Tools and Workbook:

https://goo.gl/8ardeY

Introduction to Logical Privilege Escalation on Windows

2Hr Workshop

James Forshaw - @tiraniddo

Tools and workbook at: https://goo.gl/8ardeY

Agenda of this Workshop

- Windows Internals as relevant to privilege escalation
- Attack surface analysis from sandboxes and normal user
- Bug classes and Vulnerability Exploitation
- Willing to answer questions as I go along, however it might need to be saved till the end depending on the question :-)

Sorry, only so much I can talk about in 2hrs.
Some things are going to be missed :(

Setup the Tools and Examples

- Download toolset and workbook from link below
- Ideally you want a VM of Windows 10 Creator Update
  - 32 bit preferred, but 64 bit should also work for most things
- Extract contents to *c:\workshop* in the VM
- Run *setup.ps1* to configure your environment and install the driver. Follow descriptions in the wookbook.

https://goo.gl/8ardeY

What is a Logical Vulnerability?

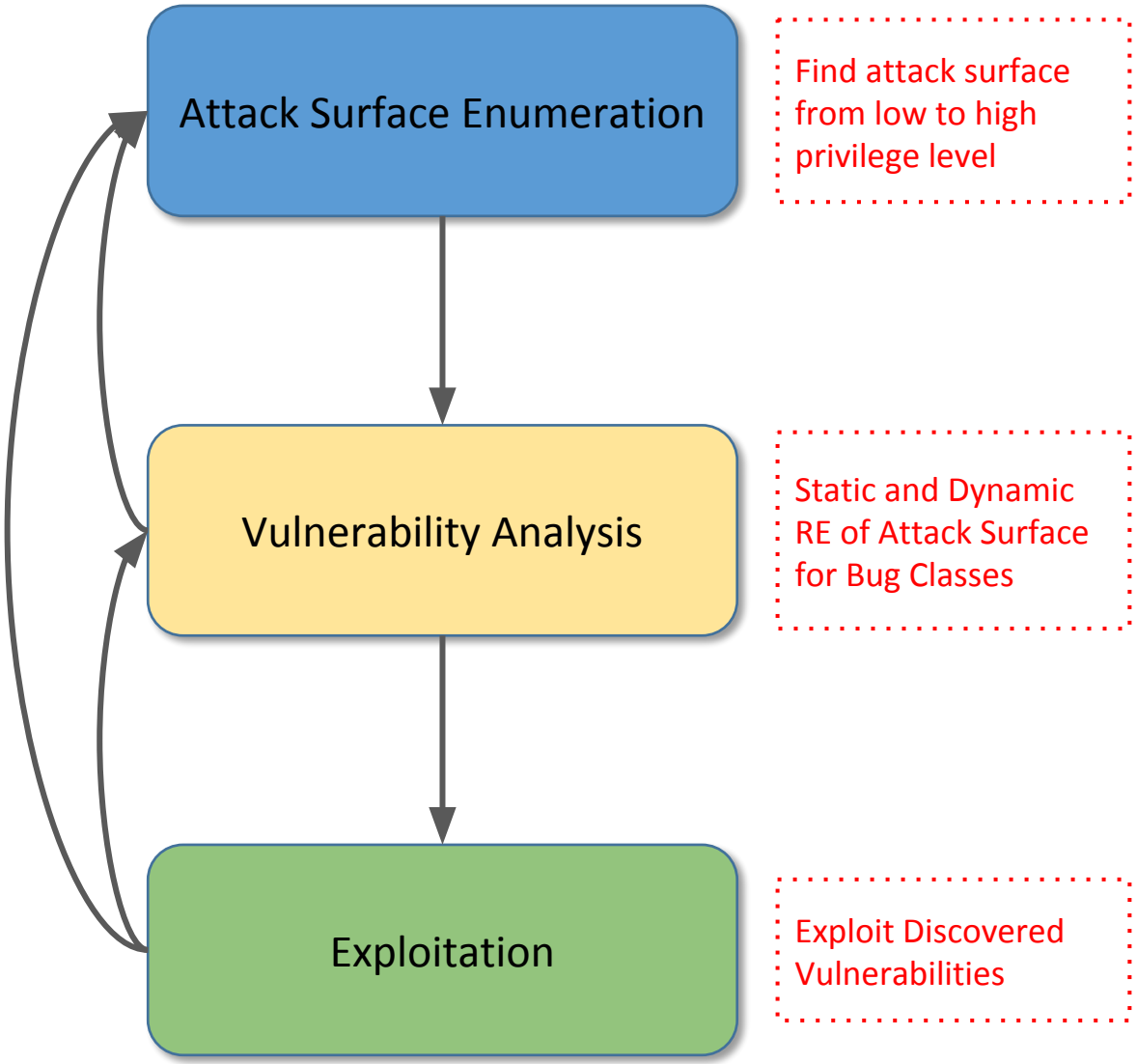
A security vulnerability which rely on subverting the programmer's original logic rather than abusing unintended behaviour.

Why?

- Why Privilege Escalation?
  - Everything is getting sandboxed!
    - Even Firefox has a sandbox!
  - Everyone is running as a normal user
    - Or should be, of course there's UAC, but well.
- Why Logical Exploitation?
  - Exploiting memory corruption is getting more difficult
    - Stack cookies, hardened heaps
    - Control Flow and Return Flow Guard
    - SMEP (Supervisor Mode Execution Protection) preventing trivial kernel code execution
    - ASLR, DEP and all that, including limiting information leakage from kernel.
  - Exploiting memory corruptions is boring ;-)

