

A vibrant night photograph of a city skyline, likely Hong Kong, featuring numerous skyscrapers with illuminated windows. In the foreground, a multi-lane highway curves through the city, with the motion of vehicles captured as long, glowing streaks of light. A bridge or overpass is visible on the left, and a modern building with a curved facade is on the right. The overall atmosphere is dynamic and urban.

Demystifying .COM

James Forshaw (@tiraniddo)
Troopers 2017

Agenda

- Component Object Model (COM) Internals
- Attacking COM
 - Enumerating attack surface for EoP
 - Reverse engineering COM components
- Bugs and “Features”

We'll be using my OleViewDotNet tool throughout.

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

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

<https://goo.gl/Bkhlos>

A classic yellow Ford Mustang is shown from a front-three-quarter angle, parked on a lawn. The hood is propped open, exposing the engine and various mechanical components. The car's iconic front grille and headlights are visible. The background features a garden with green plants and small pink flowers.

com Internals

COM is Scary!

**“Any sufficiently
complex middleware is
indistinguishable from
magic.”**

Arthur C. Clarke’s Third Law of Software Development

Mostly Documented'ish

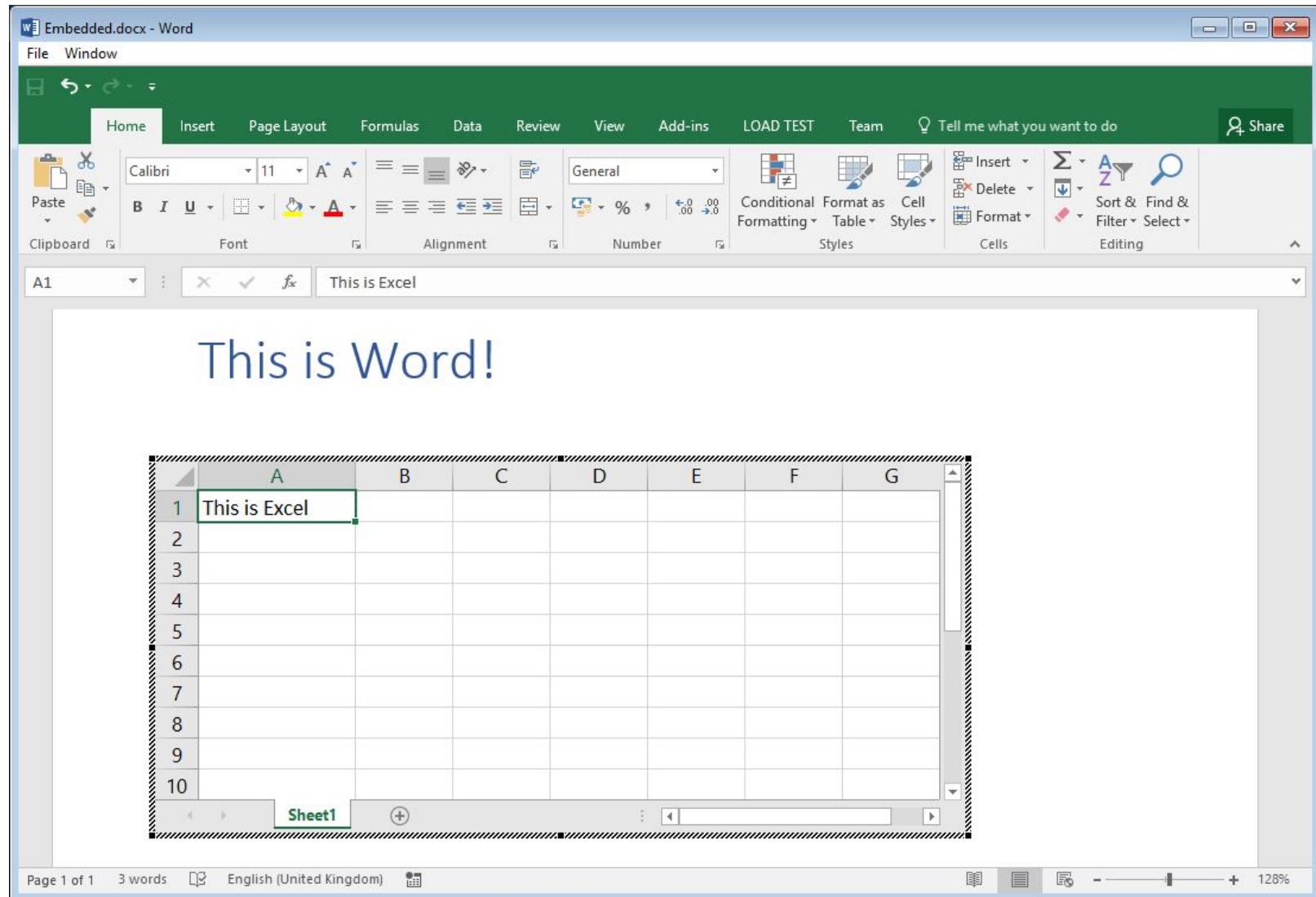
[MS-DCOM]:

Distributed Component Object Model (DCOM) Remote Protocol

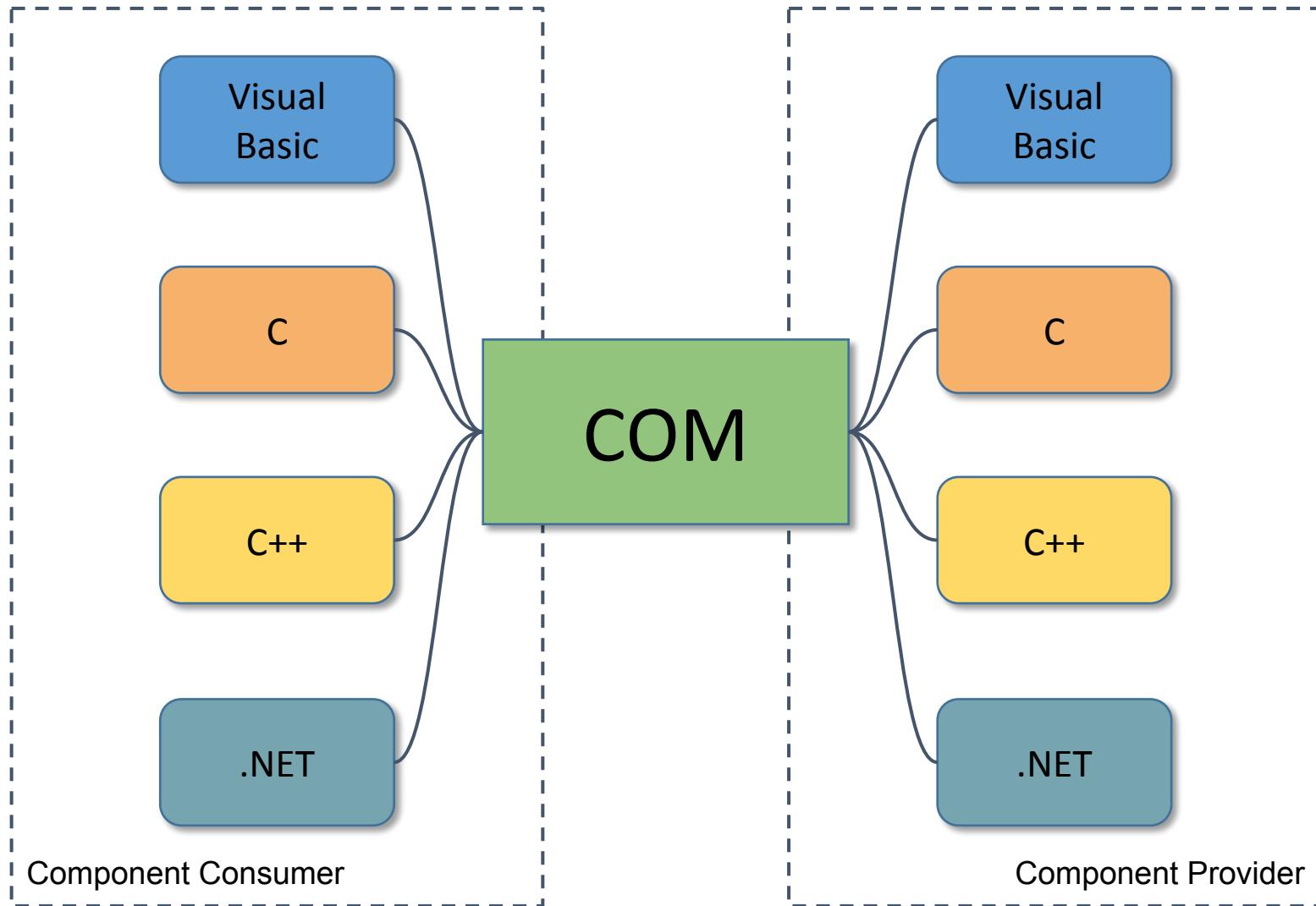
Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.

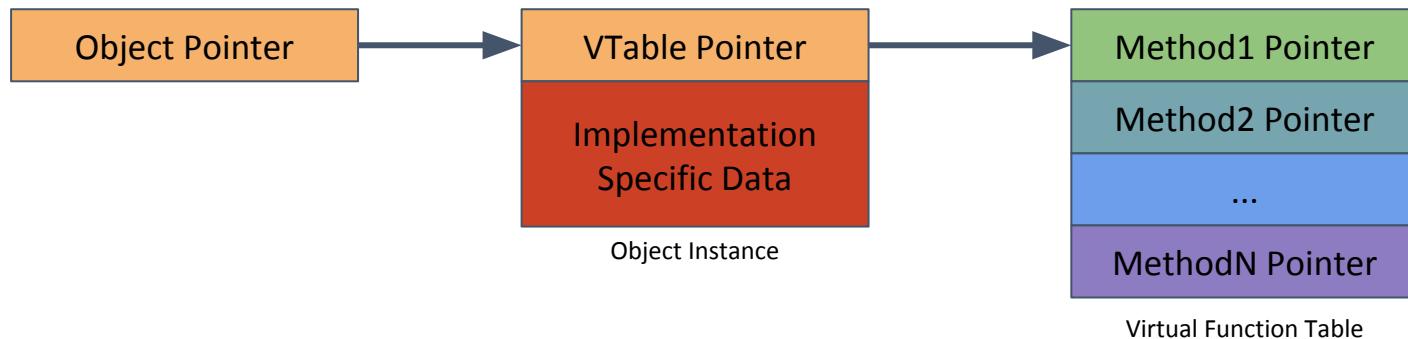
In the Beginning was OLE



Interoperability Heaven



Common ABI



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


```

Object pointer is first.
Arguments passed left to right

```
struct Object {
    struct ObjectVTable* Vtbl; // VTable Pointer at start
    // Implementation specific data follows.
};


```

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

C++ Implementation

```
struct Interface {  
    virtual void SetInt(int i) = 0;  
    virtual int GetInt() = 0;  
};  
  
class Object : public Interface {  
public:  
    void SetInt(int i) {} ← Define a pure  
    int GetInt() {} ← Derive from  
};  
  
Interface* obj = new Object;  
obj->SetInt(1234);
```

The Casting Problem

```
struct Interface1 {
    virtual void A() = 0;
};

struct Interface2 {
    virtual void B() = 0;
};

class Object : public Interface1,
               public Interface2 {
    void A() {}
    void B() {}
}
```

```
// Okay (mostly).
Interface1* intf1 = new Object;
// Incredibly bad idea.
Interface2* intf2 = (Interface2*)intf1;
```

IUnknown, the Root of all COM Evil

```
DEFINE_GUID(IID_IUnknown,
    "00000000-0000-0000-C000-00000000046") ;

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.

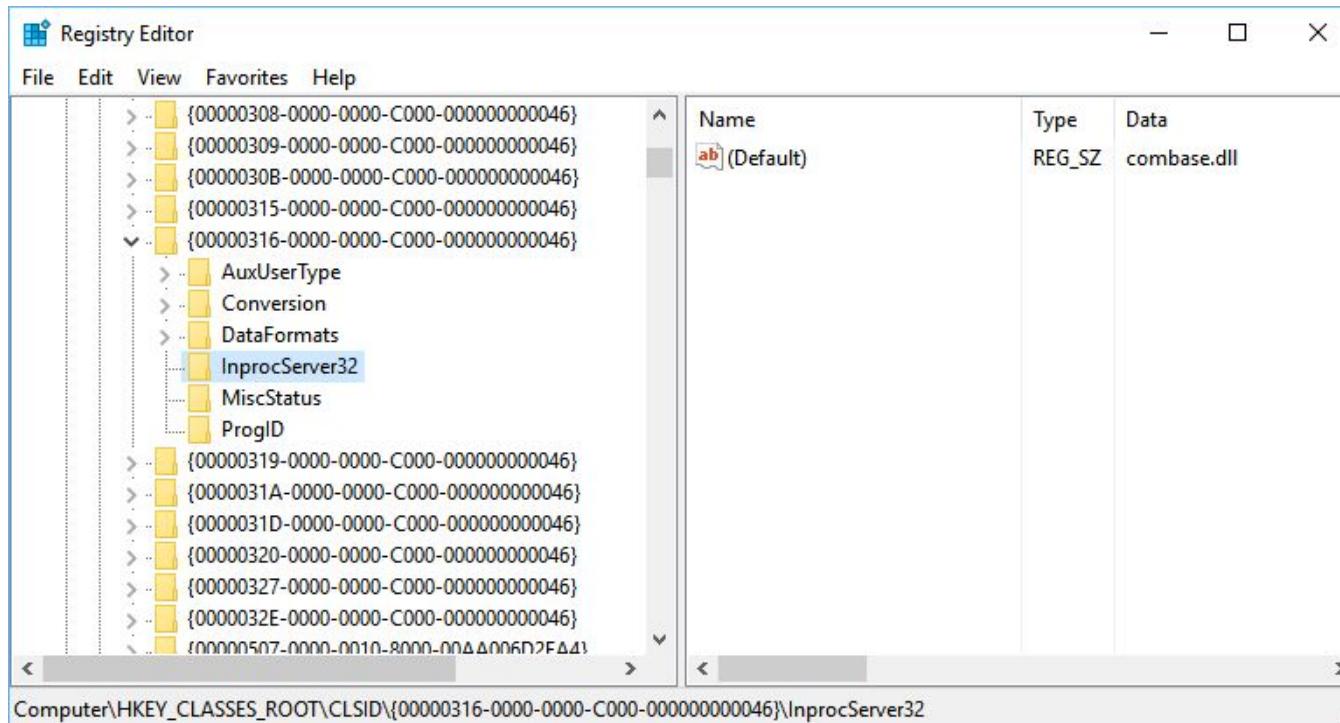
```
struct Interface1 : public IUnknown {} ;
struct Interface2 : public IUnknown {} ;
```

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

Cast is a GIANT code smell!
↖(ツ)↗

Class Registration

- Each Class Factory has a unique GUID, the Class ID (CLSID)
- Classes are registered in the Windows Registry under HKEY_CLASSES_ROOT\CLSID\{GUID}
- Registration information depends on the type of server



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,           ← Used for COM
        REFIID riid,                 Aggregation. Best
        void **ppvObject);           ignored :-)

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

Creating Class Factories and Instances

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

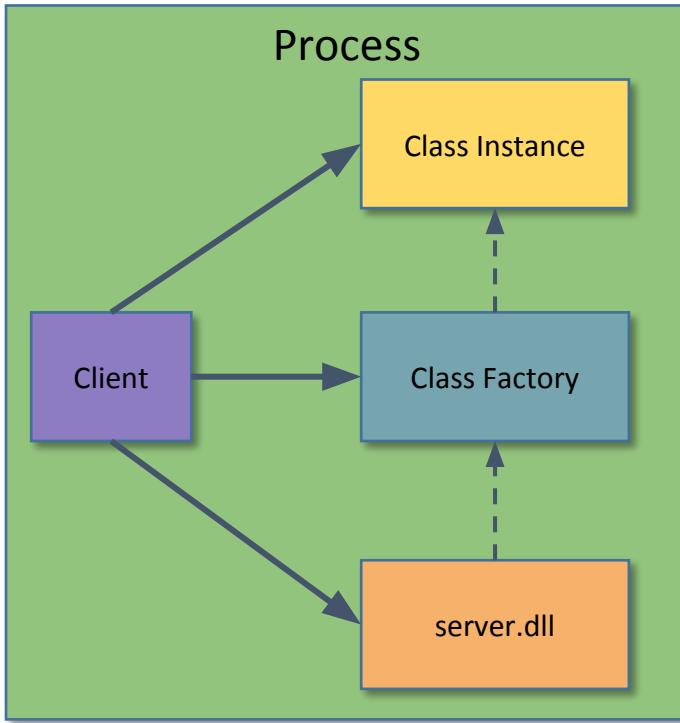
Specifies what type of server to lookup:

- CLSCTX_INPROC_SERVER
- CLSCTX_INPROC_HANDLER
- CLSCTX_LOCAL_SERVER
- CLSCTX_REMOTE_SERVER

Specify remote server information is required (more on this later).

```
HRESULT CoCreateInstanceEx(REFCLSID rclsid,  
                           IUnknown *punkOuter,  
                           DWORD dwClCtx,  
                           COSERVERINFO *pServerInfo,  
                           DWORD dwCount,  
                           MULTI_QI *pResults  
);
```

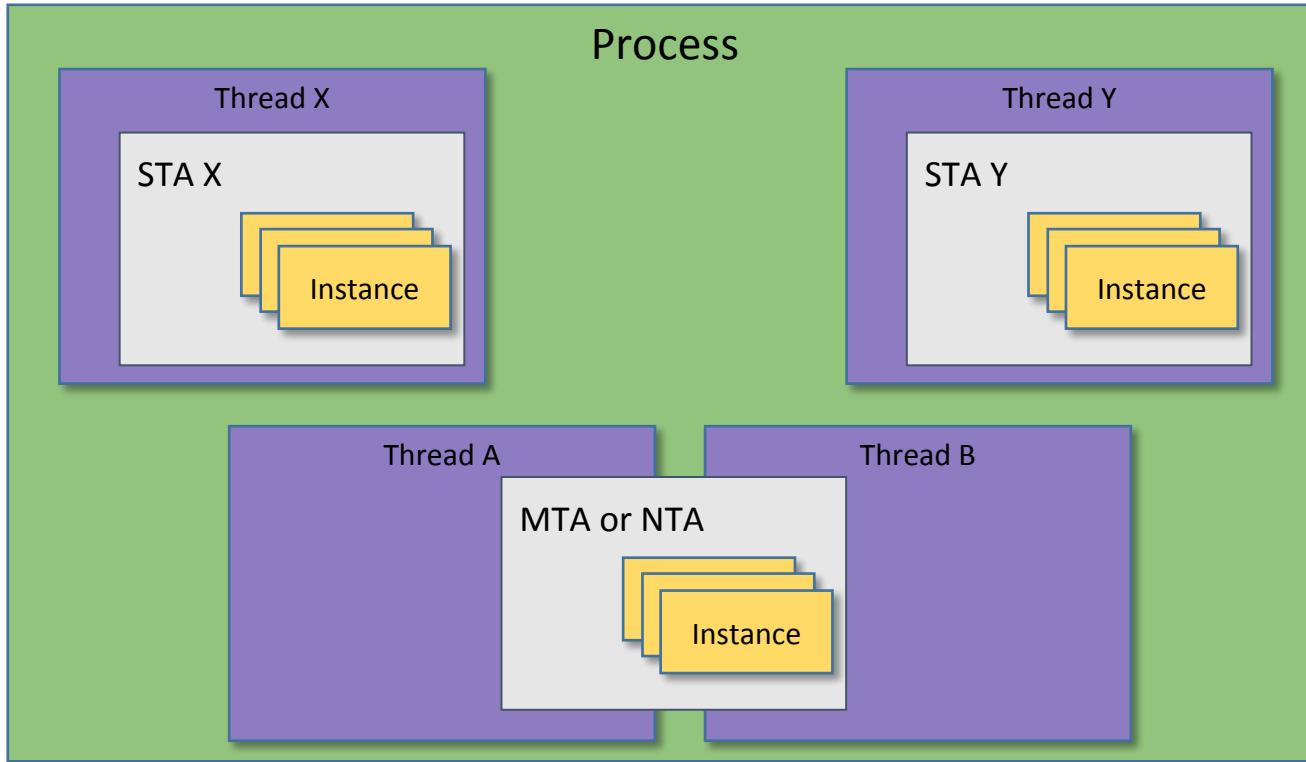
In-Process Server



- 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) ;
```

COM Apartments

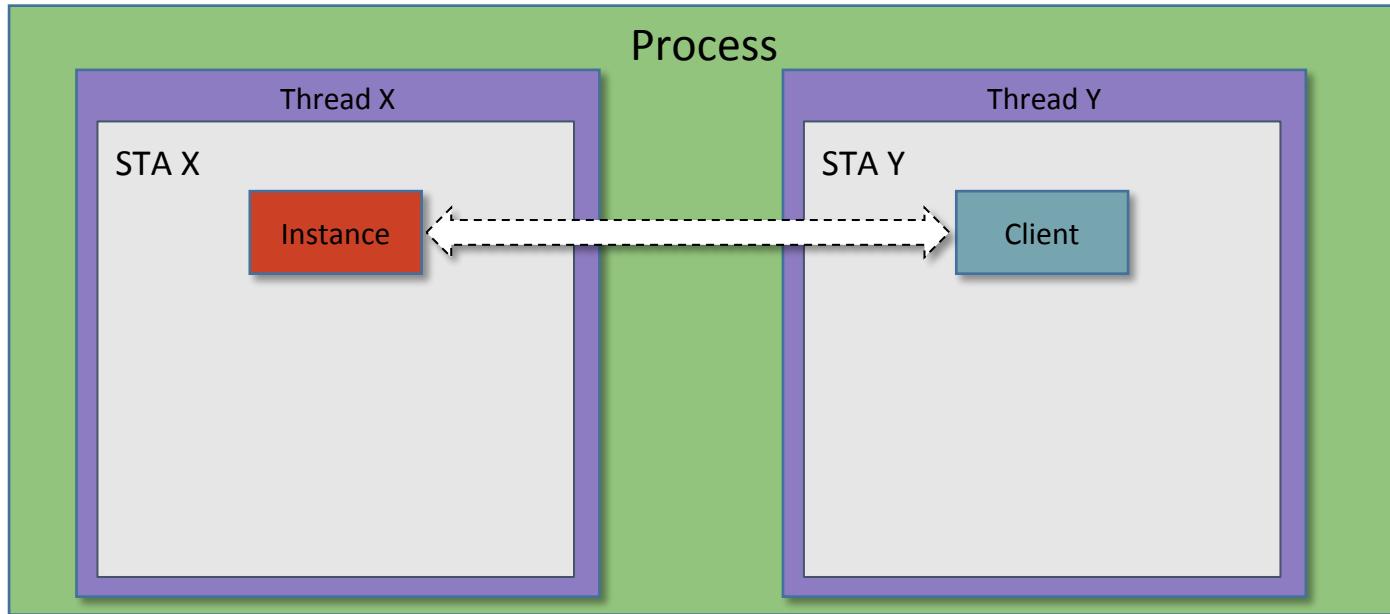


