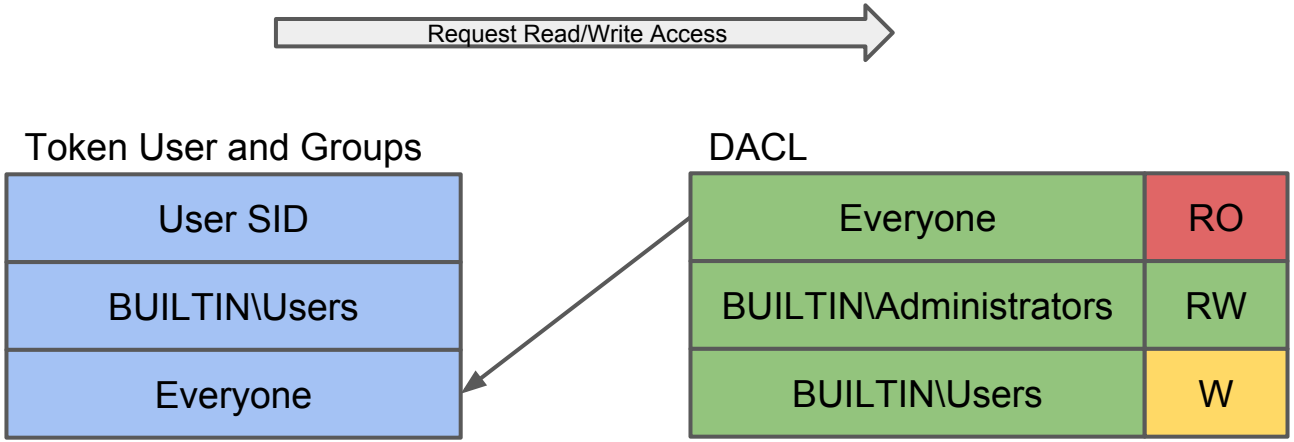
Mandatory Integrity Level Check



Owner Check

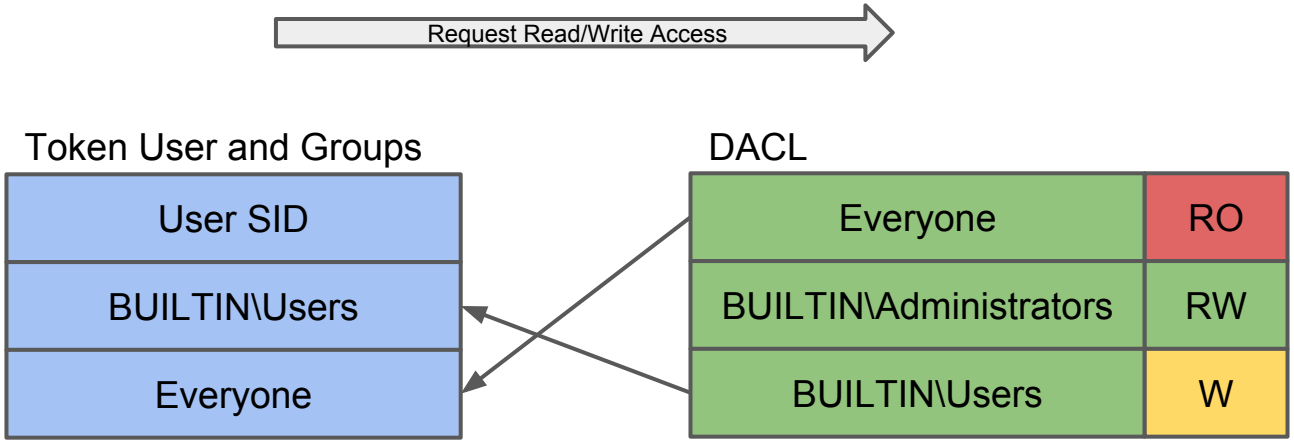

User SID
Owner
False
Is Equal
Grant Write DAC/Read Control if Requested
Can also use the OWNER RIGHTS SID ACE
All Access Granted
True
Continue to DACL Check
Grant Access

Kernel DACL Check



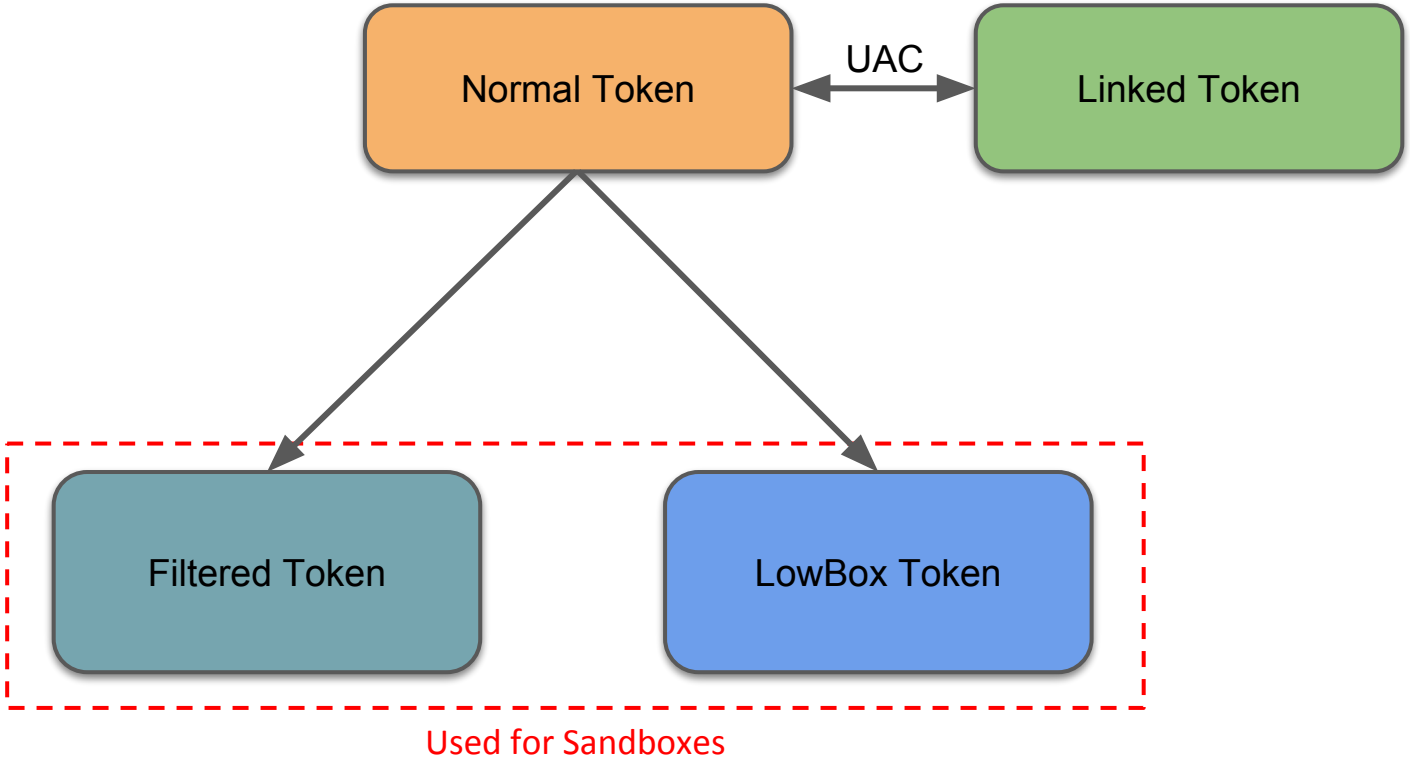
Current Granted Access: Read Only

Kernel DACL Check

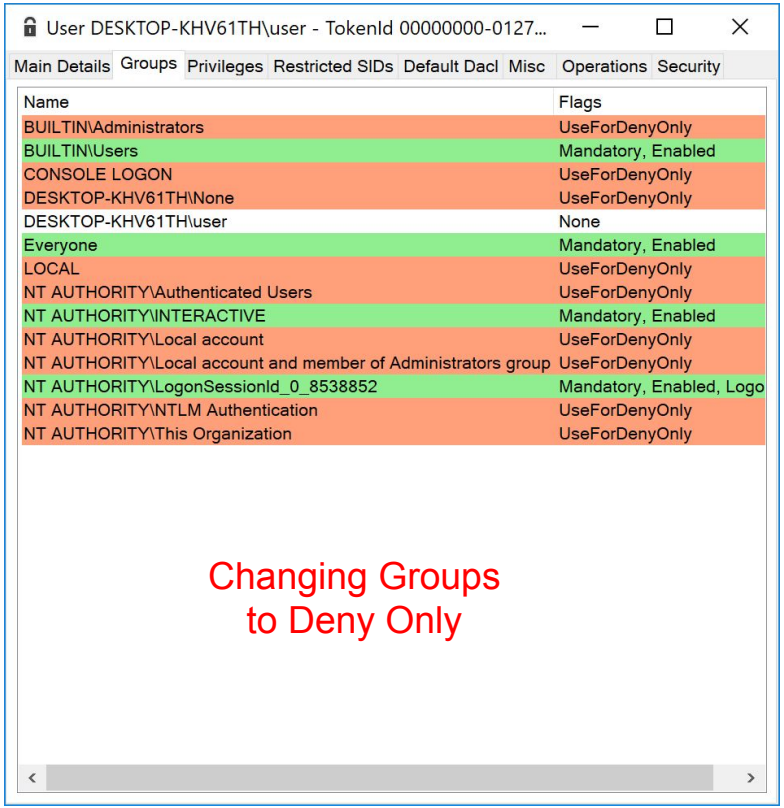


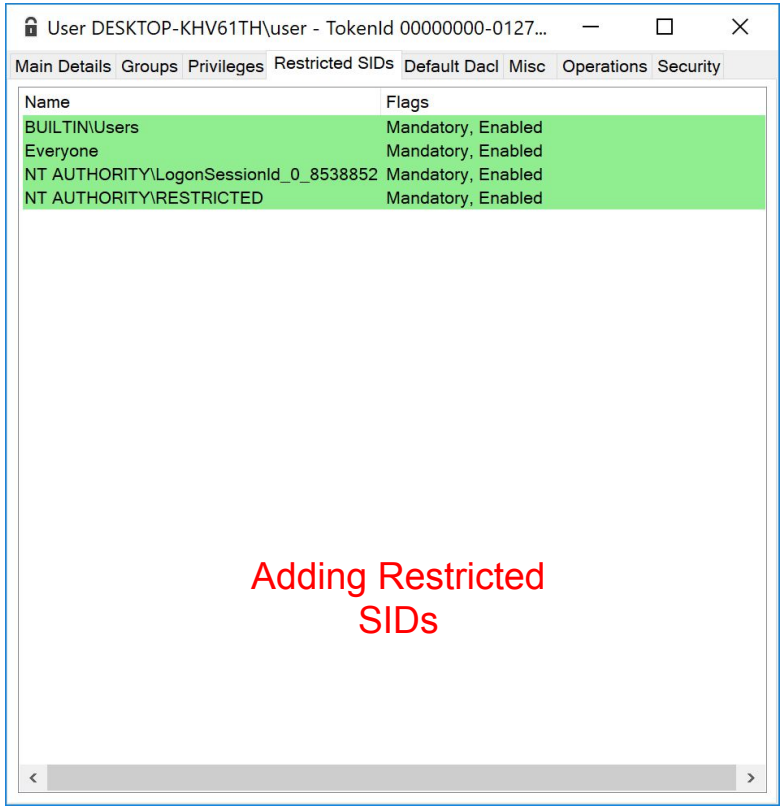
Final Granted Access: Read/Write

Sandbox Tokens

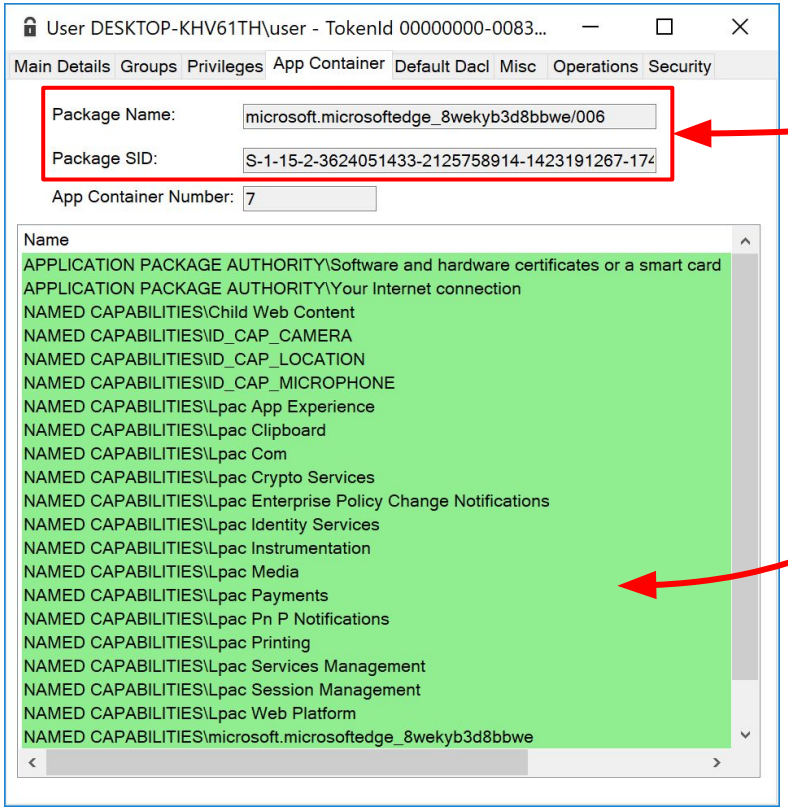


Restricted Token





LowBox Token

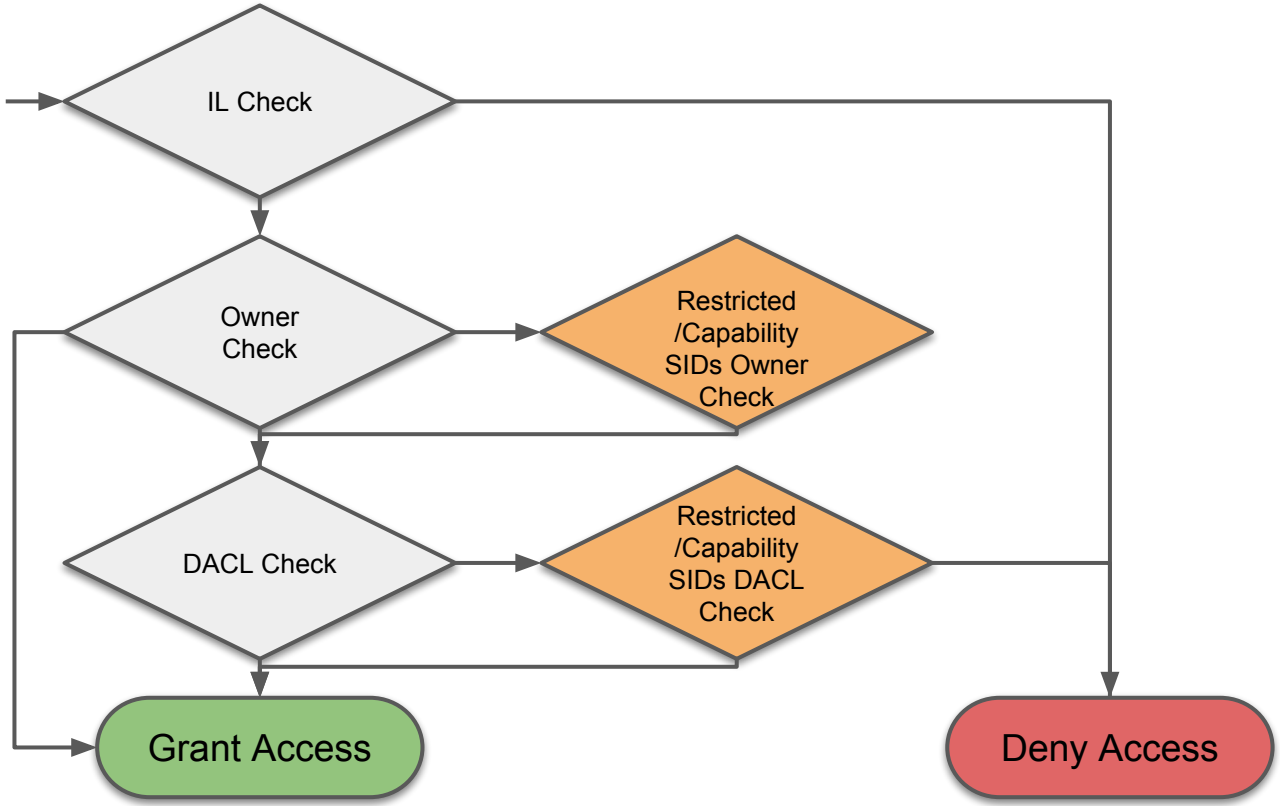


Package / AppContainer SID

Capability SIDs

Restricted/Lowbox Token Access Check

Desired Access:
e.g. Read and Write



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 brige
  - \Registry\User - Attachment point for user hives.