Tools and workbook at: https://goo.gl/8ardeY

My Approach to Finding Logical Vulnerabilities

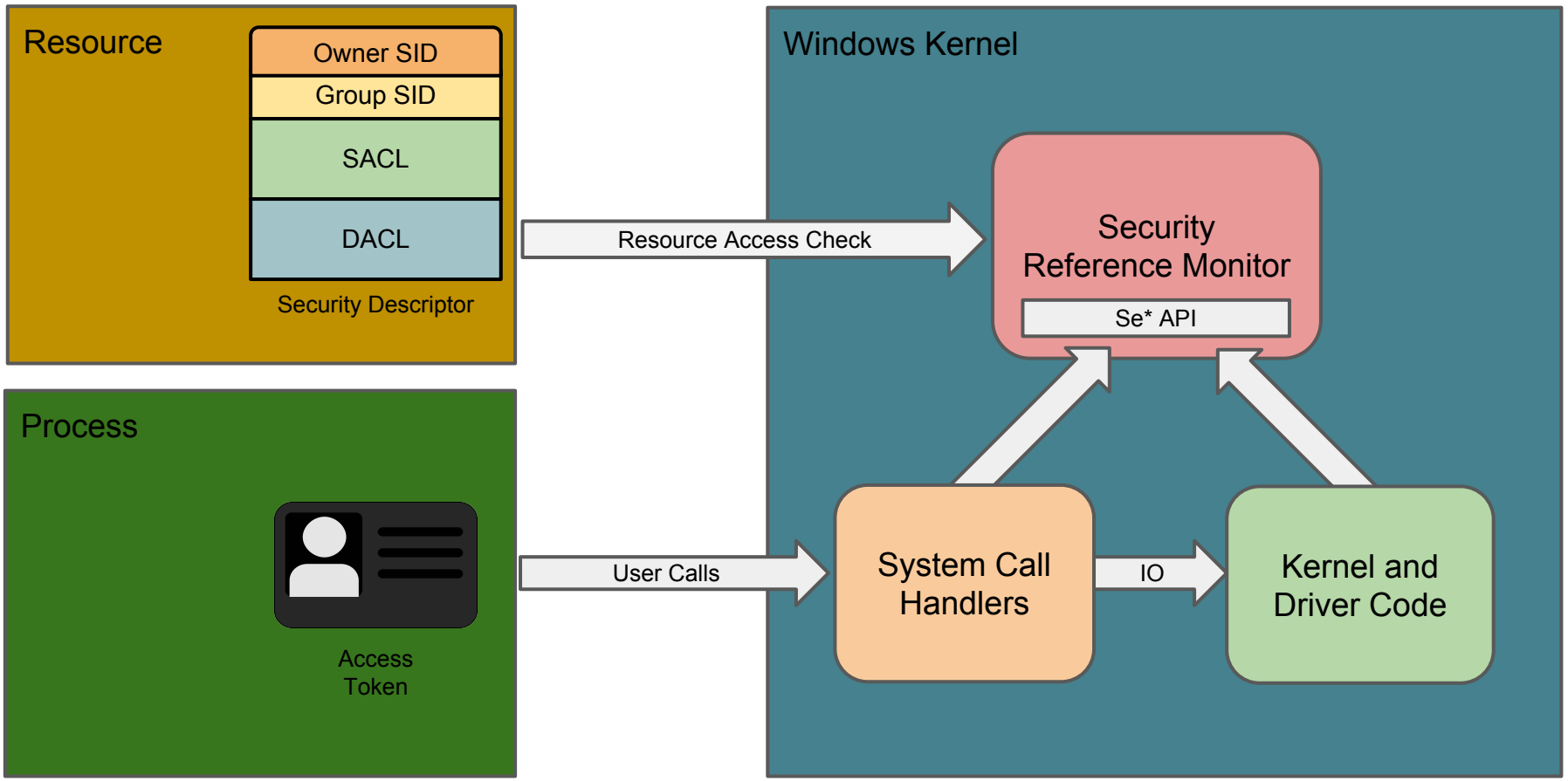


Tools and workbook at: https://goo.gl/8ardeY

Windows Internals for EoP Hunters

Tools and workbook at: https://goo.gl/8ardeY

Windows Security Components



Tools and workbook at: https://goo.gl/8ardeY

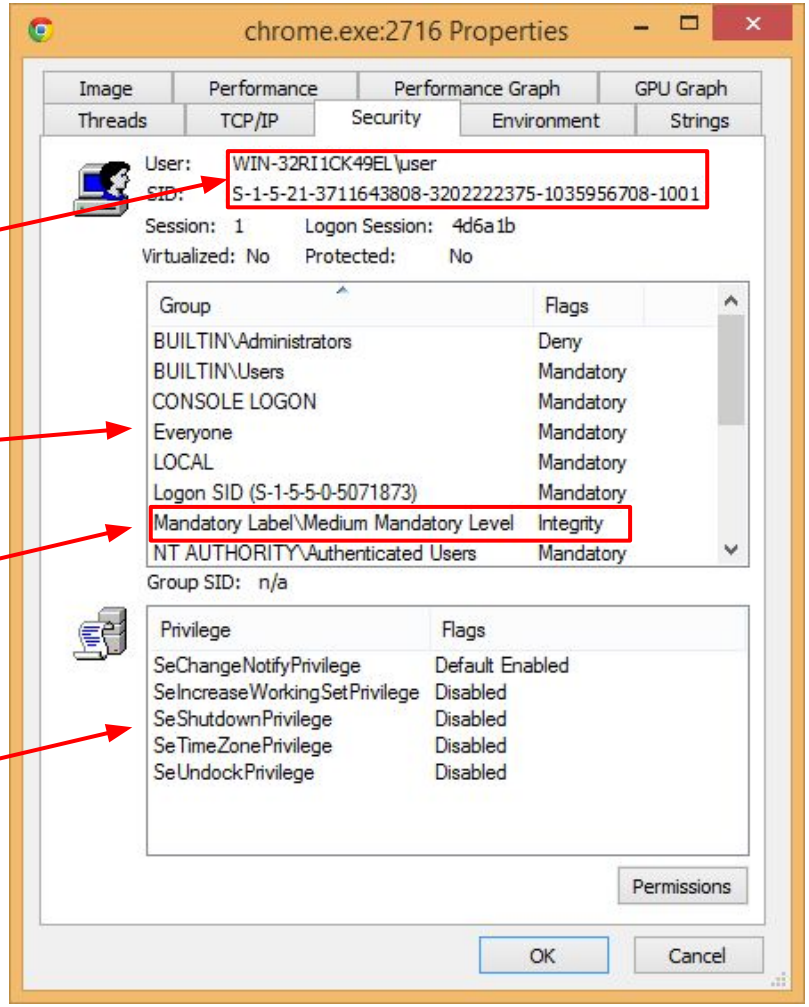
Access Token

User Security Identifier

Groups

Mandatory Label

Privileges



Security Identifiers

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

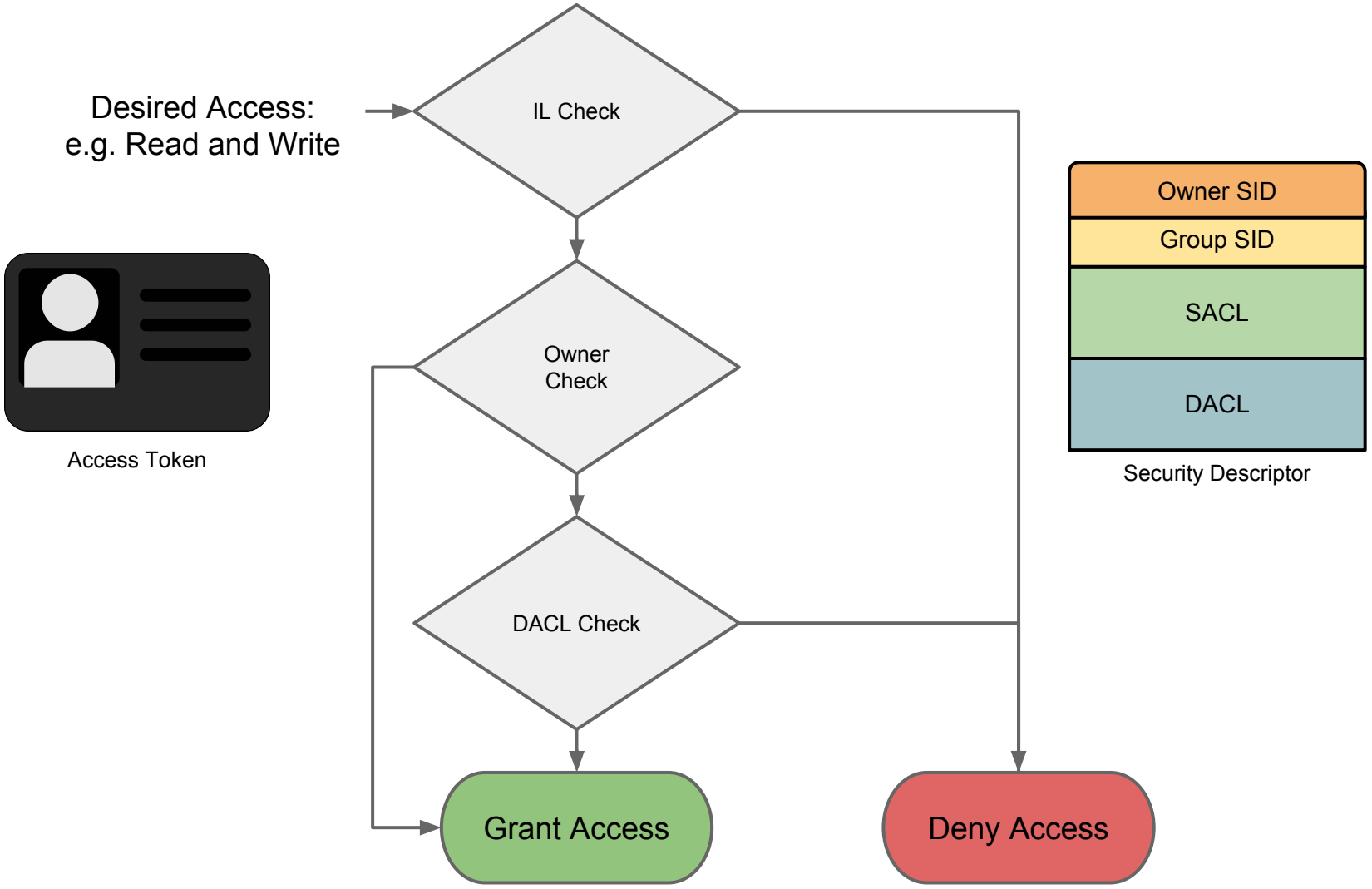
- Some well known SIDs:

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

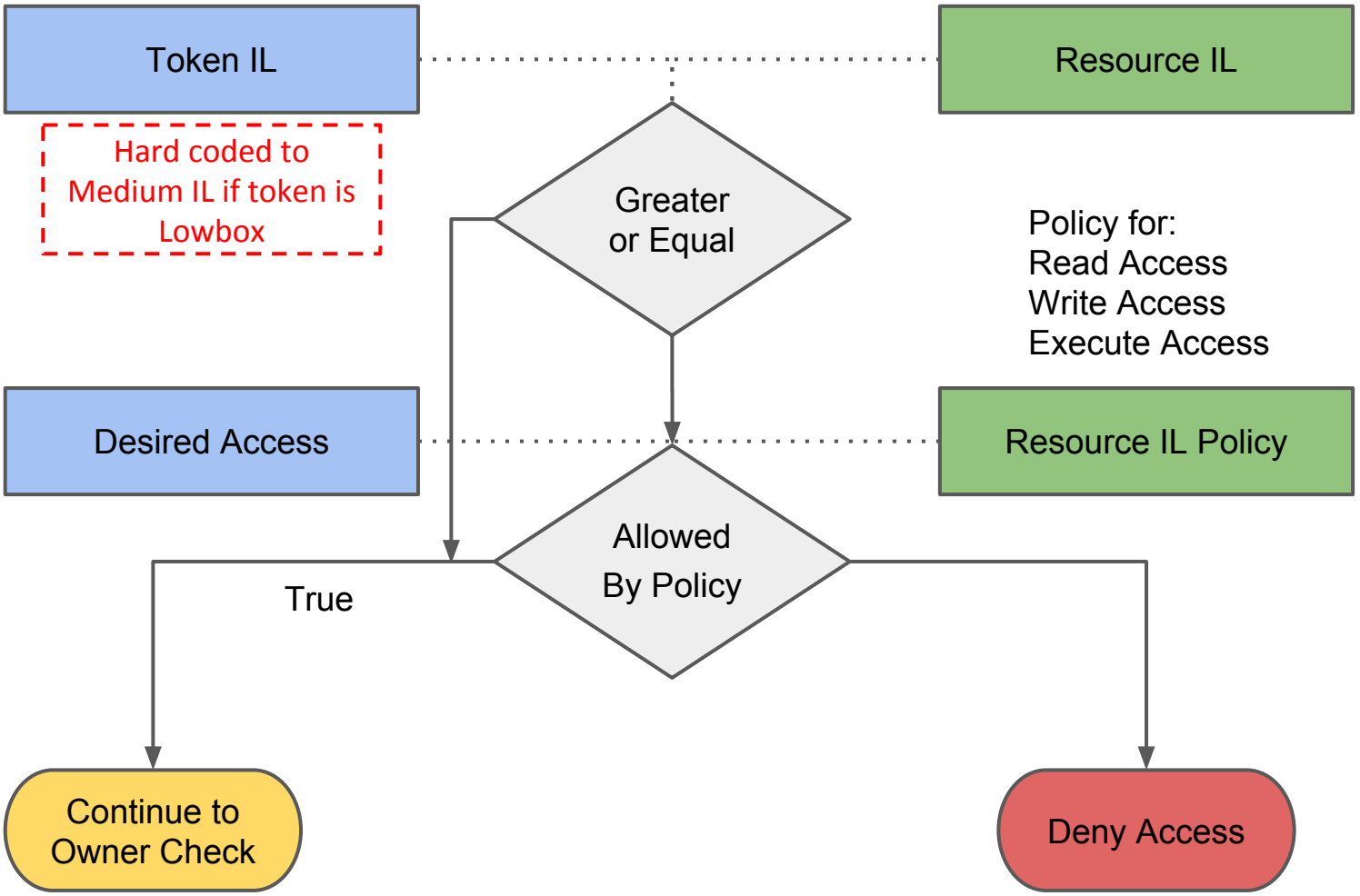
Tools and workbook at: https://goo.gl/8ardeY

Security Access Check (SeAccessCheck)



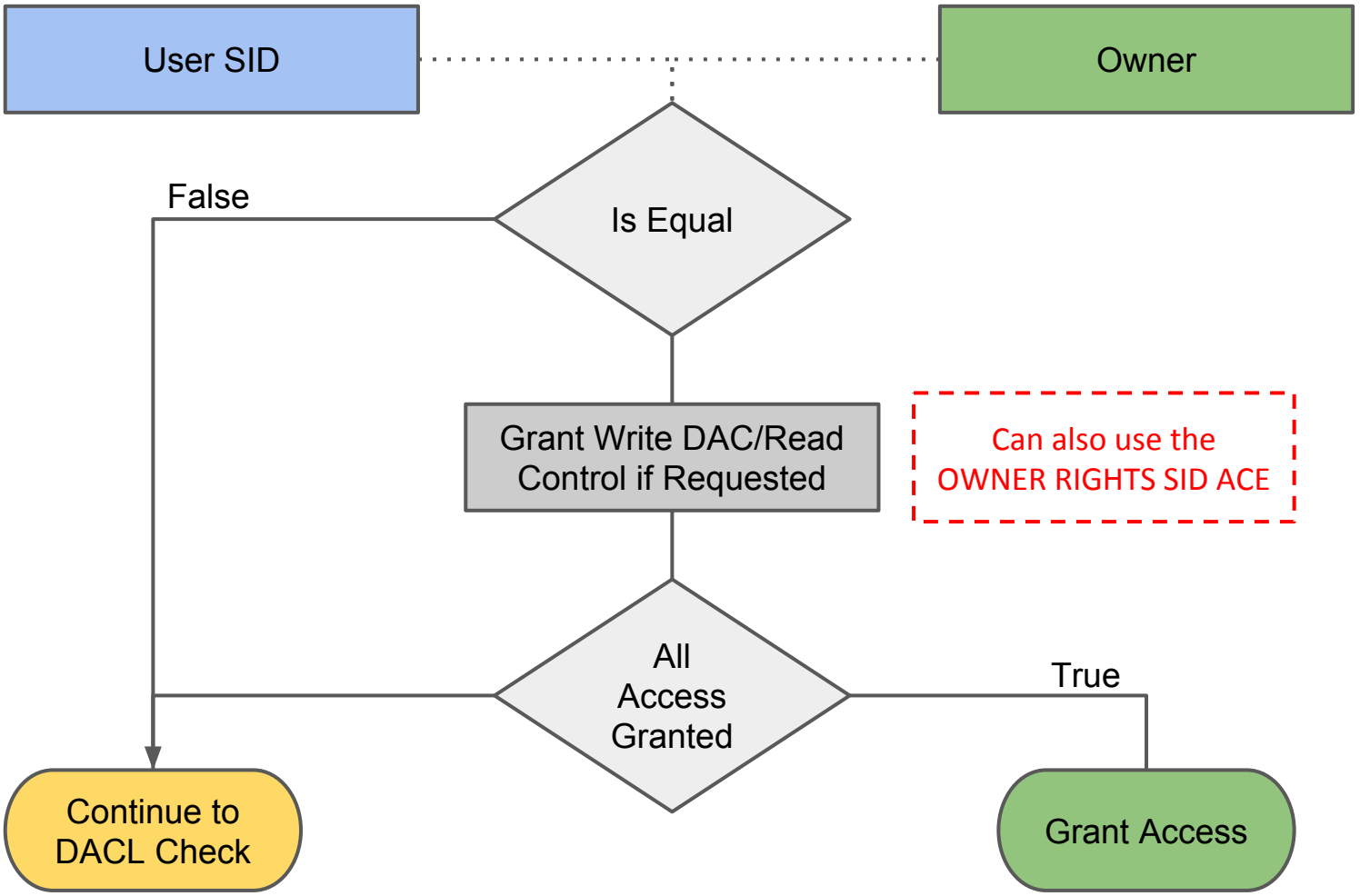
Tools and workbook at: https://goo.gl/8ardeY

Mandatory Integrity Level Check



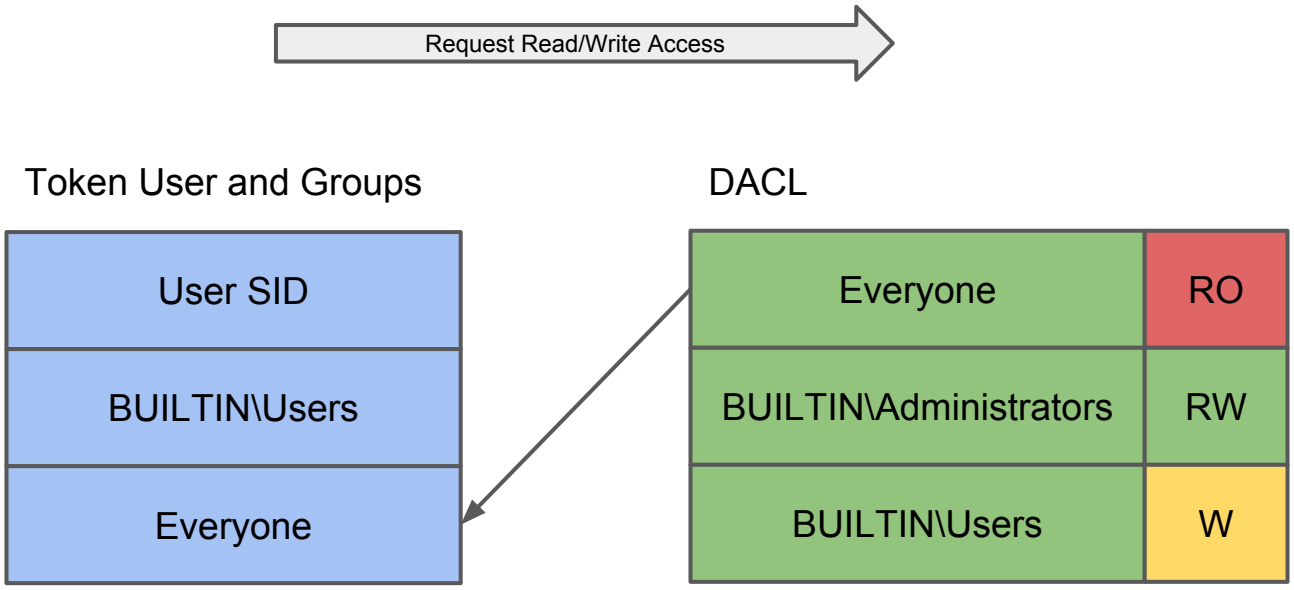
Tools and workbook at: https://goo.gl/8ardeY