```
HRESULT CoInitializeEx(   
    LPVOID pvReserved,  
    DWORD dwCoInit  
) ;
```

COINIT_APARTMENTTHREADED
or
COINIT_MULTITHREADED

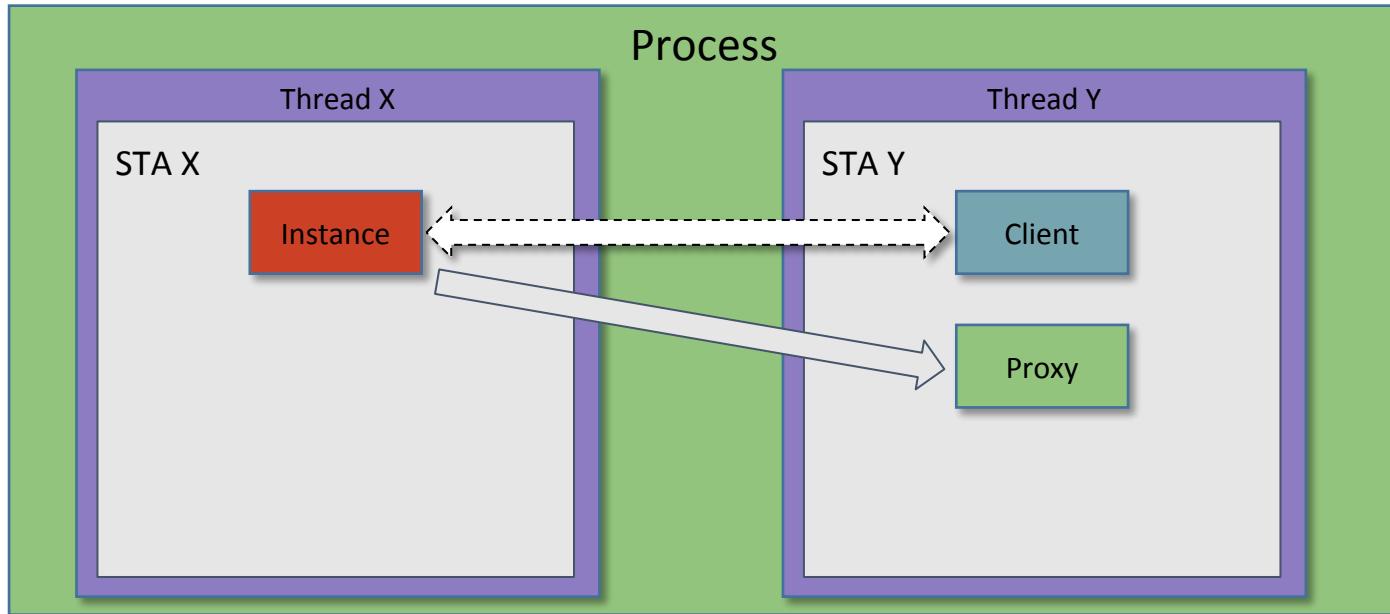
Single Threaded Apartments (STA)

- One STA per-thread. Initialized with `COINIT_APARTMENTTHREADED`.
- Created instance can only be called directly from the thread it was created on.
- Any other callers need to “Unmarshal” a Proxy to the object. Calls are sent to the original thread via a hidden Window and Window messaging



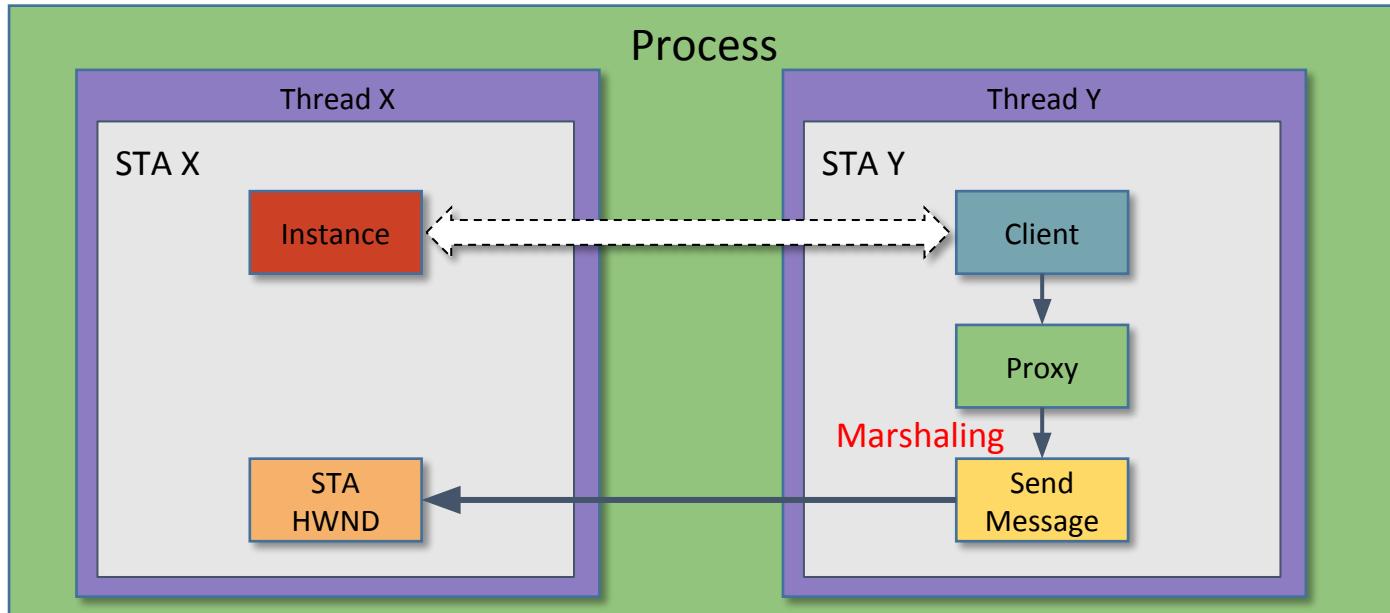
Single Threaded Apartments (STA)

- One STA per-thread. Initialized with COINIT_APARTMENTTHREADED.
- Created instance can only be called directly from the thread it was created on.
- Any other callers need to “Unmarshal” a Proxy to the object. Calls are sent to the original thread via a hidden Window and Window messaging



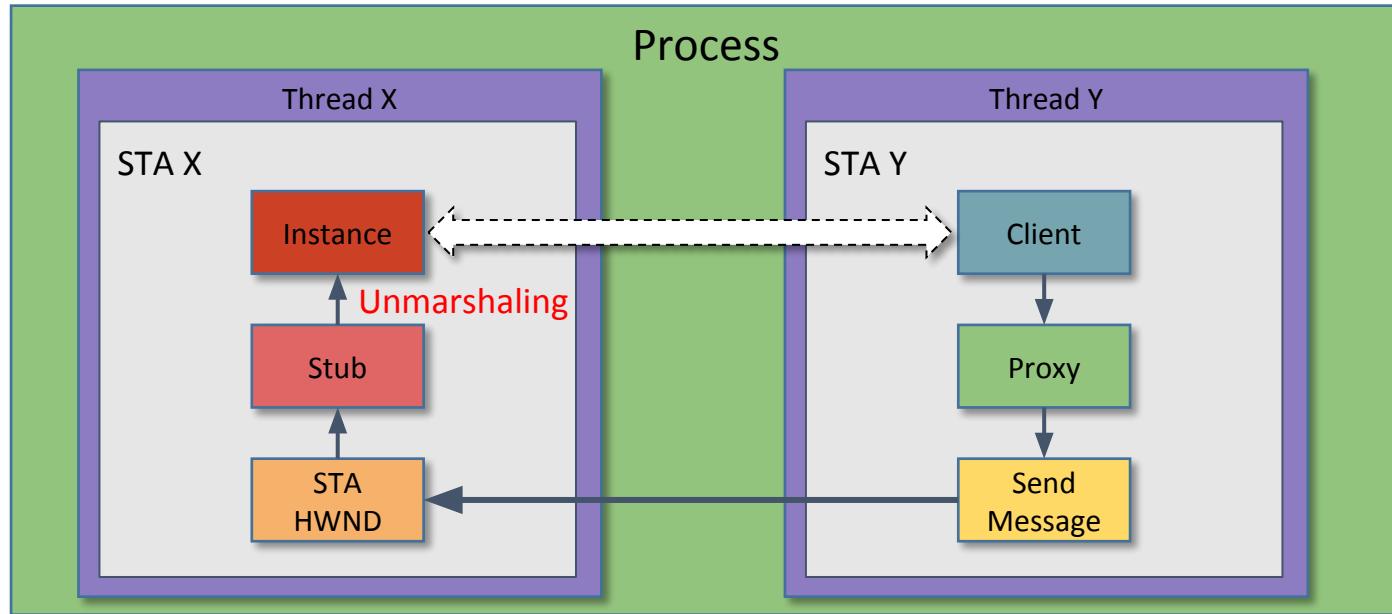
Single Threaded Apartments (STA)

- One STA per-thread. Initialized with COINIT_APARTMENTTHREADED.
- Created instance can only be called directly from the thread it was created on.
- Any other callers need to “Unmarshal” a Proxy to the object. Calls are sent to the original thread via a hidden Window and Window messaging



Single Threaded Apartments (STA)

- One STA per-thread. Initialized with COINIT_APARTMENTTHREADED.
- Created instance can only be called directly from the thread it was created on.
- Any other callers need to “Unmarshal” a Proxy to the object. Calls are sent to the original thread via a hidden Window and Window messaging



The Mystery Window

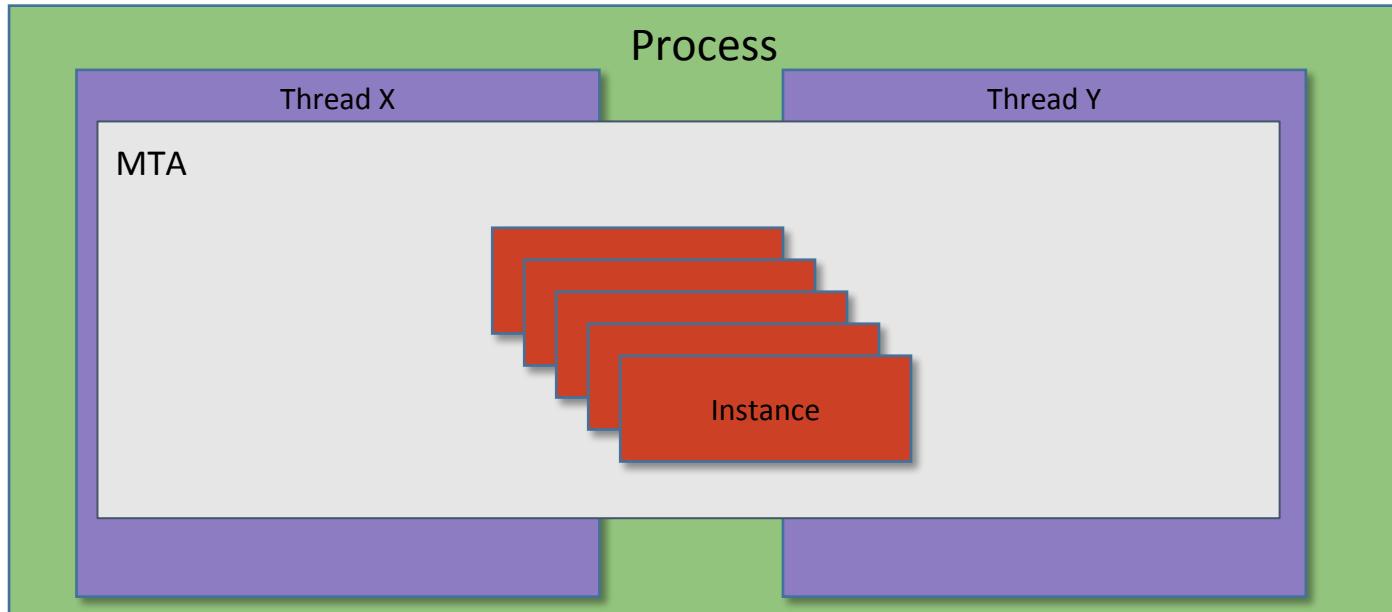
- Initializing a STA creates a hidden “Message Only” Window

```
HWND hwnd = FindWindowEx(HWND_MESSAGE, NULL, NULL, NULL);
while (hwnd) {
    WCHAR name[256] = {};
    if (GetWindowText(hwnd, name, _countof(name))
        && _wcsnicmp(name, L"ole", 3) == 0) {
        DWORD pid = 0;
        DWORD tid = GetWindowThreadProcessId(hwnd, &pid);
        printf("%p %5d %5d %ls\n", hwnd, pid, tid, name);
    }
    hwnd = GetNextWindow(hwnd, GW_HWNDNEXT);
}
```

HWND	PID	TID	NAME
002906E4	21300	13416	OleMainThreadWndName
00510942	4316	8464	OLEChannelWnd
005F0A4A	4316	21232	OLEChannelWnd
00210D3E	4316	6028	OLEChannelWnd
001A048E	5880	19768	OLEChannelWnd
005C077E	5880	16680	OLEChannelWnd
004E08A2	5880	12464	OLEChannelWnd
000702E4	19600	20756	OleMainThreadWndName
00530832	5880	12636	OleMainThreadWndName
004E07EE	10952	18772	OLEChannelWnd
00110D58	10952	19568	OleMainThreadWndName
003D03B2	5404	19032	OLEChannelWnd
004B0B7A	2244	2416	OleMainThreadWndName
001E09D8	19828	19832	OleMainThreadWndName

Multi-Threaded Apartments (MTA)

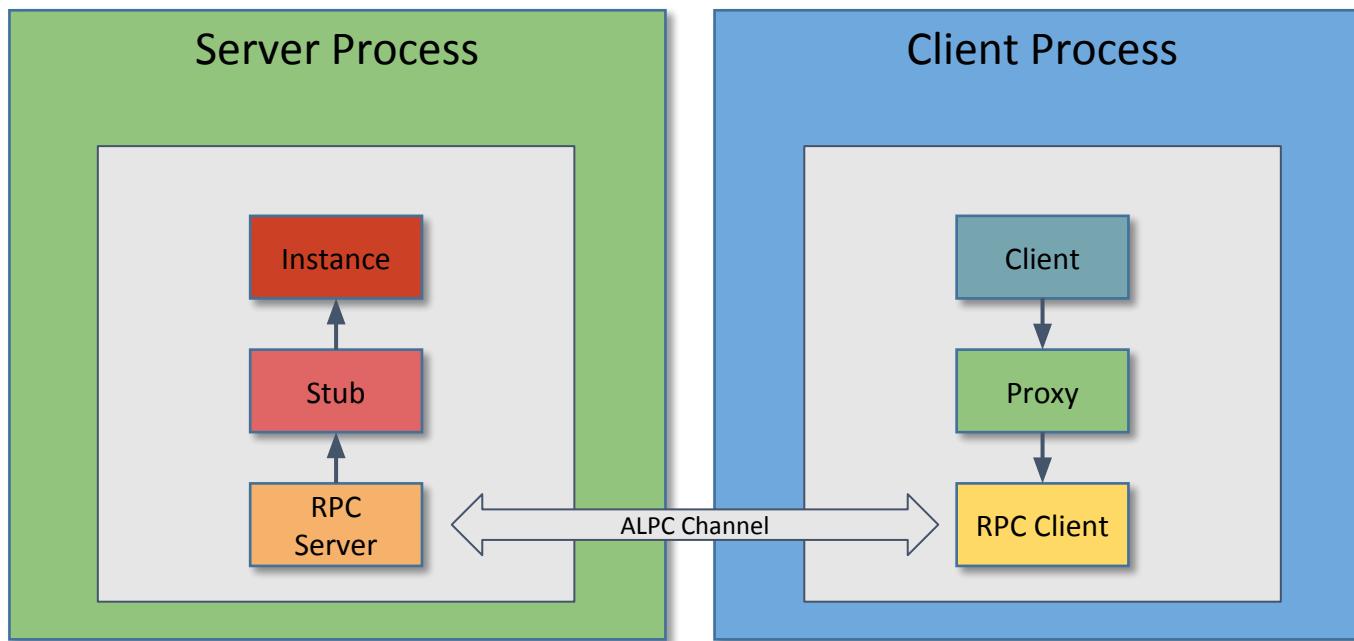
- One MTA per-process. Initialized with `COINIT_MULTITHREADED`.
- Created instance can be accessed directly in any thread in the MTA.
- STA instances need to be marshaled and vice-versa
- Also similar Neutral Apartment (NTA) which optimizes away some marshaling.



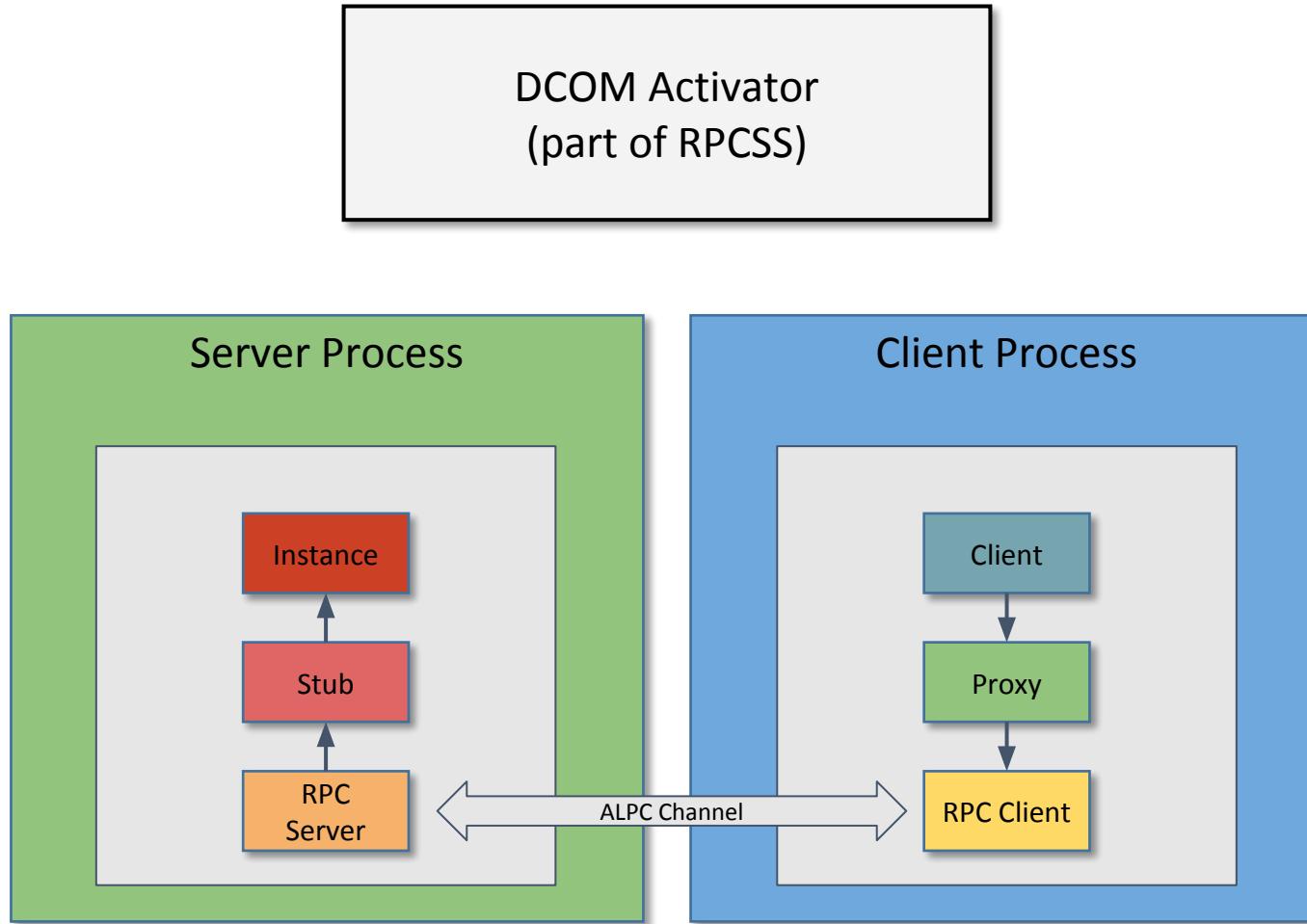
Threading Models

- In Process servers should be registered with a *ThreadingModel* registry value.
- Takes a string which must be one of the following values:
 - “Apartment” - Object must be created inside a STA
 - “Free” - Free-threaded, object must be created in the MTA
 - “Both” - Object can be created and used directly in either STA or MTA
 - “Neutral” - Object must be created in a NTA (COM+ Wizardry)
- If no *ThreadingModel* object can ONLY be used and created in the first STA in the process.
- If *Free* object created in a STA then MTA is created if it doesn’t exist and marshaled back.
- Likewise if *Apartment* object created in an MTA a new STA is created and object marshaled back

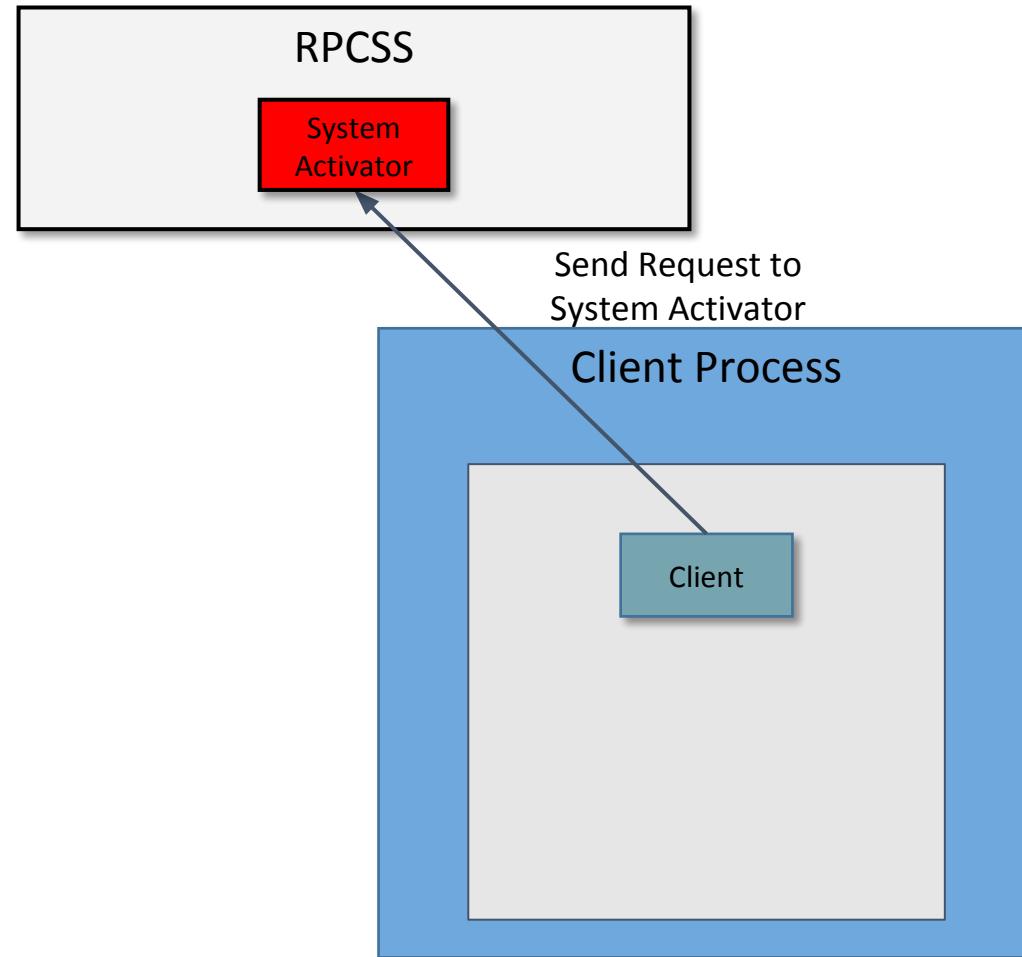
Local Server Activation



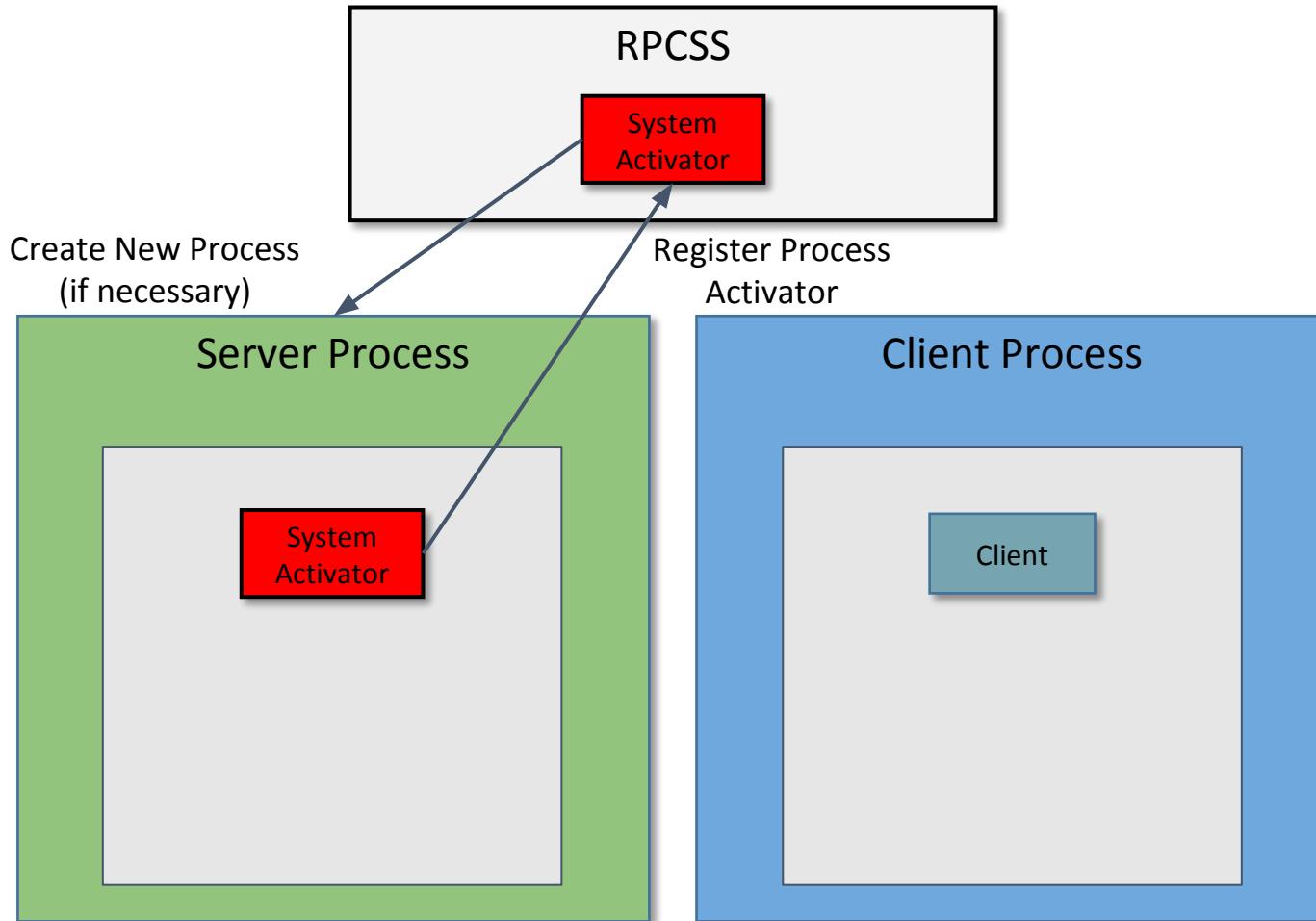
Local Server Activation



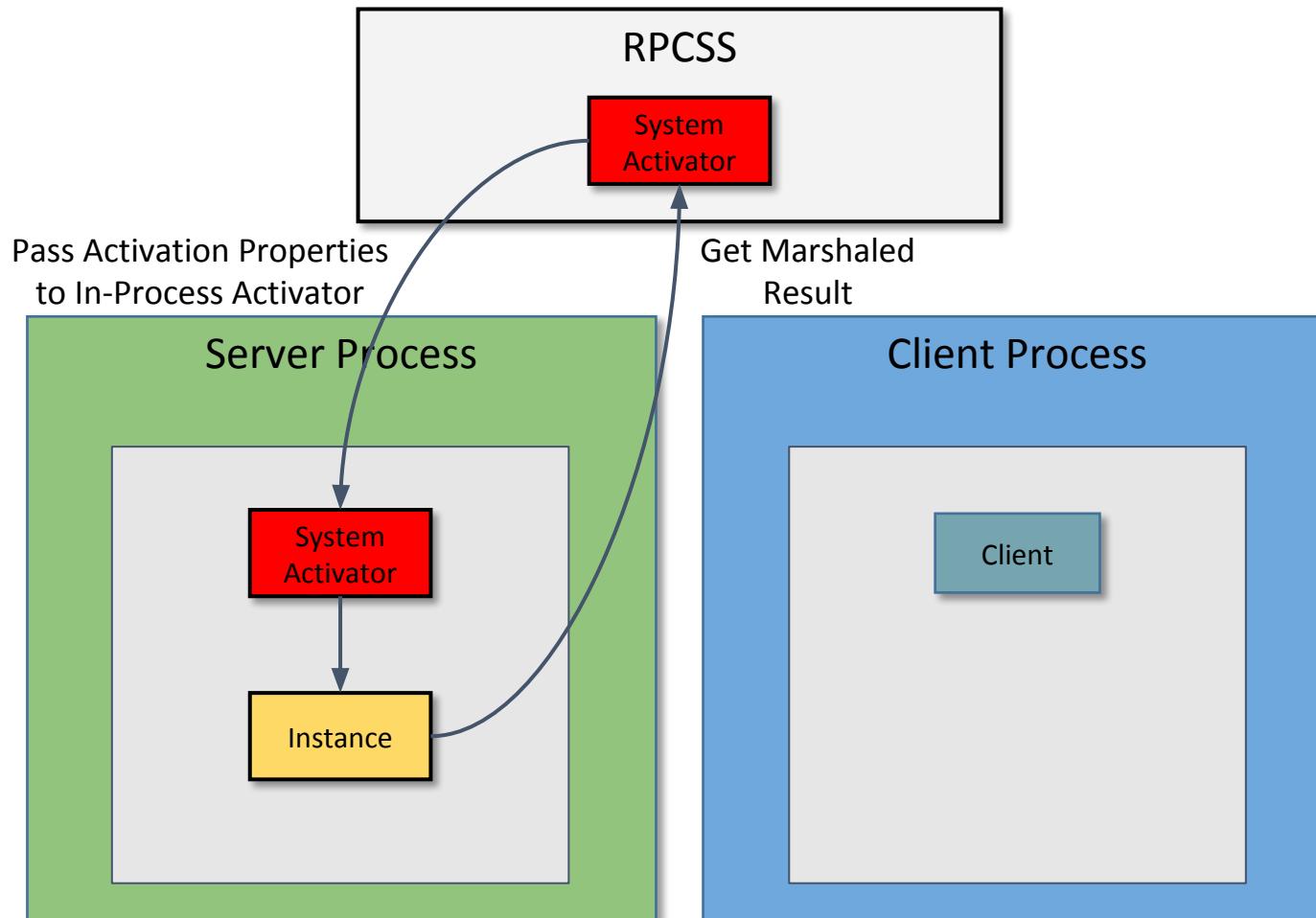
Local Server Activation



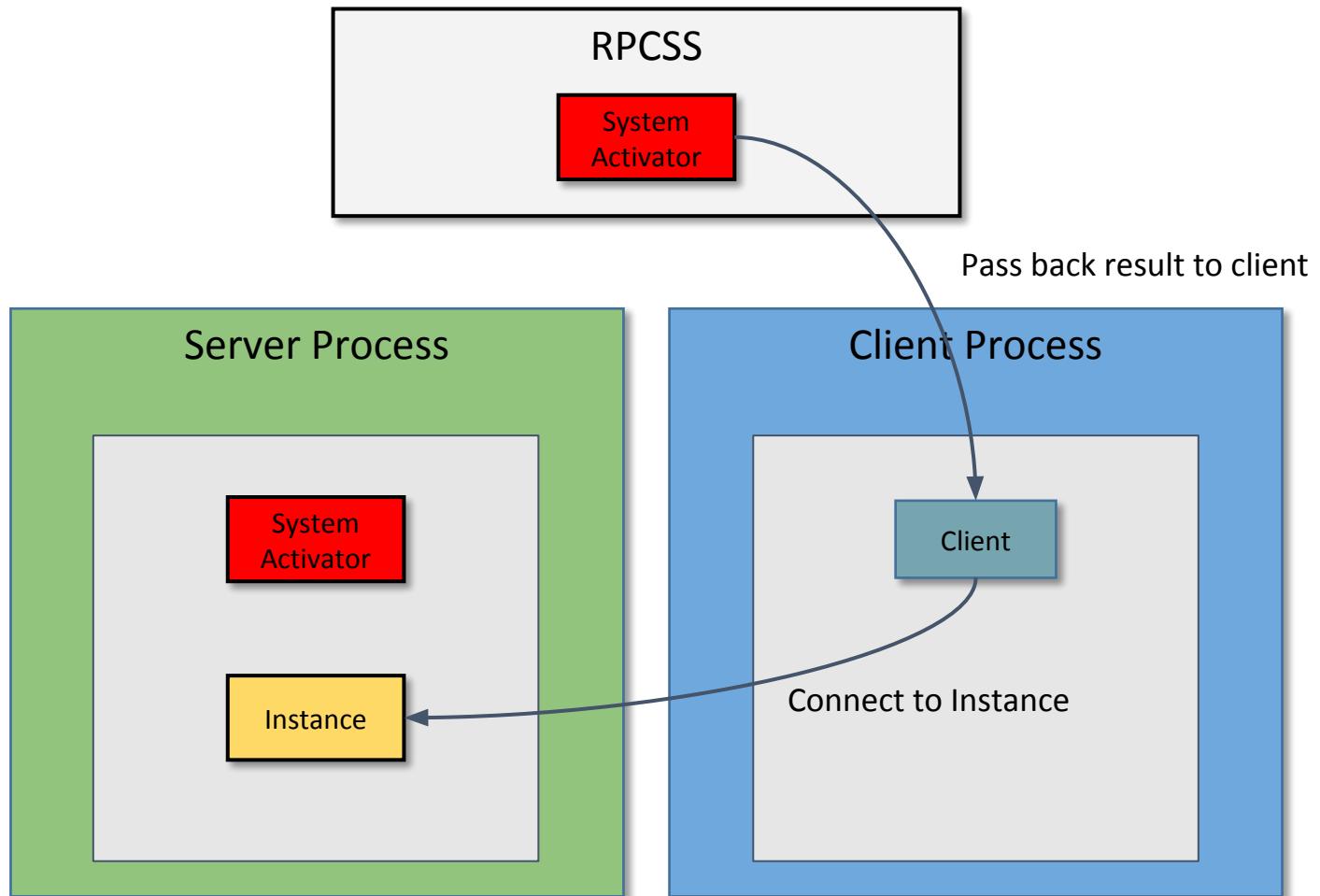
Local Server Activation



Local Server Activation



Local Server Activation



System Activator