Win32 to Native Registry Mapping

Win32 Path	***Native Path***	***Description***
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.

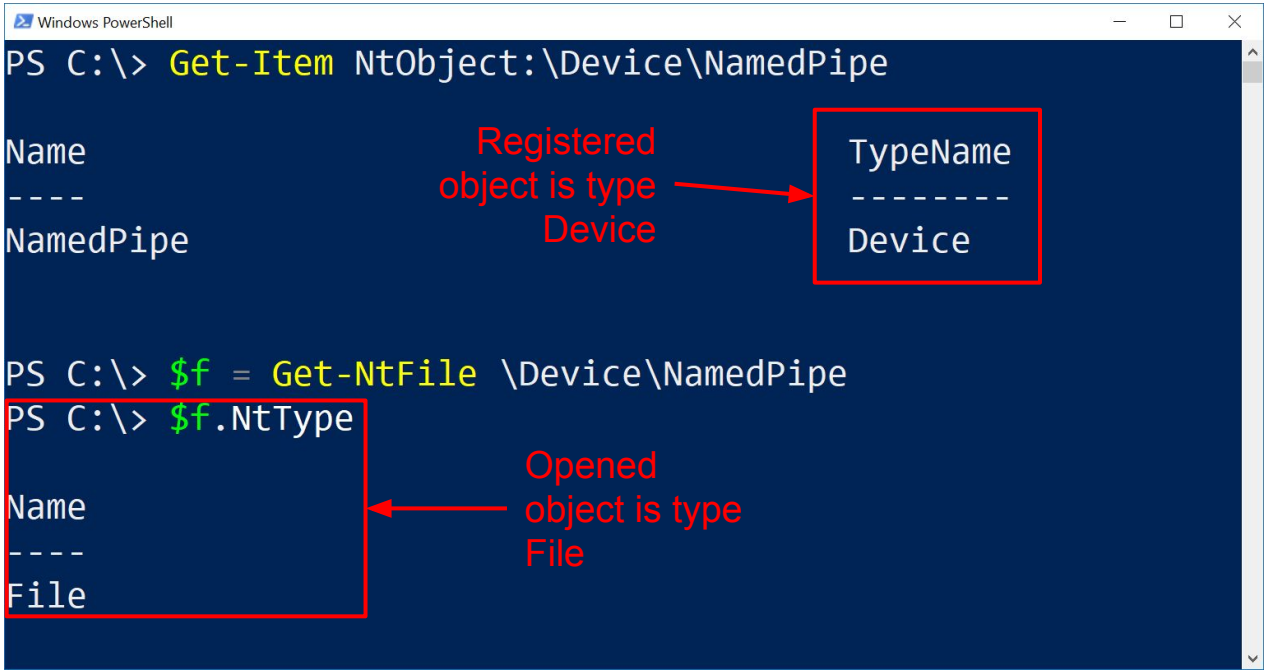
```
PS> Add-NtKey "\??\c:\temp\reg.hiv" -KeyPath `
     \Registry\A\ABC -LoadFlags AppKey
```

WORKSHOP 7

Accessing Registry Keys

Device Objects

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

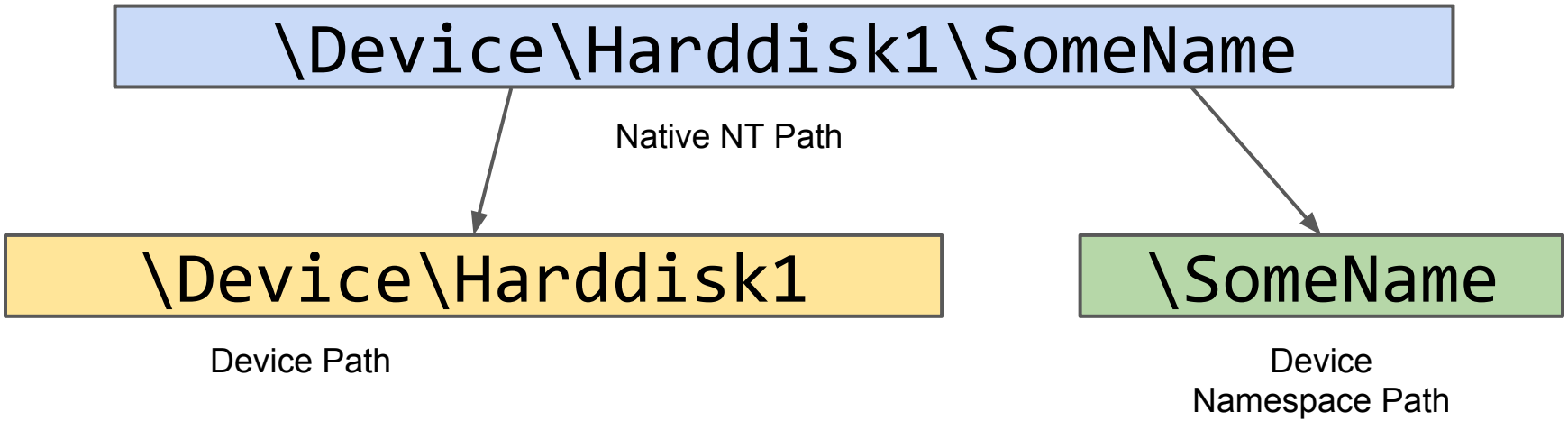


Opening a Device Name

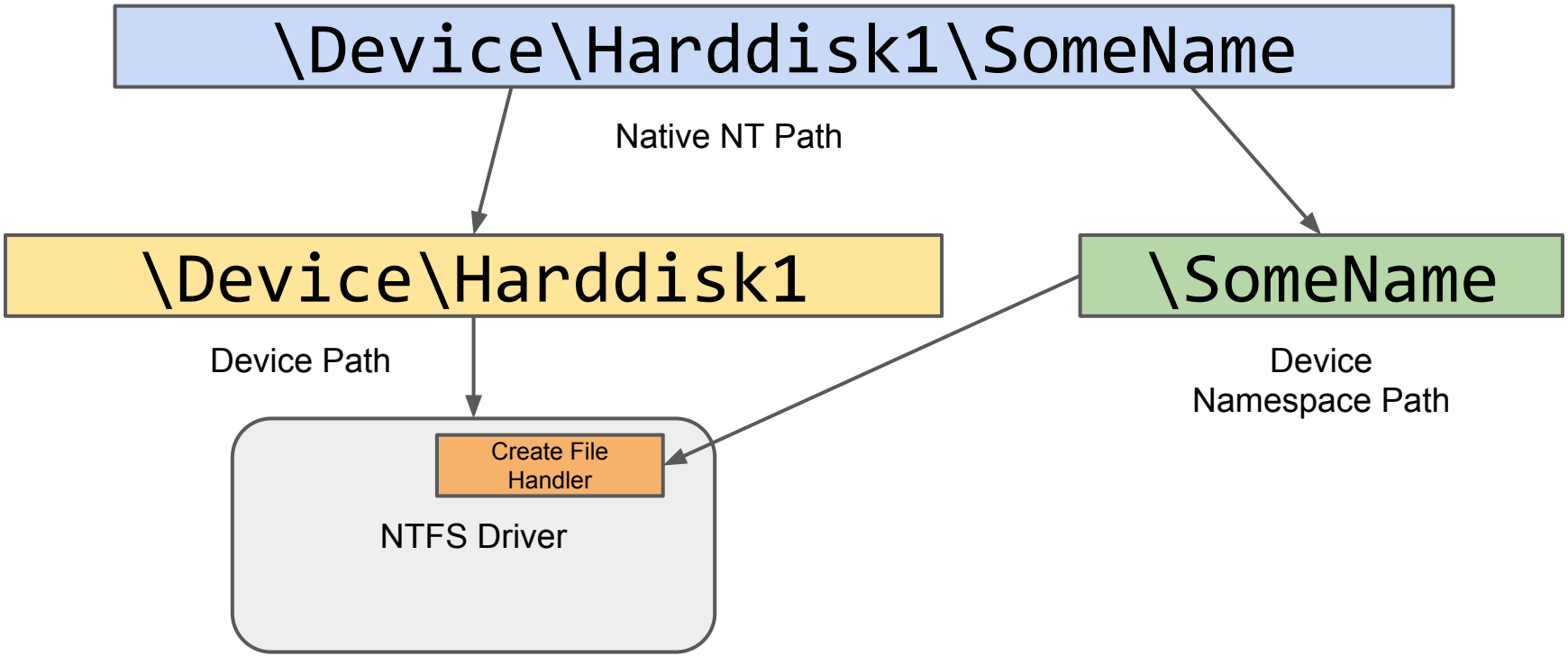
```
\Device\Harddisk1\SomeName
```

Native NT Path

Opening a Device Name

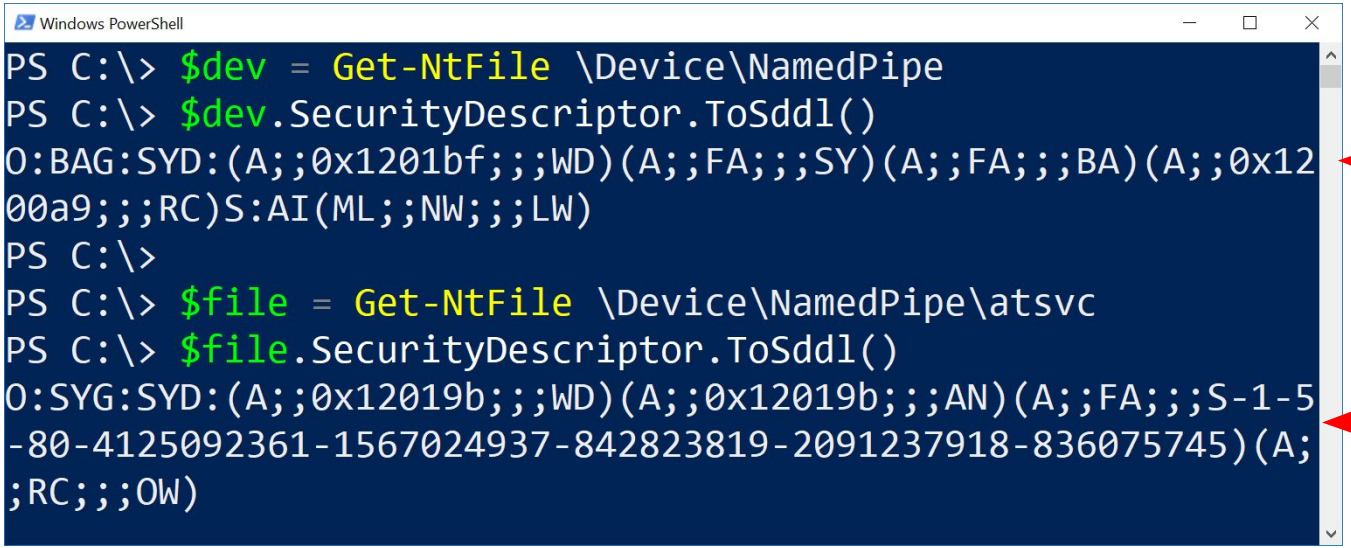


Opening a Device Name



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.



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

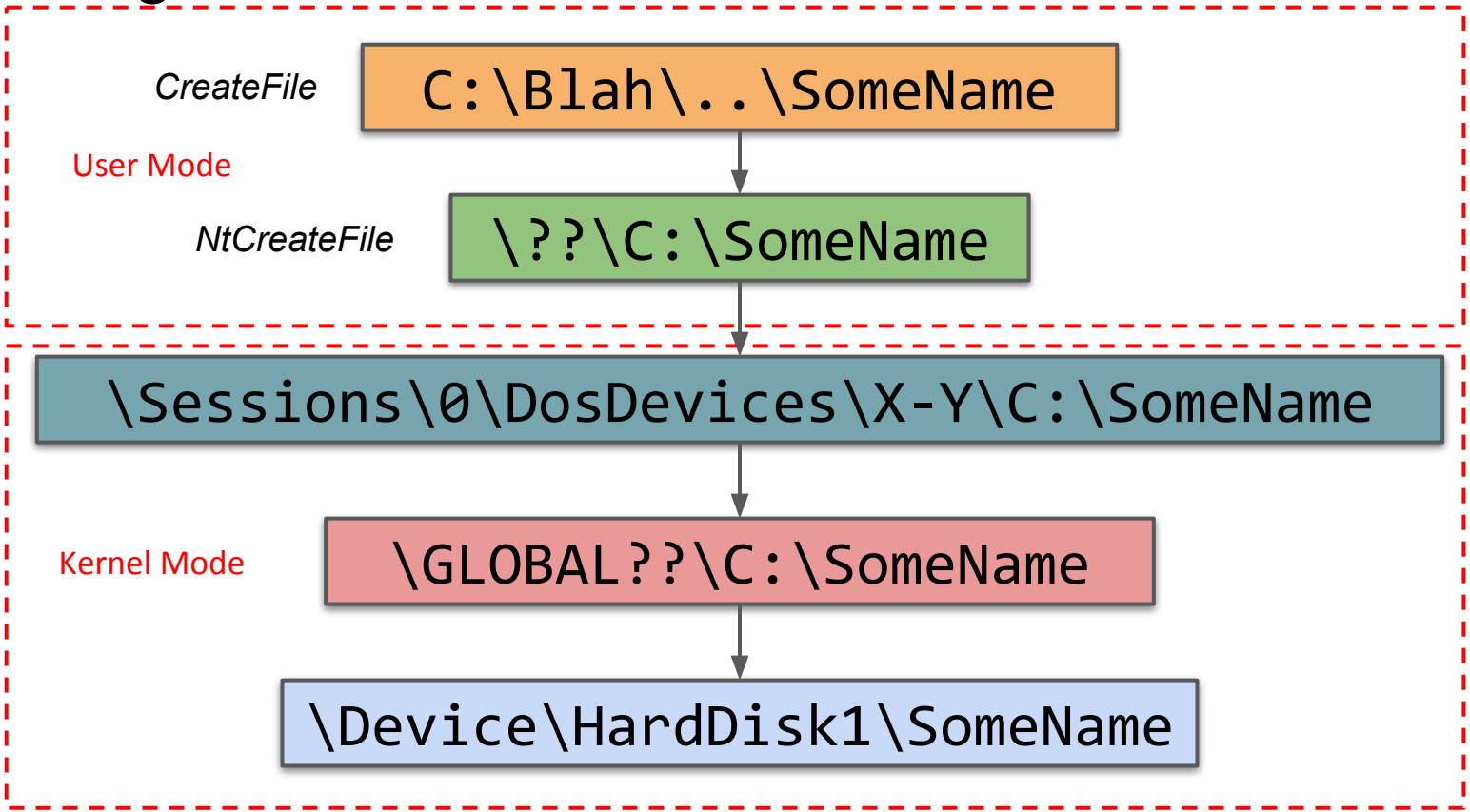
NTFS Invalid Characters

	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

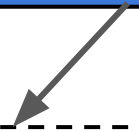
- File system calls take a sharing mode flag, determines whether a second open will succeed.
- Has three flags:
  - FILE_SHARE_READ Subsequent opens can be for read access
  - FILE_SHARE_WRITE Subsequent opens can be for write access
  - FILE_SHARE_DELETE Subsequent opens can be for delete access
- Not 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;
  DO->MajorFunction[IRP_MJ_CLEANUP] = CleanupFunction;
  DO->MajorFunction[IRP_MJ_FILE_SYSTEM_CONTROL] = FSControlFunction;
  DO->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IoControlFunction;