Owner Check



Tools and workbook at: https://goo.gl/8ardeY

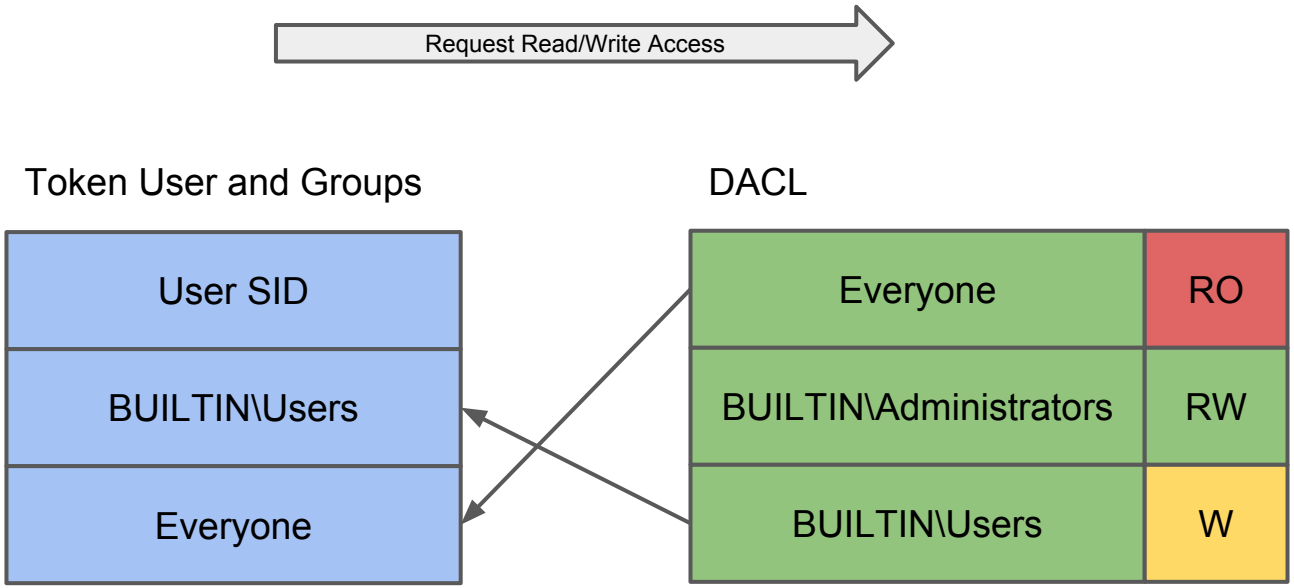
Kernel DACL Check



Current Granted Access: Read Only

Tools and workbook at: https://goo.gl/8ardeY

Kernel DACL Check



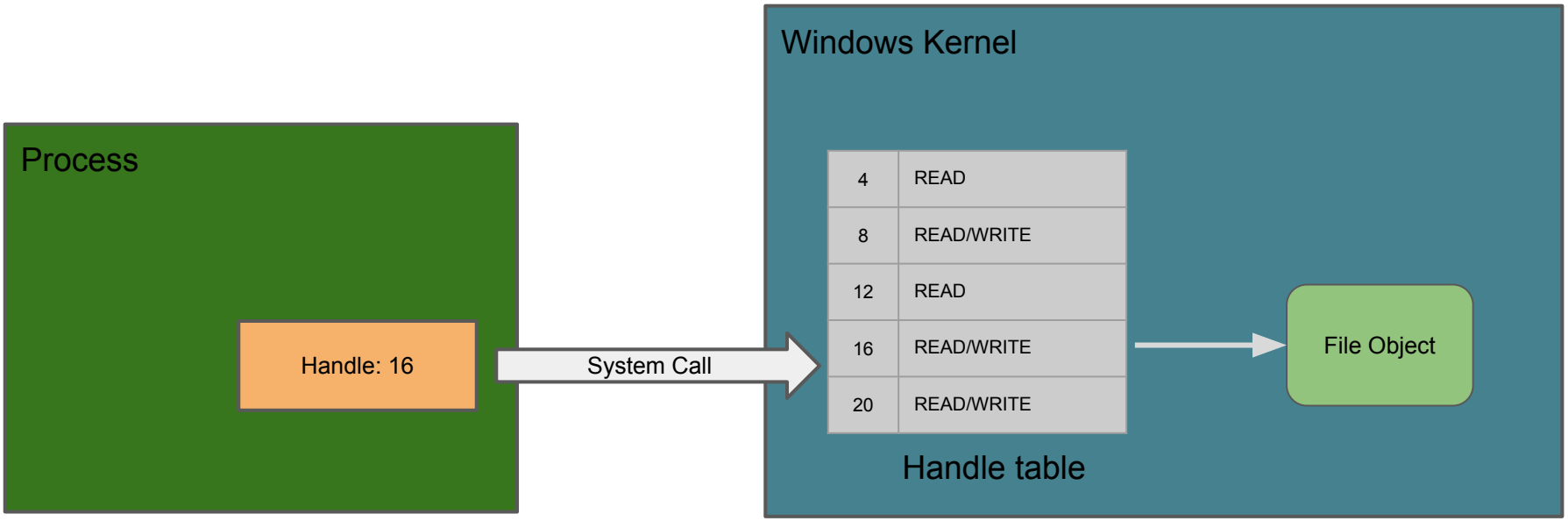
Final Granted Access: Read/Write

Security Descriptors and Inheritance

- New resources by default will inherit Security Descriptor for parent container (be it object directory/file directory/registry key etc.)
- Most resource creation calls can specify explicit SD
- 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

Tools and workbook at: https://goo.gl/8ardeY

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.

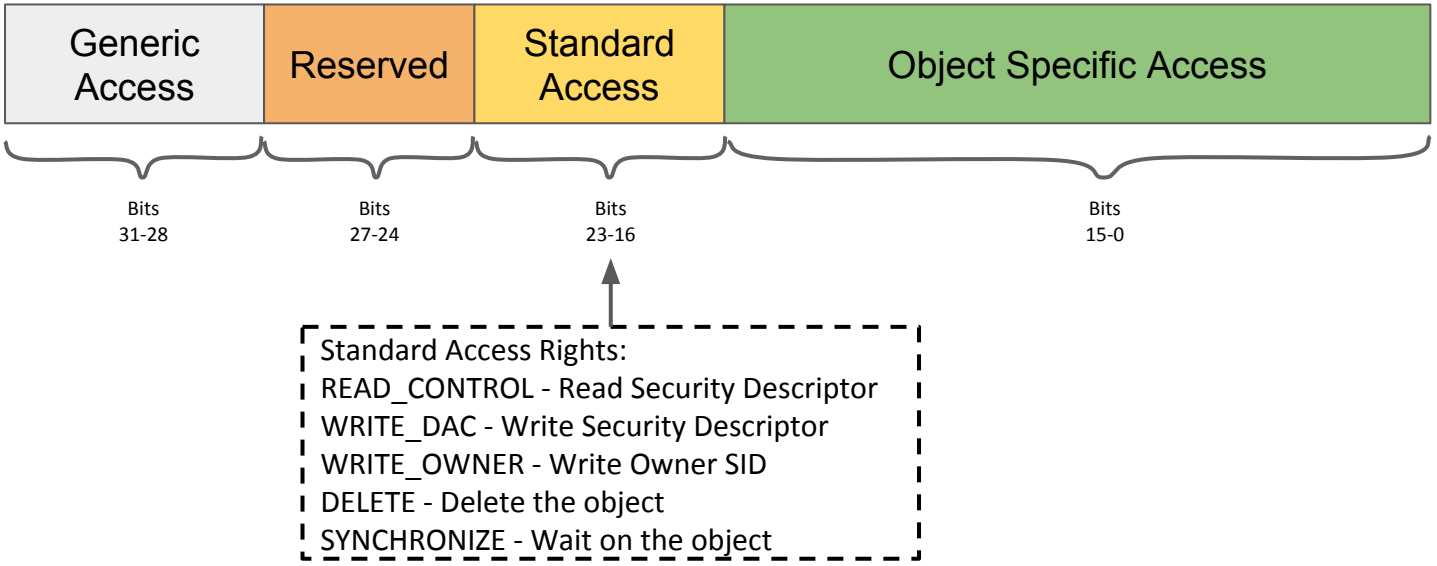
Required Access

Kernel Object Pointer

Tools and workbook at: https://goo.gl/8ardeY

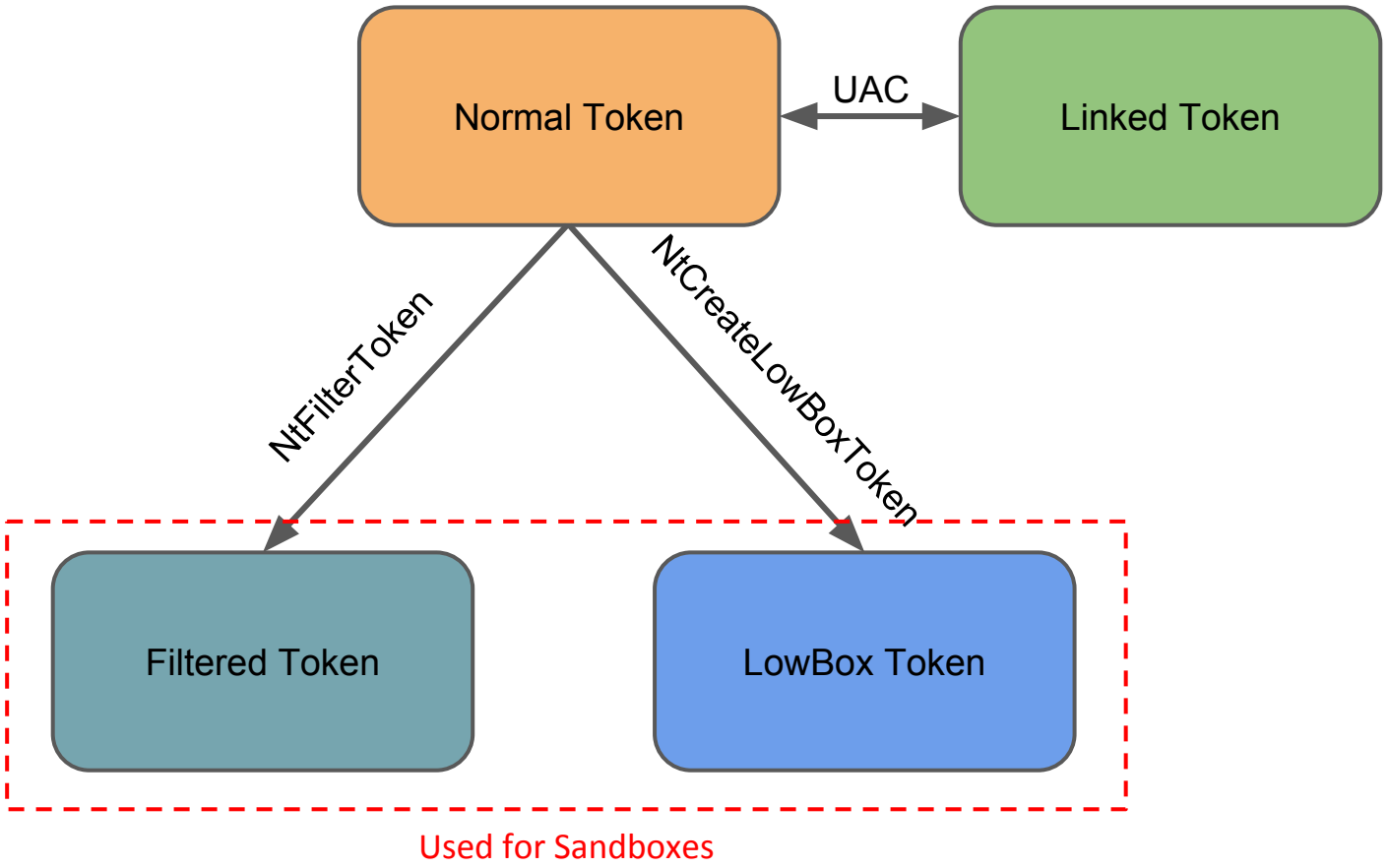
Access Masks

- When opening a handle need specify the access mask.
- Checked against the access mask in the DACL entries



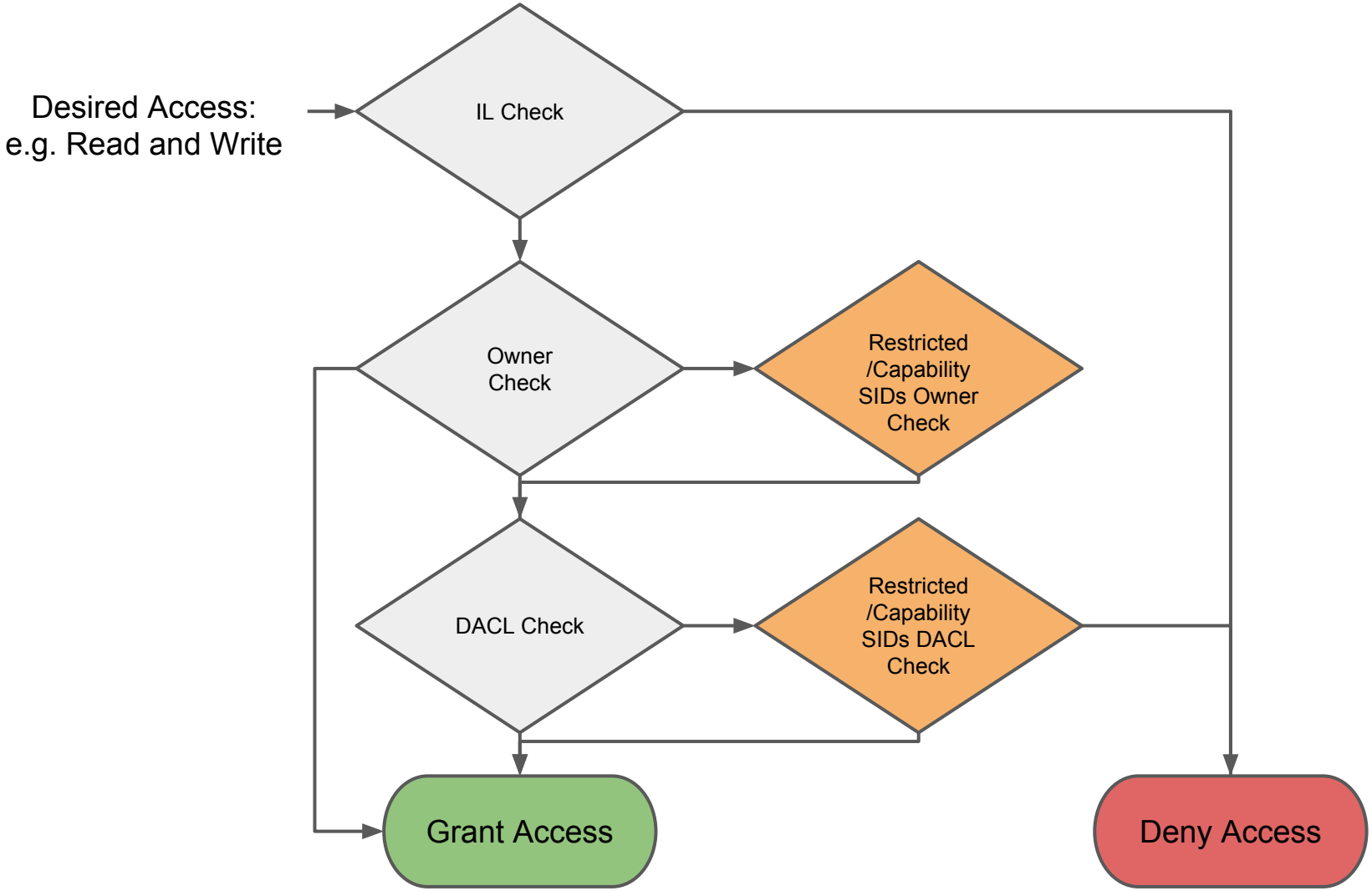
Tools and workbook at: https://goo.gl/8ardeY