```
DEFINE_GUID(IID_ISystemActivator,
            "000001a0-0000-0000-c000-000000000046")
struct ISystemActivator : public IUnknown {
    HRESULT GetClassObject(
        IActivationPropertiesIn *pActProperties,
        IActivationPropertiesOut **ppActProperties);

    HRESULT CreateInstance(
        IUnknown *pUnkOuter,
        IActivationPropertiesIn *pActProperties,
        IActivationPropertiesOut **ppActProperties);
};
```

Activation Properties In



```
struct CustomHeader {  
    DWORD totalSize;  
    DWORD headerSize;  
    DWORD dwReserved;  
    DWORD destCtx;  
    DWORD cIfs;  
    CLSID classInfoClsid;  
    CLSID *pclsid;  
    DWORD *pSizes;  
    CustomOpaqueData *opaqueData;  
};
```

List of GUIDs and Sizes of
following Property Blobs

```
struct InstantiationInfoData {  
    CLSID classId;  
    DWORD classCtx;  
    DWORD actvfllags;  
    long fIsSurrogate;  
    DWORD cIID;  
    DWORD instFlag;  
    IID *pIID;  
    DWORD thisSize;  
    COMVERSION clientCOMVersion;  
};
```

CLSID of class to create.

```
enum ACTIVATION_FLAGS {  
    ACTVFLAGS_DISABLE_AAA,  
    ACTVFLAGS_ACTIVATE_32_BIT_SERVER,  
    ACTVFLAGS_ACTIVATE_64_BIT_SERVER,  
    ACTVFLAGS_NO_FAILURE_LOG,  
    ACTVFLAGS_WINRT_LOCAL_SERVER,  
    ACTVFLAGS_WINRT_PER_USER_OK,  
    ACTVFLAGS_APPCONTAINER,  
};
```

List of GUIDs to query for.

Activation Properties Out

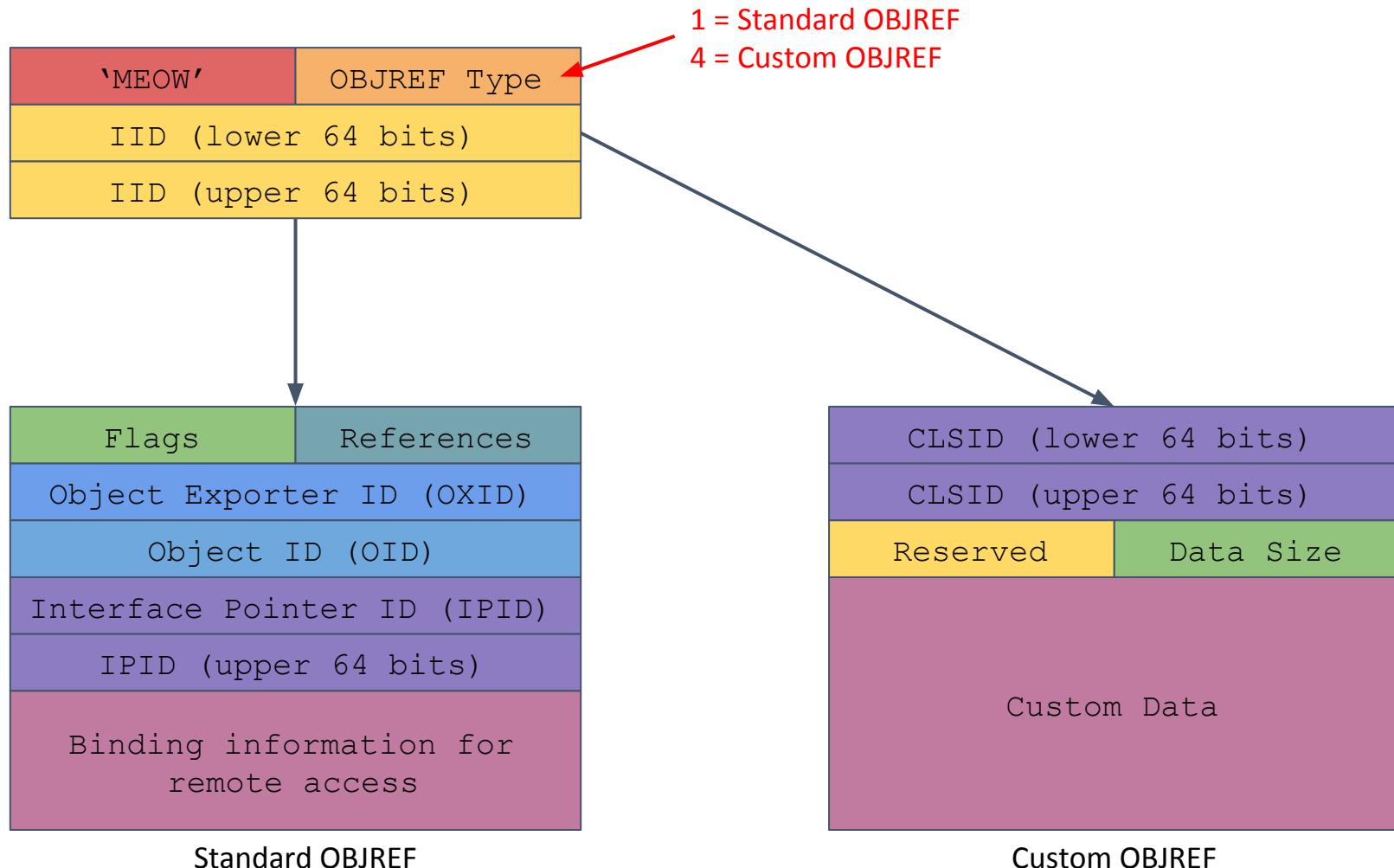
- Same structure as ActivationPropertiesIn just with different property sets.

```
struct PropsOutInfo {  
    DWORD cIfs;  
    IID *pid;  
    HRESULT *phResults;  
    MInterfacePointer **ppIntfData;  
};
```

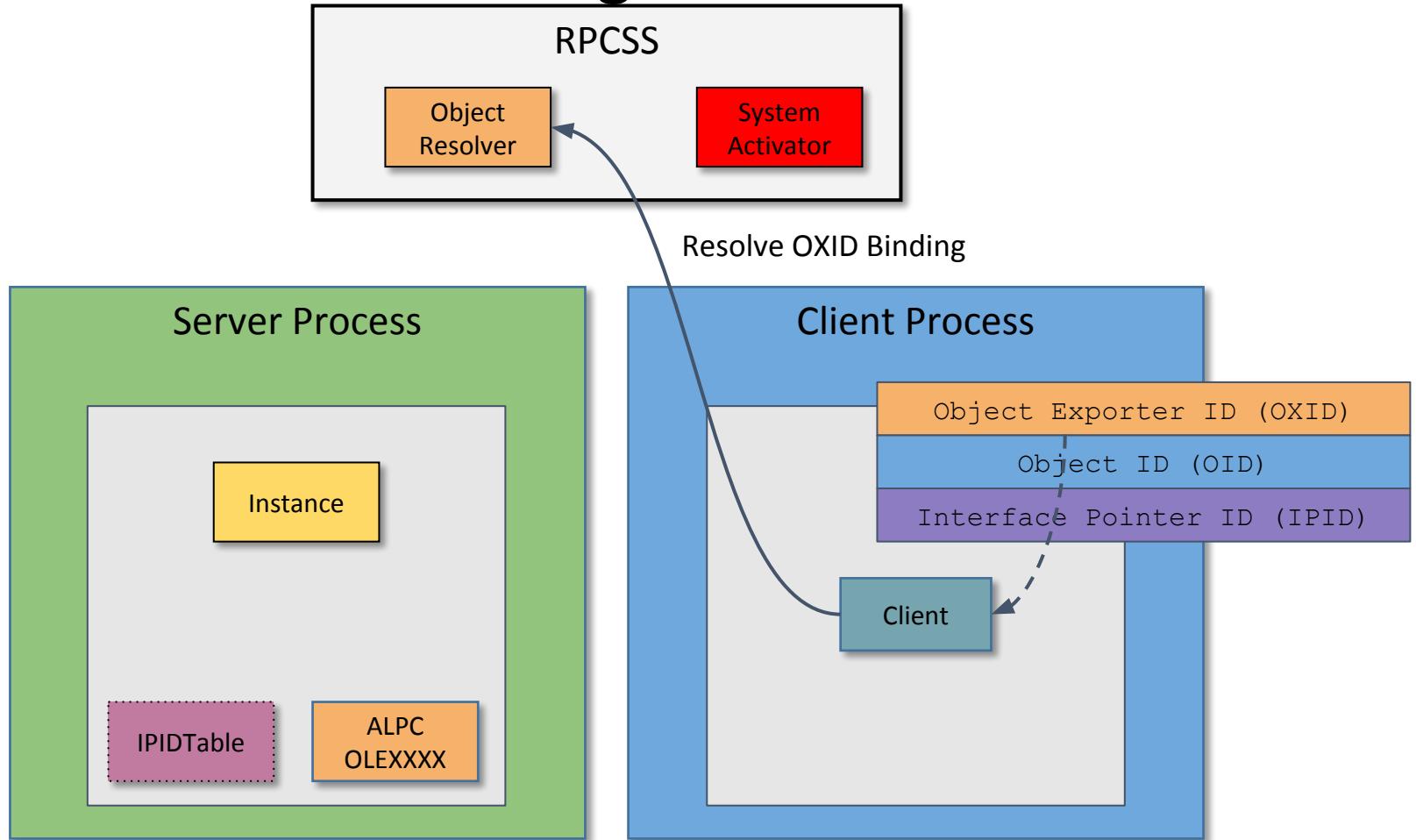
```
struct MInterfacePointer {  
    unsigned long ulCntData;  
    byte abData[];  
};
```

00000000	4D 45 4F 57 01 00 00 00 A0 01 00 00 00 00 00 00	MEOW....
00000010	C0 00 00 00 00 00 00 46 00 00 00 00 00 01 00 00 00	À.....F....
00000020	19 14 A0 F8 BF 3D 72 6A D5 41 68 BD 84 C3 08 E7	.. øj=rjÖAh‰.Ã.ç
00000030	03 5C 00 00 38 16 E4 46 37 53 80 9C 87 A9 08 45	.\\..8.äF7S...@.E
00000040	00 00 00 00

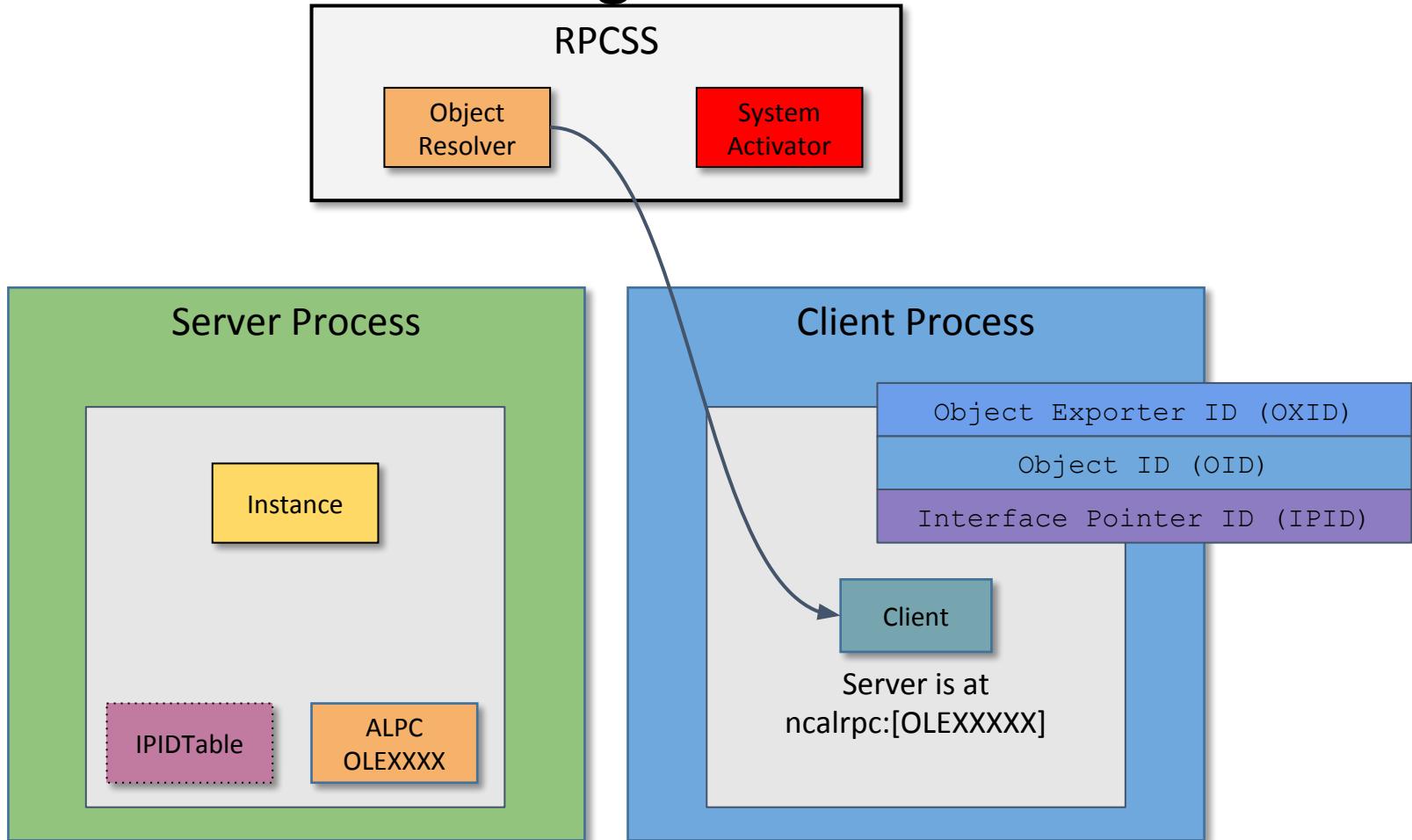
It's like Marshaling Cats



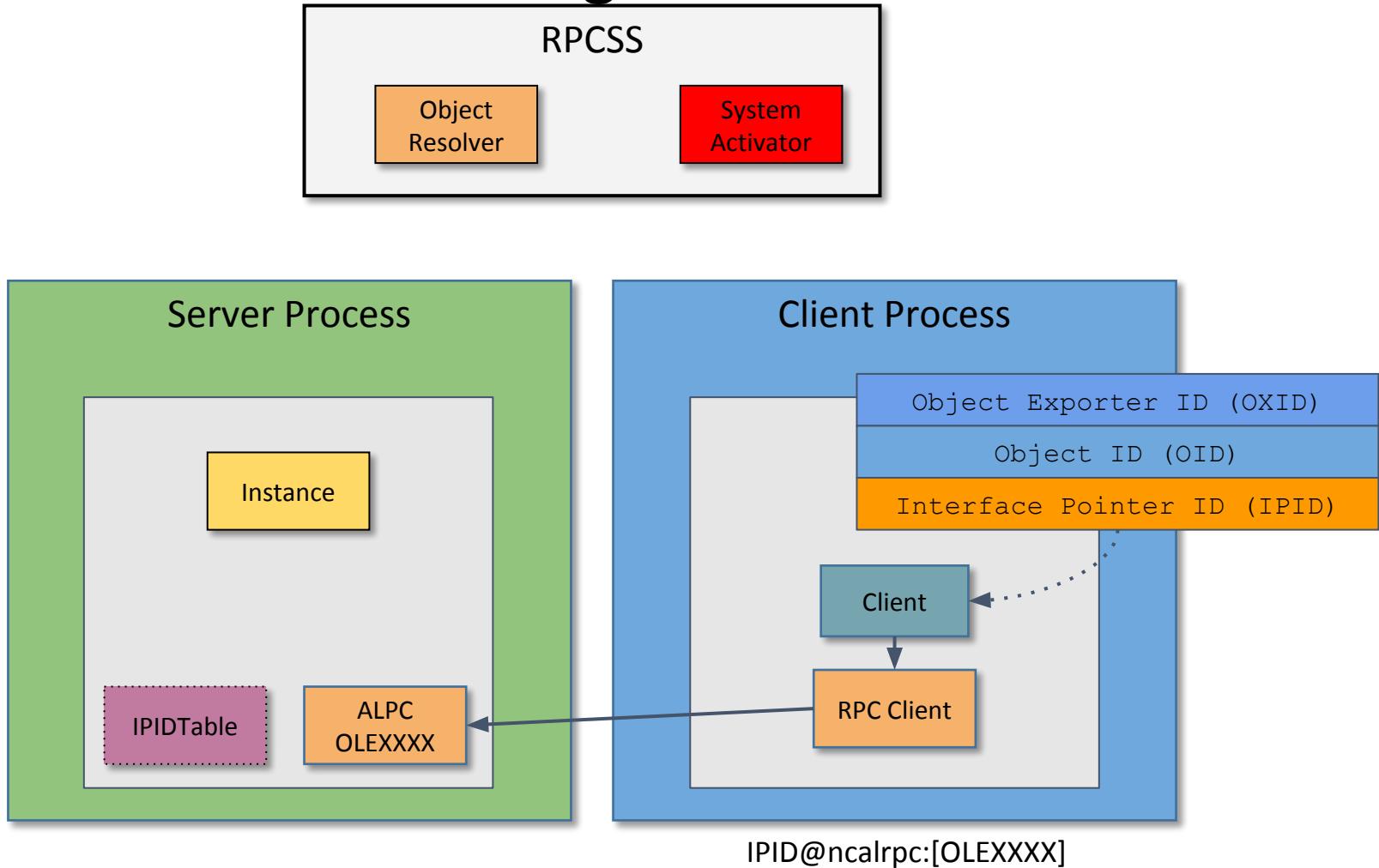
Standard Unmarshaling



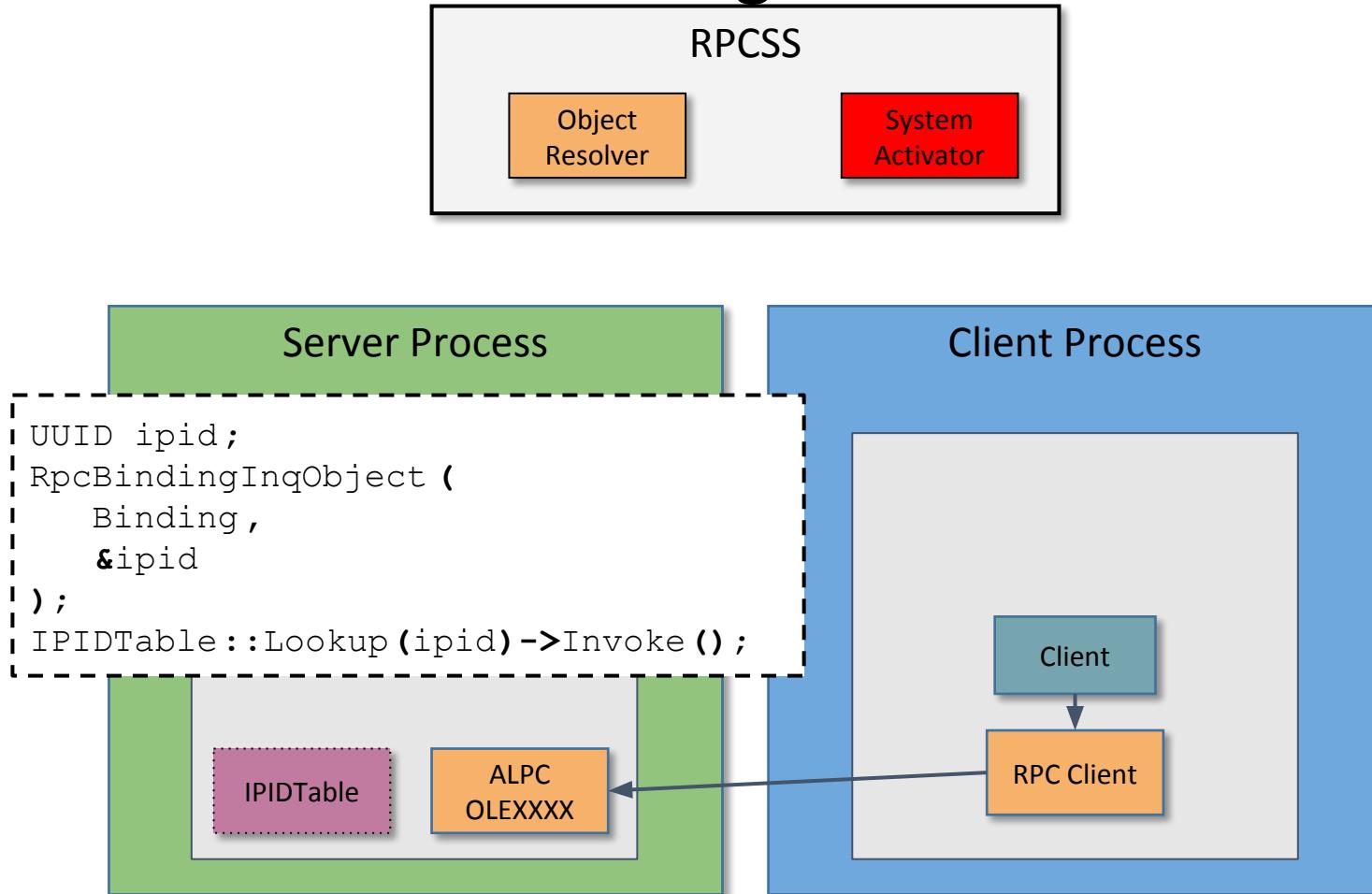
Standard Unmarshaling



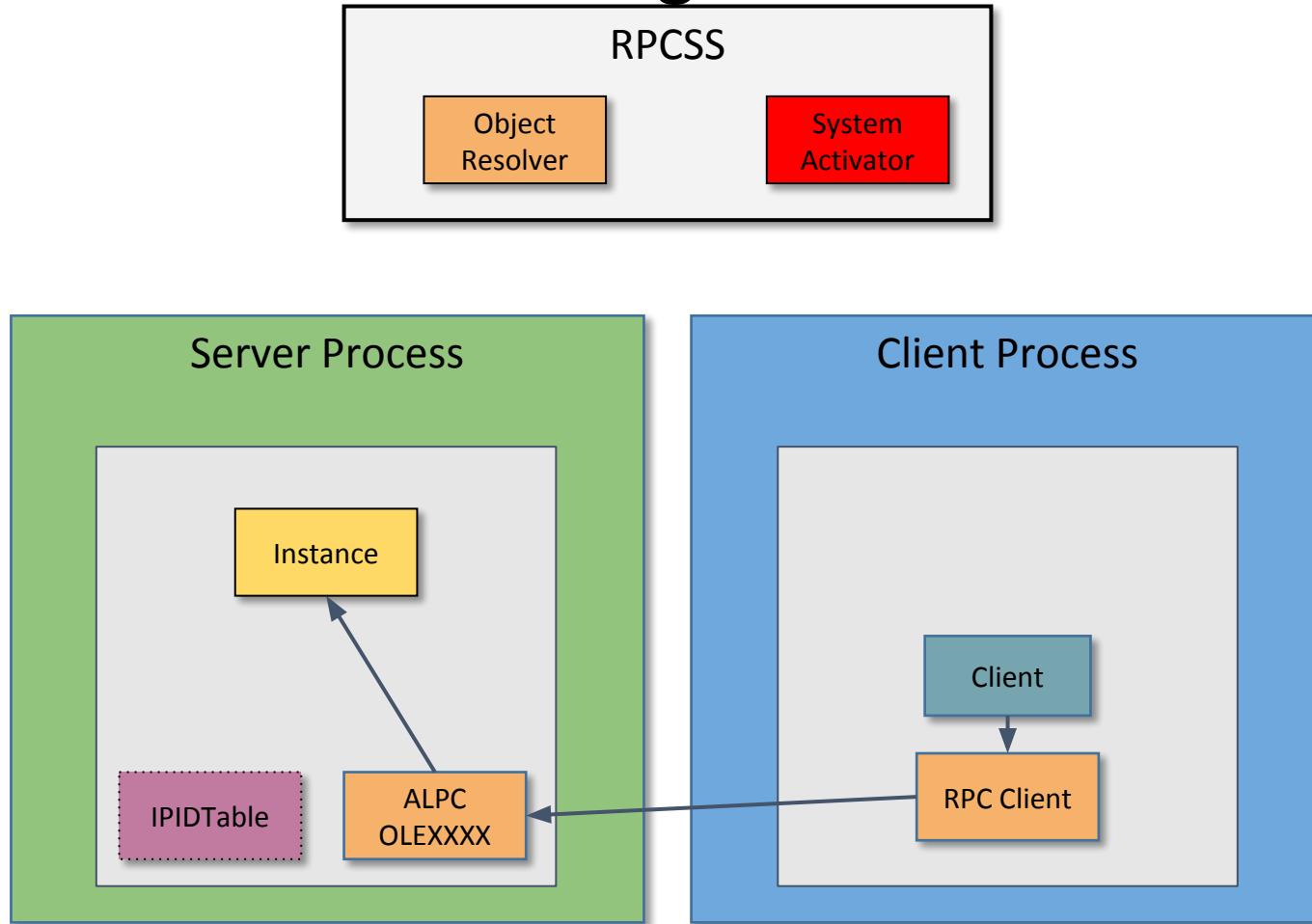
Standard Unmarshaling



Standard Unmarshaling



Standard Unmarshaling



Interface Proxies and Stubs

The screenshot shows two windows side-by-side. On the left is the Windows Registry Editor. In the center pane, under the key `Computer\HKEY_CLASSES_ROOT\Interface\{475CA8F3-9417-48BC-B9D7-4163A7844C02}`, there is a subkey named `ProxyStubClSID32`. A red arrow points from this subkey to the `Supported Interfaces` tab of the `OleViewDotNet 64bit` window on the right. The `Supported Interfaces` tab lists various COM interfaces, each with its IID. The interface `IClaimedMagneticStripeReader` is highlighted in blue. The `Object Properties` tab is visible on the far right of the `OleViewDotNet` window.

Name	IID
IMagneticStripeReaderBankCardDataReceivedEventArgs	2E958823-A31A-4763-882C-23725E39B08E
IPosPrinterStatusUpdatedEventArgs	2EDB87DF-13A6-428D-BA81-B0E7C3E5A3CD
ICashDrawerStatusUpdatedEventArgs	30AAE98A-0D70-459C-9553-87E124C52488
ITypedEventHandler_2_Windows_CDevices_CPointOfService_CClaimedPosP...	31424F6F-CFEB-5031-8A95-BEA59B09E584
IAsyncOperationCompletedHandler_1_Windows_CDevices_CPointOfService_...	32C55F7B-8EE3-555D-998B-78C98AA9627B
BarcodeScannerStatusUpdatedEventArgs	355D8586-9C43-462B-A91A-816DC97F452C
JournalPrinterCapabilities	385CCC43-E047-4463-BB58-17B5BA1D8056
IAsyncOperation_1_Windows_CDevices_CPointOfService_CClaimedMagnetic...	41630BD4-F45A-590D-8A4E-F70C9E49AD01
BarcodeScannerDataReceivedEventArgs	4234AE72-ED97-467D-AD2B-01E44313A929
IAsyncOperation_1_Windows_CDevices_CPointOfService_CCashDrawer	45007467-92F2-58FF-B191-AA5000FEDD9E
IClaimedMagneticStripeReader	475CA8F3-9417-48BC-B9D7-4163A7844C02
IClaimedBarcodeScanner	4A63B49C-8FA4-4332-BB26-945D11D81E0F
ITypedEventHandler_2_Windows_CDevices_CPointOfService_CClaimedBarc...	4F64E49A-BD8C-549D-970C-A5A250BD27CA
IMagneticStripeReaderCardTypesStatics	528F2C5D-2986-474F-8454-7CCD0592BD5F
IAsyncOperation_1_FIVectorView_1_UINT32	52C56F3C-713A-5162-9E62-362CE7ED53BE
IReceiptOrSlipJob	532199BE-C8C3-4DC2-89E9-5C4A37B34DDC
IMagneticStripeReaderEncryptionAlgorithmsStatics	53B57350-C3DB-4754-9C00-41392374A109
IAsyncOperationCompletedHandler_1_FIVectorView_1_UINT32	55772F29-DA64-5C87-871C-074337A84573
IAsyncOperationCompletedHandler_1_Windows_CDevices_CPointOfService_...	57836710-F186-5636-891D-F8C5398EA6DF
IPosPrinterCharacterSetIdsStatics	5C709EFF-709A-4FE7-B215-06A748A38B39
IBarcodeScannerReport	5CE4D8B0-A489-4B96-86C4-F0BF8A37753D
IDemandCommunication	6D11EE6E-DAA0-A1E0-0C0C-EDC9E2A0C1E5

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

Proxy-Stub Factory

- Proxy-Stub CLSID doesn't use IClassFactory. Instead uses IPSFactoryBuffer.