  return status;
}
```

IRP Handlers

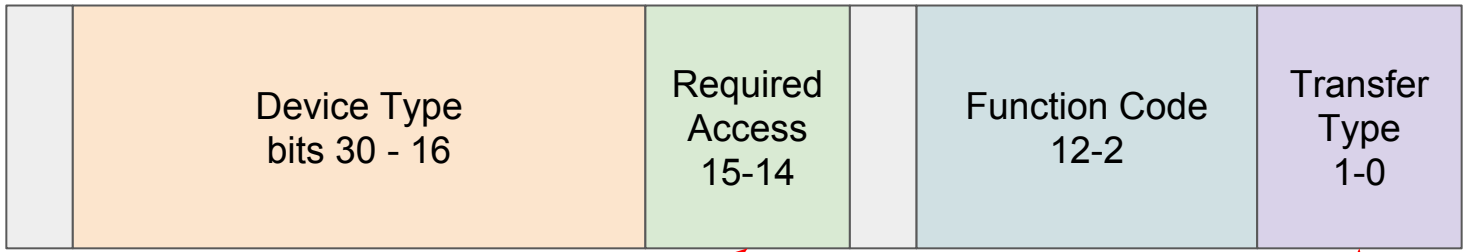
```
Device Characteristics:
SECURE_OPEN                    = 0x00100
ALLOW_APPCONTAINER_TRAVERSAL = 0x20000
```

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

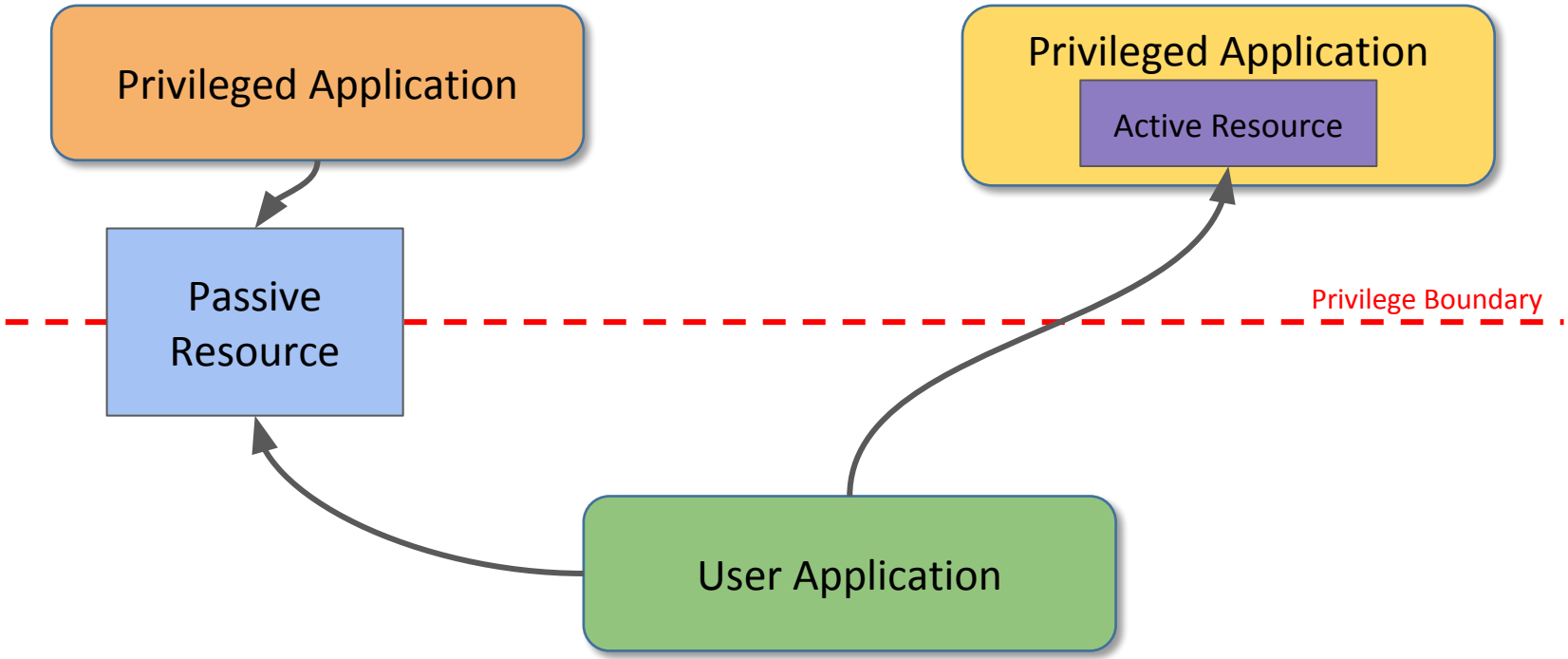
```
PS> $f.FsControl($code, [byte[]]@(1, 2, 3), 10)
```

WORKSHOP 8

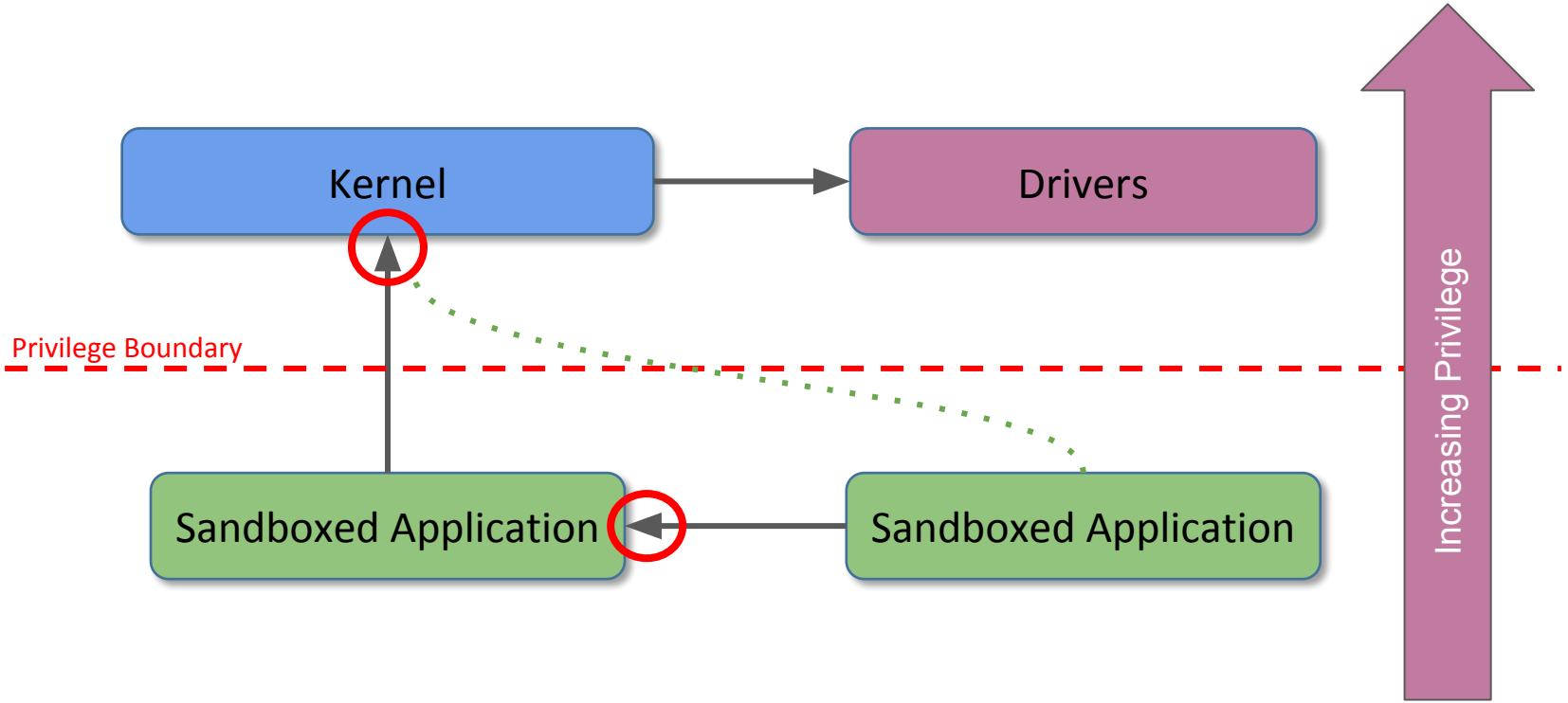
Analysing Windows IO

PART 2
Attack Surface Analysis

Passive vs Active Attack Surface



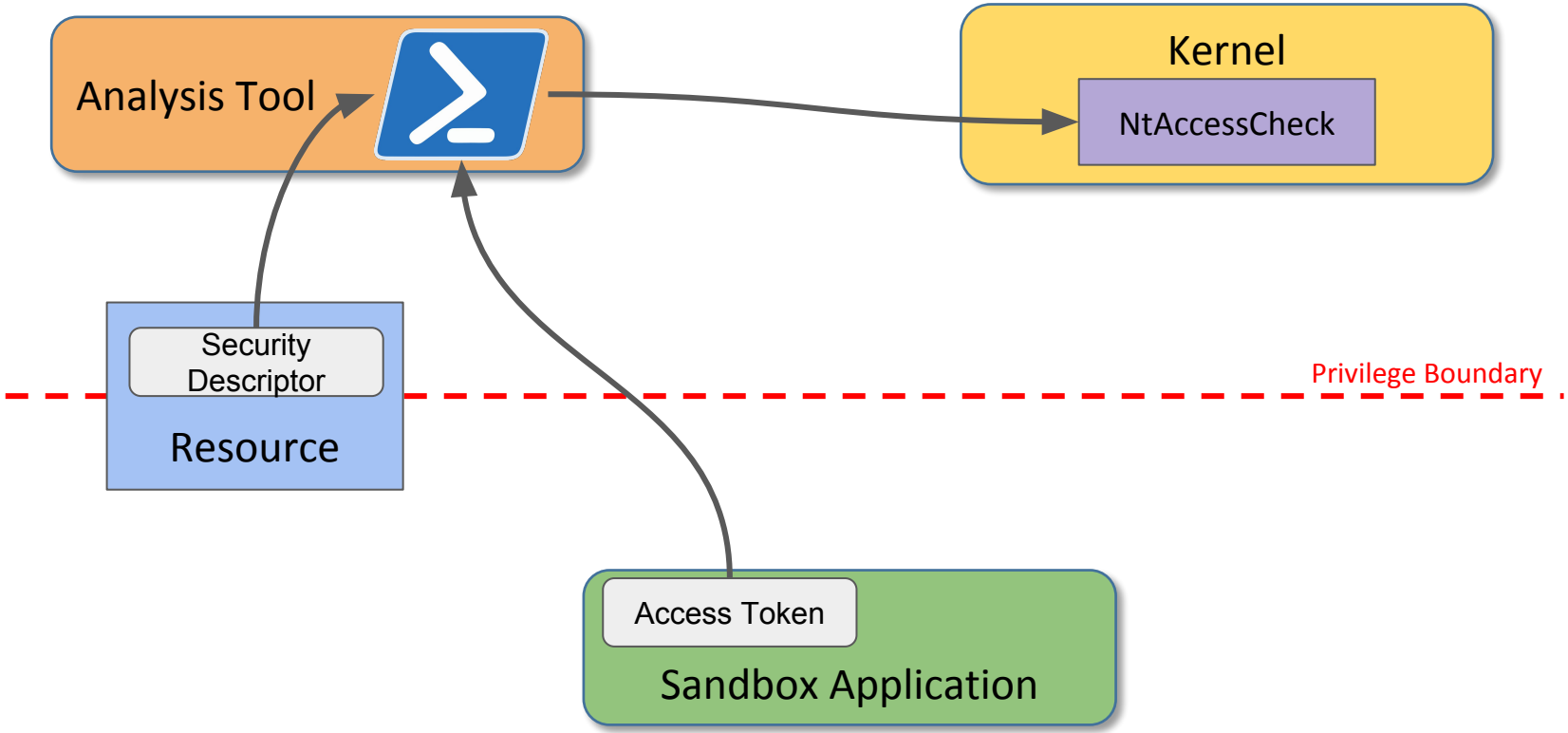
Don’t Always Think of Going Up



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

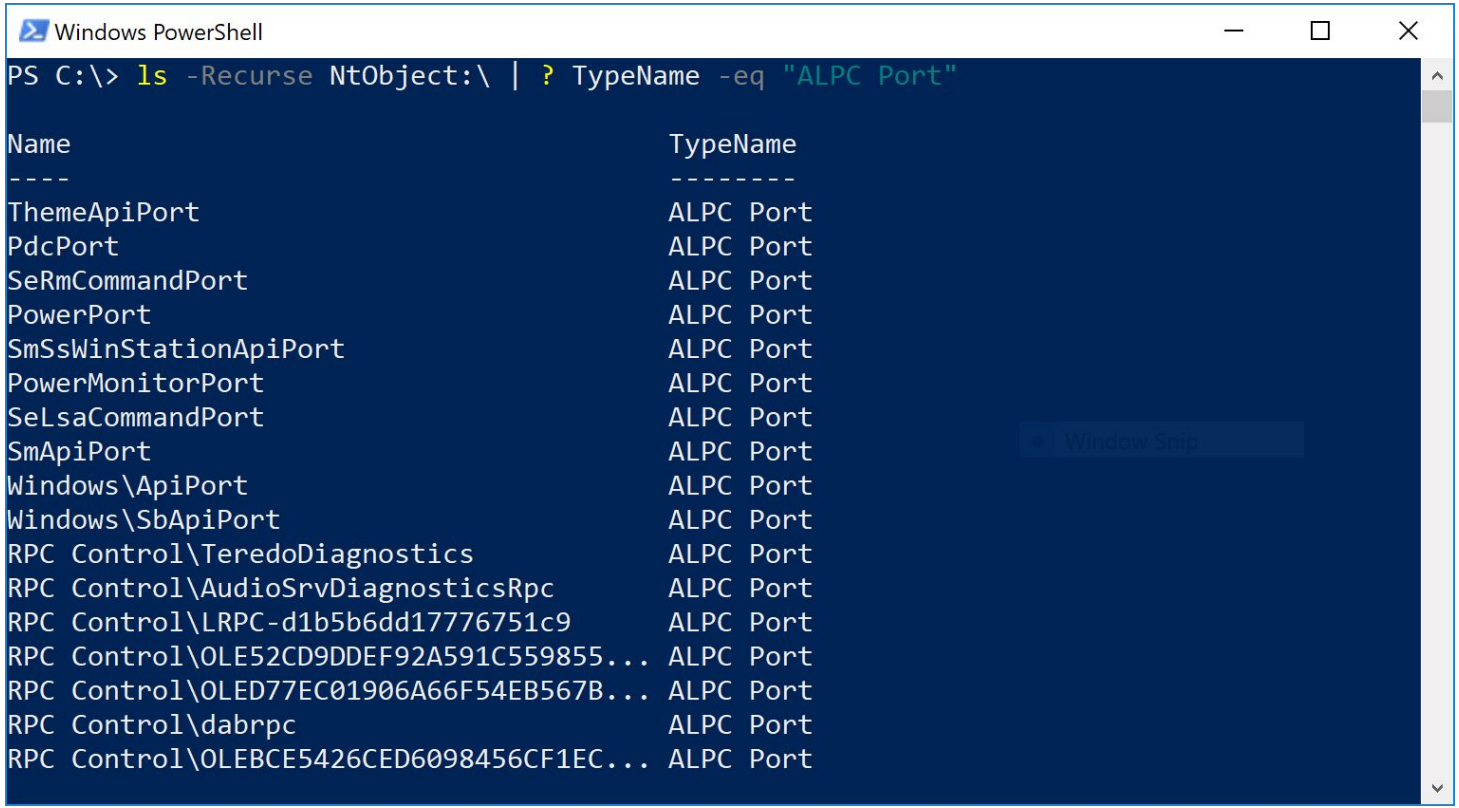
WORKSHOP 9

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


Windows PowerShell
PS C:\> ls -Recurse NtObject:\ | ? TypeName -eq "ALPC Port"
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\dabrpc 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,
  RPC_C_LISTEN_MAX_CALLS_DEFAULT,
  SecurityCallback
);
```