Token Categories



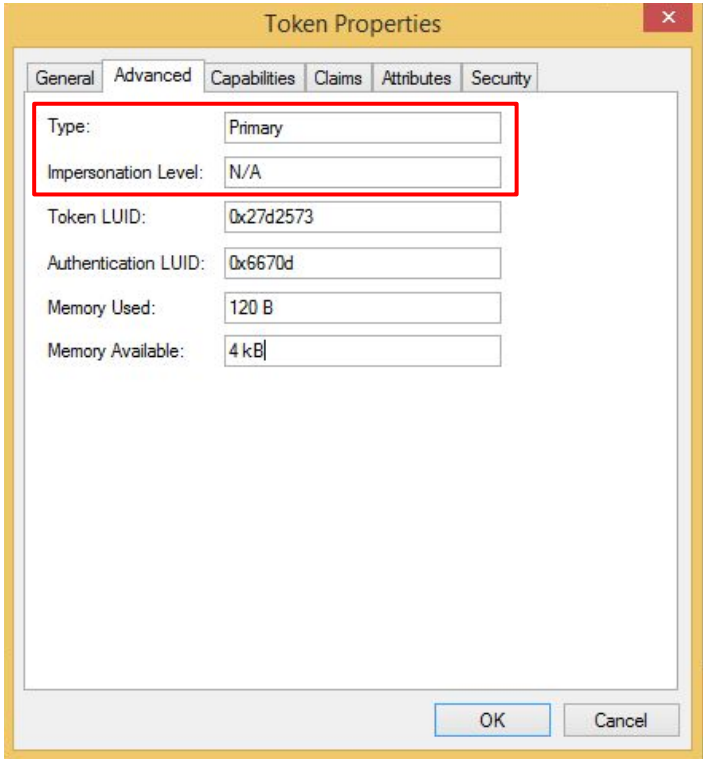
Tools and workbook at: https://goo.gl/8ardeY

Restricted/Lowbox Token Access Check

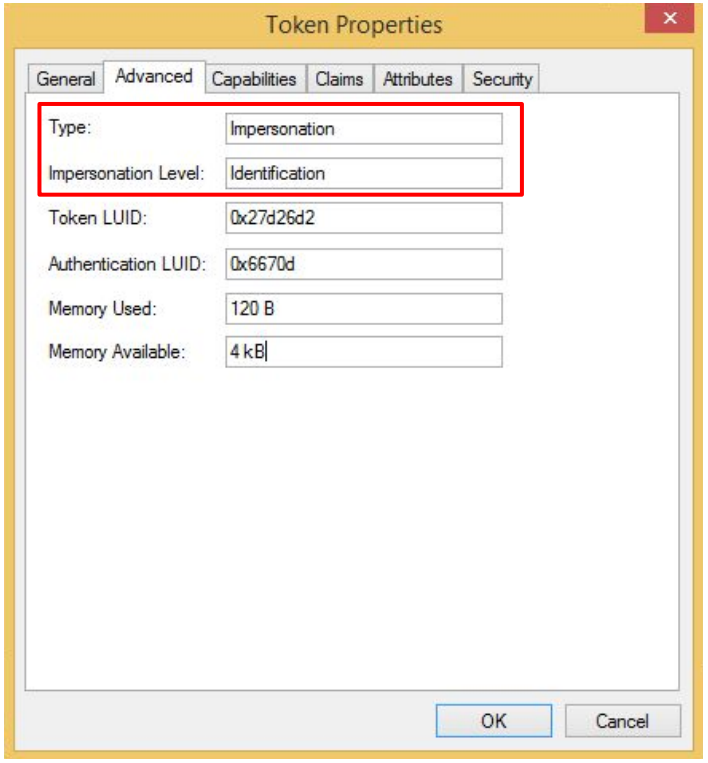


Tools and workbook at: https://goo.gl/8ardeY

Access Token Types



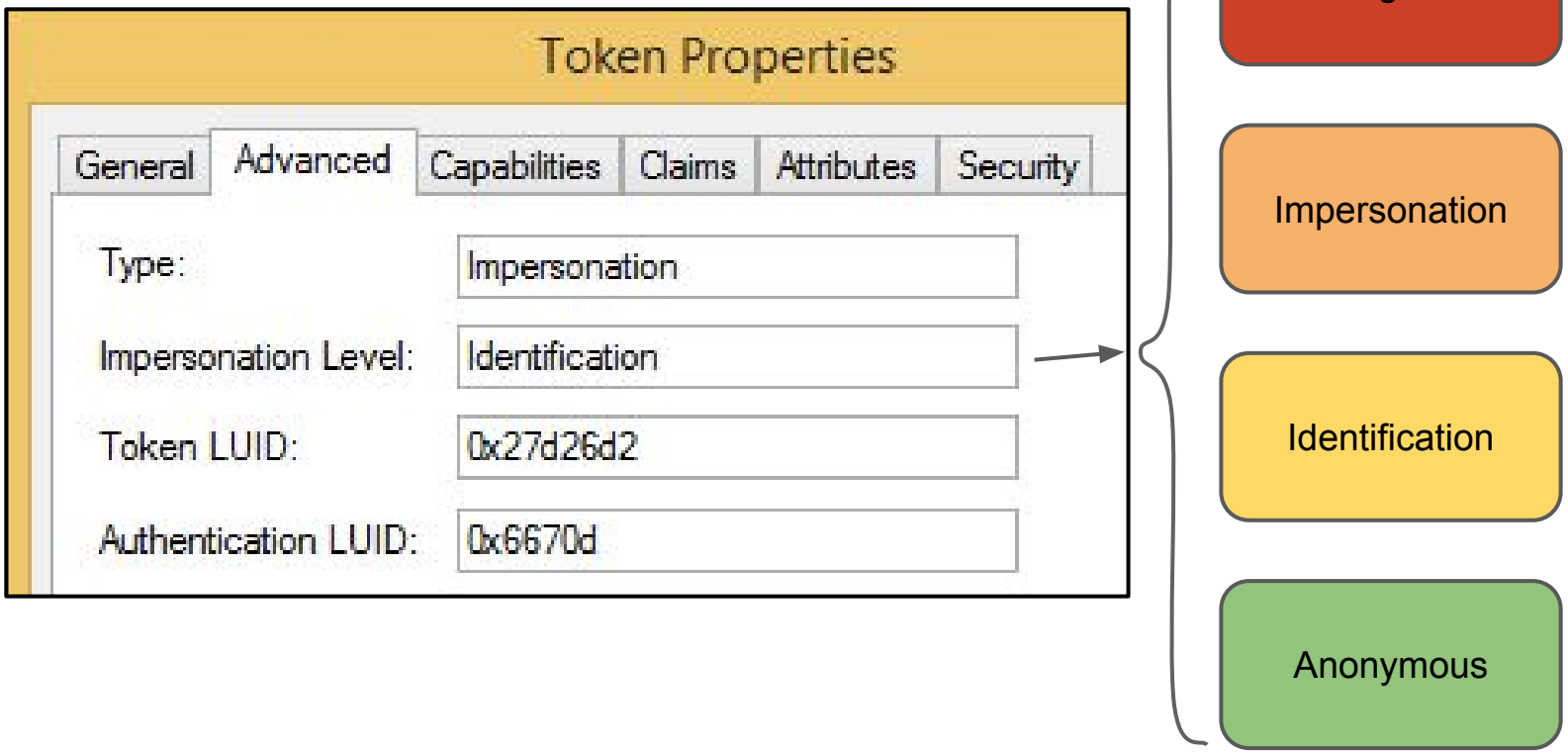
Primary



Impersonation

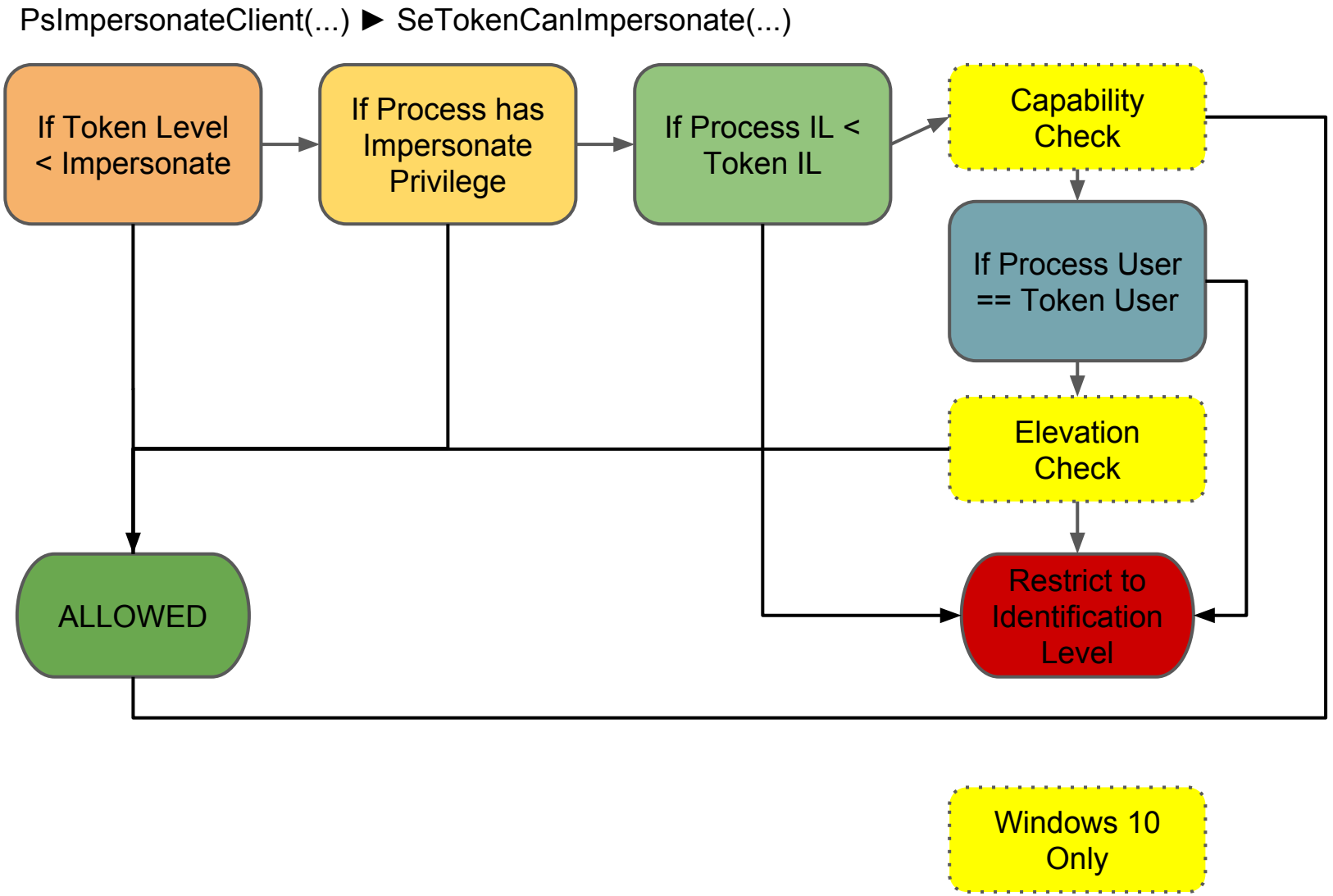
Tools and workbook at: https://goo.gl/8ardeY

Impersonation Security Level



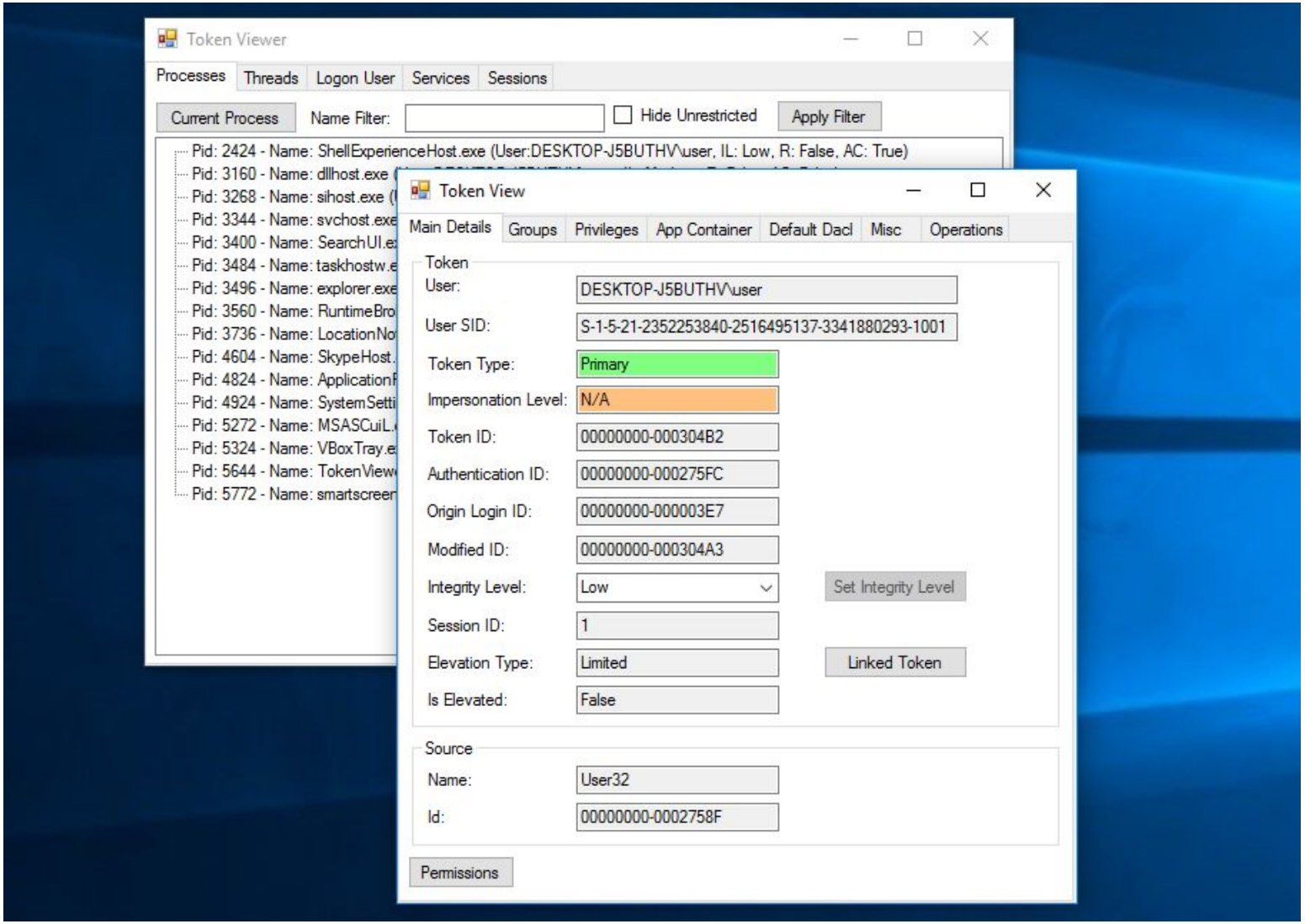
Tools and workbook at: https://goo.gl/8ardeY