```
DEFINE_GUID(IID_IPSFactoryBuffer,
    "D5F569D0-593B-101A-B569-08002B2DBF7A" )
struct IPSFactoryBuffer : public IUnknown
{
    HRESULT CreateProxy (
        IUnknown *pUnkOuter,
        REFIID riid,
        IRpcProxyBuffer **ppProxy,
        void **ppv);

    HRESULT CreateStub (
        REFIID riid,
        IUnknown *pUnkServer,
        IRpcStubBuffer **ppStub);
};
```

IRemUnknown

- Proxies don't directly forward QueryInterface/AddRef/Release to the remote object, these calls are used on the proxy object.
- Actual IUnknown calls done on special interface, IRemUnknown

```
struct customREMOTE_REPLY_SCM_INFO {
    OXID             Oxid;
    DUALSTRINGARRAY *pdsaOxidBindings;
    IPID             ipidRemUnknown;
    DWORD            authnHint;
    COMVERSION       serverVersion;
};
```

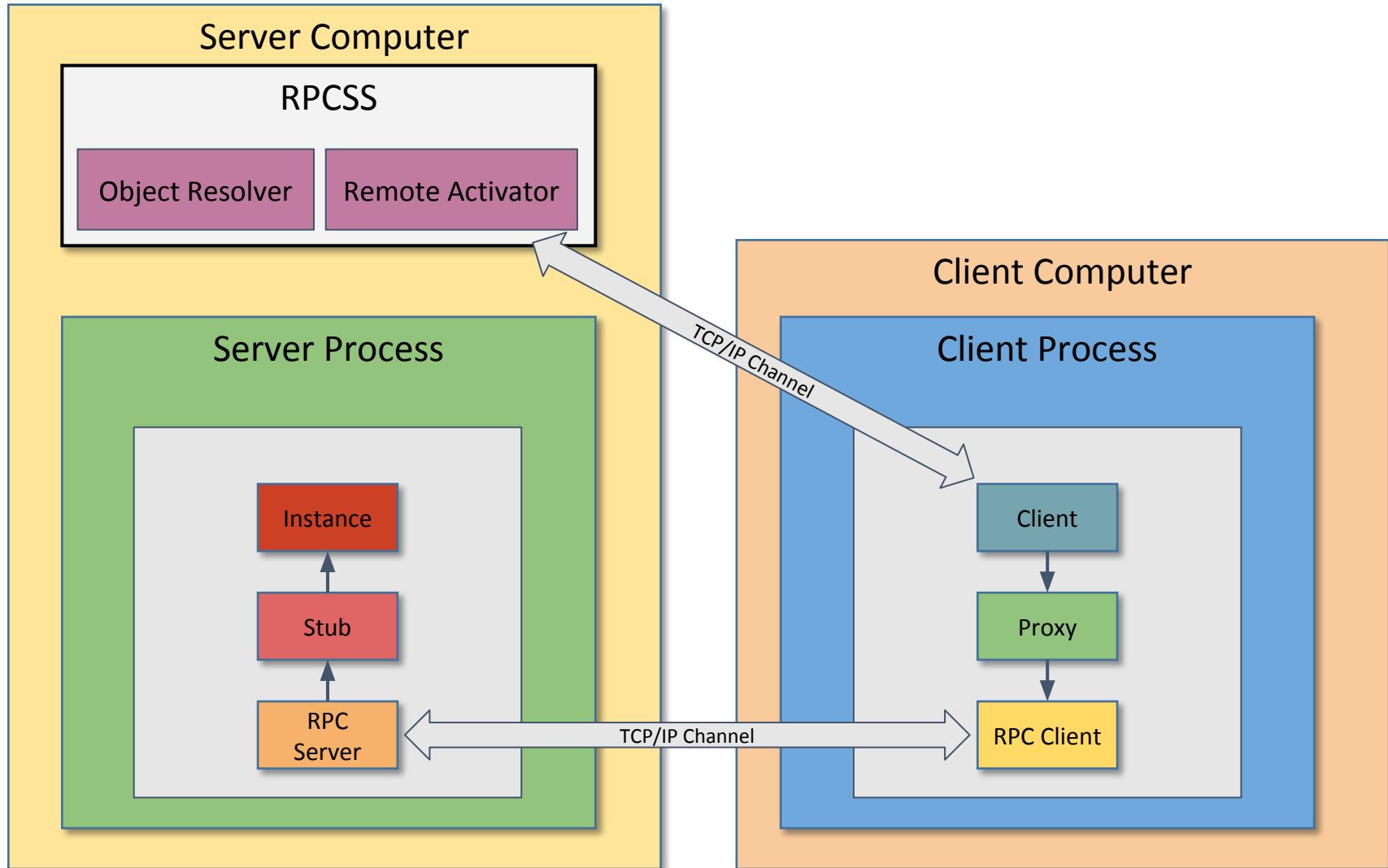
Returned in ActivationPropertiesOut

```
DEFINE_GUID(IID_IRemUnknown,
    "00000131-0000-0000-C000-00000000046")
struct IRemUnknown : public IUnknown {
    HRESULT RemQueryInterface(
        REFIID ripid,
        unsigned long cRefs,
        unsigned short cLids,
        IID *iids,
        PREMQIRESULT *ppQIResults);

    HRESULT RemAddRef(
        unsigned short cInterfaceRefs,
        REMINTERFACEREF InterfaceRefs[],
        HRESULT *pResults);

    HRESULT RemRelease(
        unsigned short cInterfaceRefs,
        REMINTERFACEREF InterfaceRefs[]);
};
```

Remote Server



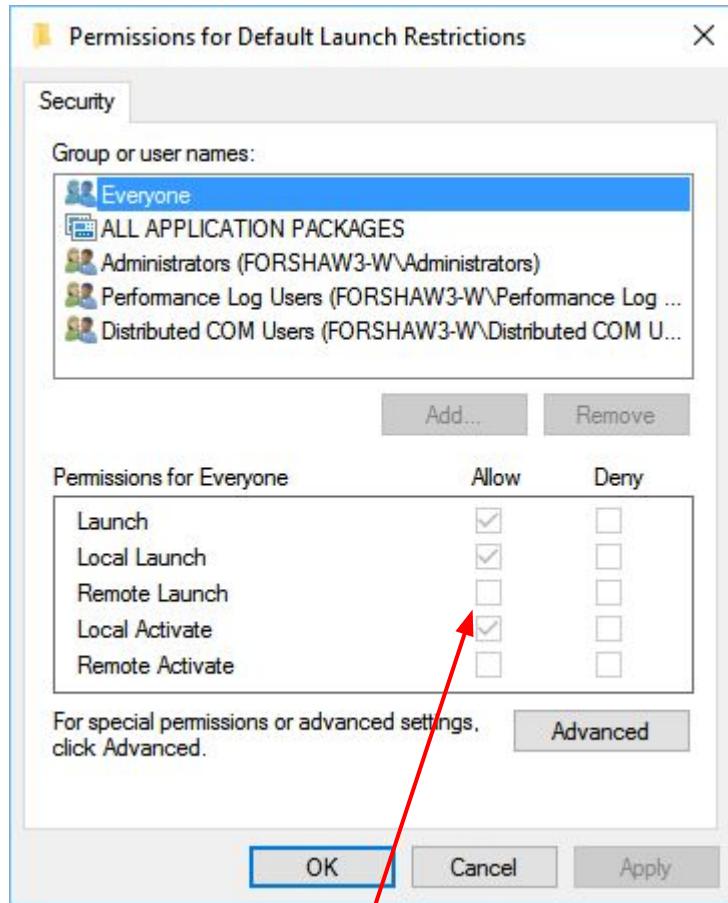
COSERVERINFO

```
struct COSERVERINFO {  
    DWORD dwReserved1;  
    LPWSTR pwszName;  
    COAUTHINFO *pAuthInfo;  
    DWORD dwReserved2;  
};
```

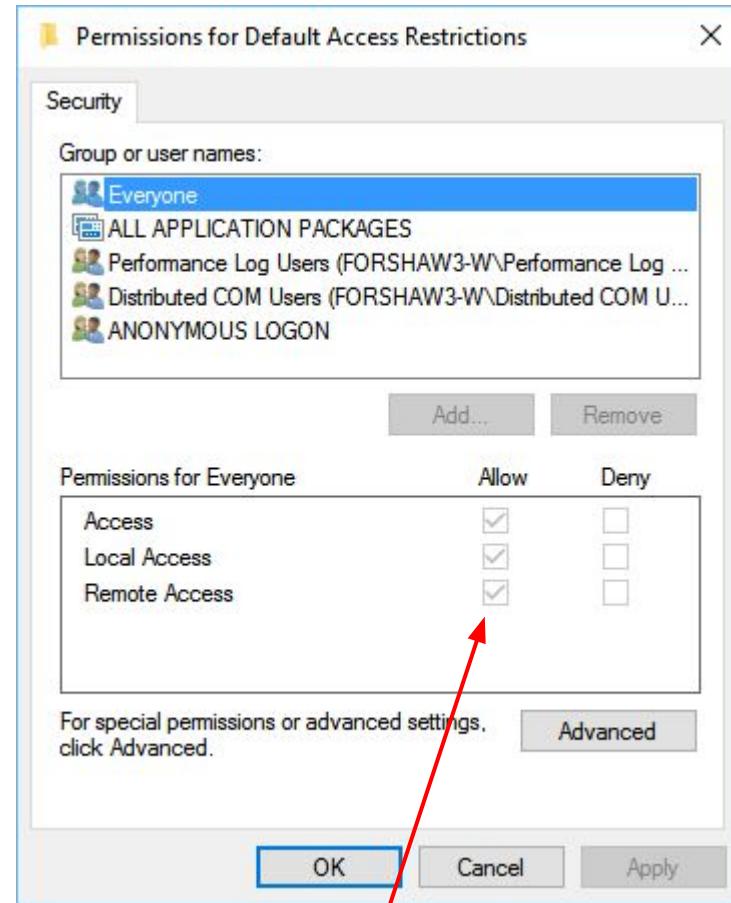
Can specify better authentication level when connecting such as CALL or even better PKT_PRIVACY