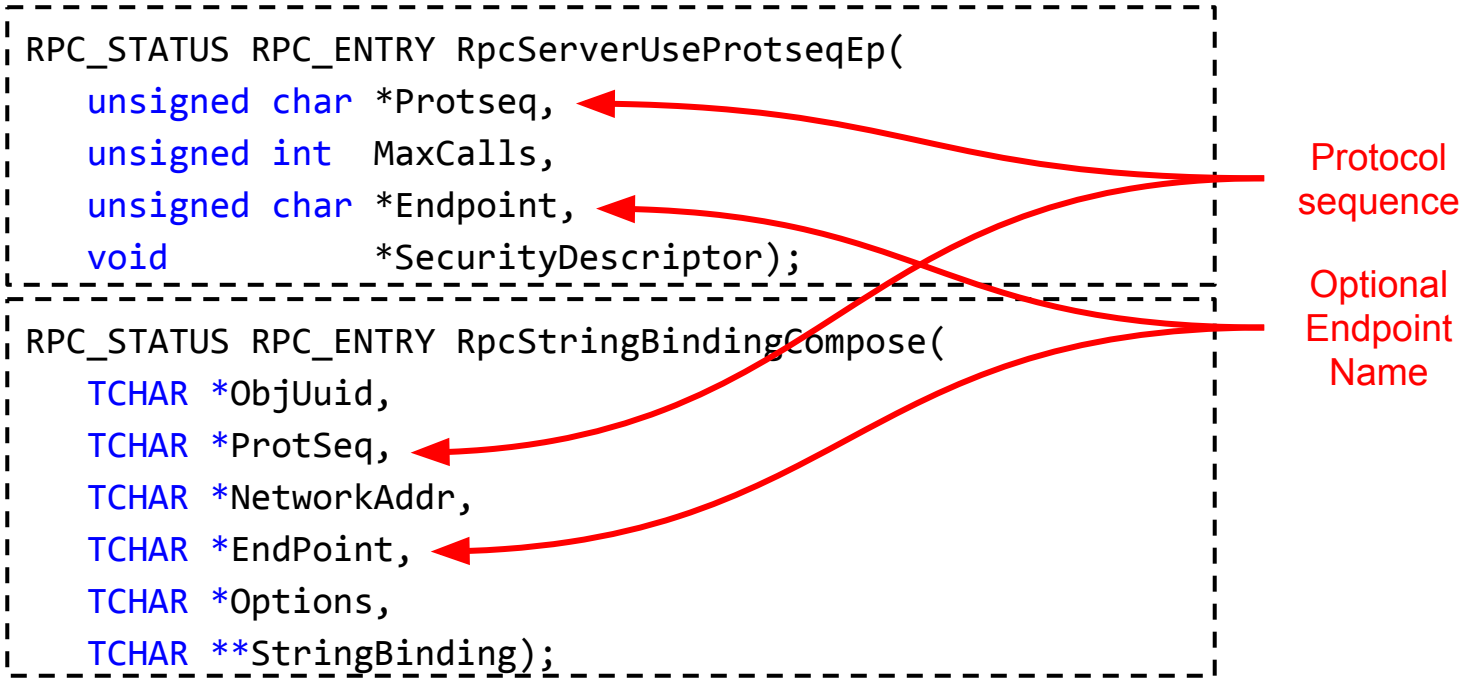
Server registration structure (generally auto-generated)

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

Optional security callback.

RPC Endpoints

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



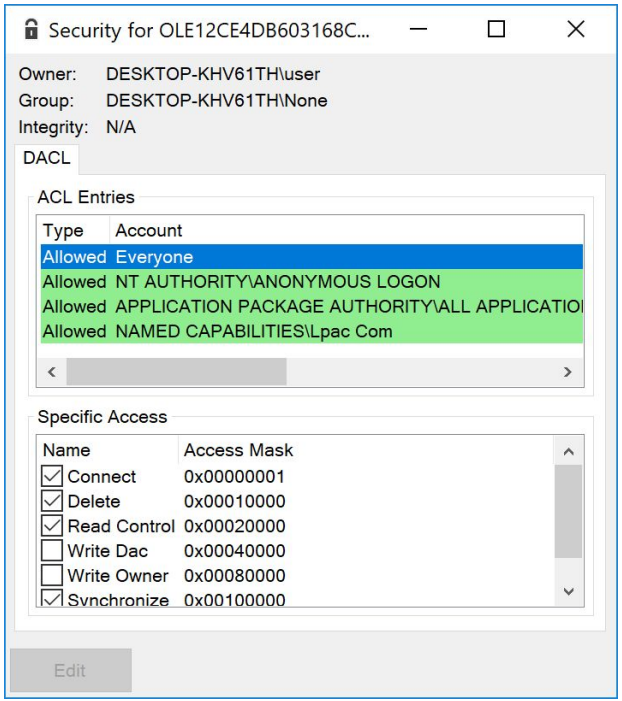
Protocol Sequences

Protocol Sequence	*Optional Endpoint Name*	*Description*
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)



Runtime Security

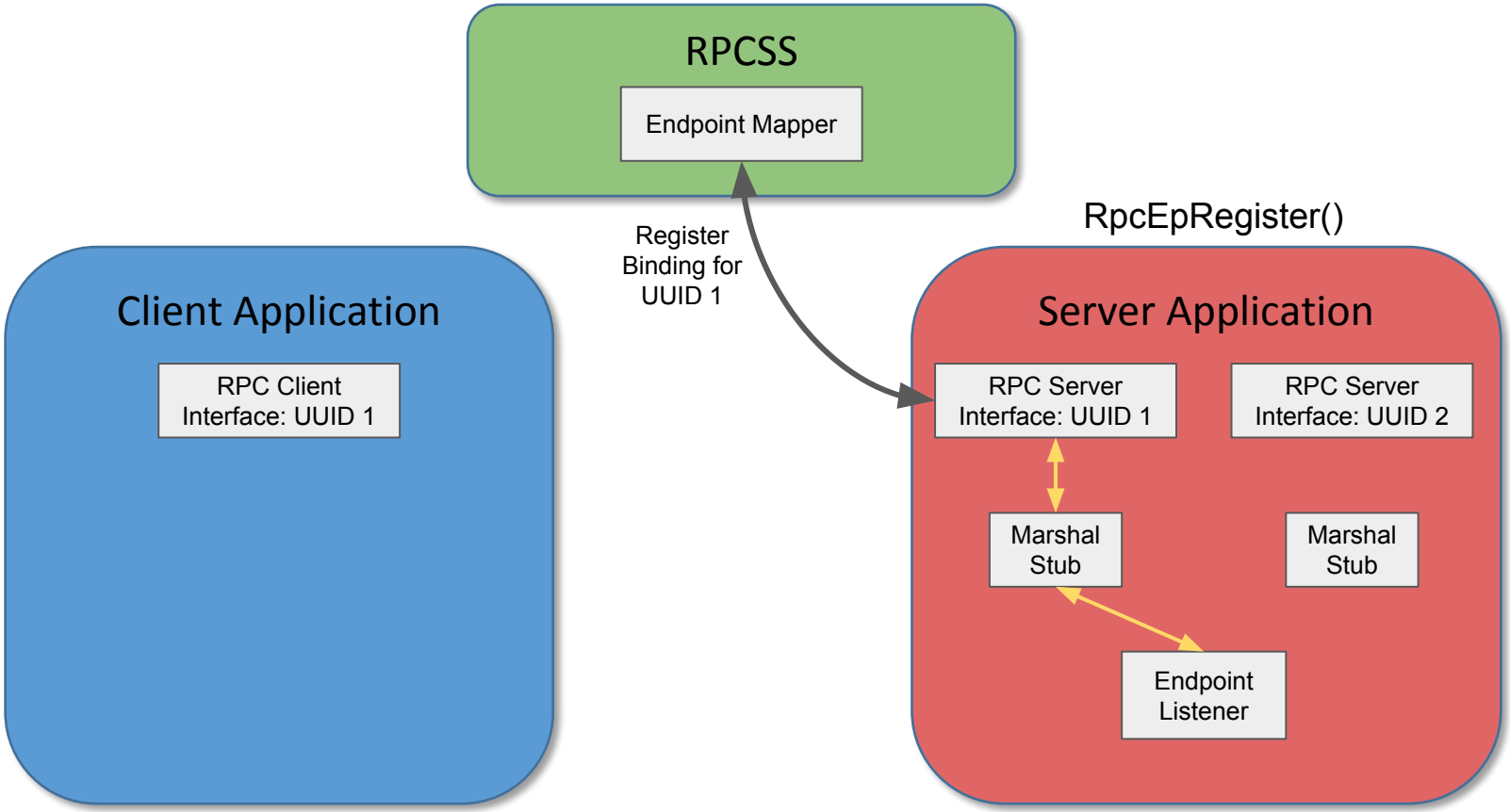
```
RPC_STATUS RPC_ENTRY RpcServerRegisterIf3(
  _In_     RPC_IF_HANDLE      IfSpec,
  _In_opt_ UUID               *MgrTypeUuid,
  _In_opt_ RPC_MGR_EPV        *MgrEpv,
  _In_     unsigned int       Flags,
  _In_     unsigned int       MaxCalls,
  _In_     unsigned int       MaxRpcSize,
  _In_opt_ RPC_IF_CALLBACK_FN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

Security callback, run code to verify the client

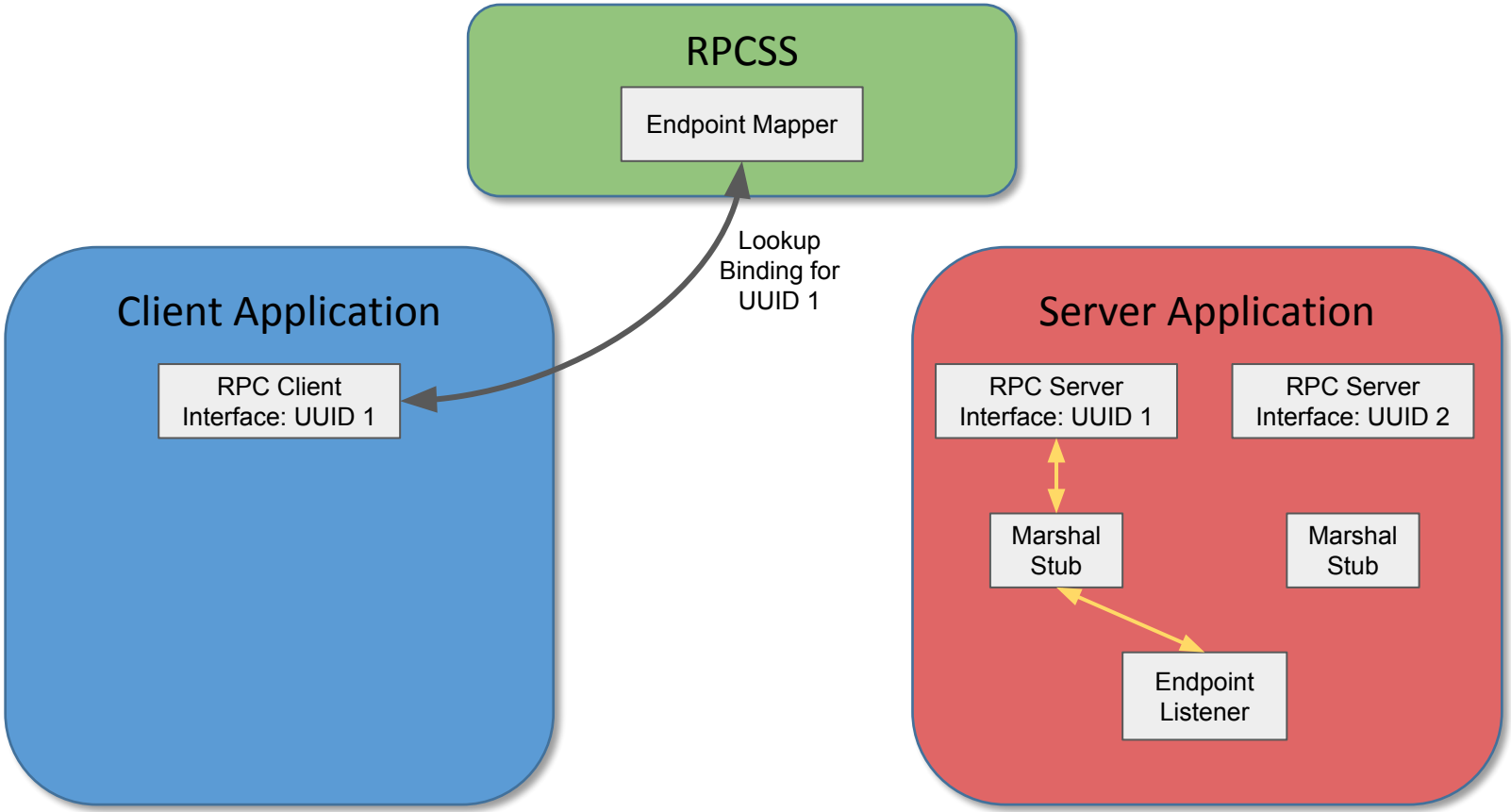
Static Security Descriptor

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

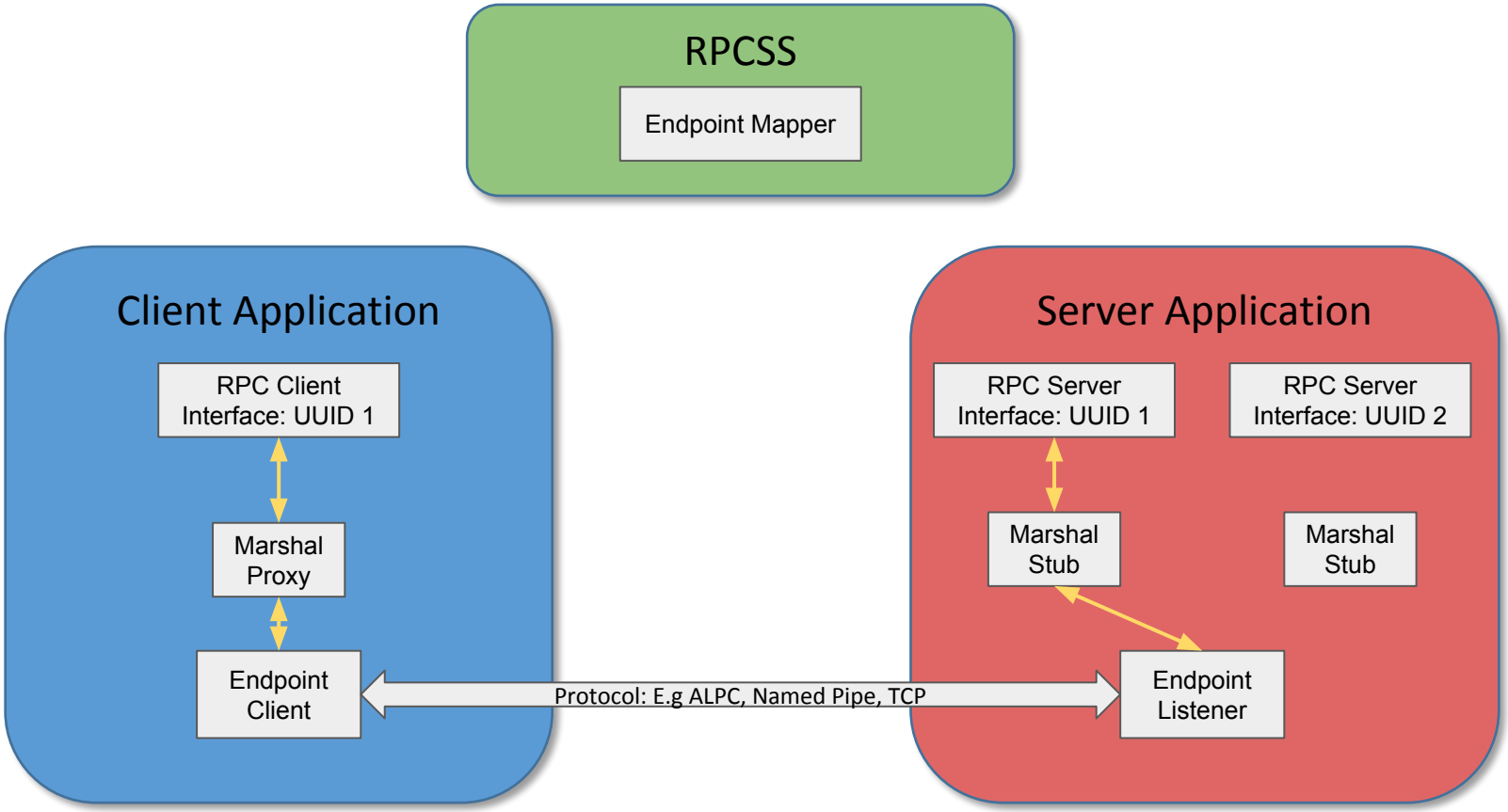
RPC Endpoint Mapping



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
```