Impersonation Security



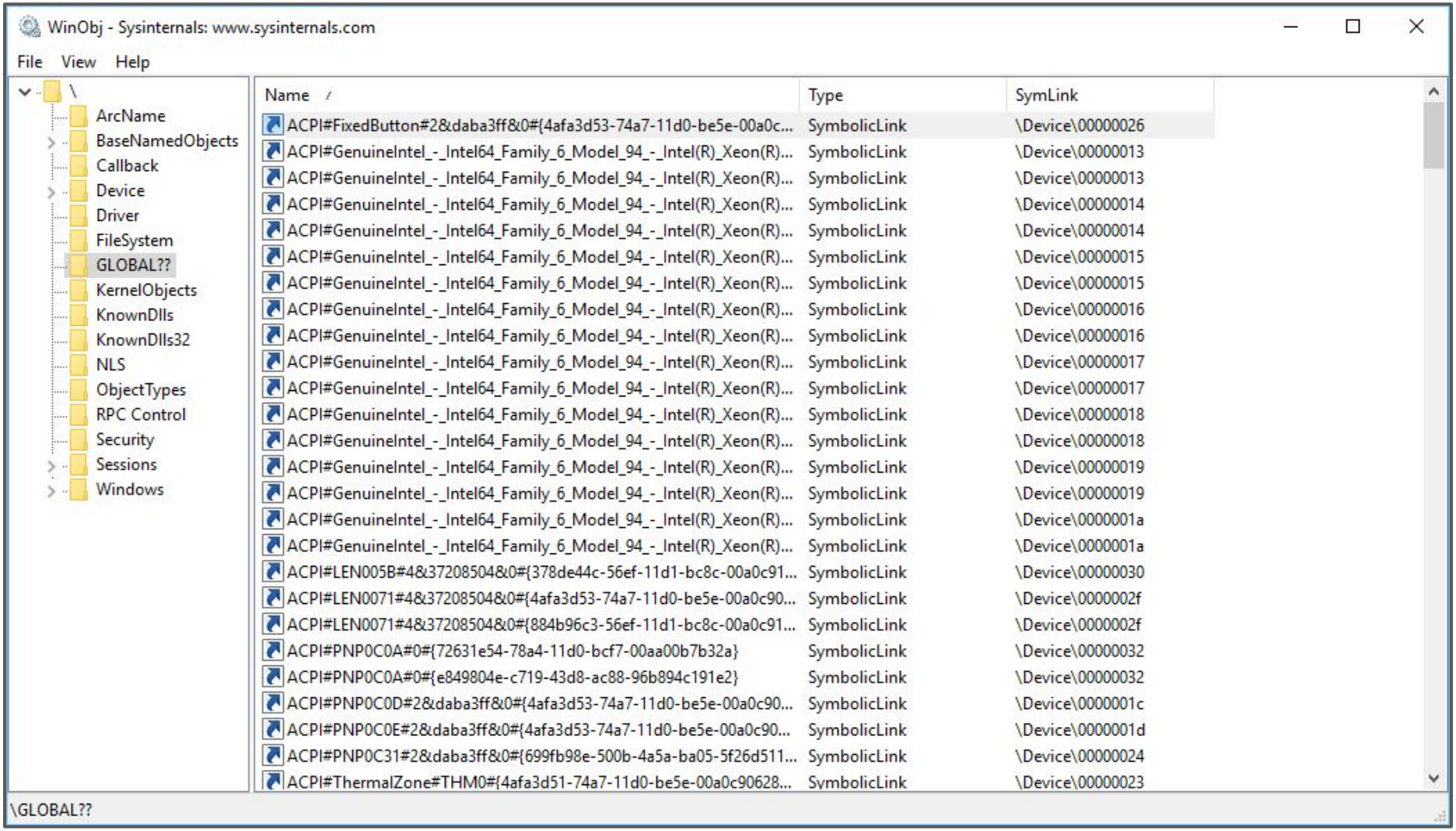
Tools and workbook at: https://goo.gl/8ardeY

DEMO 1: Viewing Token and Security Descriptors



Tools and workbook at: https://goo.gl/8ardeY

Object Manager Namespace



Important Object Directories

Path	*Description*
\Device	Default location for kernel driver Device Objects
\GLOBAL??	System location for symbolic links to devices including drive letters
\BaseNamedObjects	System location for named resources
\Sessions\X	Directory for the login session X
\Session\0\DosDevices	Directory for the “Dos Devices” for each logged in user.
\??	“Fake” prefix which refers to per-user Dos Devices.

Win32 Path Support

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

Tools and workbook at: https://goo.gl/8ardeY

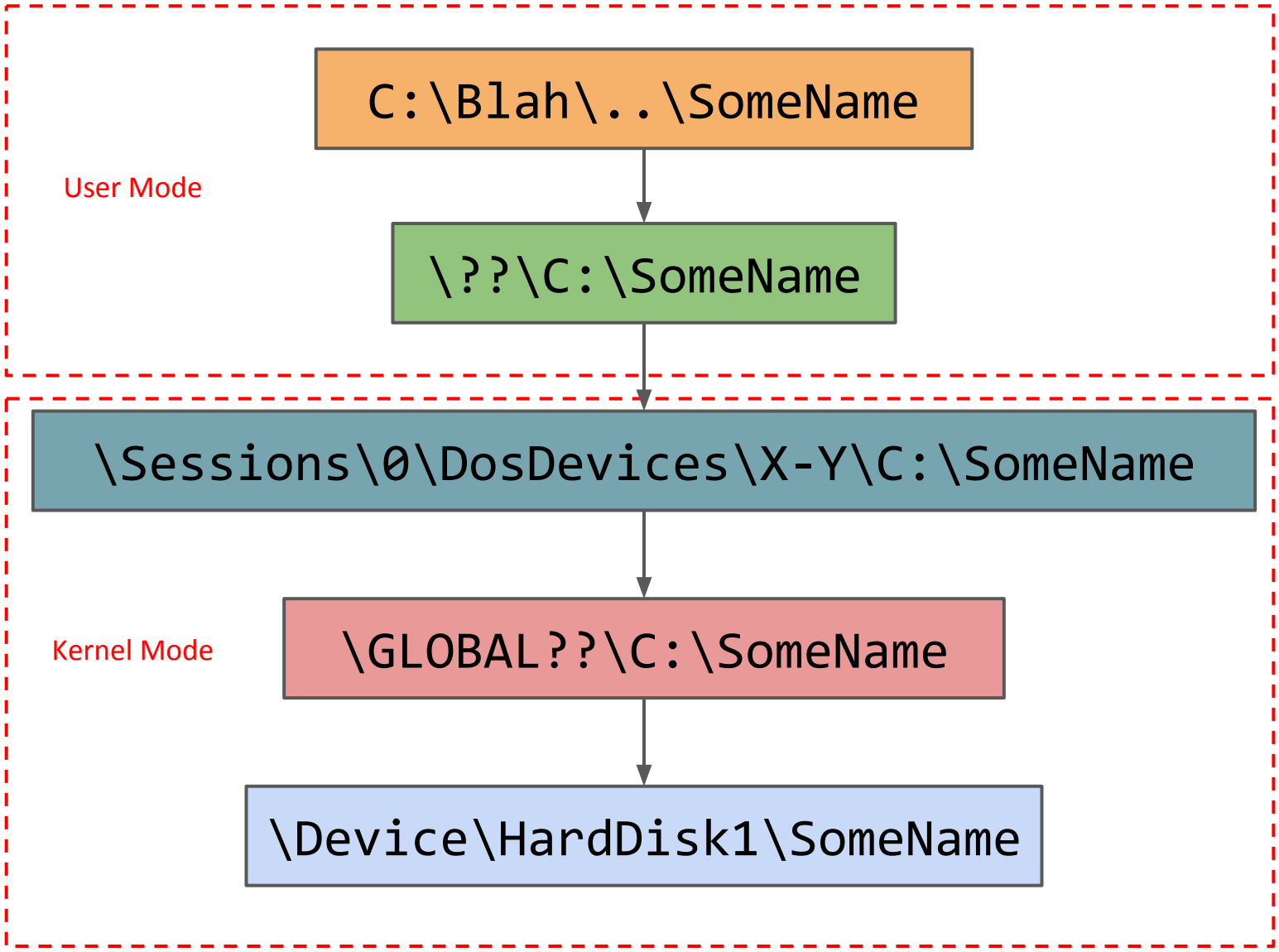
Canonicalization

- Type of Win32 path affects canonicalization behaviour

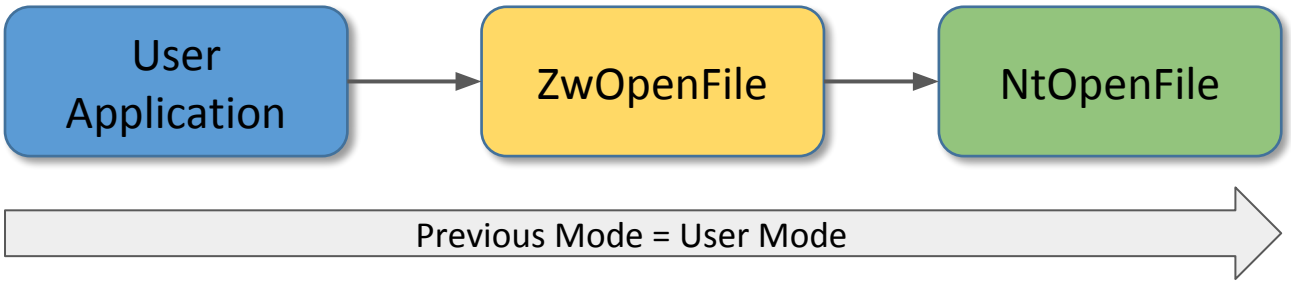
Path	Result of Canonicalization
c:\path\../badgers	c:\badgers
c:\..\d:/badgers	c:\d:\badgers
\\.\c:\path\../badgers	c:\badgers
\\.\c:\..\d:/badgers	\\.\d:\badgers (WTF!)
\\?\c:\path\../badgers	\\?\c:\path\../badgers

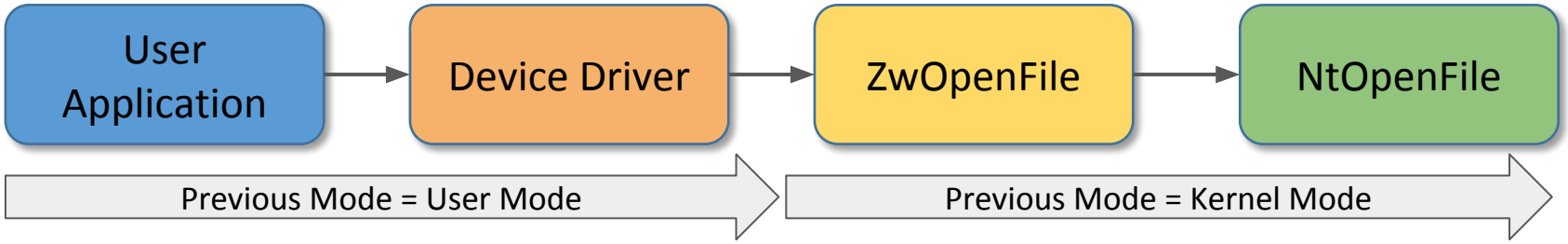
Tools and workbook at: https://goo.gl/8ardeY