```
struct COAUTHINFO {  
    DWORD dwAuthnSvc;  
    DWORD dwAuthzSvc;  
    LPWSTR pwszServerPrincName;  
    DWORD dwAuthnLevel;  
    DWORD dwImpersonationLevel;  
    void* pAuthIdentityData;  
    DWORD dwCapabilities;  
};
```

COM Security Restrictions

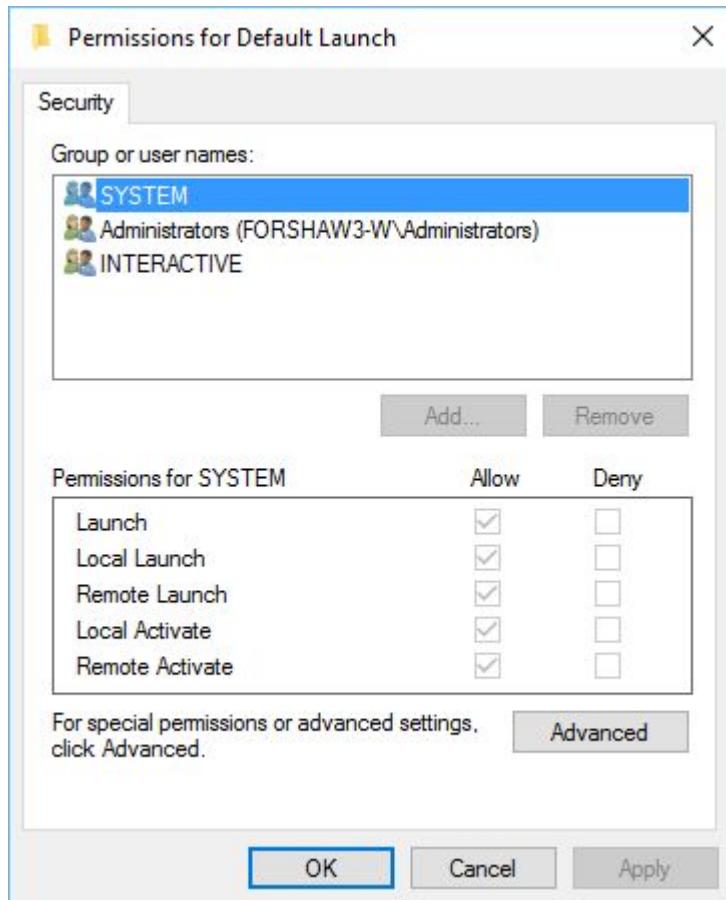


'Everyone' not allowed to launch or activate a new object remotely

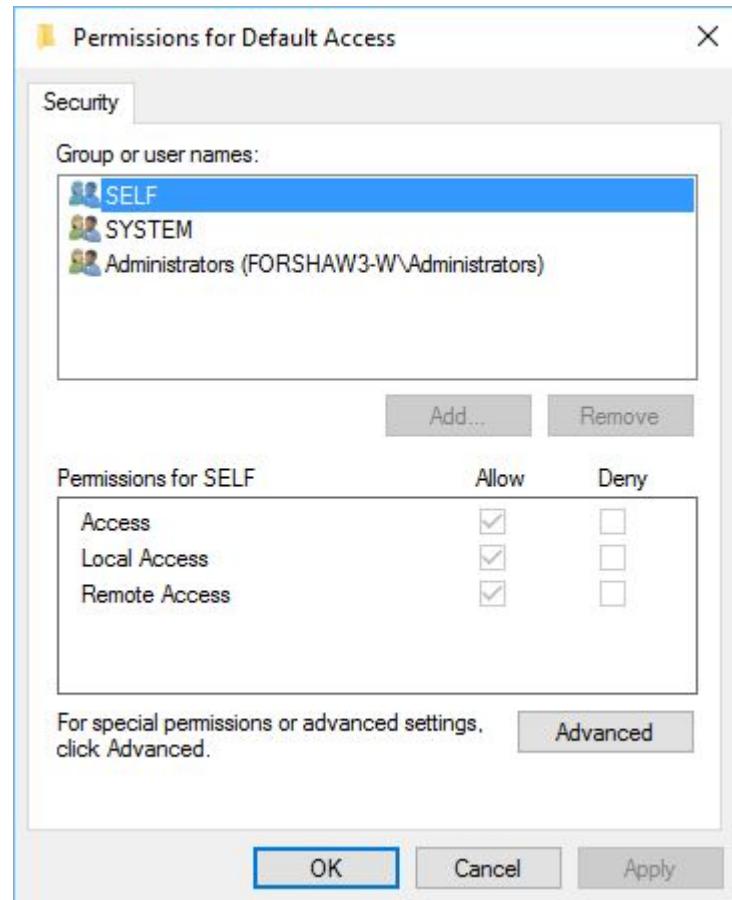


'Everyone' can access an existing object remotely

COM Security



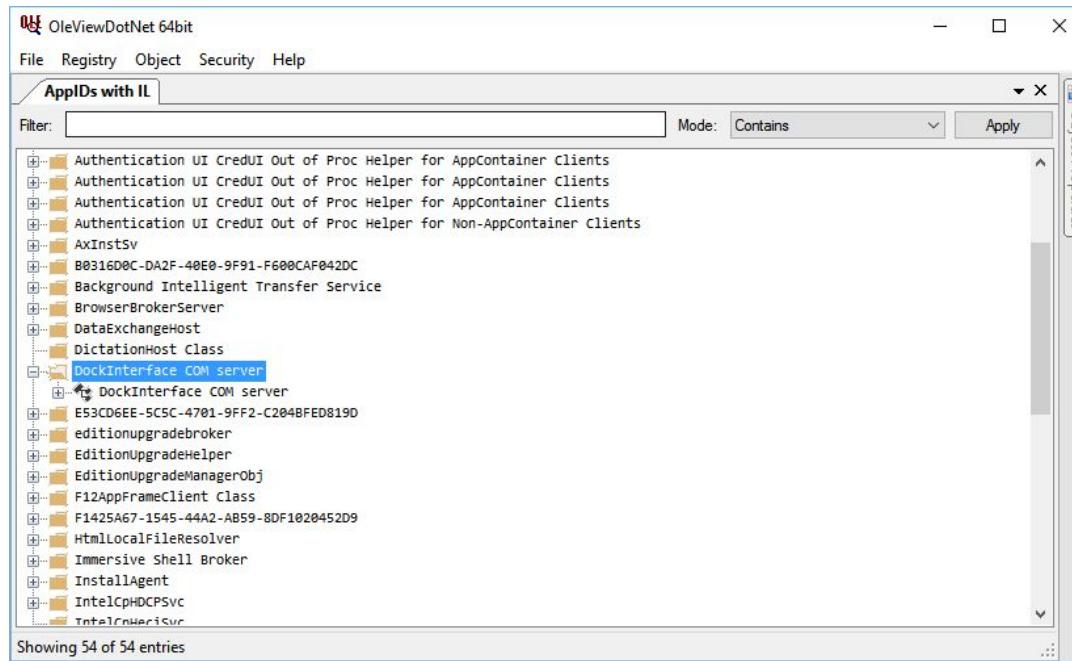
Launch = Create a new instance of the server.
Activate = Create new object on existing server.
Enforced in RPCSS



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

Integrity Levels

- Launching/Activating an OOP COM object from a Low IL needs an explicit entry in the SD for that IL.
- Accessing an object however DOES NOT need an IL label in the Access SD.



Security Through CoInitializeSecurity

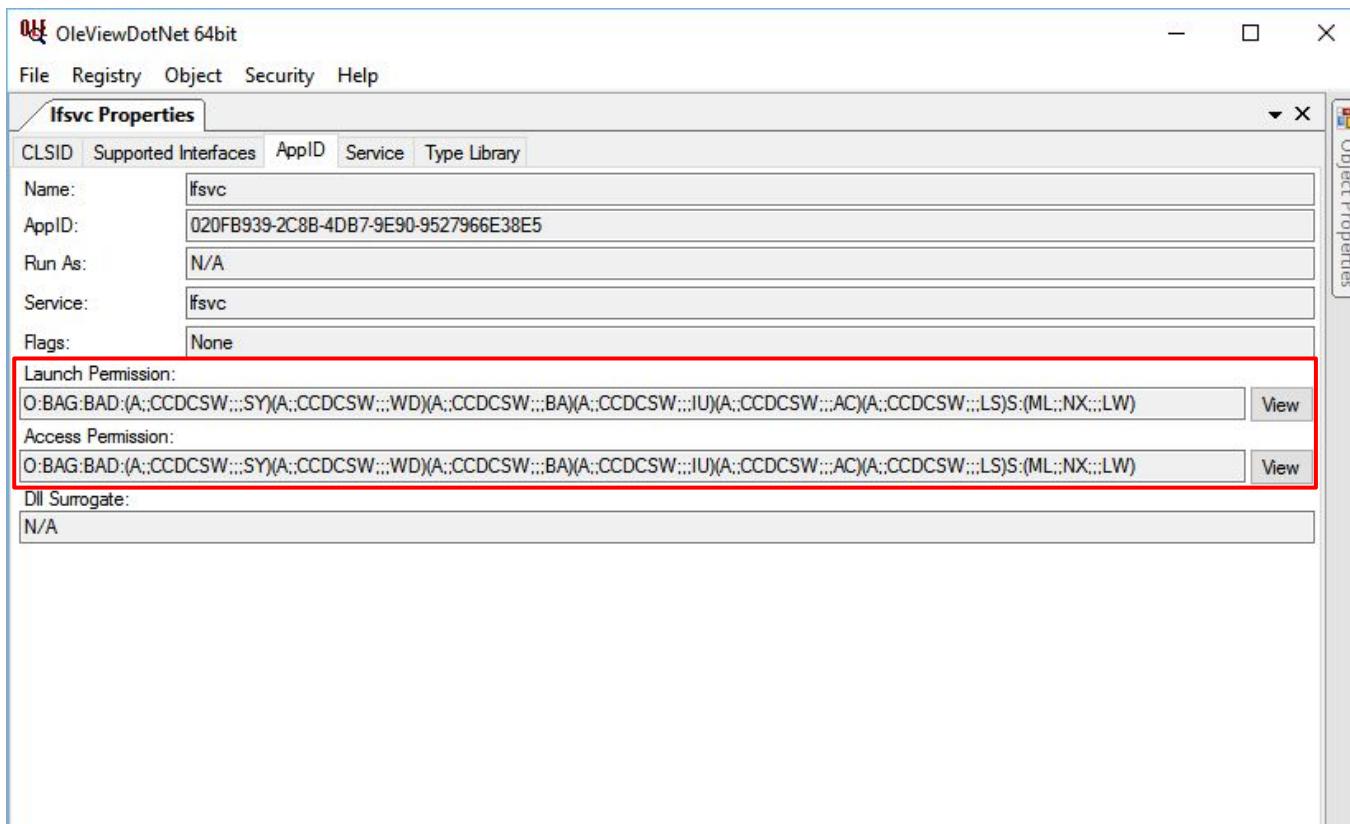
```
HRESULT CoInitializeSecurity( PSECURITY_DESCRIPTOR pSecDesc,  
    LONG cAuthSvc,  
    SOLE_AUTHENTICATION_SERVICE * asAuthSvc,  
    void * pReserved1,  
    DWORD dwAuthnLevel,  
    DWORD dwImplLevel,  
    void * pAuthList,  
    DWORD dwCapabilities,  
    void * pReserved3  
);
```

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.

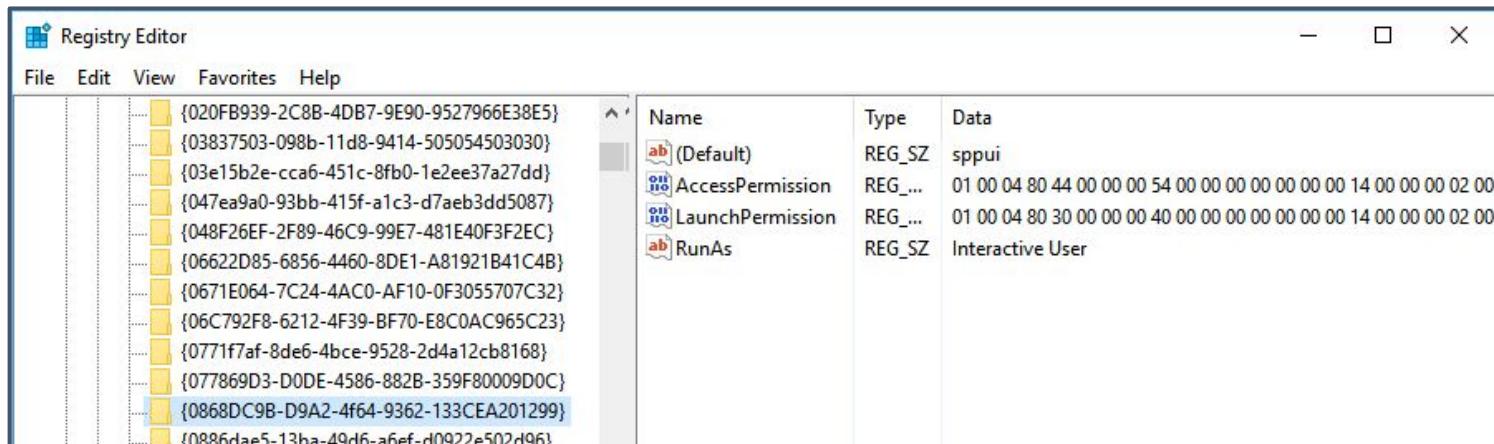
Security Through AppID

- Launch/Activation and Access Security can be specified through the AppID registration for the Class

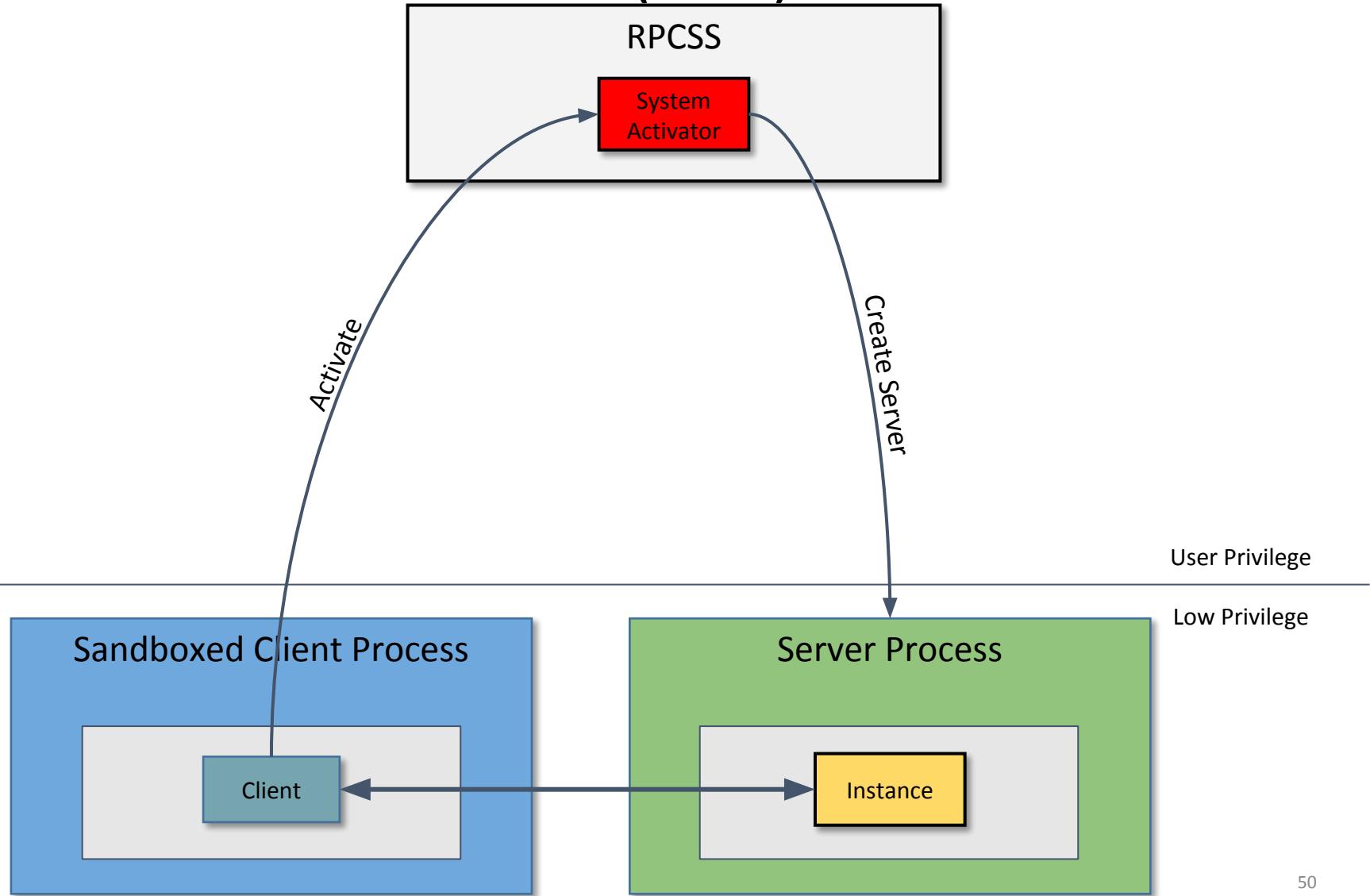


Application IDs

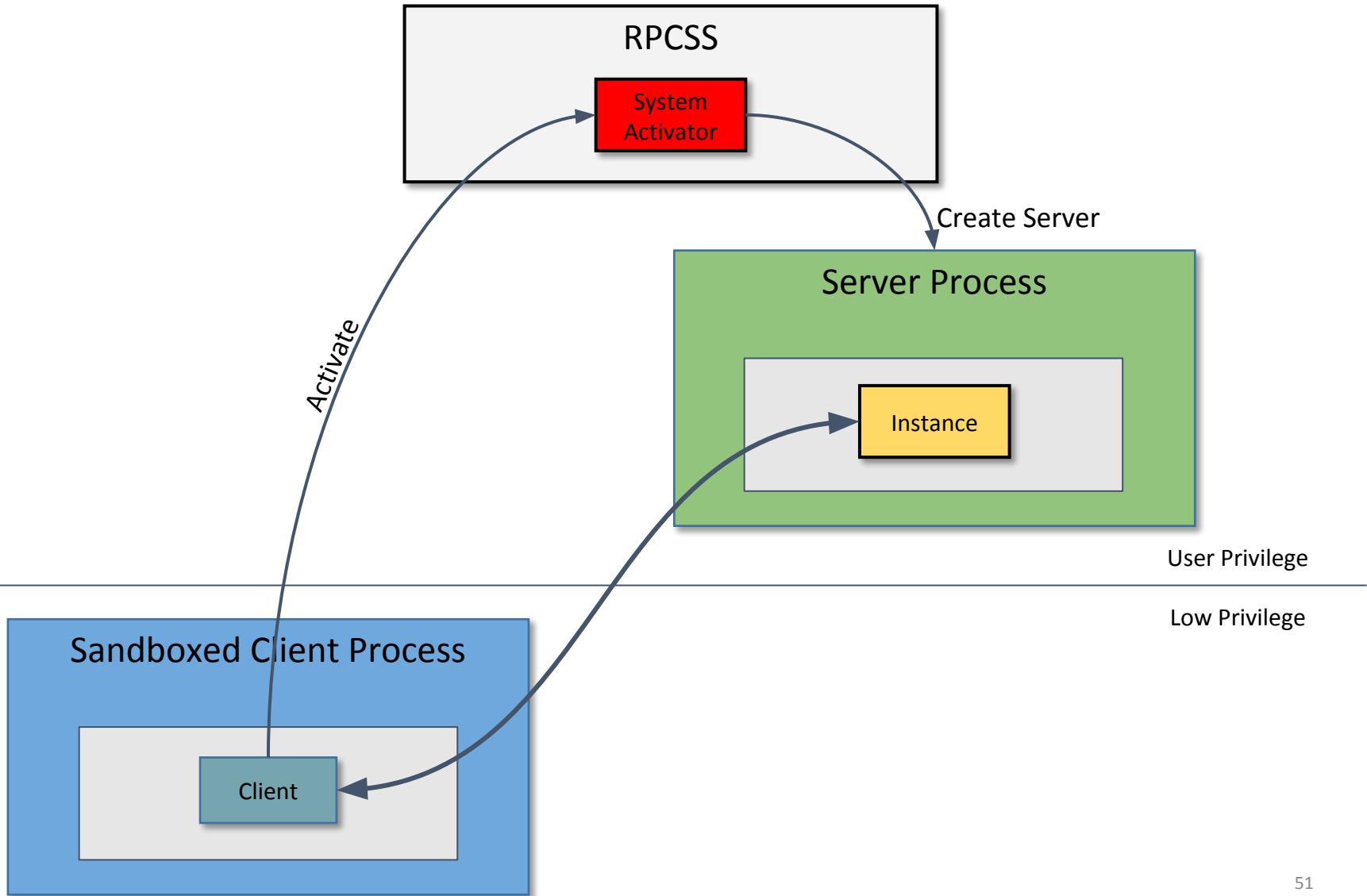
- AppIDs can configure more than just Launch/Access security
- Used to specify a number of features:
 - Allows you to specify a DLL Surrogate. This allows you to create in-process DLL servers as OOP local servers
 - Specify the object is hosted in a Windows service rather than a separate executable
 - Specify running as interactive user.



Activate as Activator (AAA)



RunAs Interactive User

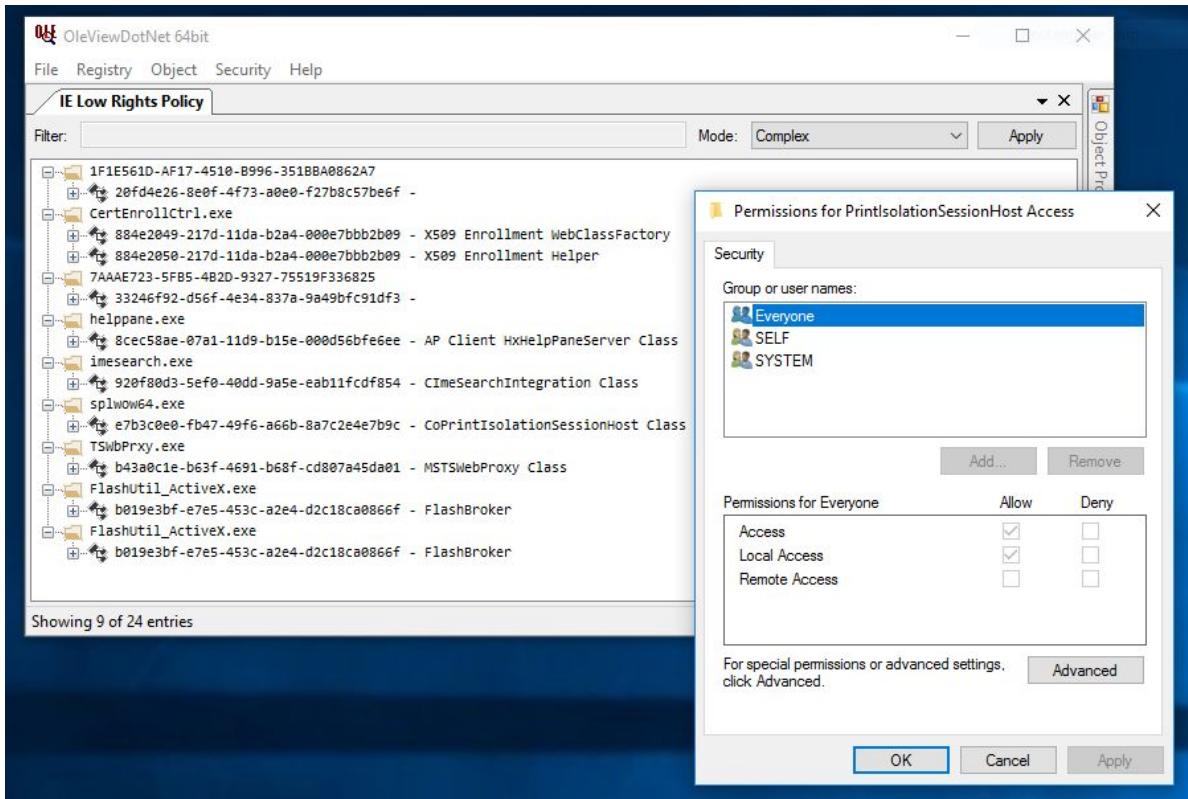




Enumerating Attack Surface and Reverse Engineering

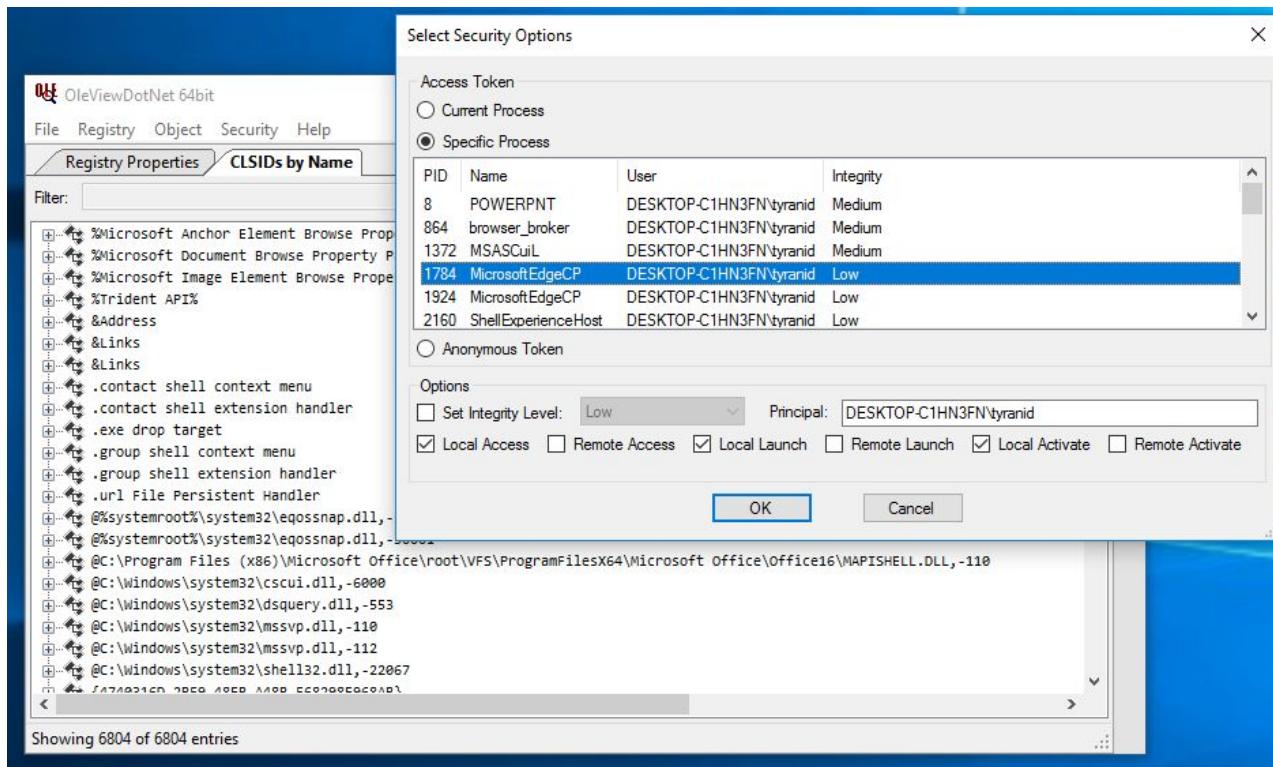
Accessible Objects from a Sandbox

- There's two ways to get a reference to a COM object
 - Activate via a broker (such as IE elevation policy)



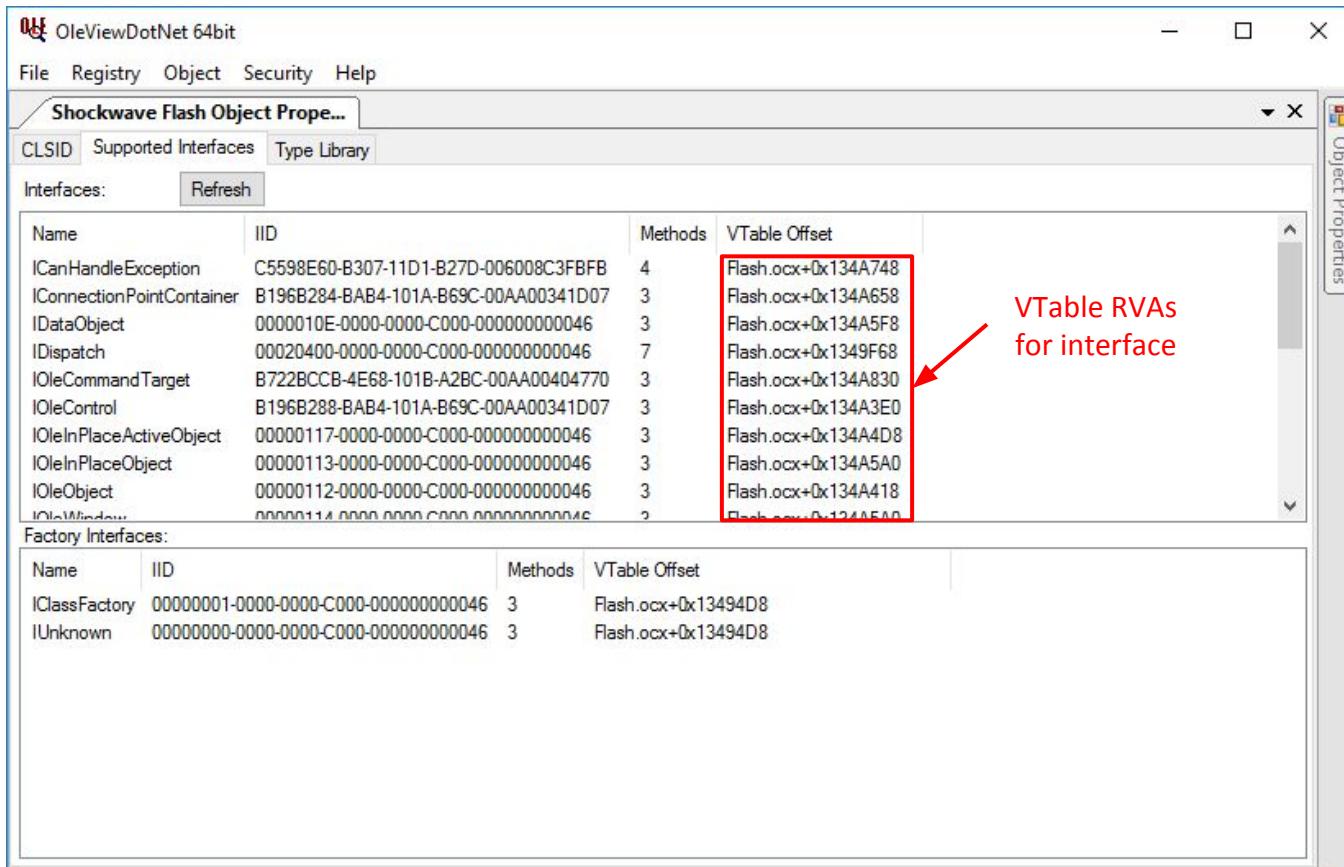
Accessible Objects from a Sandbox

- There's two ways to get a reference to a COM object
 - Activate via a broker (such as IE elevation policy)
 - Launch/Activate one directly



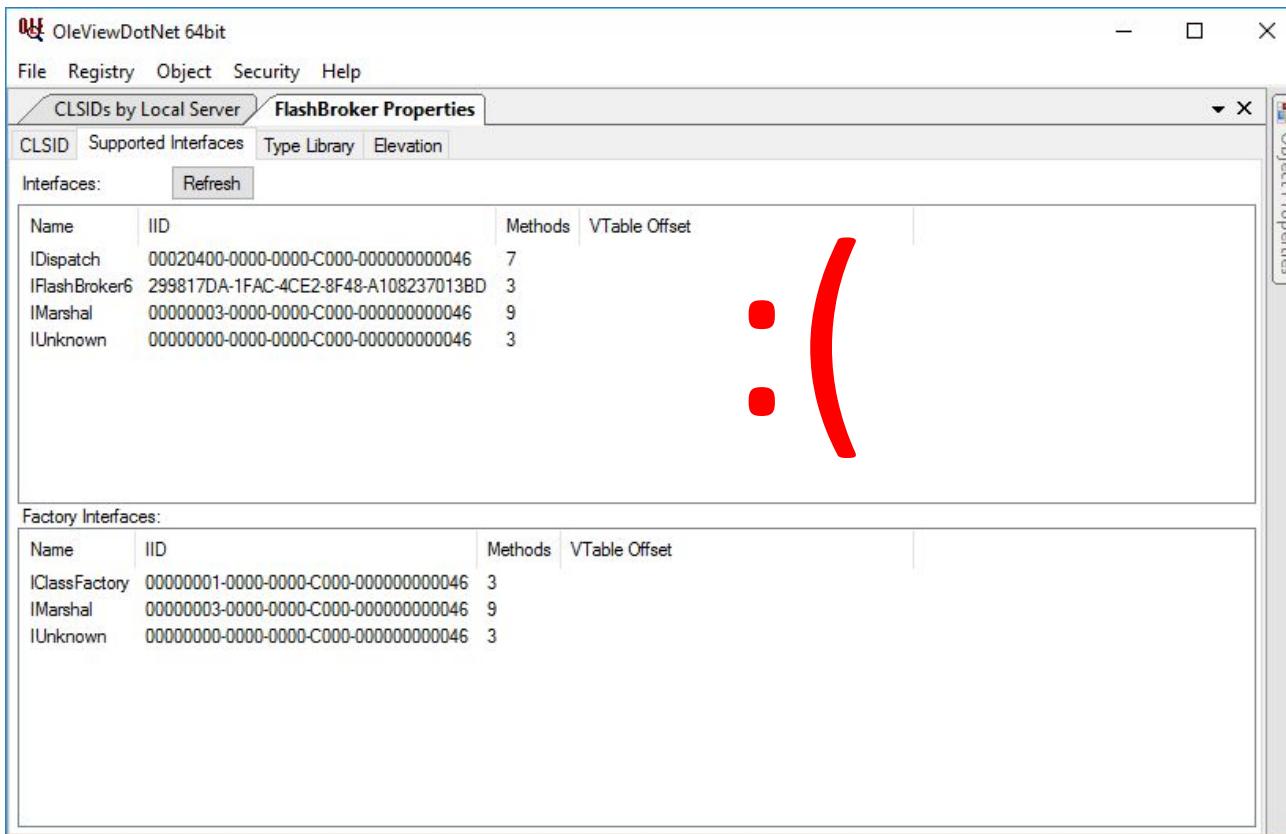
Reverse Engineering COM Components

- In-process you can just inspect the vtable address.



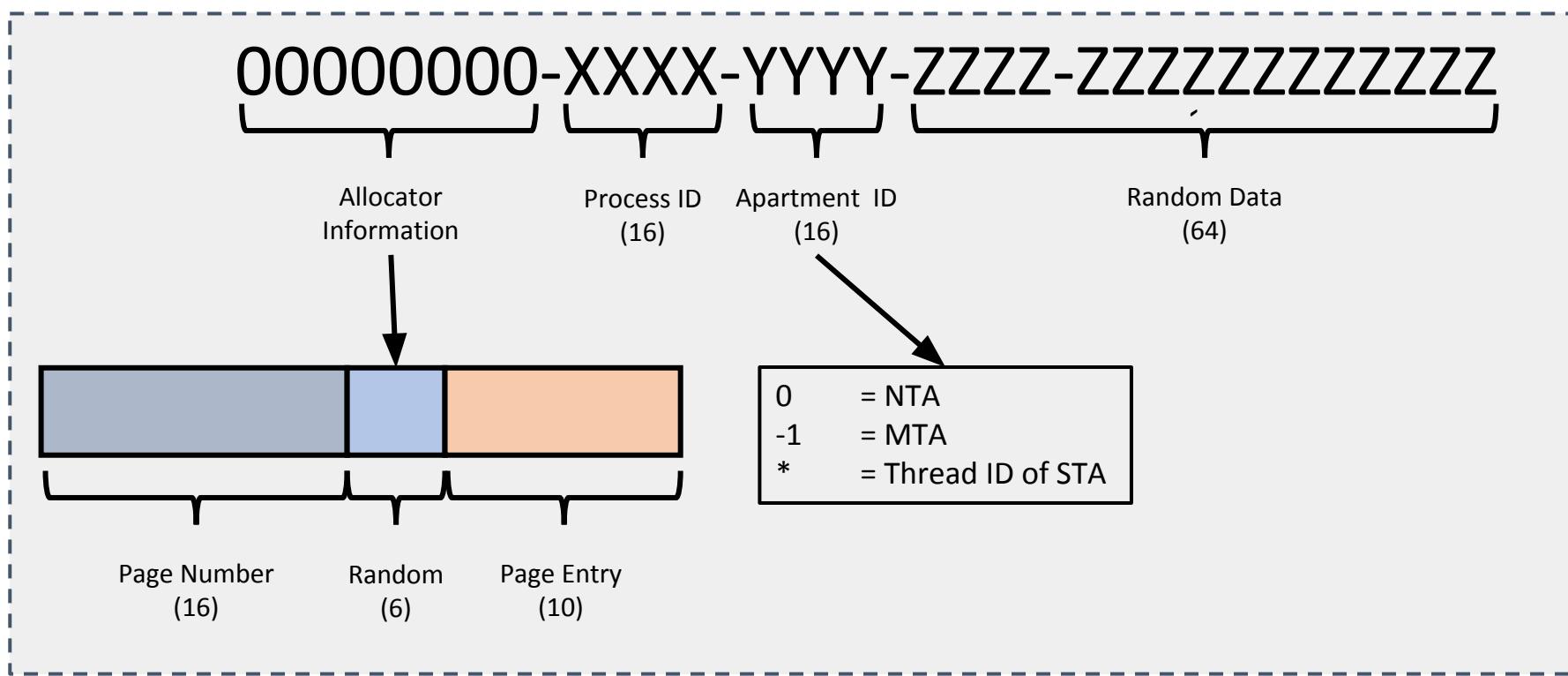
Reverse Engineering COM Components

- No such luck for OOP objects.

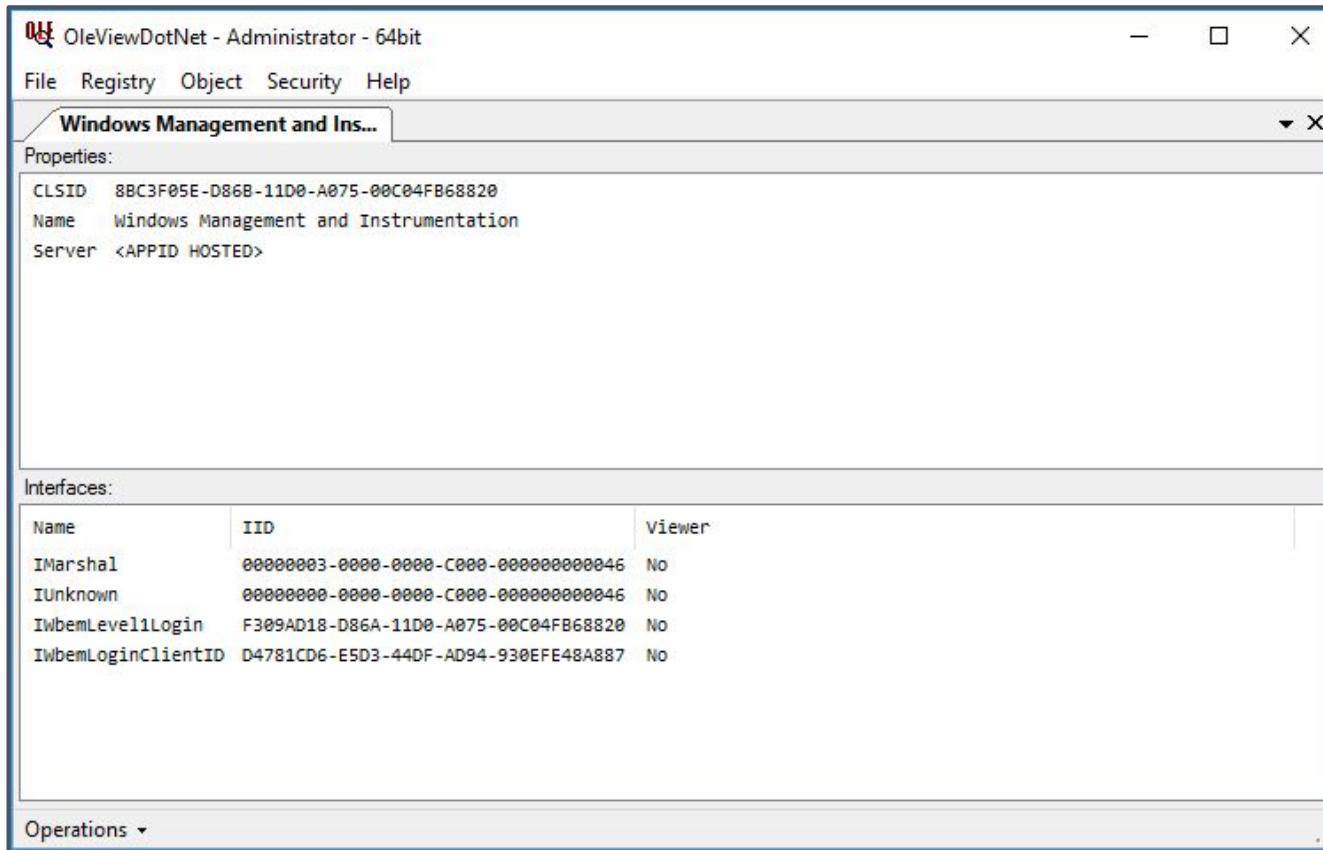


IPID Structure

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



Tracking Down OOP VTable



Tracking Down OOP VTable

The image shows two windows of the OleViewDotNet application. The top window is titled "Windows Management and Instr..." and displays properties for a service named "Windows Management and Instrumentation". The bottom window is titled "Marshal Viewer - Standard" and provides detailed information about the "IMarshal" interface.

Properties:

- CLSID: 8BC3F05E-D86B-11D0-A075-00C04FB68820
- Name: Windows Management and Instrumentation
- Server: <APPID HOSTED>

Interfaces:

Name	IID	Marshal	Viewer
IMarshal	00000003-0000-0000-C000-000000000046	No	
IUnknown	00000000-0000-0000-C000-000000000046	No	
IWbemLevel1Login	F309AD18-D86A-11D0-A075-00C04FB68820	No	
IWbemLoginClientID	D4781CD6-E5D3-44DF-AD94-930EFE48A887	No	

Marshal Viewer - Standard

OBJREF Type:	Standard	IID:	F309AD18-D86A-11D0-A075-00C04FB68820	IID Name:	IWbemLevel1Login
Standard Flags:	0x0	Public Refs:	1	OID:	0x2DA8A0652473E24F
OXID:	0x4DDFA1BDC588A3DB	Apartment:	NTA	Process Name:	svchost
IPID:	0001482F-01A8-0000-F9F2-3464FAED95DA	String Bindings:		Security Bindings:	
Process ID:	424	Tower ID	Network Address	Authentication Service	Principal Name
		Tcp	DESKTOP-C1HN3FN	GSS_Negotiate	
		Tcp	10.0.2.15	NegoExtender	
		Tcp	192.168.56.103	GSS_Kerberos	
				WinNT	
				22	
				PKU2U	
				GSS_SChannel	

A red arrow points from the IPID value in the "String Bindings:" section of the "Marshal Viewer" window to the "PID from IPID" text at the bottom right of the image.

Tracking Down OOP VTable

OleViewDotNet - Administrator - 64bit

File Registry Object Security Help

Windows Management and Instr... Marshal Viewer - Standard COM Processes

Filter: Mode: Contains Apply

Find IPID in Process List

IPID	IID
0001E822-01A8-1860-201F-388915ABA257	INotifyNetworkListManagerEvents
0001DC25-01A8-0C10-B9FD-96998B9D361D	INotifyNetworkInterfaceEvents
0001E826-01A8-0C10-5A1D-914532F13B4C	IConnectionProvider
0001C428-01A8-0000-A69E-AC93C53B7739	ILocalSystemActivator
00017429-01A8-0000-8C3C-18452FEFBC21	IApplicationStateChangeHandler
0001042A-01A8-FFFF-D514-DE4E8E21260C	IBackgroundAccessStateChangeHandler
0001A42D-01A8-0000-7C28-E4B4A7FC544F	INotifyNetworkInterfaceEvents
00016C2E-01A8-FFFF-8FEC-9ADA33A7C4C5	IWpnAppEndpoint
0001482F-01A8-0000-F9F2-3464FAED95DA	IWbemLevel1Login
0001F030-01A8-0C10-F6AD-C870F3AACB09	INotifyNetworkListManagerEvents
00019431-01A8-FFFF-F8F1-A726FFE8C121	IConnectionManagerCallbacks2
00029400-01A8-1860-4EBB-2D48E0D39C90	INotifyNetworkEvents
00028401-01A8-1860-7544-6CFF90C6CC0B	INotifyNetworkInterfaceEvents
00028402-01A8-0000-EEFA-F3265F191235	INotifyNetworkEvents
0002F003-01A8-0000-368F-A5B8C0221761	IUnknown
0002DC04-01A8-0000-2DC3-02A1FAC540A8	INotifyNetworkListManagerEvents
00020008-01A8-0000-82DC-CA1895D39166	INetworkEvents
0002BC0A-01A8-0000-E591-02EAAE6261D1	INotifyNetworkInterfaceEvents
0002400C-01A8-0C10-2BDB-44DEC0901080	IUPnPDeviceFinder
0002BC0E-01A8-0000-DD8C-B15F7334B223	INotifyNetworkGlobalCostEvents
00029010-01A8-0000-5699-106337F32C27	INotifyNetworkGlobalCostEvents

Showing 29 of 29 entries

Tracking Down OOP VTable

The screenshot shows two windows of the OleViewDotNet application. The main window displays a list of COM objects with their IPID and IID. A specific entry for IPID: 0001482F-01A8-0000-F9F2-3464FAED95DA is selected. A red arrow points from the 'Properties' button in the list view to the 'Properties' tab in the details view. Another red arrow points from the 'VTable' field in the properties tab to the value 'wbemcore+0x0A448', which is highlighted in blue.

Properties

IPID:	0001482F-01A8-0000-F9F2-3464FAED95DA
IID:	F309AD18-D86A-11D0-A075-00C04FB68820
Flags:	IPIDF_SERVERENTRY
Interface:	0x1AFD765910
Stub:	0x1AF52B64F0
OXID:	C588A3DB-A1BD-4DDF-6117-1CF7D8ED199D
References:	Strong: 5, Weak: 0, Private: 0
PID:	424
STA HWND:	0x0
VTable:	wbemcore+0x0A448
VTable:	wbemsrvc+0x7650

VTable RVA

VTable Reverse Engineering