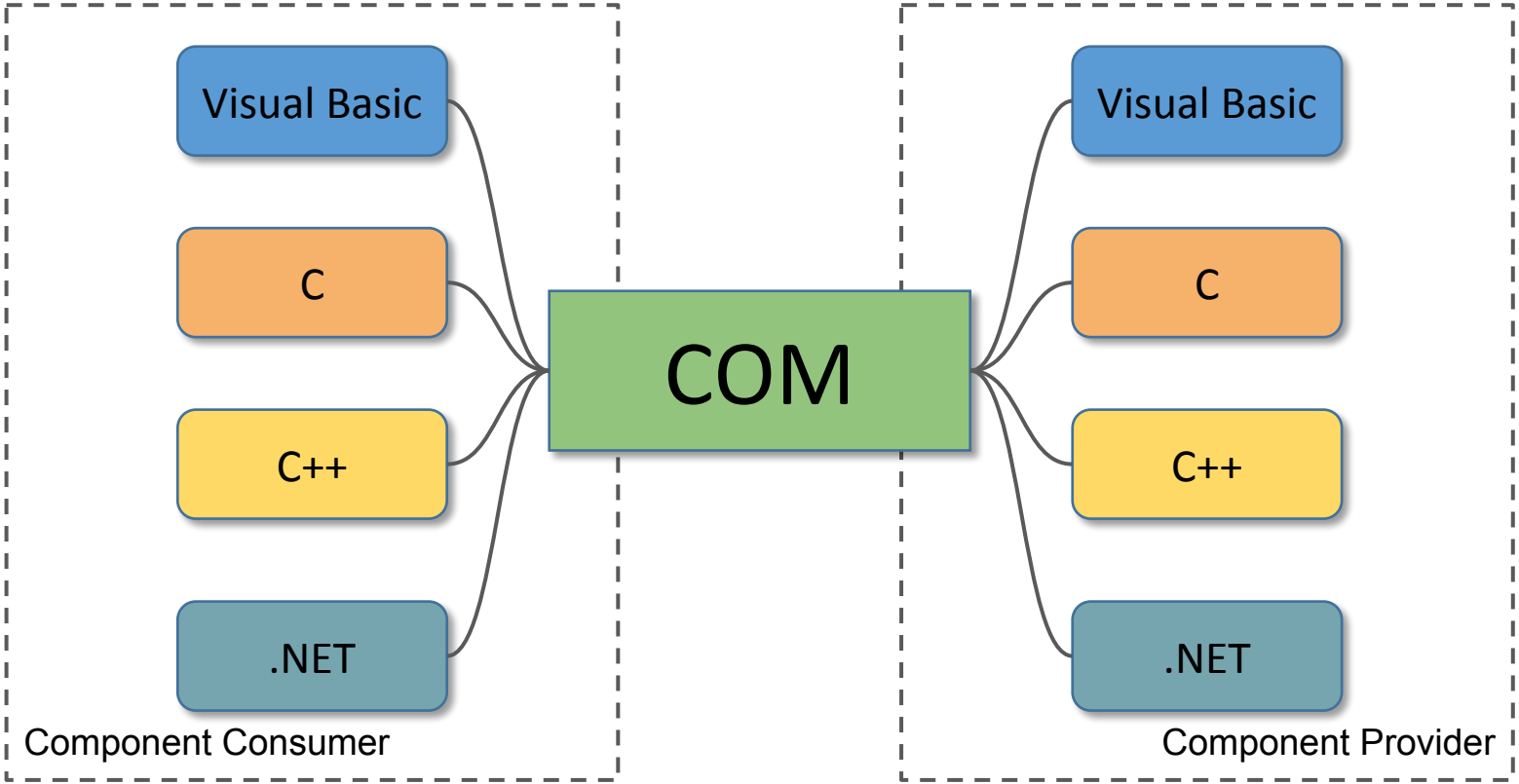
Format an RPC endpoint as text.

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

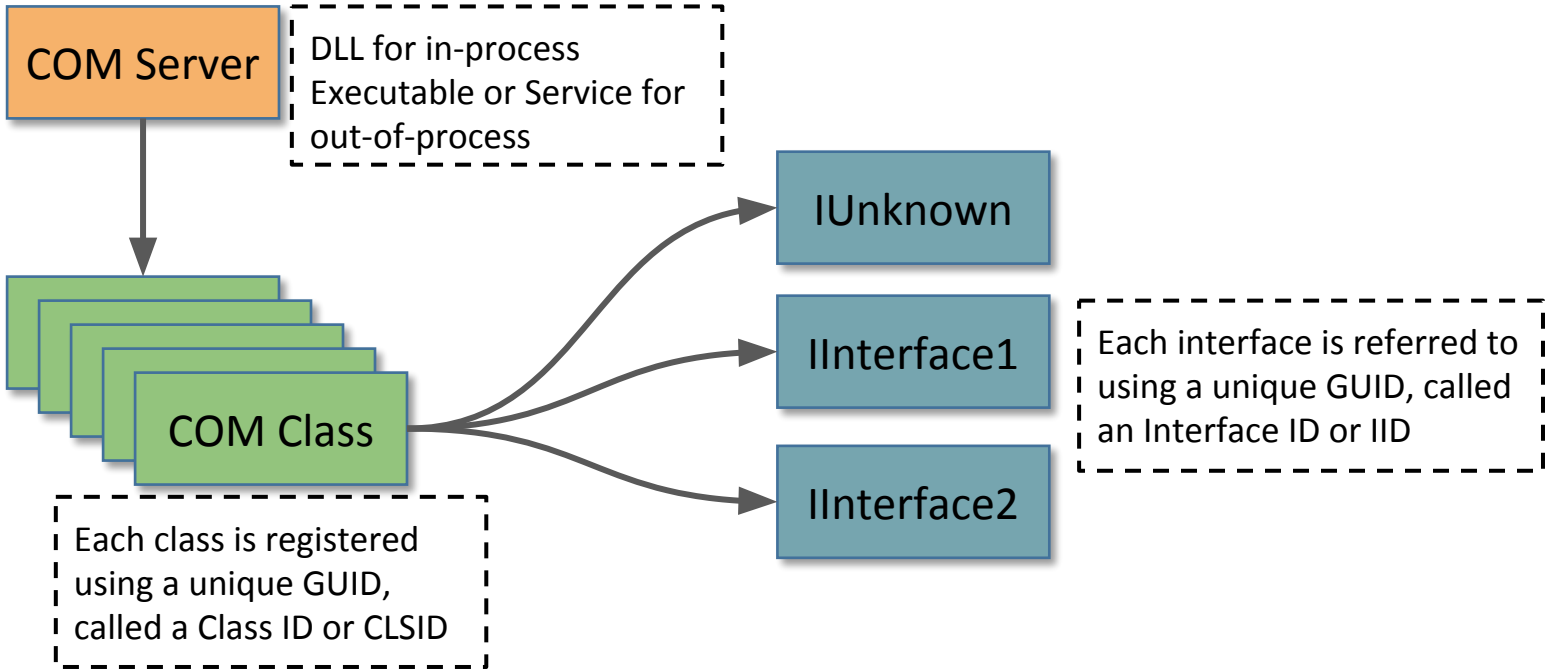
WORKSHOP 10

Finding and Analysing RPC Interfaces

COM is Middleware



COM Services



IUnknown, the Root of all COM Evil

```
DEFINE_GUID(IID_IUnknown,
     "00000000-0000-0000-C000-000000000046");
struct IUnknown {
  HRESULT QueryInterface(GUID& iid, void** ppv);
  LONG AddRef();
  LONG Release();
};
```

Standard COM error code. >= 0 is Success

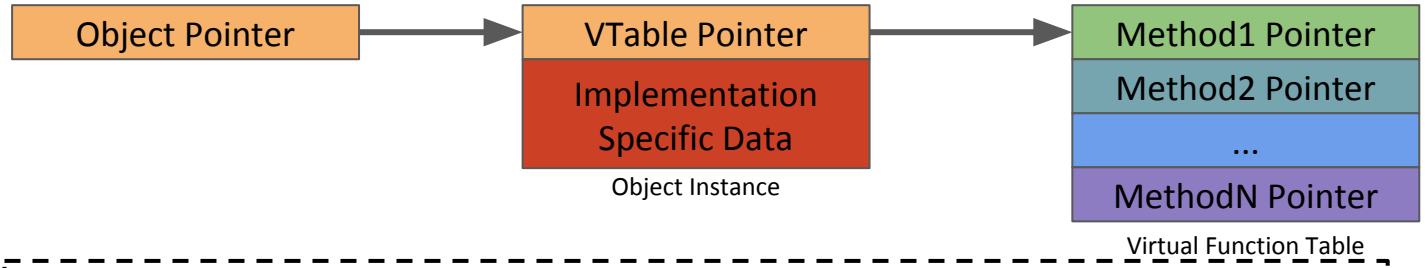
Used to reference count the object.

Interfaces defined using a 128 bit Globally Unique ID.

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

Cast is a GIANT code smell! ¯_(ツ)_/¯

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

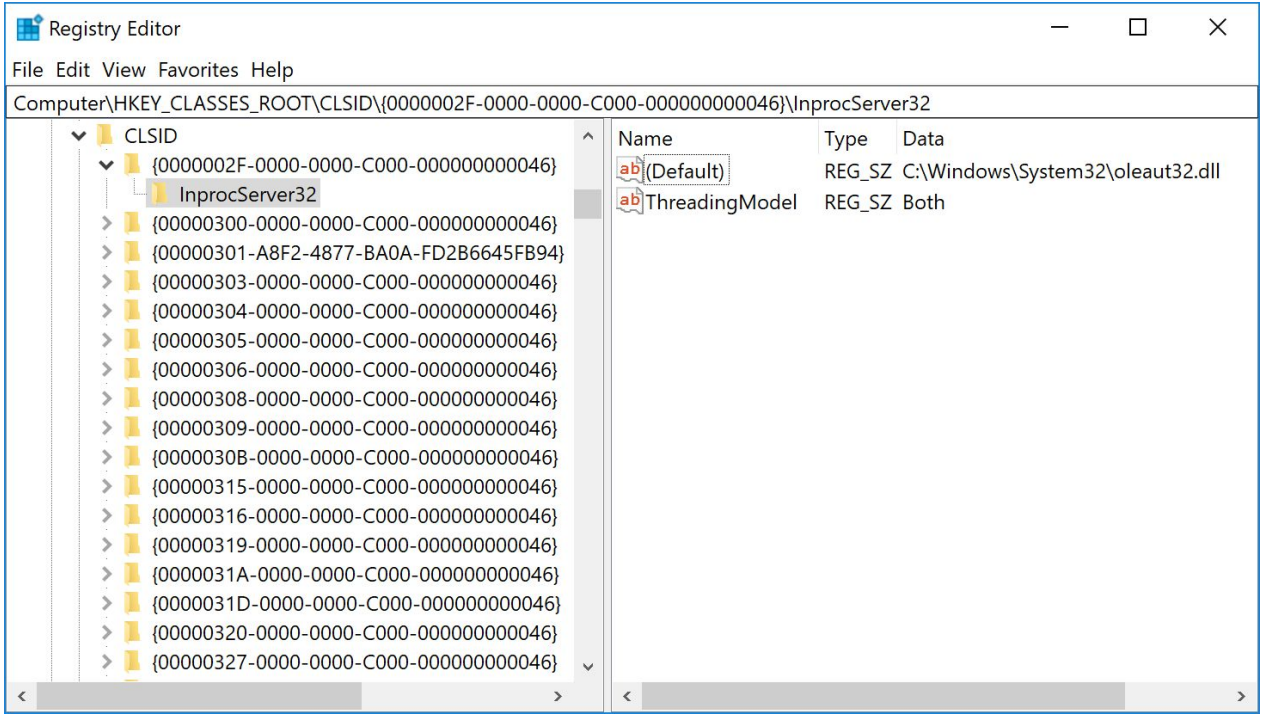
Class Factories

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

```
DEFINE_GUID(IID_ClassFactory,
     "00000001-0000-0000-C000-000000000046");

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

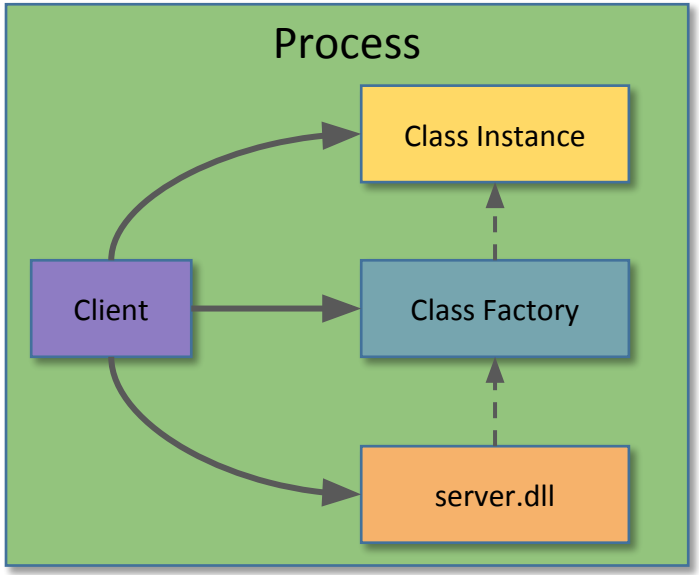
```
HRESULT CoGetClassObject(
  REFCLSID      rclsid,
  DWORD         dwClsContext,
  COSERVERINFO *pServerInfo,
  REFIID        riid,
  LPVOID        *ppv);
```

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);
```