File Path Handling



Previous Processor Mode

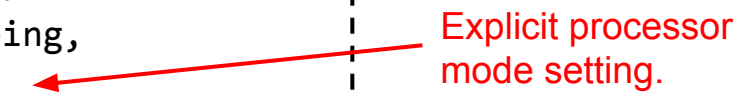




Previous Processor Mode

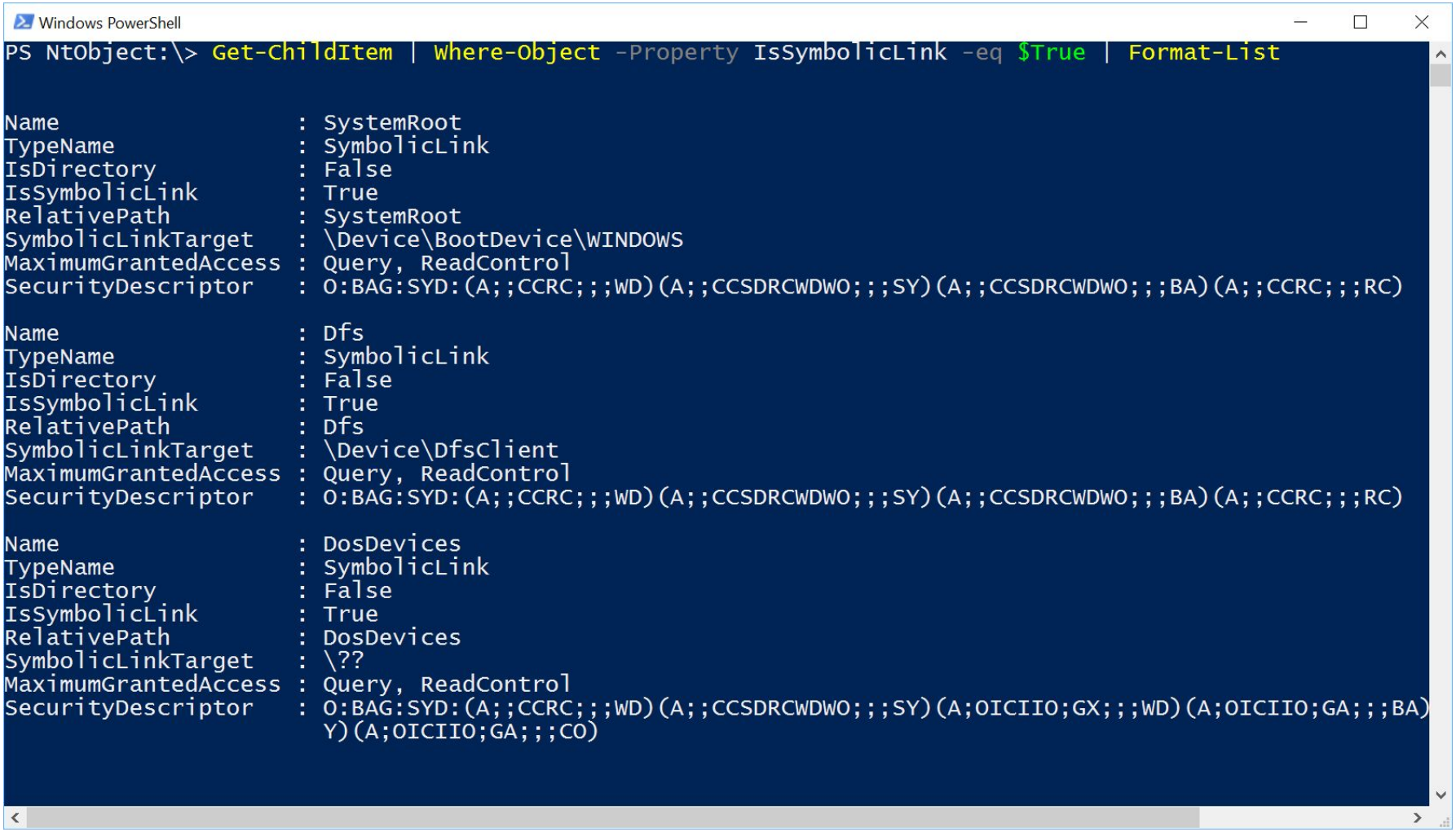
- Previous processor mode used to determine whether to enforce security!

```
BOOLEAN SeAccessCheck(
  _In_  PSECURITY_DESCRIPTOR      SecurityDescriptor,
  _In_  PSECURITY_SUBJECT_CONTEXT SubjectSecurityContext,
  _In_  BOOLEAN                   SubjectContextLocked,
  _In_  ACCESS_MASK               DesiredAccess,
  _In_  ACCESS_MASK               PreviouslyGrantedAccess,
  _Out_ PPRIVILEGE_SET            *Privileges,
  _In_  PGENERIC_MAPPING          GenericMapping,
  _In_  KPROCESSOR_MODE           AccessMode,
  _Out_ PACCESS_MASK              GrantedAccess,
  _Out_ PNTSTATUS                 AccessStatus
);
```



Tools and workbook at: https://goo.gl/8ardeY

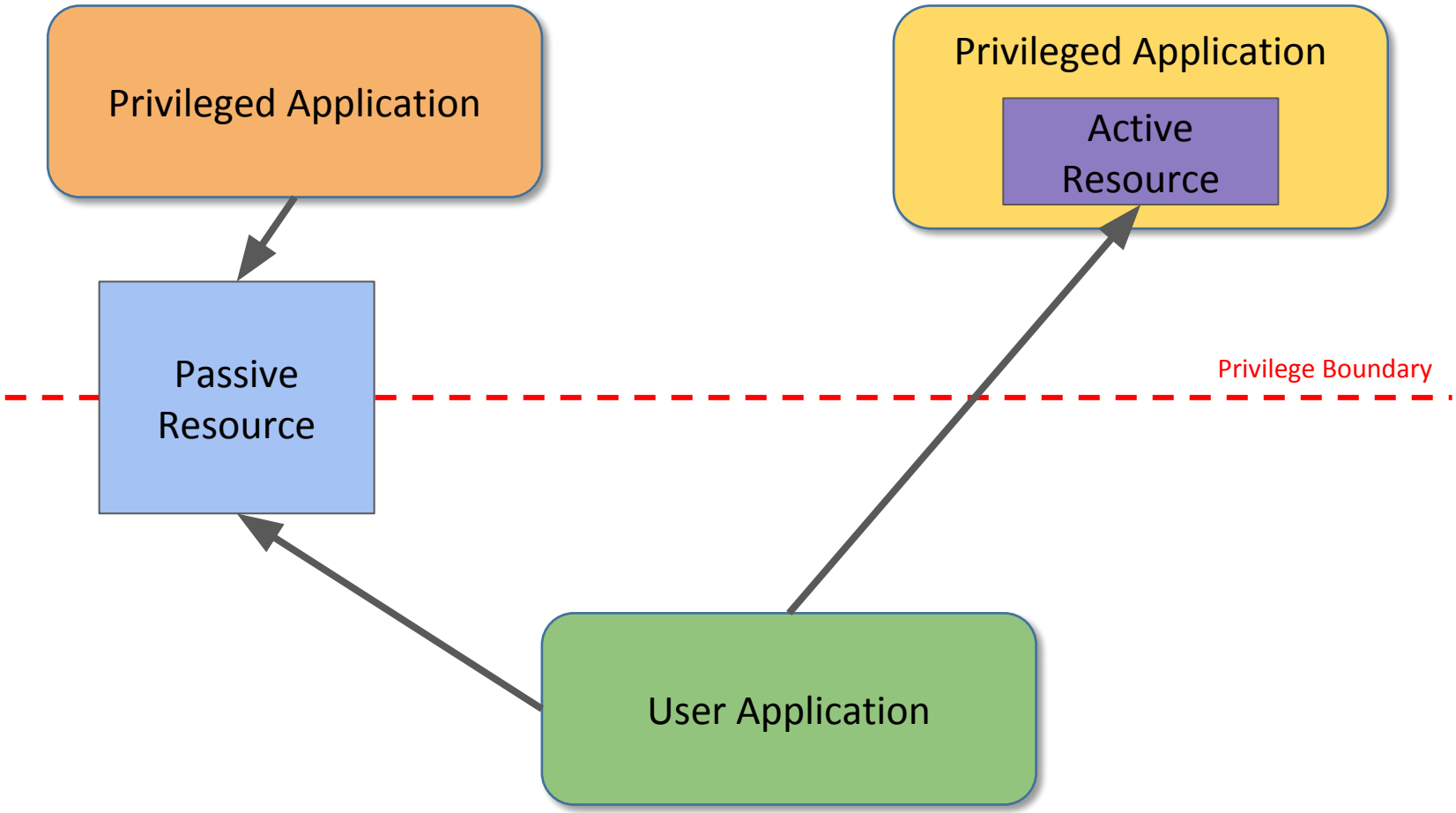
DEMO 2: Displaying Object Namespace



Tools and workbook at: https://goo.gl/8ardeY

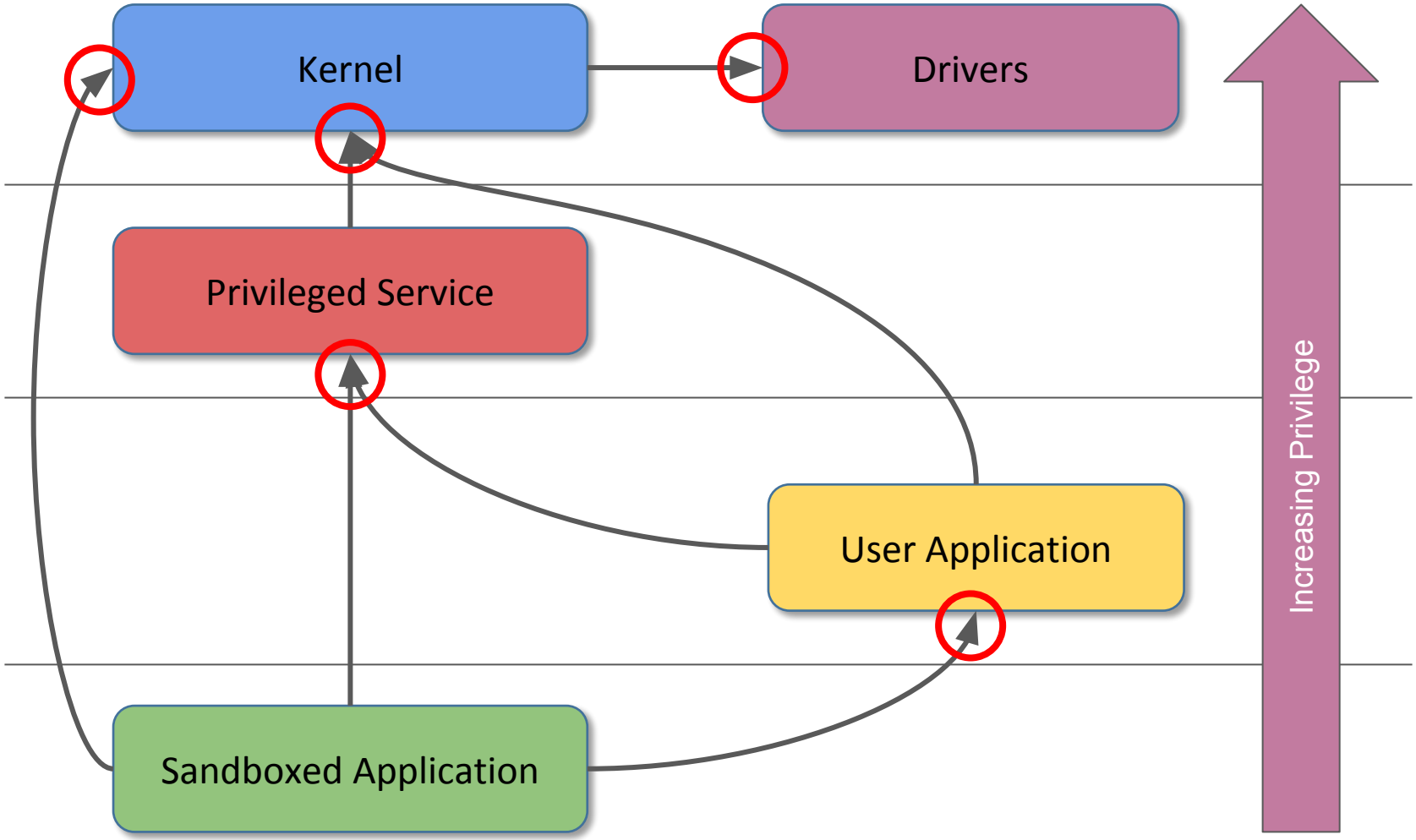
Hunting for Attack Surface

Passive vs Active Attack Surface

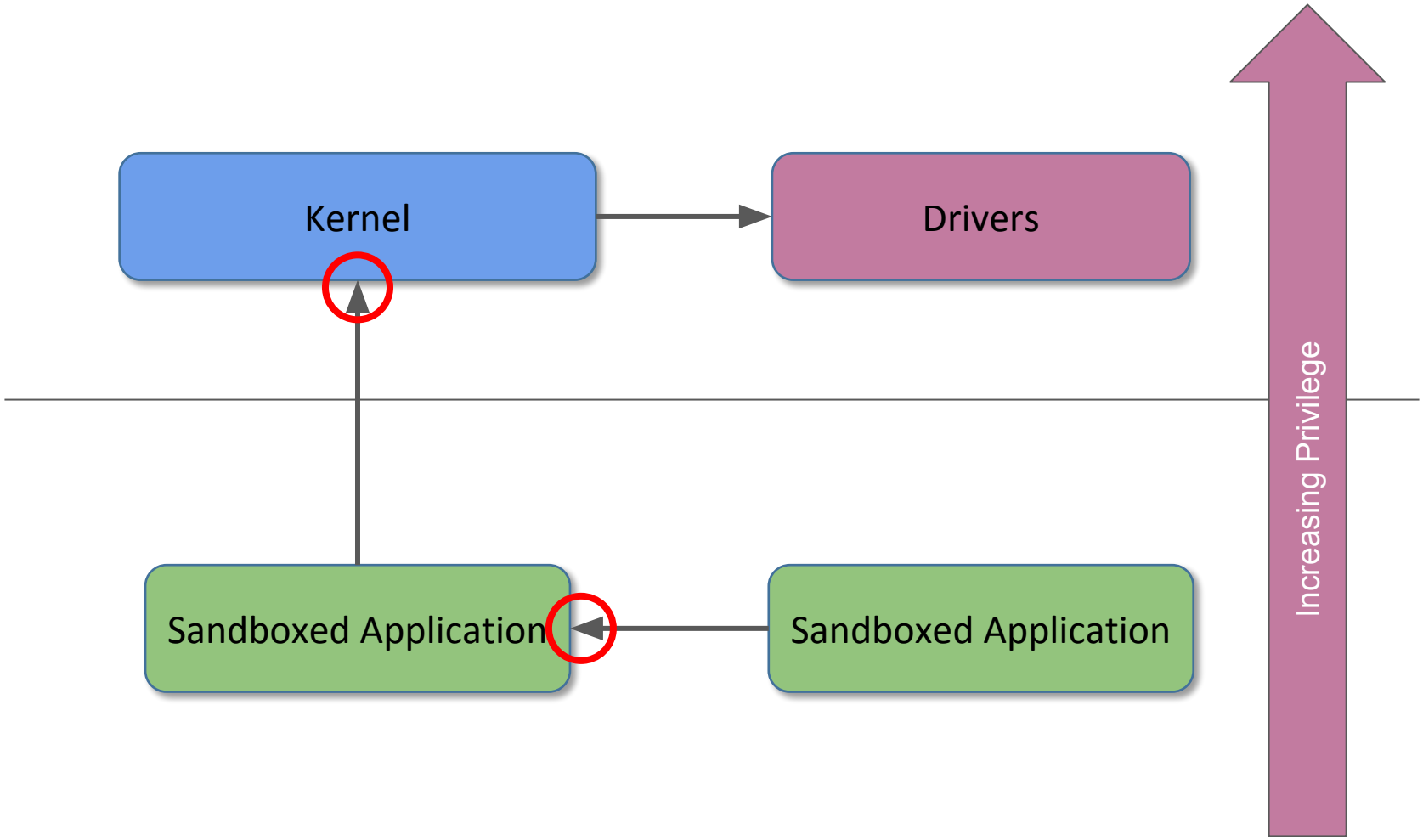


Tools and workbook at: https://goo.gl/8ardeY

Privilege Escalation Routes



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

Tools and workbook at: https://goo.gl/8ardeY

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 (such as Sections/Mutexes/Events etc.)