```
.rdata:00000001800D79E0 ; const CJobManagerExternal::`vtable'
.rdata:00000001800D79E0 ??_7CJobManagerExternal@6B@ dq offset ?QueryInterface@?$RuntimeClass@U?$InterfaceList@U
.rdata:00000001800D79E0
.rdata:00000001800D79F0
.rdata:00000001800D79F0
.rdata:00000001800D79F8
.rdata:00000001800D79F8
.rdata:00000001800D7A00
.rdata:00000001800D7A00
.rdata:00000001800D7A08
.rdata:00000001800D7A08
.rdata:00000001800D7A10
.rdata:00000001800D7A10
.rdata:00000001800D7A18
.rdata:00000001800D7A18
```

IUnknown

Interface Specific

QueryInterface Implementations

```
HRESULT QueryInterface (REFIID riid, void** ppv) {
    if (riid == IID_IUnknown || riid == IID_Interface1)
    {
        *ppv = this;
    } else if (riid == IID_Interface2) {
        *ppv = static_cast<Interface2*>(this);
    } else {
        return E_NOINTERFACE;
    }
    ((IUnknown*)*ppv)->AddRef();
    return S_OK;
}
```

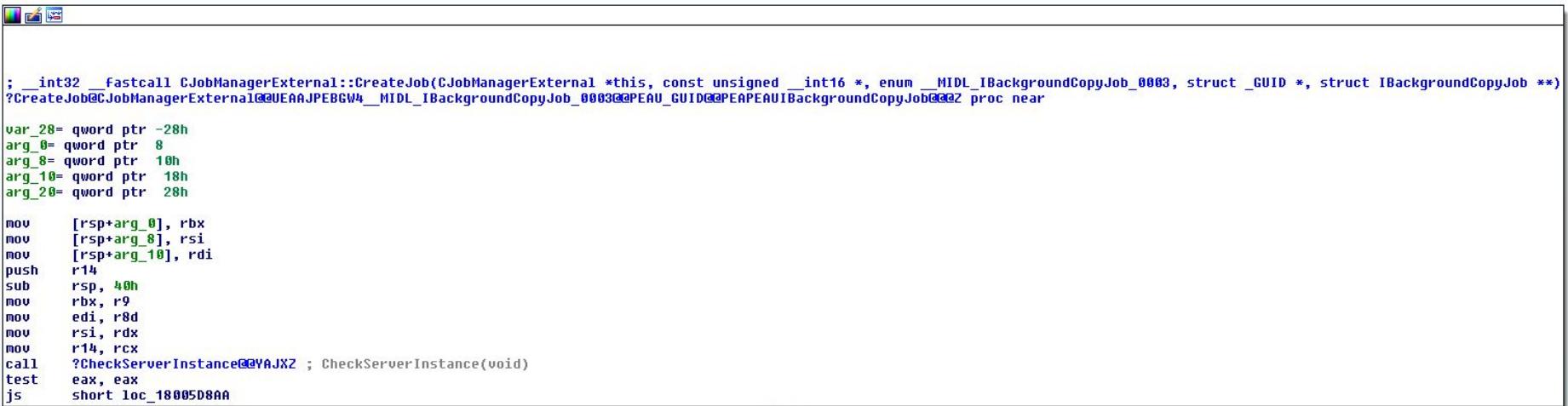
Explicit Implementation

```
HRESULT QueryInterface (REFIID riid, void** ppv) {
    QITAB qit[] = {{ &IID_Interface1, 0 },
                   { &IID_Interface2, 8 } { NULL, 0 }};
    return QISearch(this, qit, riid, ppv);
}
```

Using QISearch Helper

Interface Information - Public Symbols

- Most Windows components come with public symbols available
- Most COM code written in C++
- So, use C++ managed names to recover some parameter information



The screenshot shows a debugger interface with assembly code. The code is for a function named `CJobManagerExternal::CreateJob`. The assembly instructions include `mov`, `push`, `sub`, `mov`, `call`, and `test`. The debugger highlights memory addresses in blue, such as `0EAAJPEBGW4` and `0003@0EAU_GUID@0EAPEAUBackgroundCopyJob@00Z`. Register names like `arg_0`, `arg_8`, `arg_10`, `arg_18`, and `arg_20` are shown in green. The assembly code ends with a jump to `short loc_18005D8AA`.

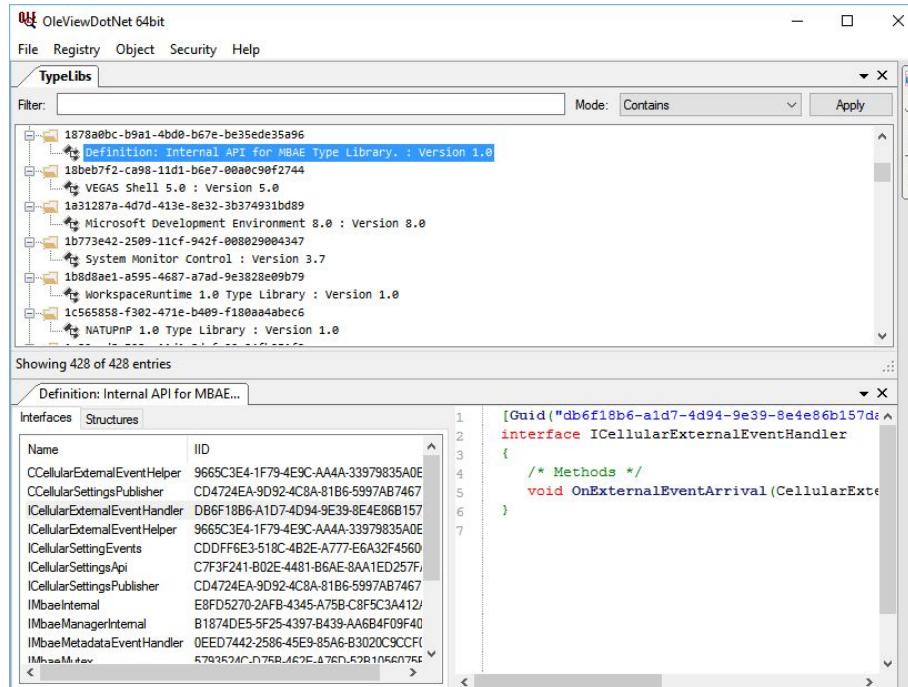
```
; __int32 __fastcall CJobManagerExternal::CreateJob(CJobManagerExternal *this, const unsigned __int16 *, enum __MIDL_IBackgroundCopyJob_0003, struct _GUID *, struct IBackgroundCopyJob **)
?CreateJob@CJobManagerExternal@@UEAAJPEBGW4__MIDL_IBackgroundCopyJob_0003@@PEAU_GUID@@PEAPEAUBackgroundCopyJob@00Z proc near

var_28= qword ptr -28h
arg_0= qword ptr 8
arg_8= qword ptr 10h
arg_10= qword ptr 18h
arg_18= qword ptr 20h
arg_20= qword ptr 28h

mov    [rsp+arg_0], rbx
mov    [rsp+arg_8], rsi
mov    [rsp+arg_10], rdi
push   r14
sub    rsp, 40h
mov    rbx, r9
mov    edi, r8d
mov    rsi, rdx
mov    r14, rcx
call   ?CheckServerInstance@@YAJXZ ; CheckServerInstance(void)
test   eax, eax
js     short loc_18005D8AA
```

Interface Information - Type Libraries

- Some objects have type library information which is a structured format for describing COM interfaces.
- “import” the library into a C++ project with `#import <name.tlb>` preprocessor directive



Interface Information - Proxy/Stub NDR

- Proxies and Stubs contains NDR bytecode which describe how to marshal parameters back and forth.
- Contains information about interfaces and structures

The screenshot shows the OleViewDotNet 64bit application window. The title bar reads "OleViewDotNet 64bit". The menu bar includes File, Registry, Object, Security, Help. The toolbar has tabs for "Proxy CLSIDs" and "tpcps.dll", with "Proxy CLSIDs" currently selected. Below the toolbar are two tabs: "Interfaces" and "Structures", with "Interfaces" selected. A table lists various COM interfaces with their GUIDs and IIDs. On the right side of the interface, there is a large block of C code representing the NDR bytecode for the IRecoManager interface.

Name	IID
IInkManager	A5558507-9B96-46BA-94ED-982E684A9A
IInkObject	B9C4A0C1-16ED-4DC2-B34A-4E830323658
IInkPoint	3776F33D-6BF8-4ADD-9C7E-946AB4A771
IInkStroke	251F1257-1DCB-4A0D-A826-4F9E326FE49
ILattice	946665F9-71E3-447B-A896-3359DA41153
IPenService	9E358D23-02B2-4CCD-9FEE-6B75EE8DD4
IPenServiceSampleCollector	B0EC4A1-FF78-43FA-B06E-EBEAAA1BFC
IPenServiceTablet	5EE46C56-1537-4CC5-8A93-A7ED4D1393
IRecoAsyncResults	250CEF9C-121F-493D-ADCD-7A2A6823C0
IRecoContext	E6DAB875-75AF-4C8A-9665-2B6A44DD0F
IRecognizer	3A182AD6-596A-4070-A574-73941817B67
IRecoManager	C2E8F101-5D03-42EE-B90A-35255783103
IRecoWordList	37ADC645-ACE6-4A31-B6A1-FD1F4EF480
IRenderingContext	4E6B4F16-5A0C-4815-9AA2-DE231F5AAA
IRenderInk	538A9C7B-858A-4FF1-9769-62B6D74993C
IStrokeGeometry	11F962C5-242E-4D4D-B205-0F3AB3562F
IStrokeSet	2080FF4F-297F-4F66-AA83-CACA65F6721
ITablet	1CB2EFC3-ABC7-4172-8FCB-3BC9CB93E2
ITablet2	C247F616-BBEB-406A-AED3-F75E656599
ITablet3	AC0E3951-0A2F-448E-88D0-49DA0C45341
ITabletContext	45AAAF04-9D6F-41AF-8FD1-FCD6D4B2F

```
[Guid("c2e8f101-5d03-42ee-b90a-352557831039")]
interface IRecoManager : IUnknown {
    HRESULT Proc3(/* Stack Offset: 8 */ [In, Out] int* p0,
    HRESULT Proc4(/* Stack Offset: 8 */ [In] struct Struct,
    HRESULT Proc5(/* Stack Offset: 8 */ [In] GUID* p0, /* Stack Offset: 8 */ [Out] GUID* p0,
    HRESULT Proc6(/* Stack Offset: 8 */ [Out] GUID* p0, /* Stack Offset: 8 */ [In] GUID* p0,
    HRESULT Proc7(/* Stack Offset: 8 */ [In] GUID* p0, /* Stack Offset: 8 */ [In] GUID* p0,
    HRESULT Proc8(/* Stack Offset: 8 */ [In] GUID* p0, /* Stack Offset: 8 */ [Out] IRecoWordLis
```

A close-up photograph of a shield-shaped stink bug with prominent red and black horizontal stripes on its back. The insect is positioned centrally, facing towards the bottom left. It is resting on a large, broad green leaf with visible veins.

Bugs and "Features"

Activation Properties In- SPD

```
struct SpecialPropertiesData {  
    unsigned long dwSessionId; // Choosing a Session ID?  
    long fRemoteThisSessionId;  
    long fClientImpersonating;  
    long fPartitionIDPresent;  
    DWORD dwDefaultAuthnLvl;  
    GUID guidPartition;  
    DWORD dwPRTFlags;  
    DWORD dwOrigClсх;  
    DWORD dwFlags; // UAC Related Stuff  
    DWORD dwPid;  
    unsigned __int64 hwnd;  
    DWORD ulServiceId;  
    DWORD Reserved [ 4 ];  
};
```

Choosing a Session ID?

UAC Related Stuff