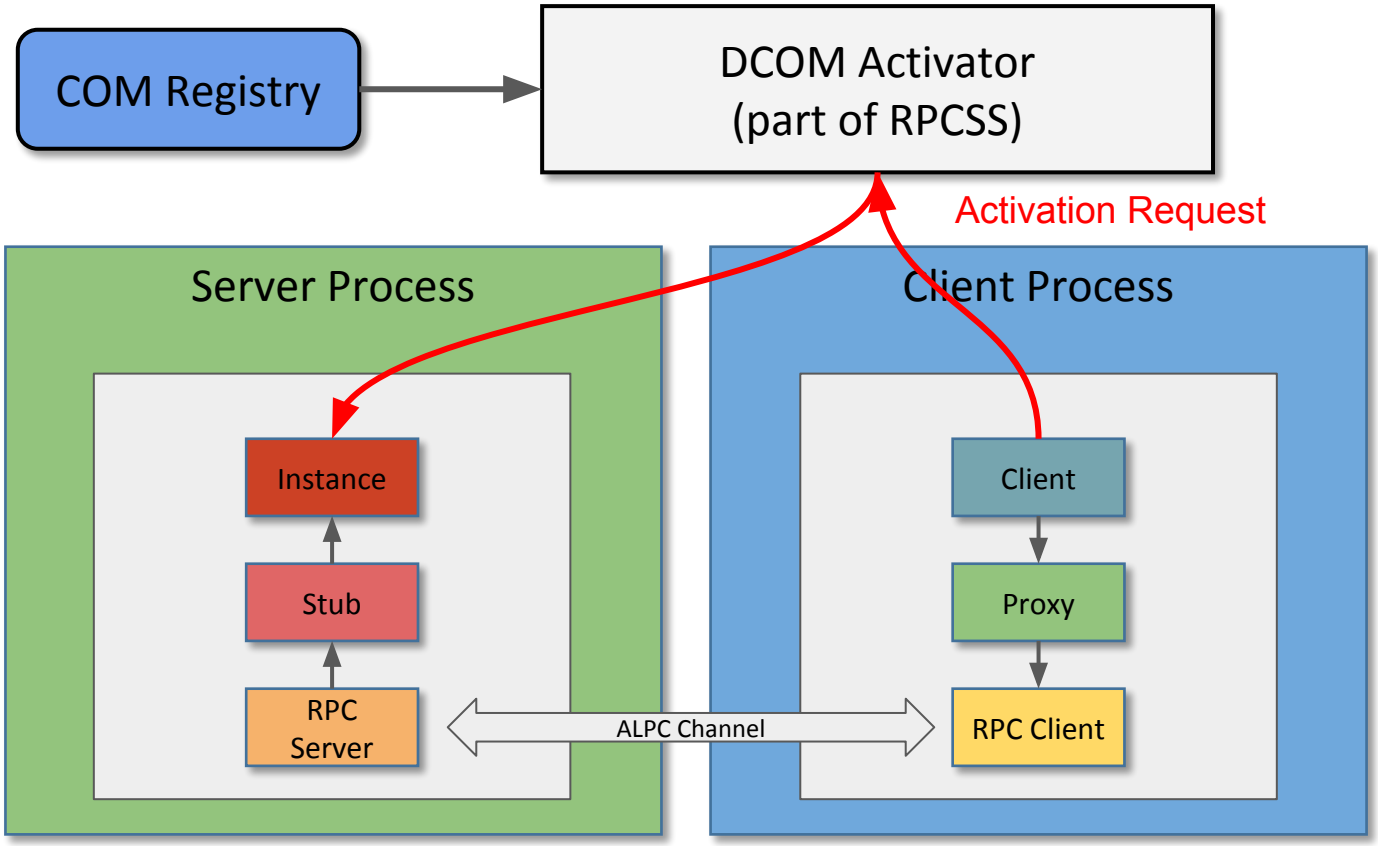
In-Process Server



- 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
```

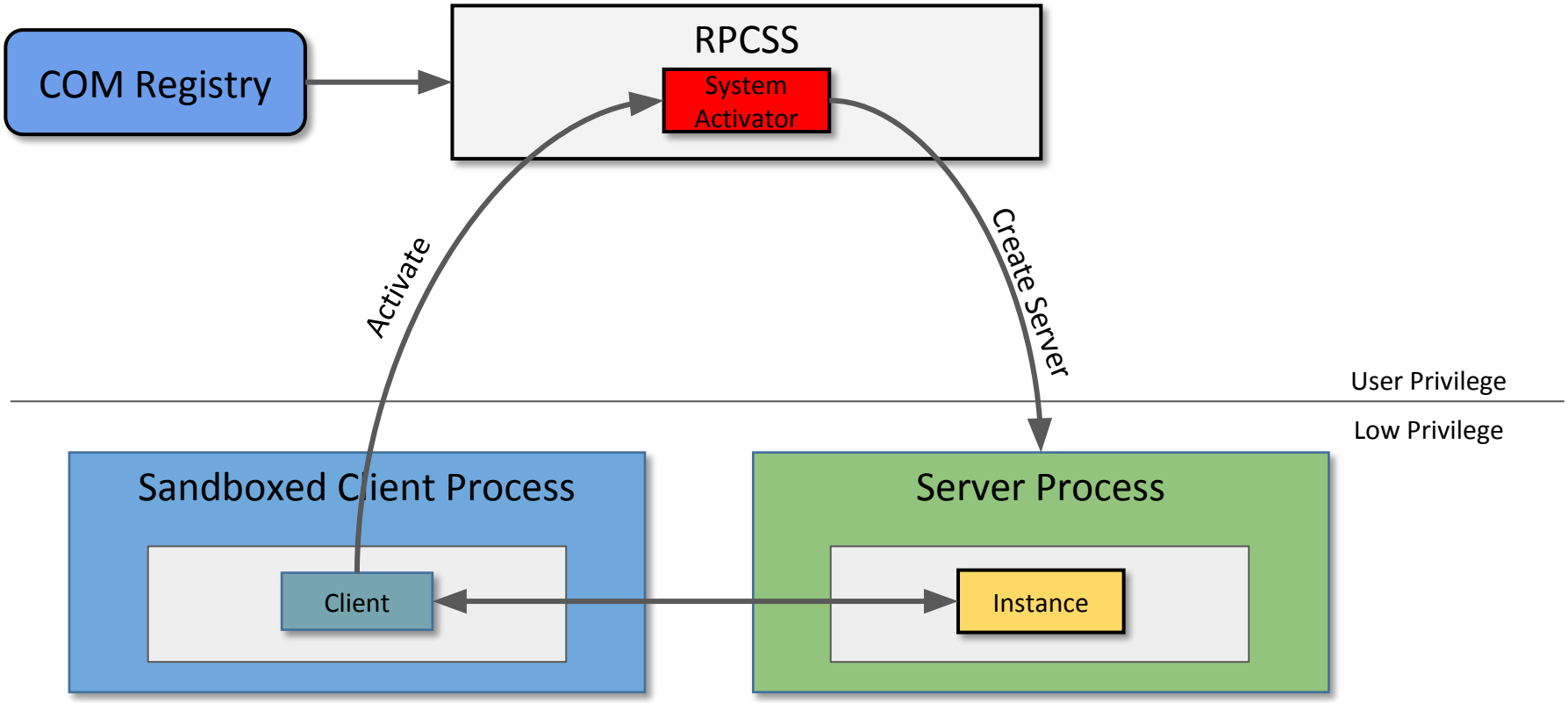
Get all COM classes which have the name Flash in their registration

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

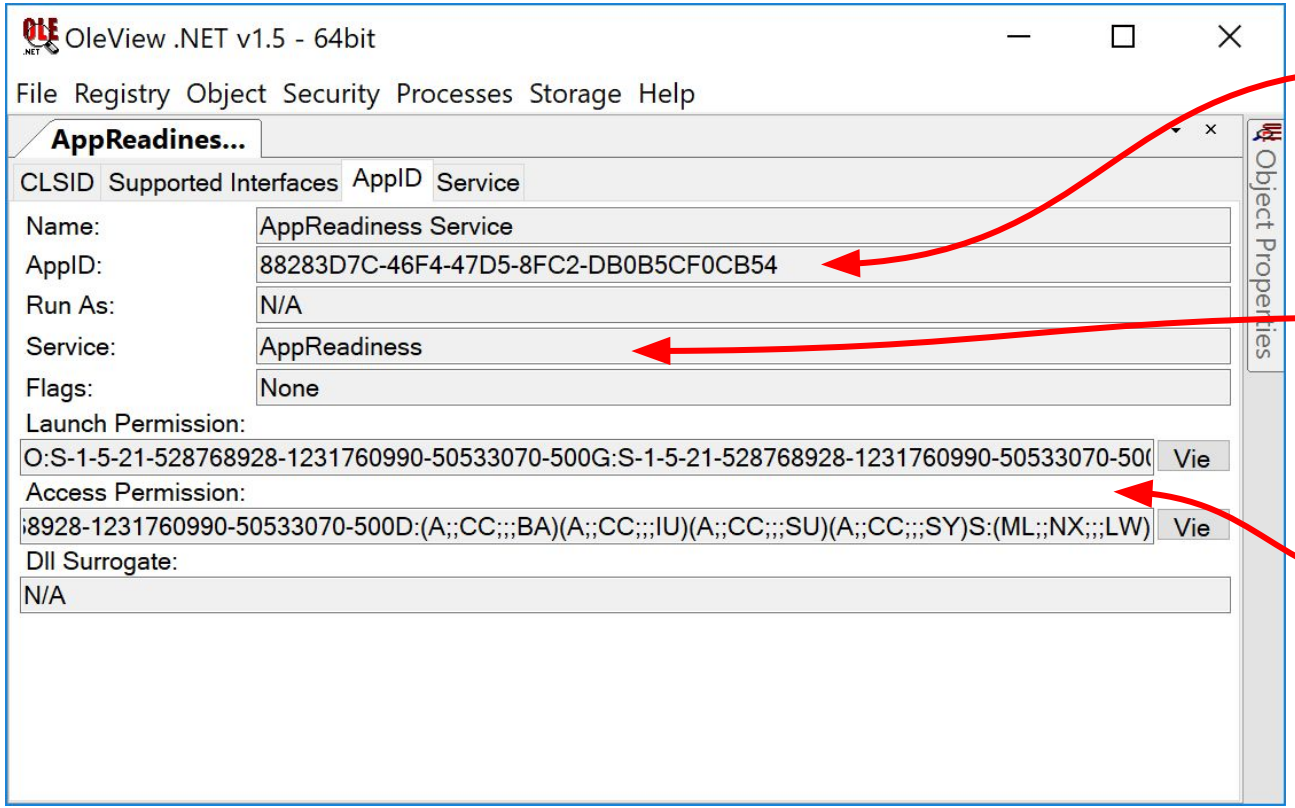
Enumerate all accessible interfaces for a class.

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

Local Server Activate as Activator (AAA)



Activation Via AppID

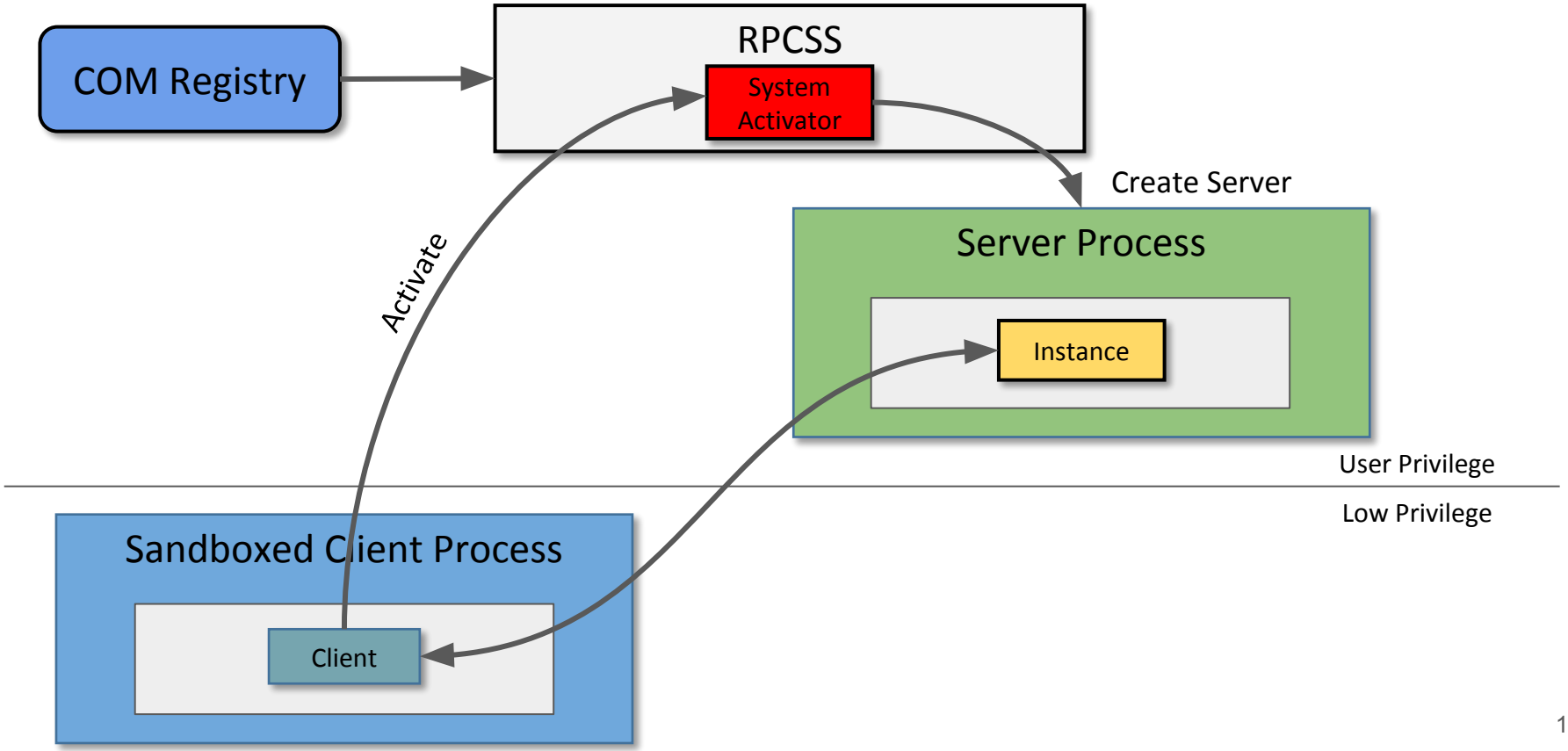


Classes are registered with the AppID GUID

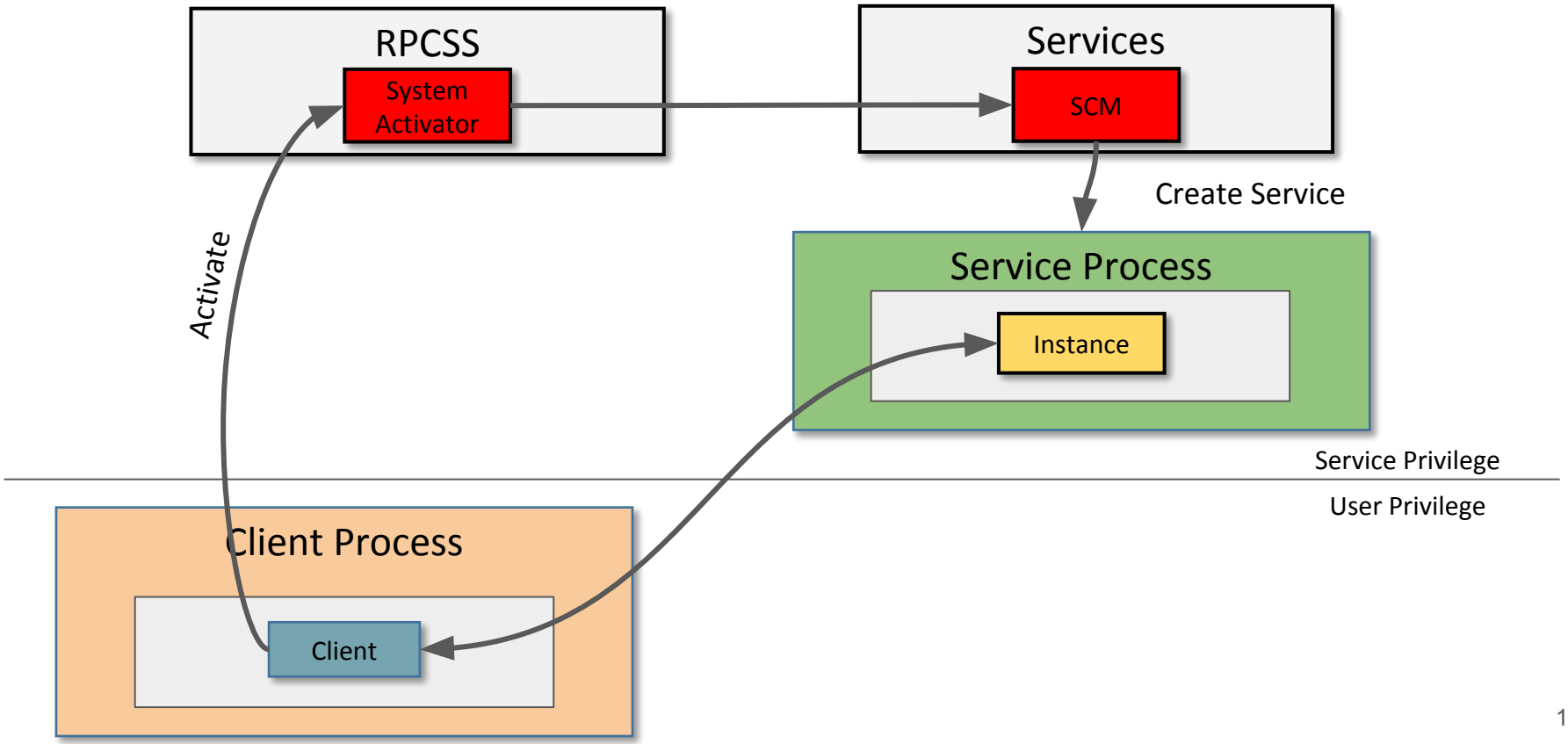
RunAs field or *Service name* start

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 a registered in the AxInstSv service.

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

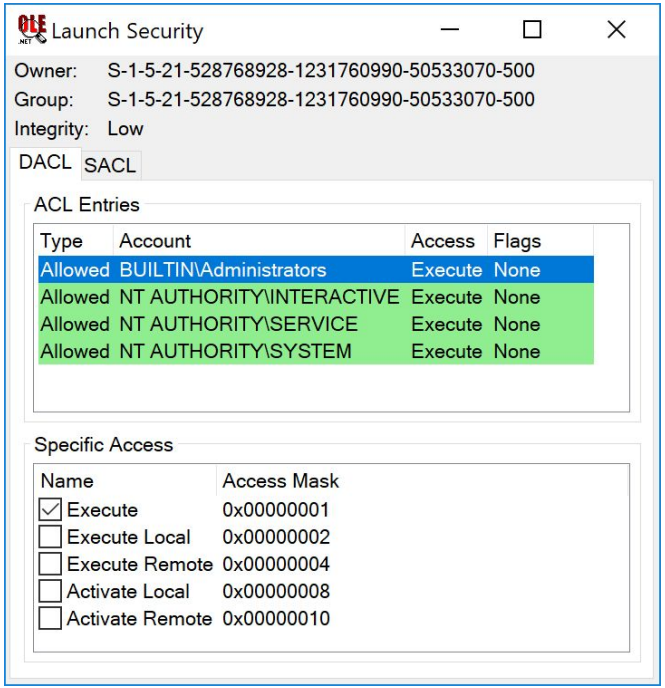
Get com classes which run in a service and group by the service name.

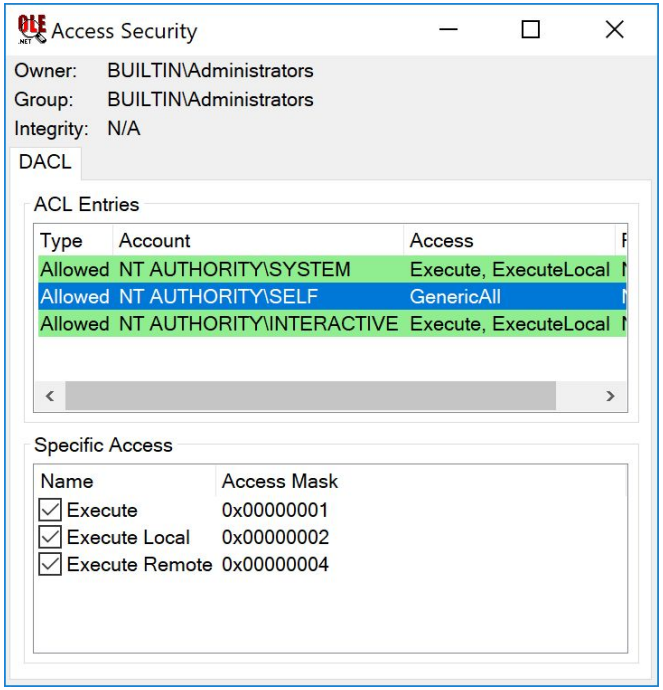
```
PS> Get-ComClass -Service | `
      group {$_.AppIDEntry.ServiceName}
```

Parse COM information from processes and return registered class information.