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

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" });
```

- "Get-AccessibleService -AccessRights Start" is your friend to find what you can start.

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}
```

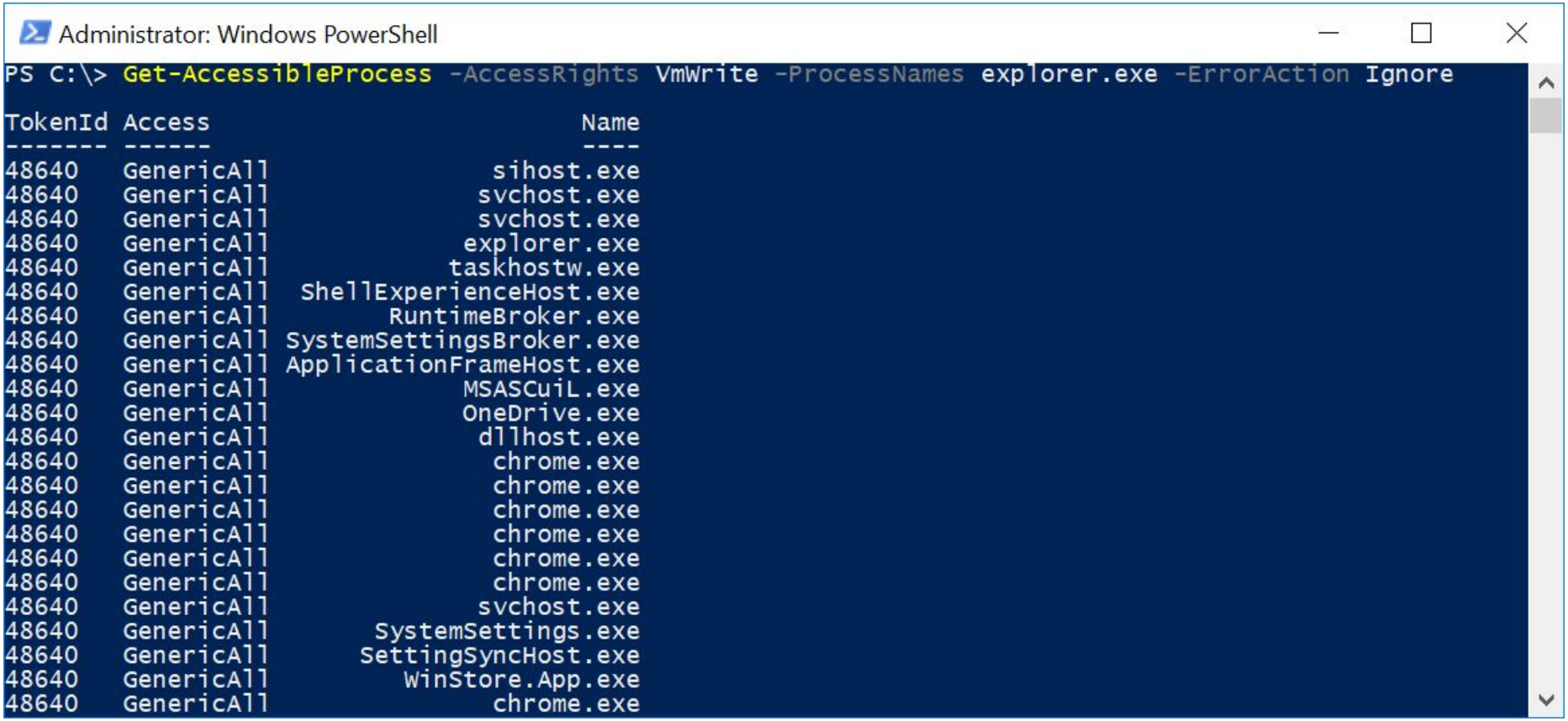
- Use the following C++ code to start the WebClient service

```
const GUID _MS_Windows_WebClntLookupServiceTrigger_Provider =
        { 0x22B6D684, 0xFA63, 0x4578,
           { 0x87, 0xC9, 0xEF, 0xFC, 0xBE, 0x66, 0x43, 0xC7 } };
REGHANDLE Handle;

if (EventRegister(
        &_MS_Windows_WebClntLookupServiceTrigger_Provider,
        nullptr, nullptr, &Handle) == ERROR_SUCCESS) {
        EVENT_DESCRIPTOR desc;
        EventDescCreate(&desc, 1, 0, 0, 4, 0, 0, 0);
        EventWrite(Handle, &desc, 0, nullptr) == ERROR_SUCCESS;
        EventUnregister(Handle);
}
```

Tools and workbook at: https://goo.gl/8ardeY

DEMO 3 - Enumerating Accessible Resources

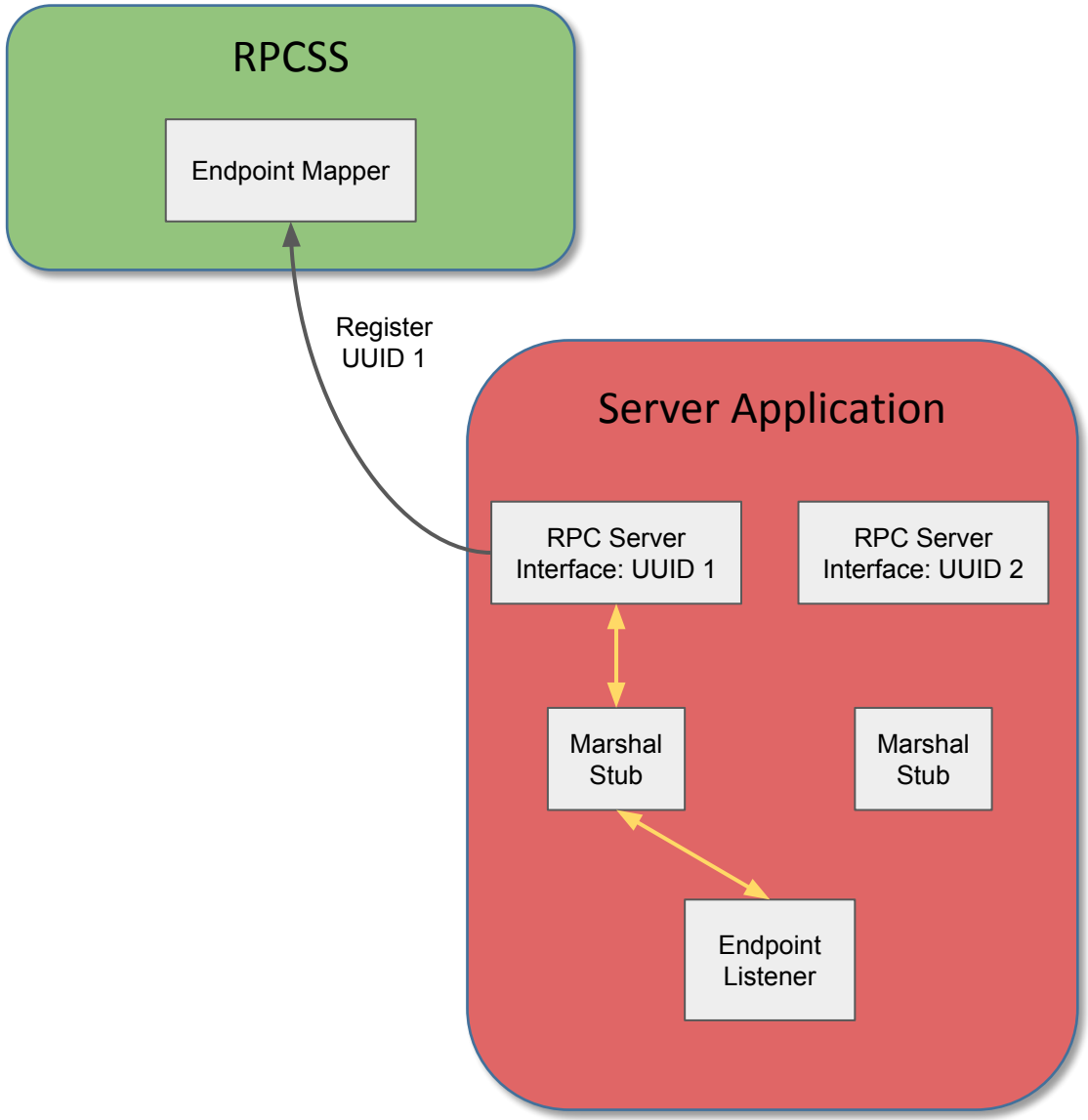


RPC Services

- 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

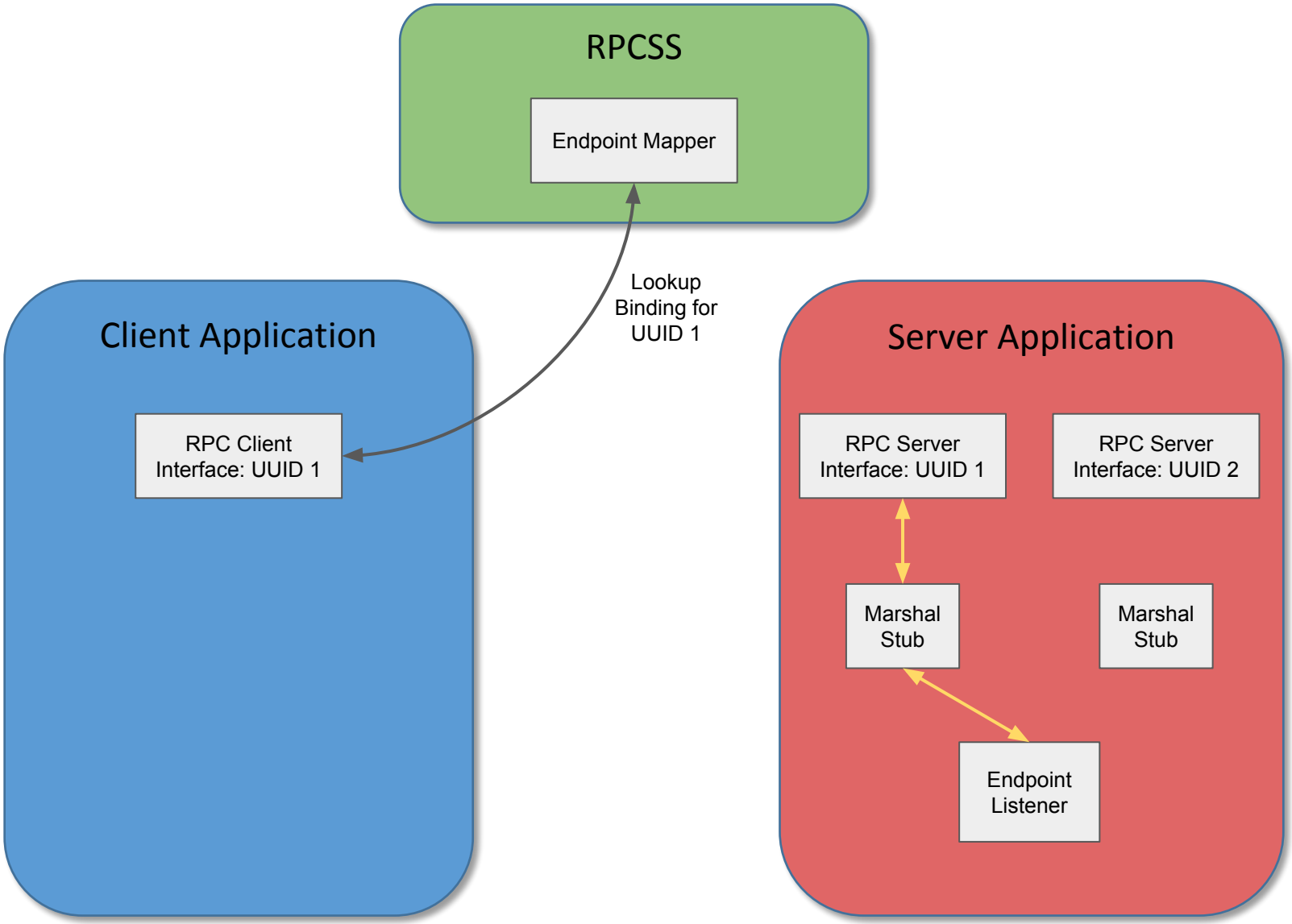
Tools and workbook at: https://goo.gl/8ardeY