```
enum SPD_FLAGS {  
    SPD_FLAG_USE_CONSOLE_SESSION,  
    SPD_FLAG_USE_DEFAULT_AUTHN_LVL,  
    SPD_FLAG_USE_SERVER_PID,  
    SPD_FLAG_USE_LUA_LEVEL_ADMIN,  
    SPD_FLAG_COAUTH_USER_IS_NULL,  
    SPD_FLAG_COAUTH_DOMAIN_IS_NULL,  
    SPD_FLAG_COAUTH_PWD_IS_NULL,  
    SPD_FLAG_USE_LUA_LEVEL_HIGHEST  
};
```

Session and Elevation Monikers

Session-to-Session Activation with a Session Moniker

Session-to-session activation (also called cross-session activation) allows a client process to start (activate) a local server process on a specified session. This feature is available for applications that are configured to run in the security context of the interactive user, also known as the "RunAs Interactive User" object activation mode. For more information about security contexts, see [The Client's Security Context](#).

Distributed COM (DCOM) enables object activation on a per-session basis by using a system-supplied [Session Moniker](#). Other system-supplied monikers include [file monikers](#), [item monikers](#), generic [composite monikers](#), anti-monikers, pointer monikers, and [URL monikers](#).

The COM Elevation Moniker

The COM elevation moniker allows applications that are running under user account control (UAC) to activate COM classes with elevated privileges. For more information, see [Focus on Least Privilege](#).

When to Use the Elevation Moniker

The elevation moniker is used to activate a COM class to accomplish a specific and limited function that requires elevated privileges, such as changing the system date and time.

Elevation requires participation from both a COM class and its client. The COM class must be configured to support elevation by annotating its registry entry, as described in the Requirements section. The COM client must request elevation by using the elevation moniker.

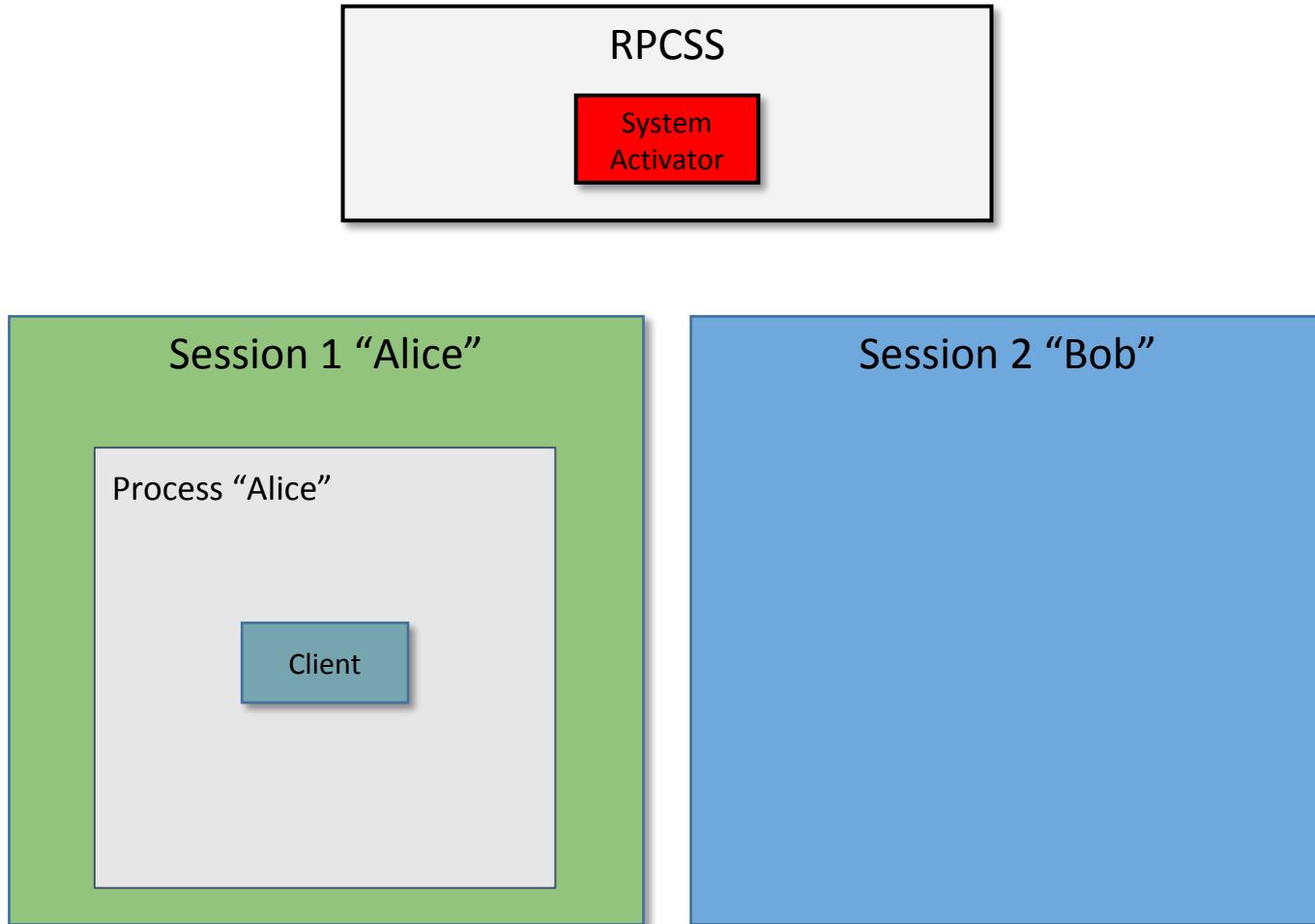
Monikers and Binding

- Monikers are string representation of COM objects.
- Pass string to CoGetObject to “Bind” to the underlying object.

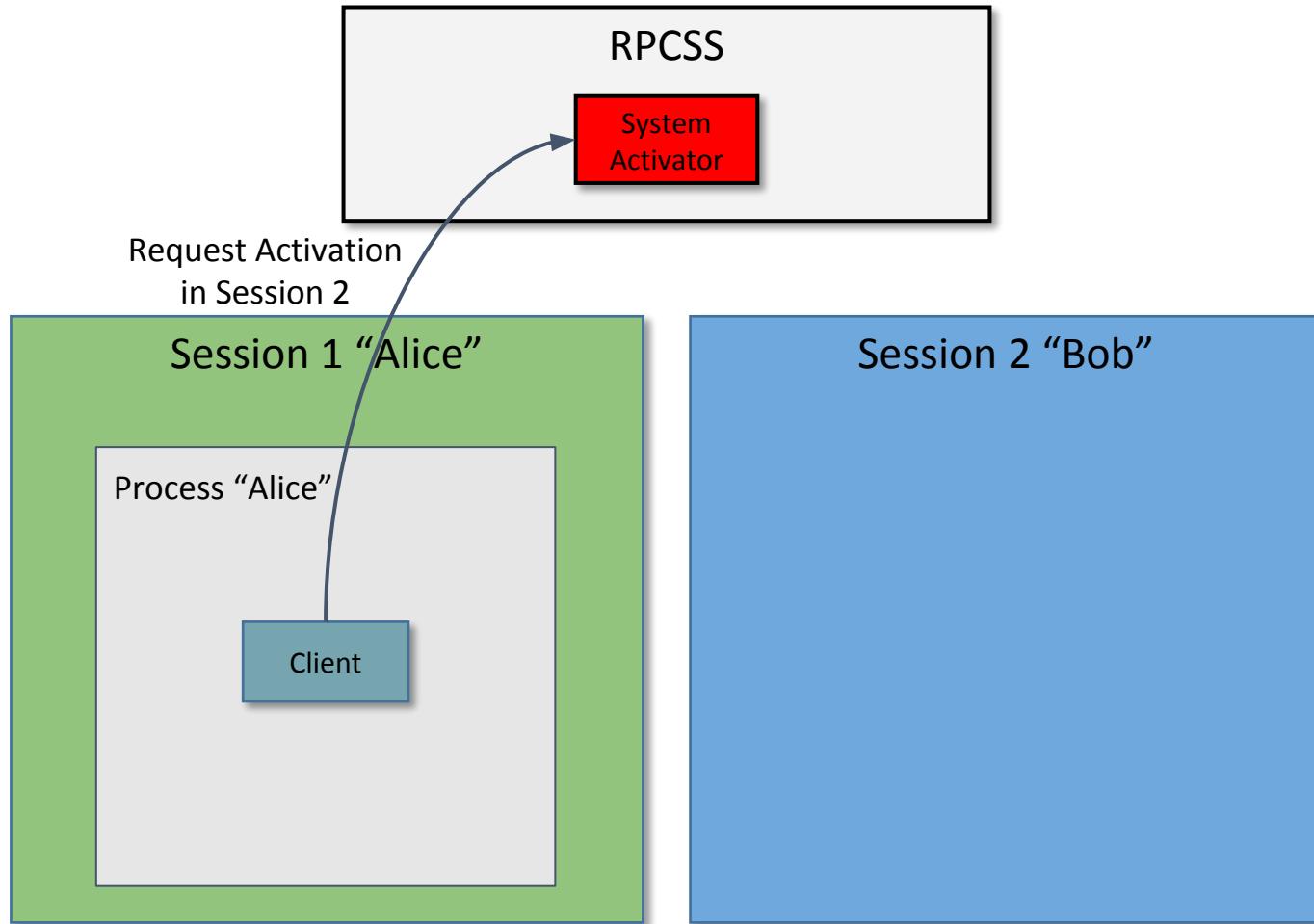
```
HRESULT CoCreateInstanceInSession (DWORD session,
    REFCLSID rclsid, REFIID riid, void ** ppv) {
    BIND_OPTS3 bo = {};
    WCHAR wszCLSID [50];
    WCHAR wszMonikerName [300];