```
PS> Get-ComRegisteredClass
```

DCOM Security





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
```

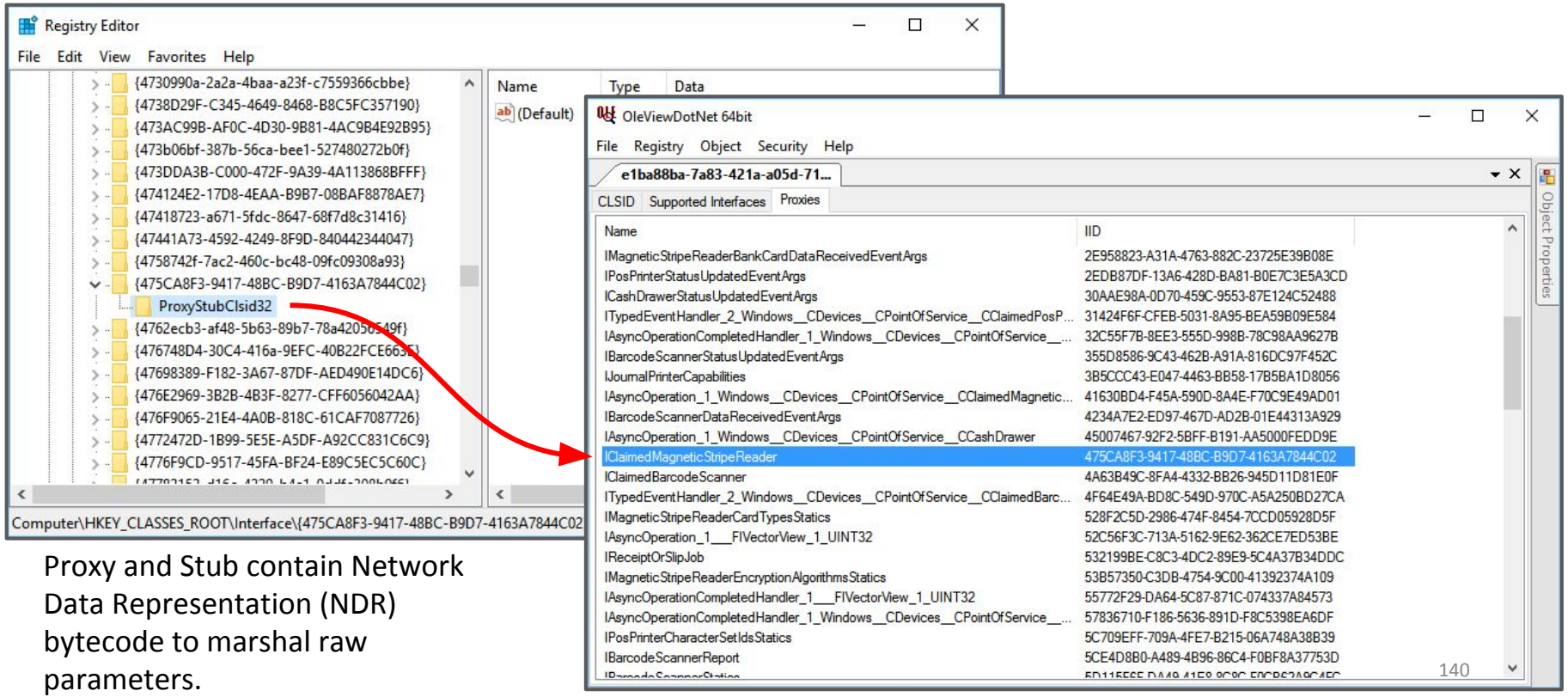
Get service hosted classes accessible by the current user.

```
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



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

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

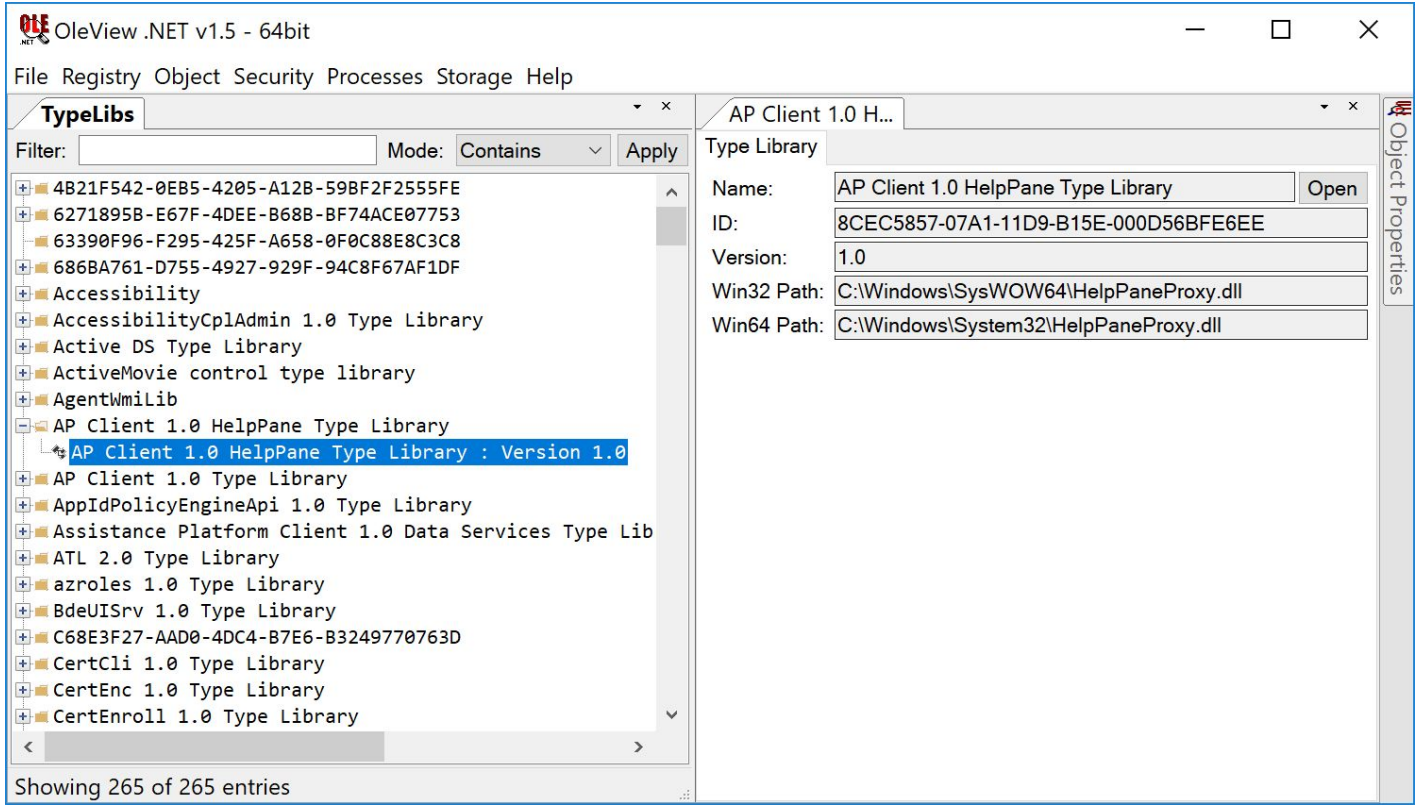
```
[uuid("00020400-0000-0000-c000-000000000046")]
class IDispatch : public IUnknown {
  HRESULT GetTypeInfo(unsigned int iTInfo,
                      ITypeInfo ** ppTInfo);

  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



Get all COM type libraries

```
PS> Get-ComTypeLib
```

Get the COM type library which implements an IID proxy.

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

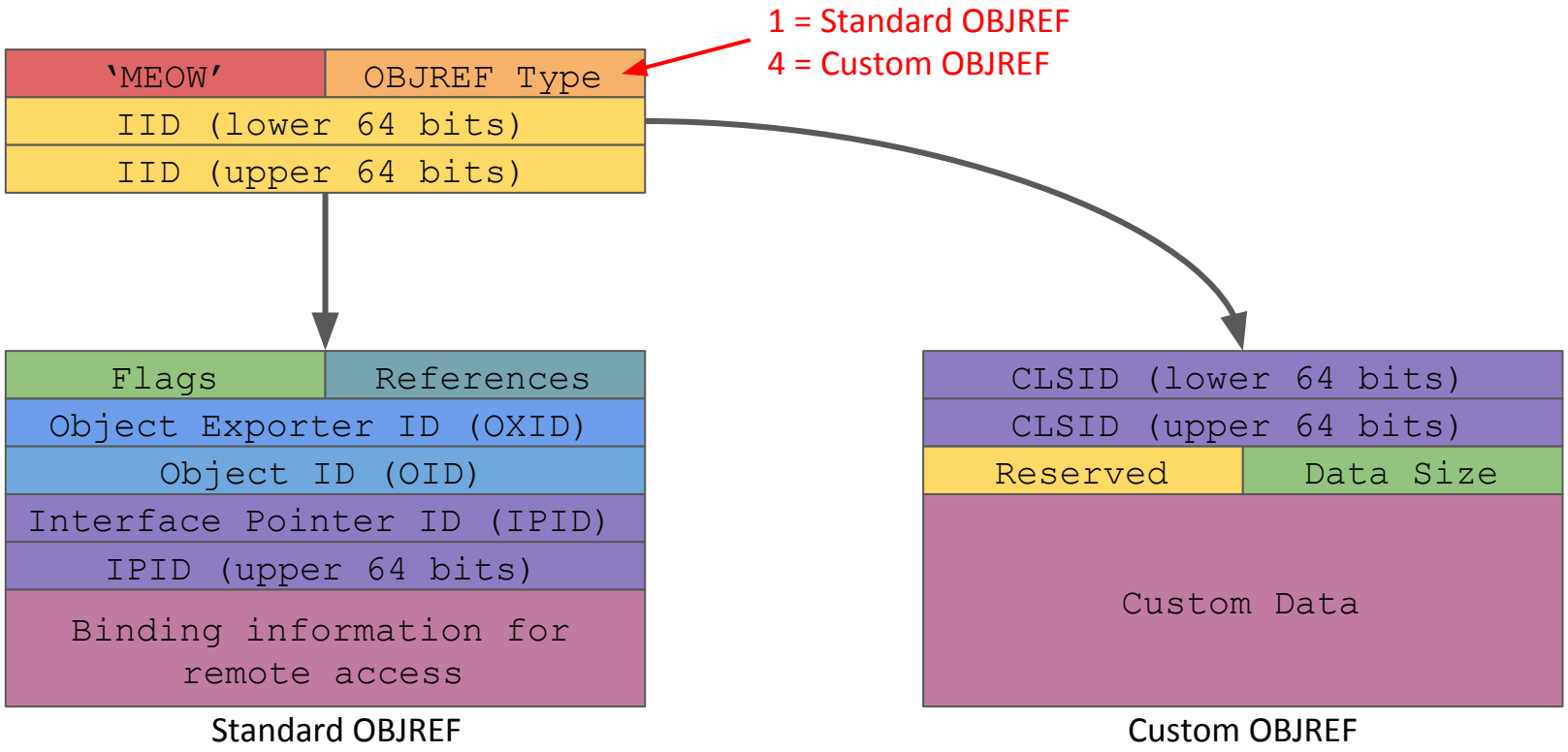
Convert a COM type library to a .NET assembly.