    StringFromGUID2 (rclsid, wszCLSID, _countof (wszCLSID));
    StringCchPrintf (wszMonikerName, _countof (wszMonikerName),
        L"session:%d!new:%s", session, wszCLSID);
    bo.cbStruct = sizeof(bo);
    bo.dwClassContext = CLSCTX_LOCAL_SERVER;
    return CoGetObject (wszMonikerName, &bo, riid, ppv);
}
```

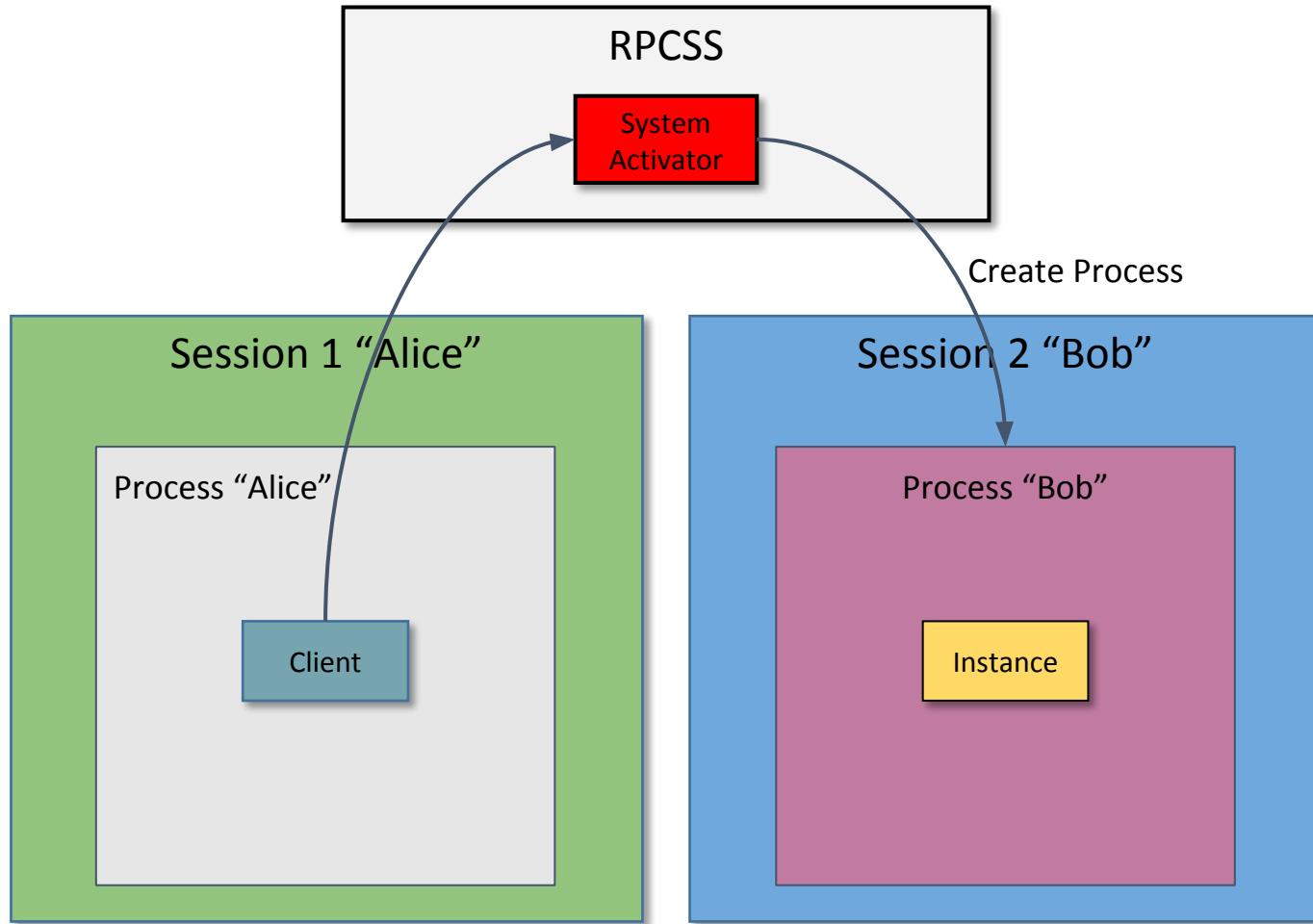
Session Moniker in Action



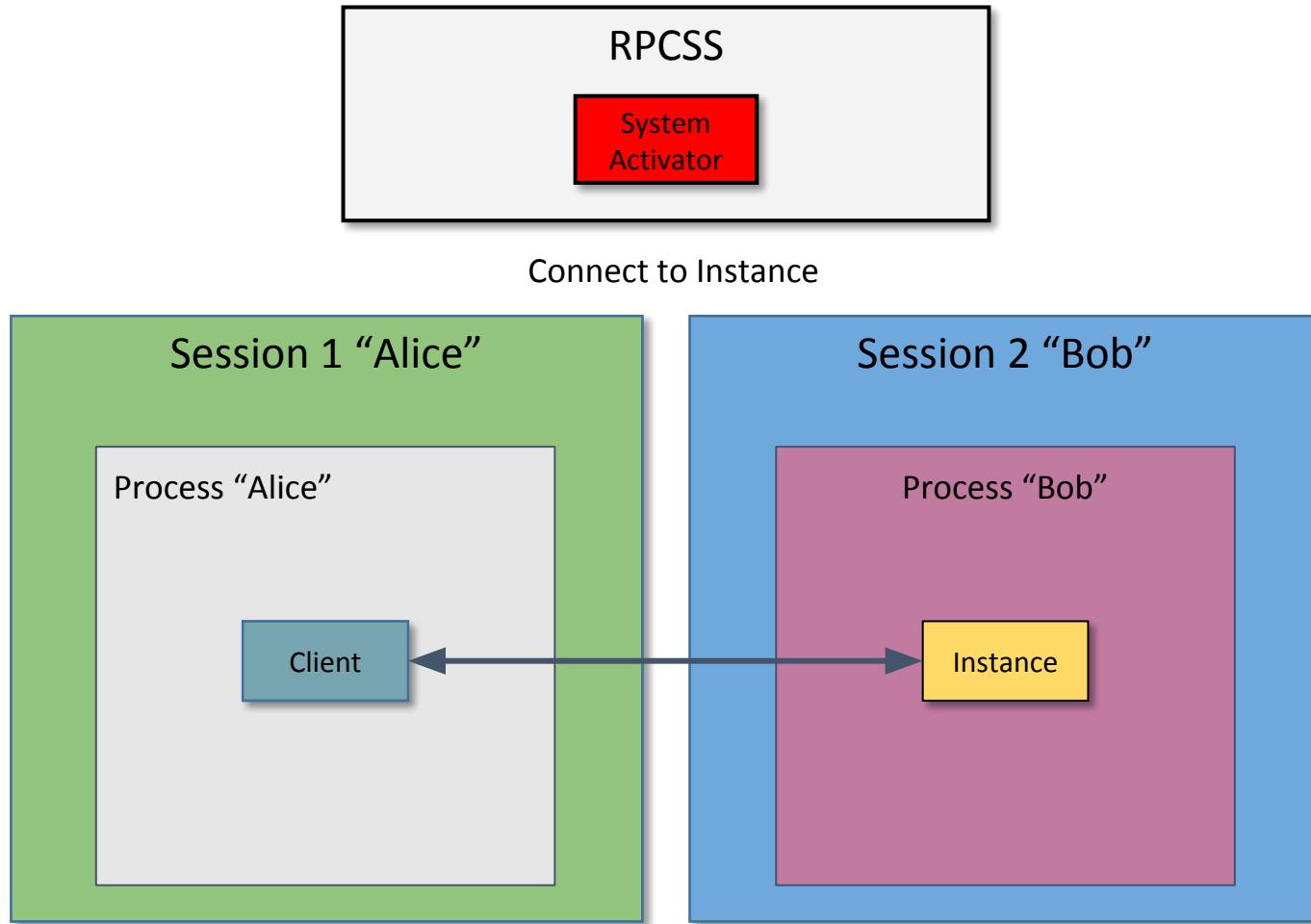
Session Moniker in Action



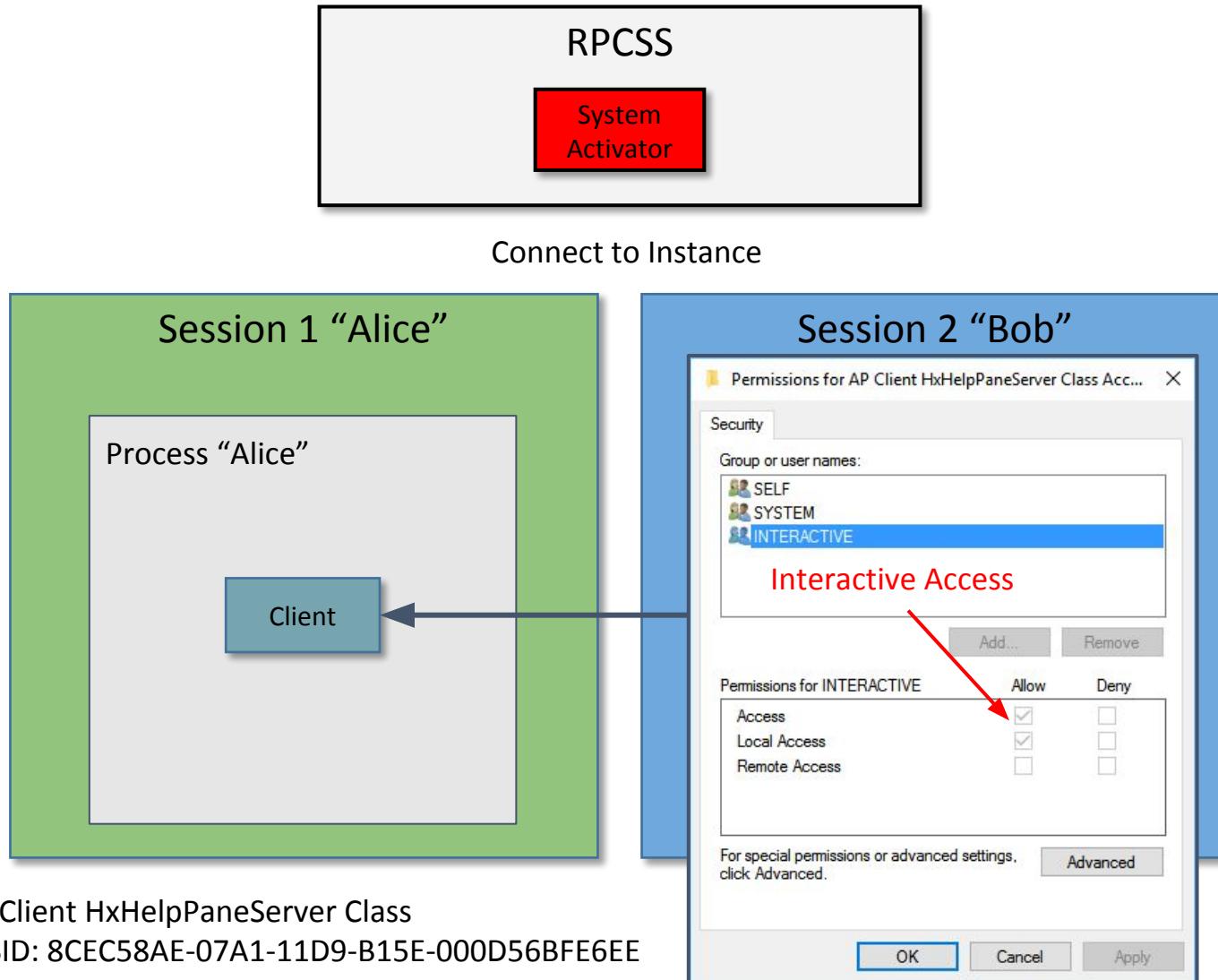
Session Moniker in Action



Session Moniker in Action



Session Moniker in Action



Abusing Information Disclosure

The screenshot shows the OleViewDotNet application interface. On the left, the 'COM Processes' tab is active, displaying a list of processes with their corresponding IPIDs and IIDs. A red arrow points from the highlighted entry in the list to the 'Properties' tab on the right. The 'Properties' tab shows the selected object's CLSID and its interfaces.

COM Processes

- 5324 - ApplicationFrameHost - DESKTOP-C1HN3FN\tyranid
- 5872 - backgroundTaskHost - DESKTOP-C1HN3FN\tyranid
- 864 - browser_broker - DESKTOP-C1HN3FN\tyranid
- 6312 - dllhost - DESKTOP-C1HN3FN\tyranid
- 3684 - explorer - DESKTOP-C1HN3FN\tyranid

Filter: Mode: Contains Apply

Properties:

IPID 0000140b-0e64-0e68-e75c-d4cd455824f1

Properties:

CLSID 00000000-0000-0000-0000-000000000000

Interfaces:

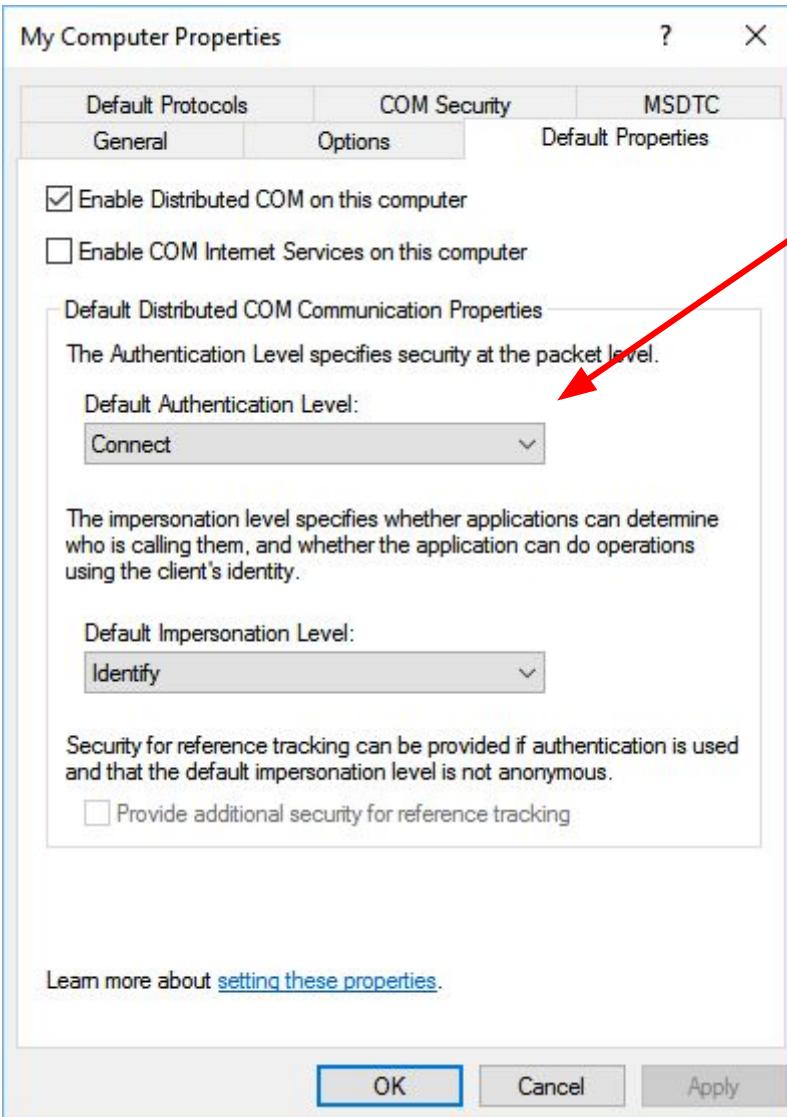
Name	IID	Viewer
ILocalSystemActivator	00000132-0000-0000-C000-000000000046	No
IMarshal	00000003-0000-0000-C000-000000000046	No
ISystemActivator	000001A0-0000-0000-C000-000000000046	No
IUnknown	00000000-0000-0000-C000-000000000046	No

Operations ▾

You Can't Help Loving DCOM



Plain Text Communications?



Defaults to CONNECT
Authentication Level?

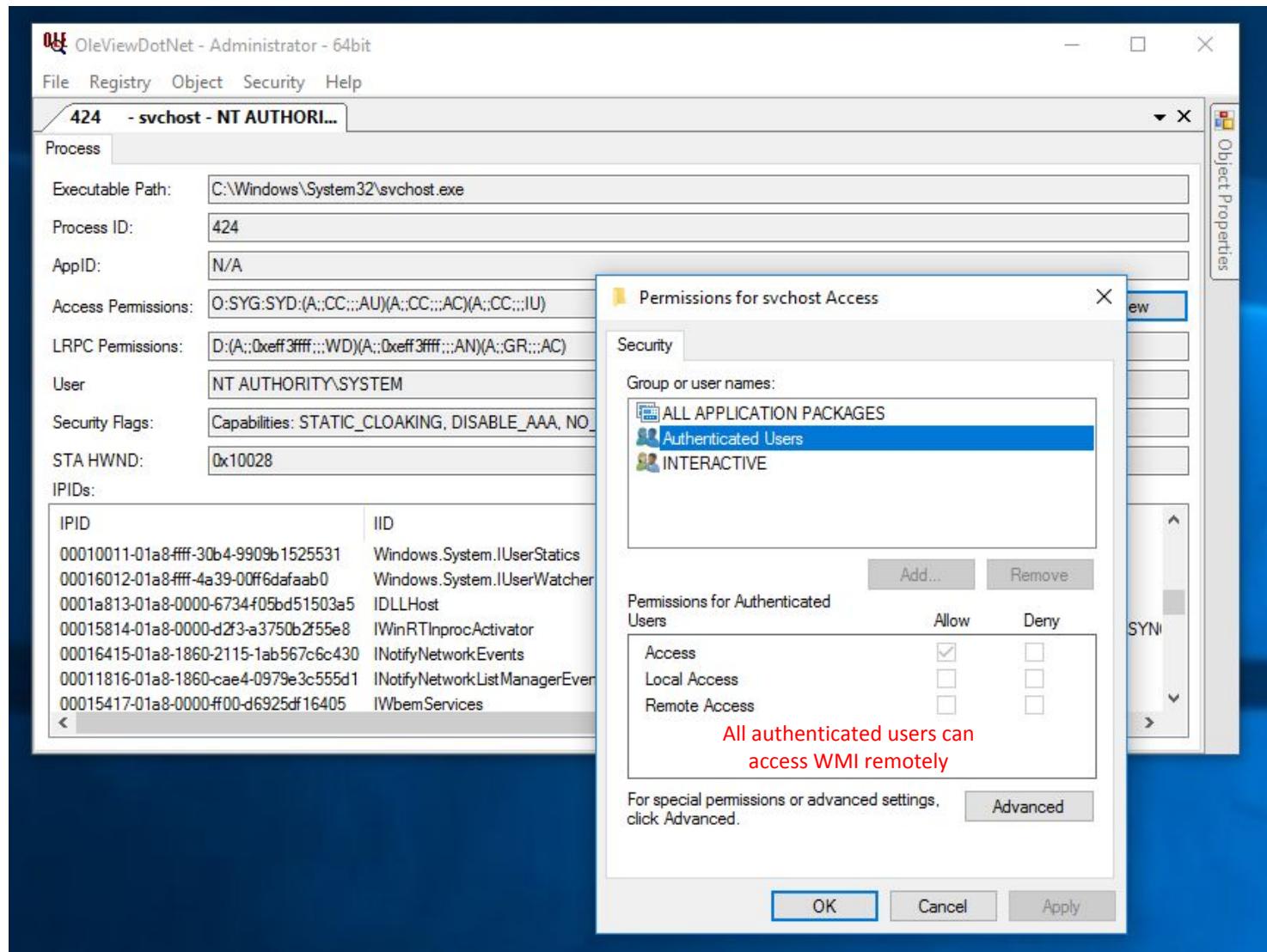
<i>Authentication Level</i>	
CONNECT	Authenticates the credentials of the client and server.
CALL/PKT	Same as CONNECT but also prevents replay attacks.
PKT_INTEGRITY	Same as CALL/PKT but also verifies that none of the data transferred between the client and server has been modified.
PKT_PRIVACY	Same as PKT_INTEGRITY but also ensures that the data transferred can only be seen unencrypted by the client and the server.

Good old Wireshark

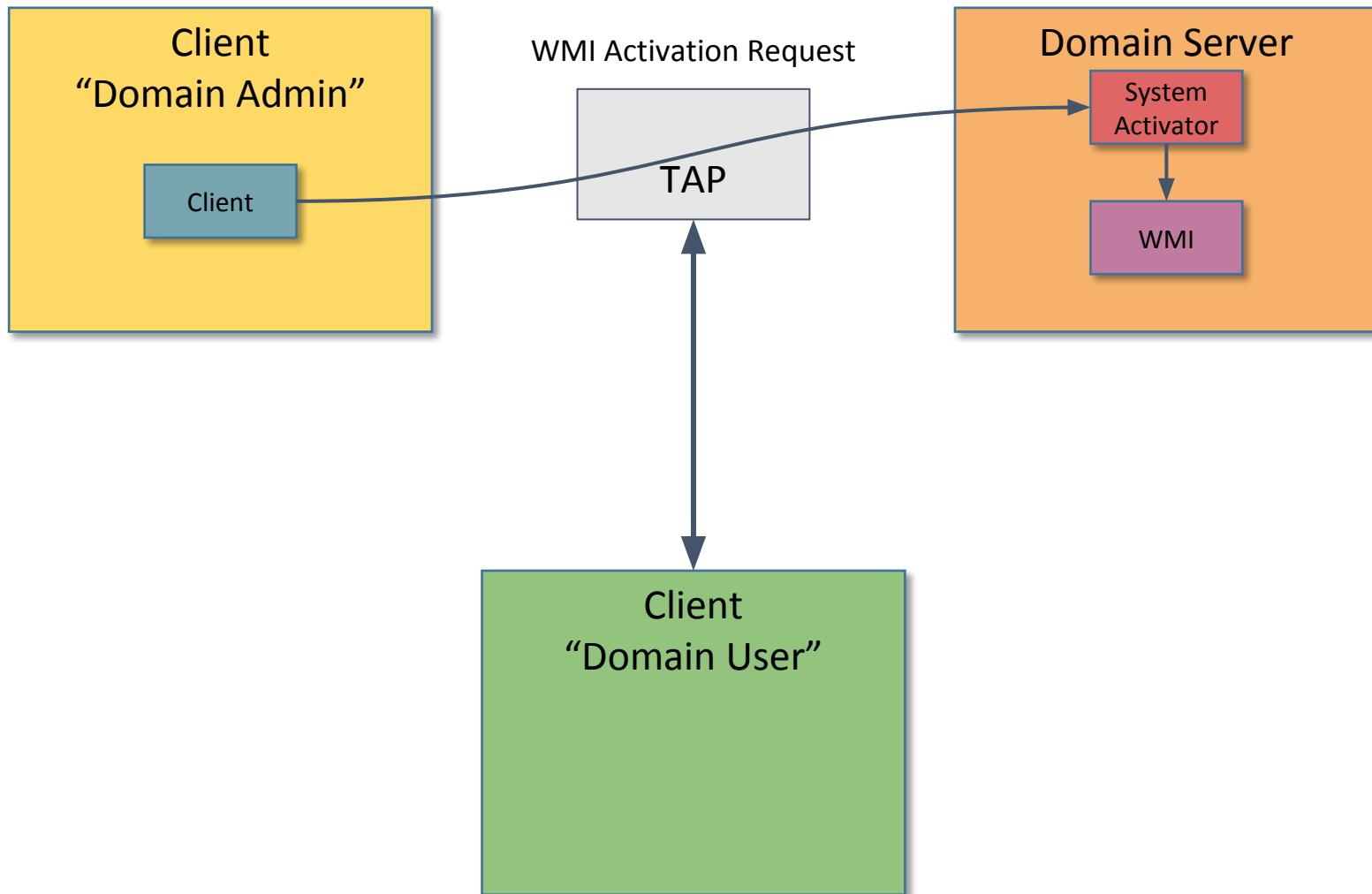
The screenshot shows the Wireshark interface with the following details:

- Network Interface:** *Ethernet 2
- Filter:** dcercp && ip.addr == 192.168.56.102
- Packets:** 478 total, 216 displayed (45.2%)
- Protocol:** DCERPC, IOXIDResolver, ISystemActivator
- Selected Packet:** 151 (58.101186) - ISystemActivator request (RemoteCreateInstance)
- Selected Hex View:** Shows the raw bytes of the selected packet, which is identified as an OBJREF (dcom.objref) object.
- Selected ASCII View:** Shows the ASCII representation of the selected packet's payload.
- Selected Bytes View:** Shows the binary representation of the selected packet's payload.
- Selected Structure View:** Shows the detailed structure of the selected packet, identifying it as an IActProperties object with CntData: 728 and an OBJREF component.

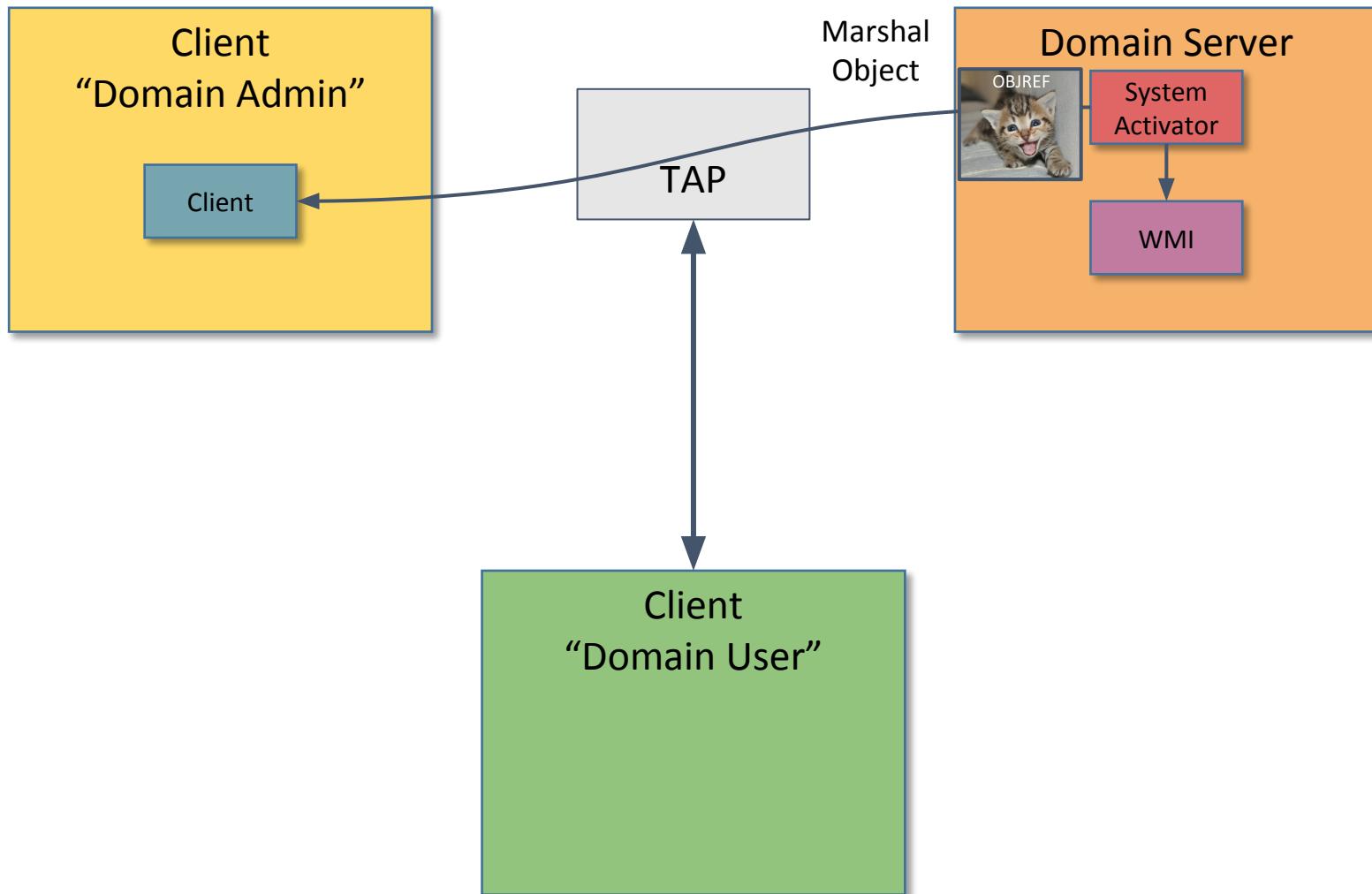
Access Permissions for WMI



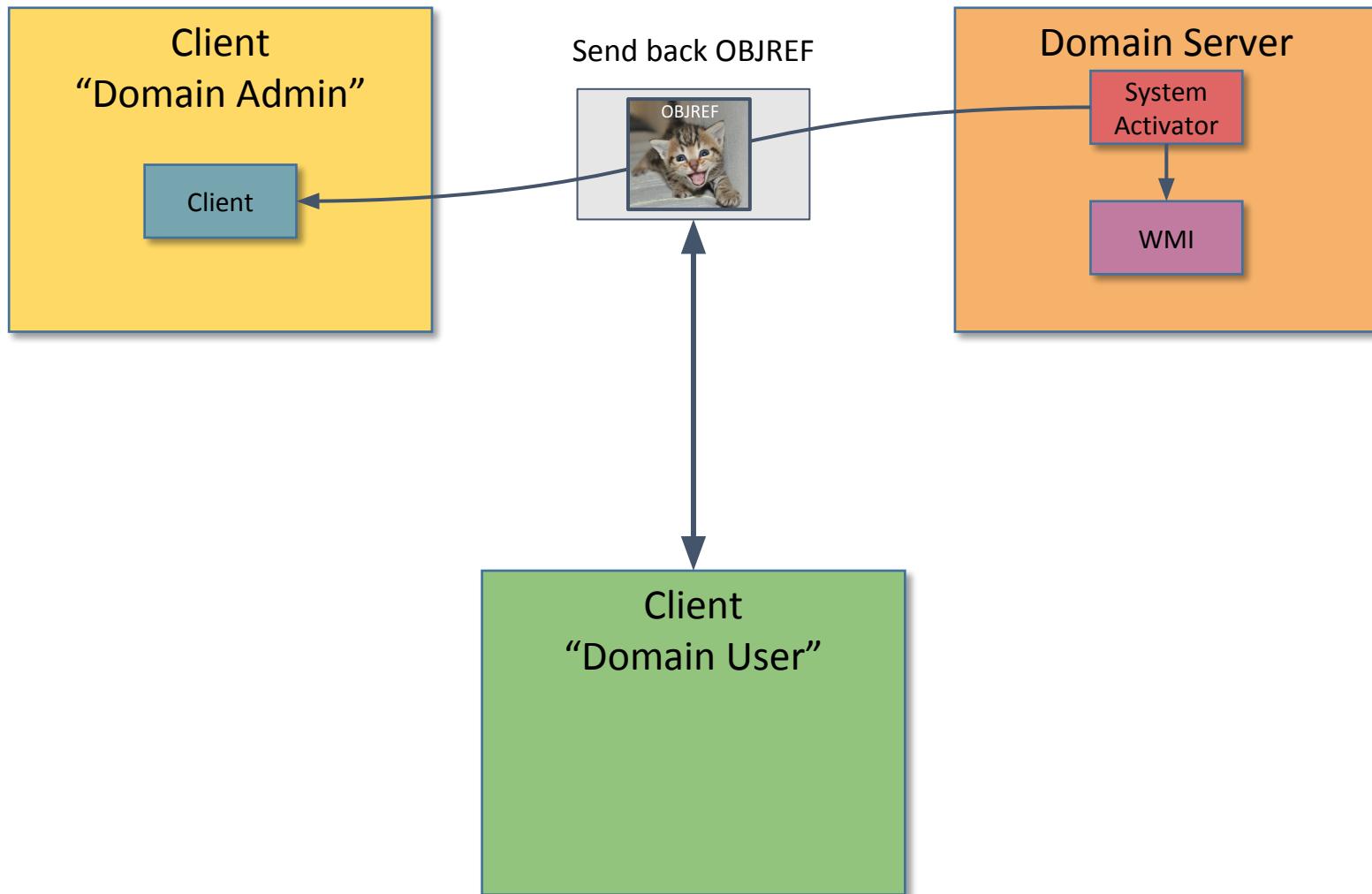
MEOW-Jacking!



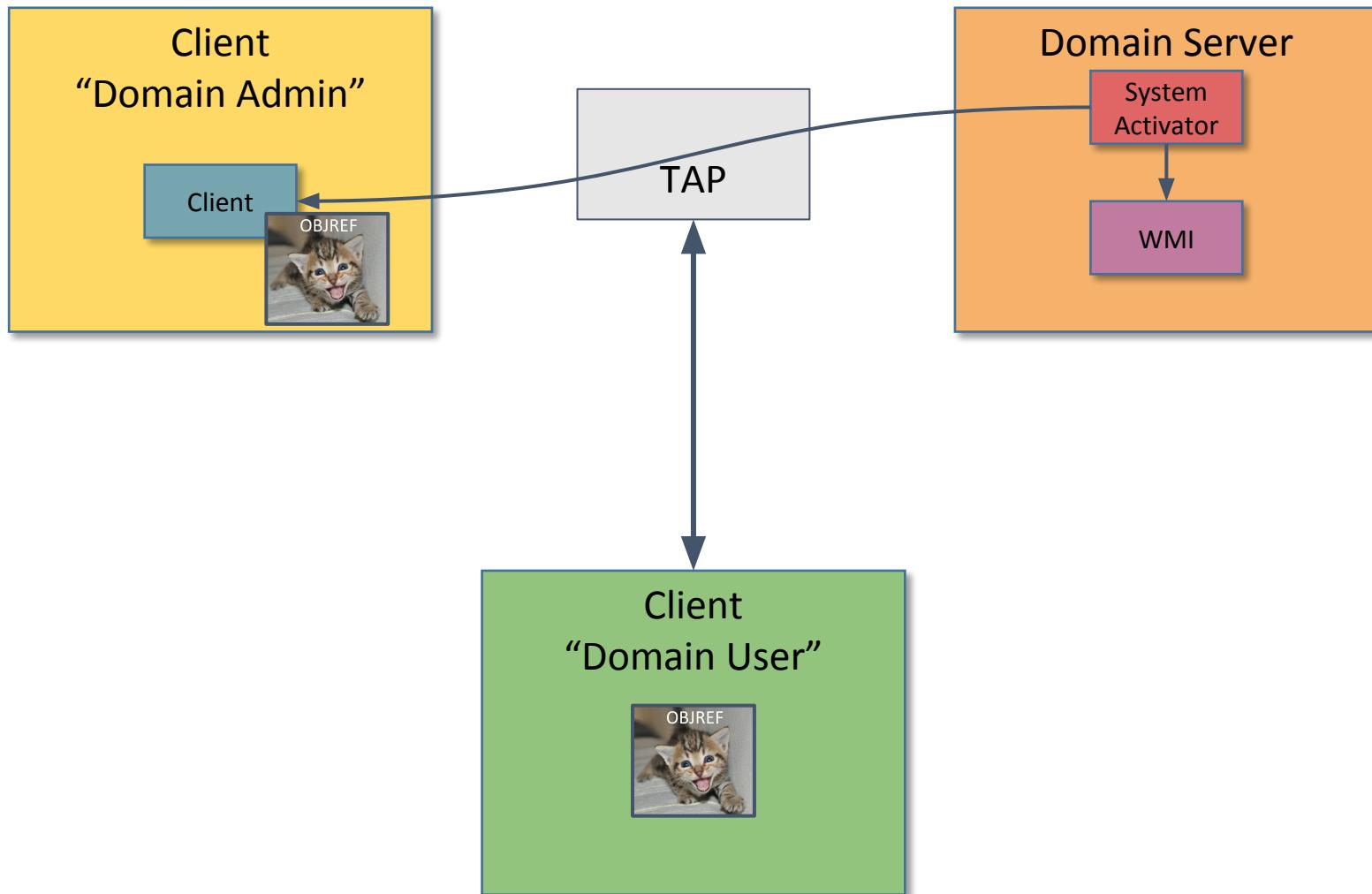
MEOW-Jacking!



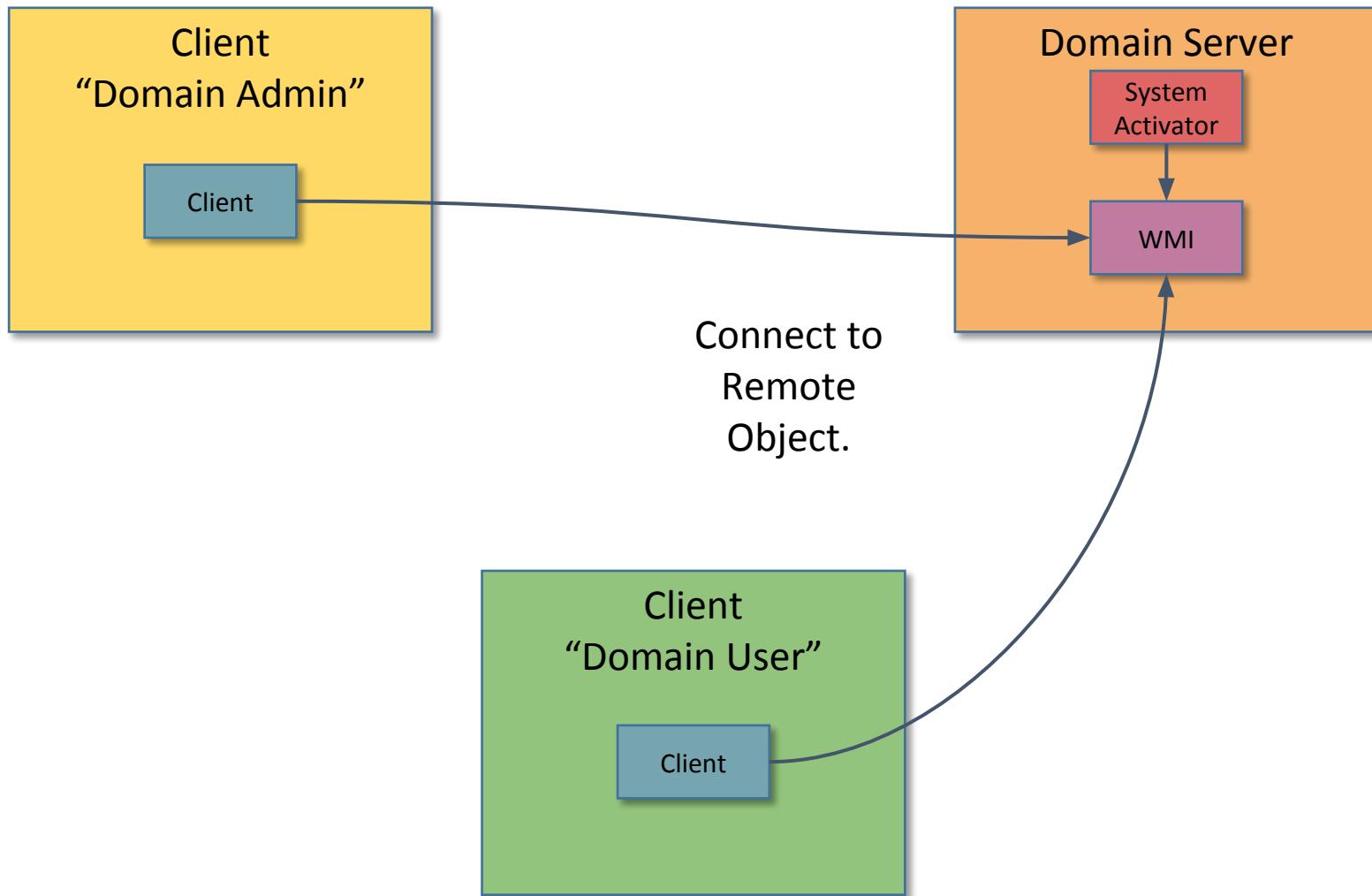
MEOW-Jacking!



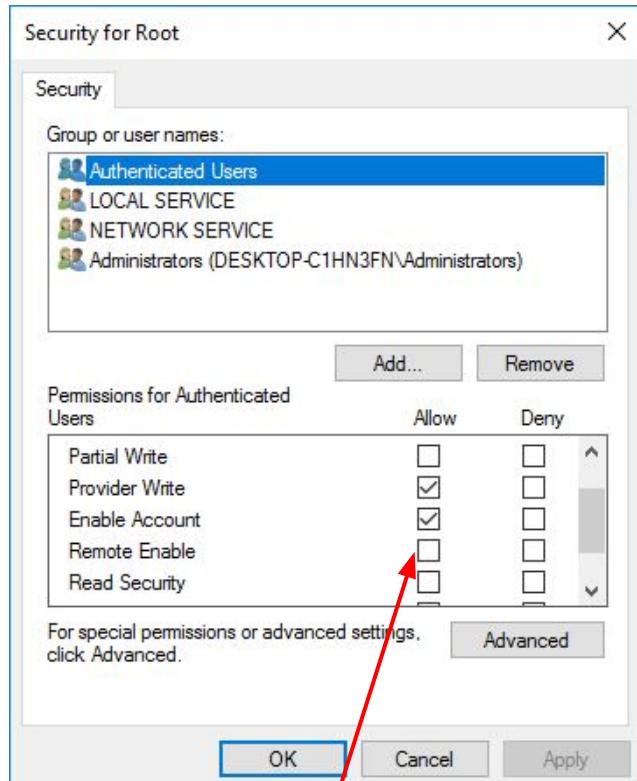
MEOW-Jacking!



MEOW-Jacking!



Not as Bad as it Could Be



User's not allowed to access WMI
remotely.

The screenshot shows the OleViewDotNet application window titled "OleViewDotNet - Administrator - 64bit". The main pane displays a list of COM processes, with entries for process IDs 8, 92, and 424. A red box highlights the "Services" node under process ID 424, which lists several system services: Delivery Optimization, Geolocation Service, User Profile Service, Shell Hardware Detection, Windows Management Instrumentation, Windows Push Notifications System Service, and Windows Update. To the right of the highlighted area, a red annotation reads: "Plenty of other potential COM objects though." At the bottom of the list, there are many entries starting with IPID: followed by various GUIDs and interface IDs like IRunDown, ILocalSystemActivator, INotifyNetworkInterfaceEvents, and IUnknown.



DEMO TIME



**Thanks for Listening
Any Questions?**