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

Format a COM type library as text.

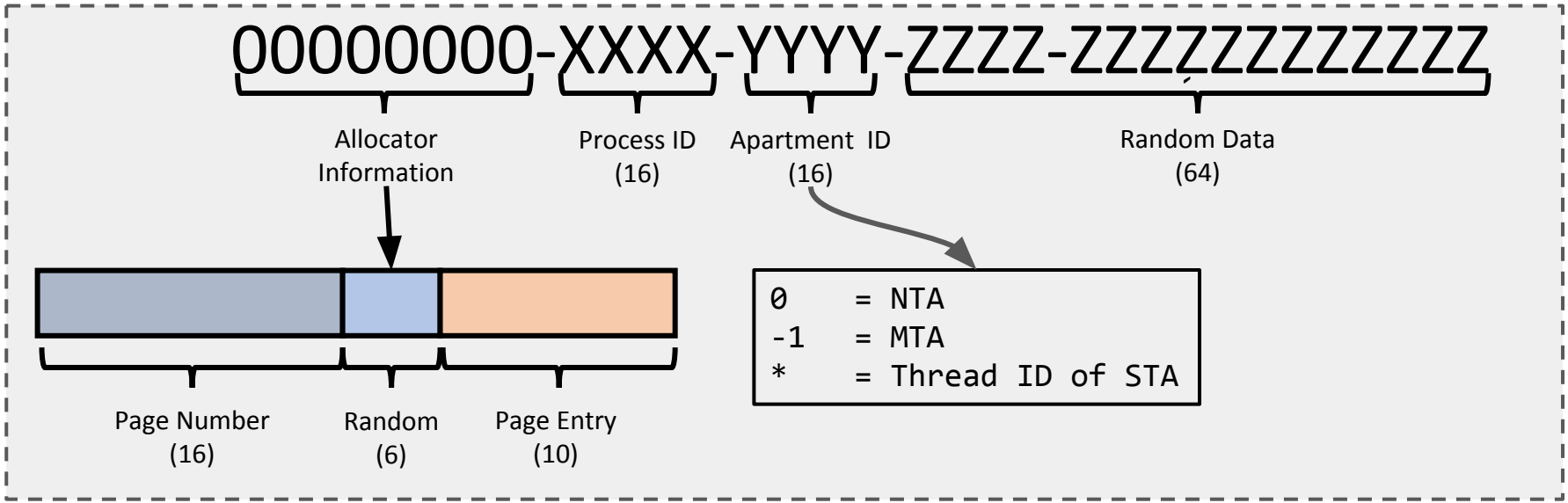
```
PS> Format-ComTypeLib -TypeLib $tlb
```

COM Marshaling

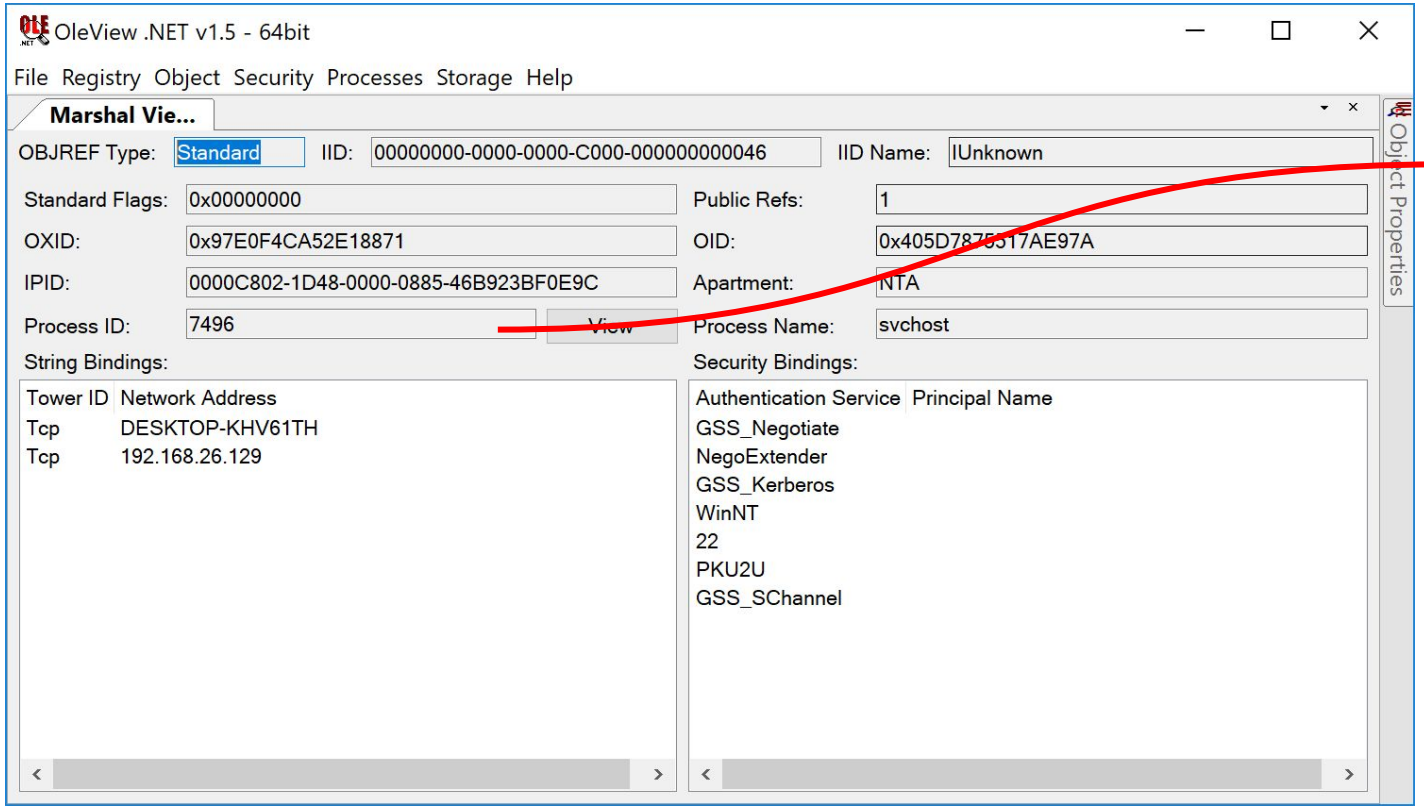


IPID Structure

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



Unpacking the Marshal OBJREF



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-AccessibleService -AccessRights Start
```

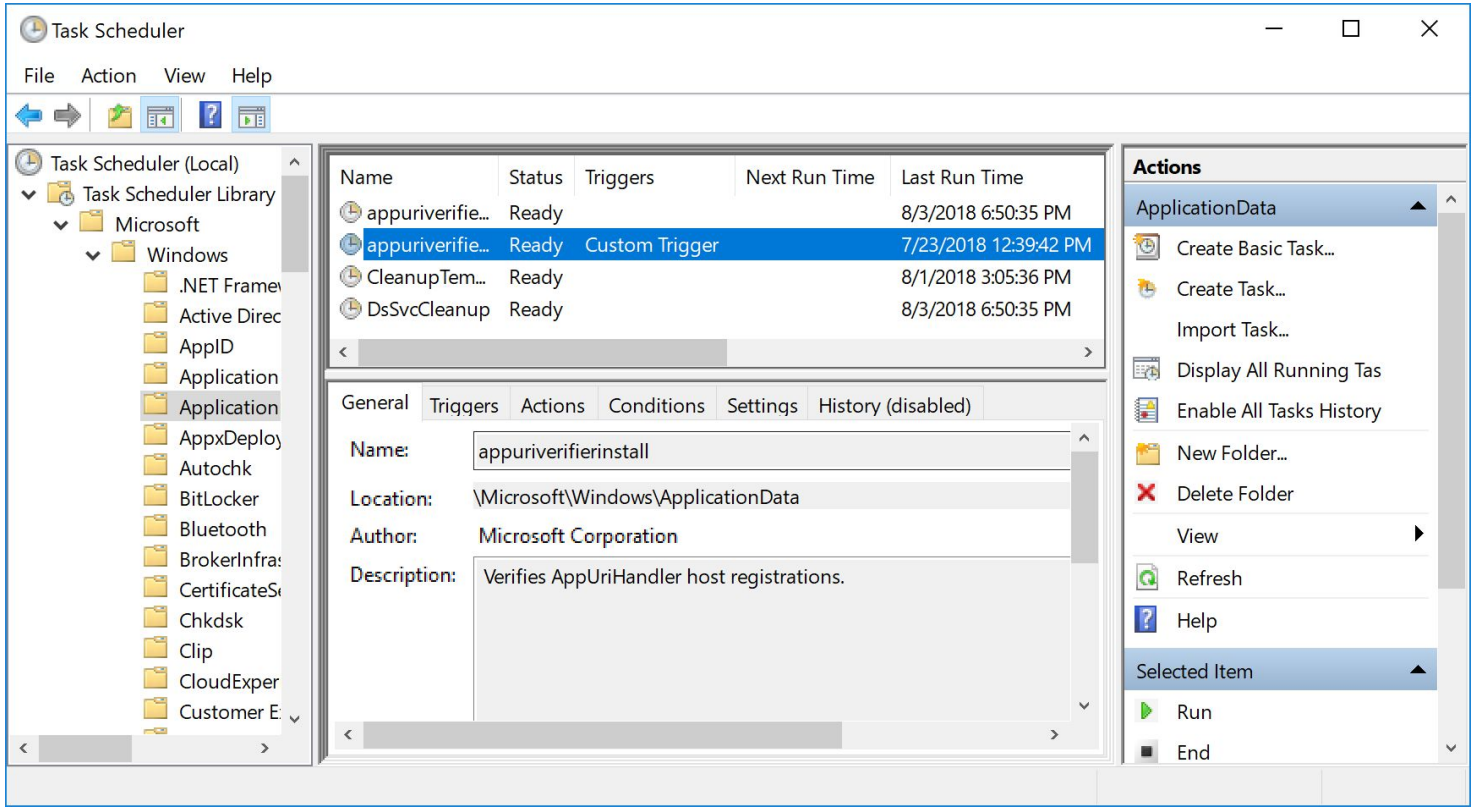
Get all services which have triggers set.

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

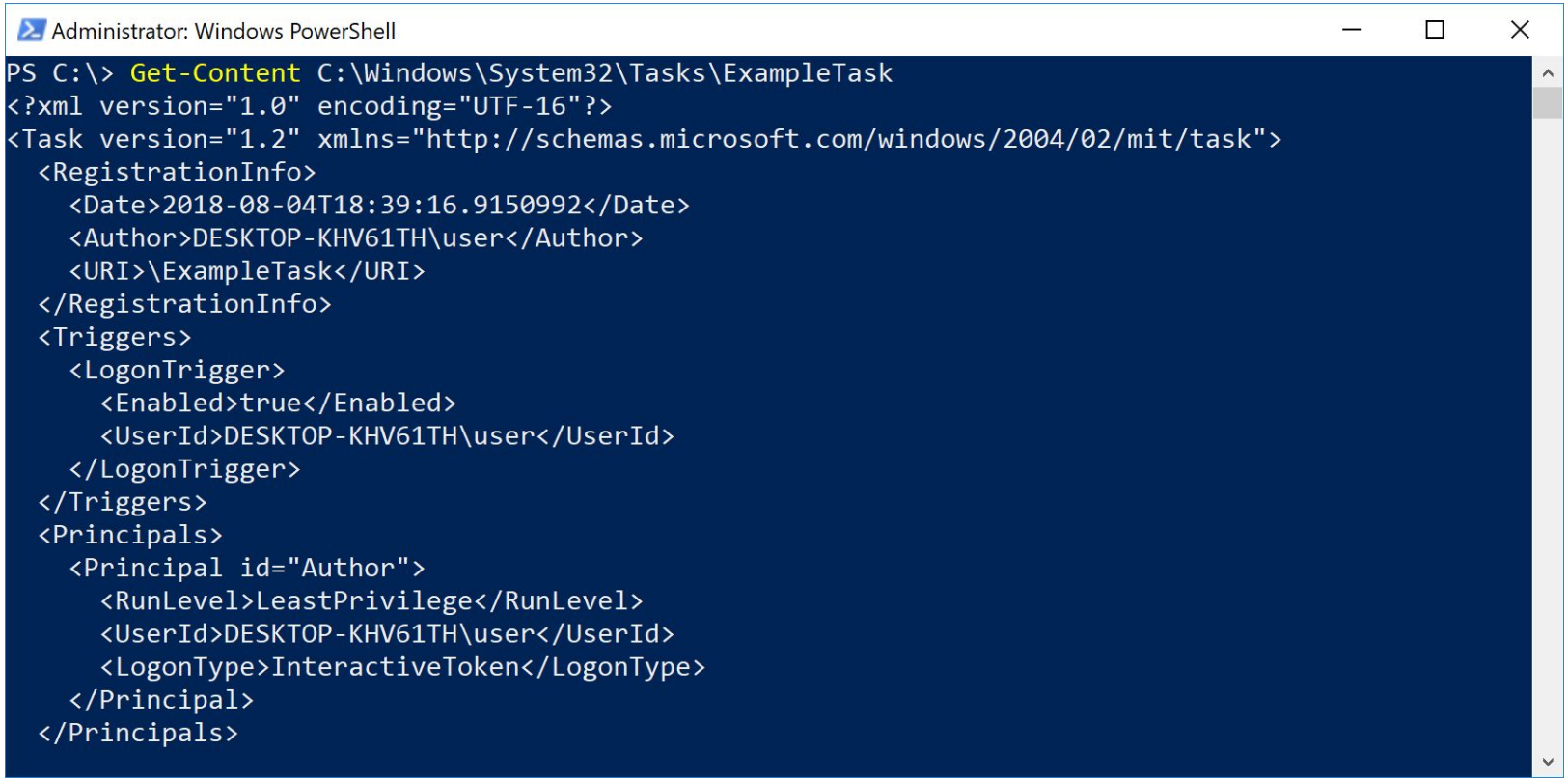
Start a service by name.

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

Scheduled Tasks



Task Files


Administrator: Windows PowerShell
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 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