Windows RPC Architecture



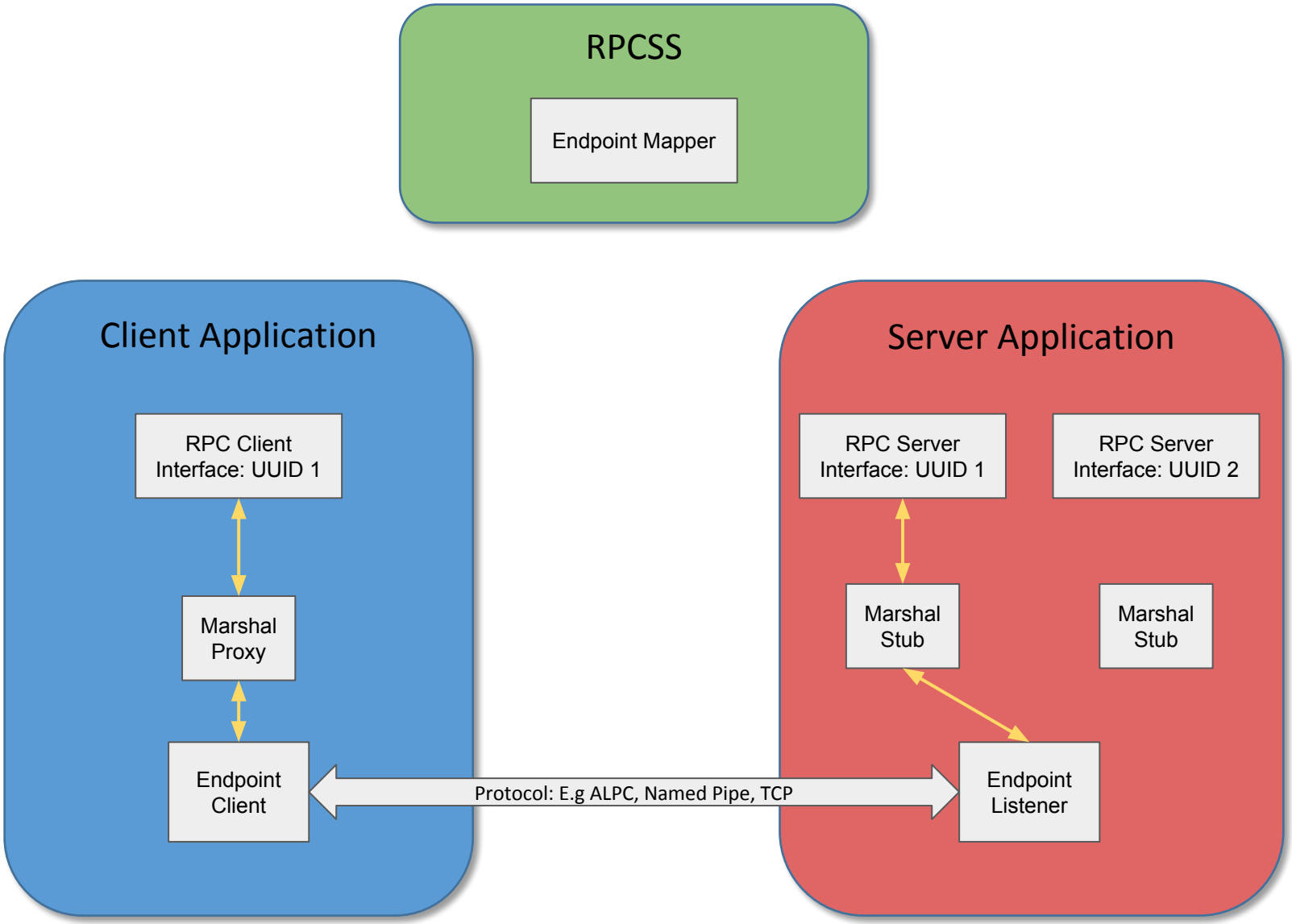
Tools and workbook at: https://goo.gl/8ardeY

Windows RPC Architecture



Tools and workbook at: https://goo.gl/8ardeY

Windows RPC Architecture



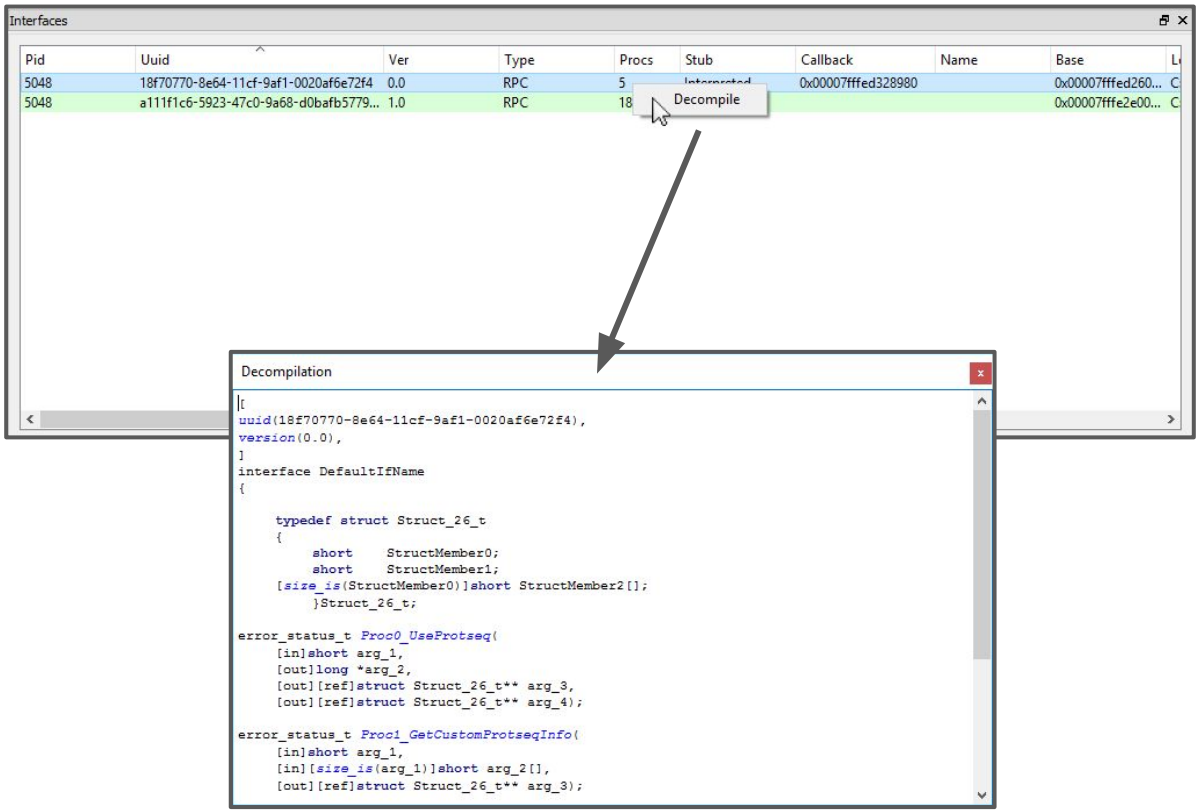
Network Data Representation (NDR)

- 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)

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

Working with RPC Interfaces

- RPCView Supports basic Decompilation of interface definitions.
- Right click interface and choose Decompile



RPC Endpoints

- RPC Supports multiple different endpoint protocols.
- Configured on server using RpcServerUseProtseqEp
- Configured on client using RpcStringBindingCompose

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

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

Protocol sequence

Optional Endpoint Name

Tools and workbook at: https://goo.gl/8ardeY

Protocol Sequences

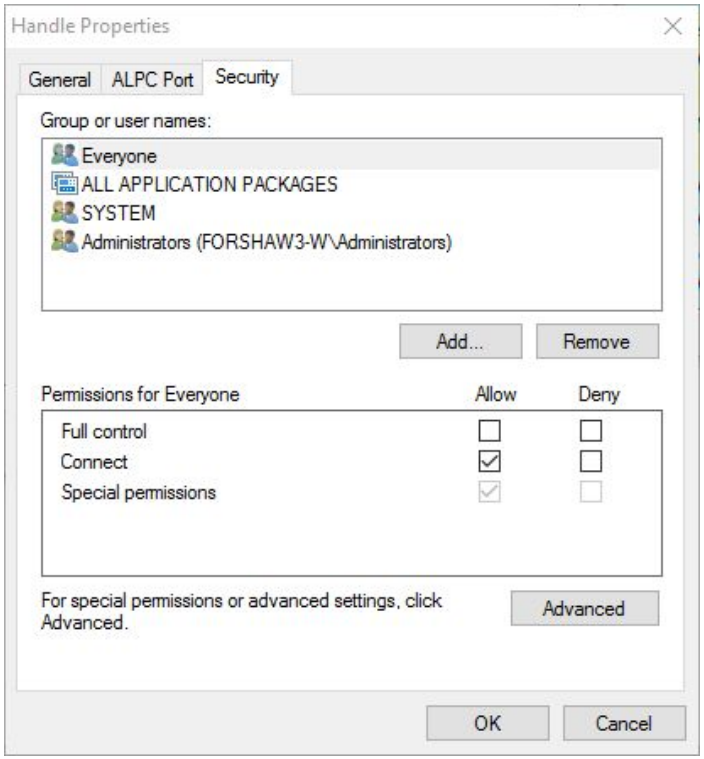
Protocol Sequence	*Optional Endpoint Name*	*Description*
ncalrpc	NAME	Local RPC (ALPC)
ncacn_np	\pipe\NAME	Windows Named Pipe
ncacn_ip_tcp	(port number)	TCP/IP
ncacn_ip_udp	(port number)	UDP/IP
ncacn_http	(port number)	HTTP

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

Tools and workbook at: https://goo.gl/8ardeY

RPC Security

Connect Time Security (Local)



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

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

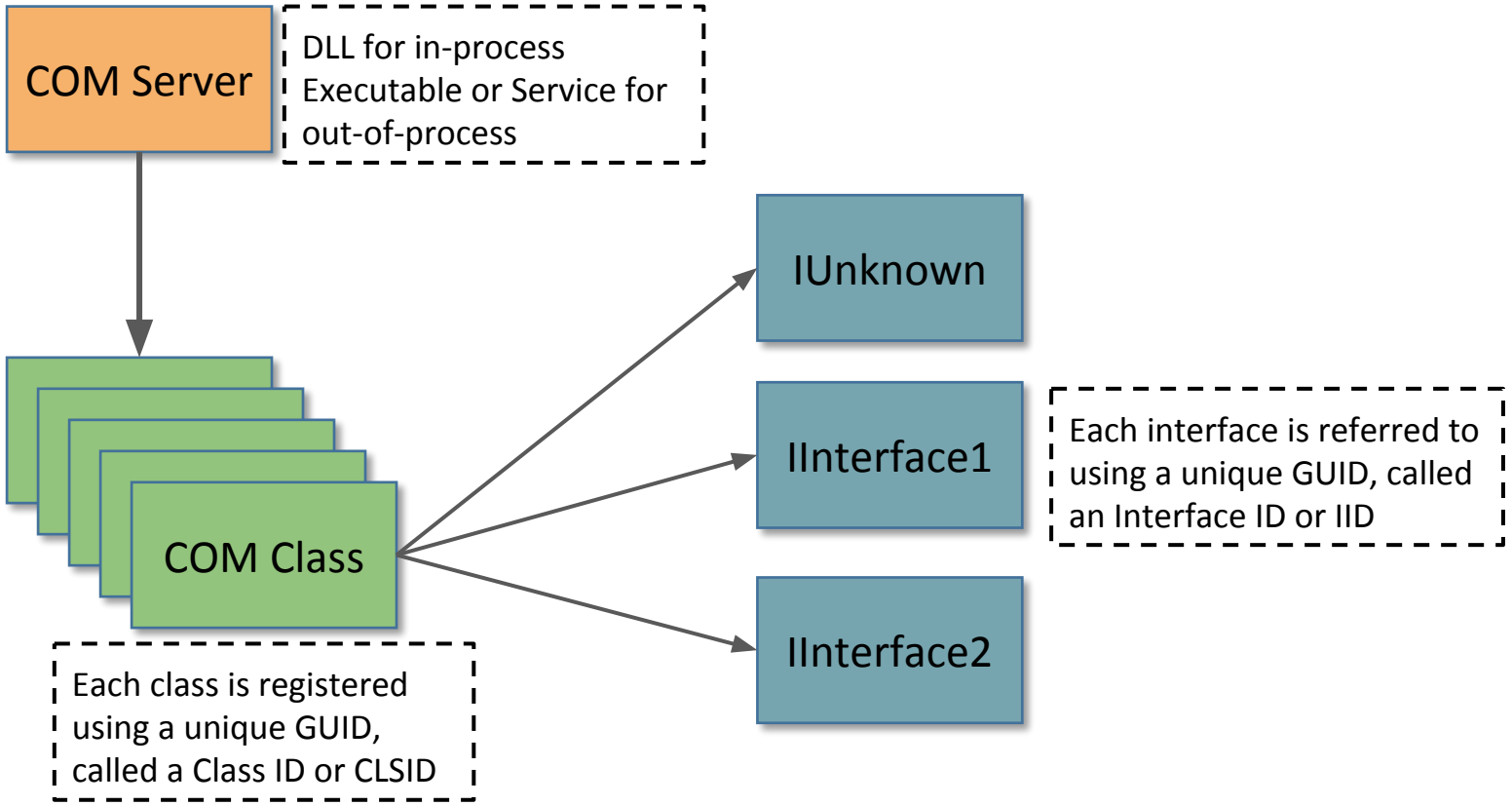
Tools and workbook at: https://goo.gl/8ardeY

Configuring RPCView Symbols

- Getting Symbol Information gives you a better idea on what functions are exported.
- Configure local path to symbols through Options -> Configure Symbols

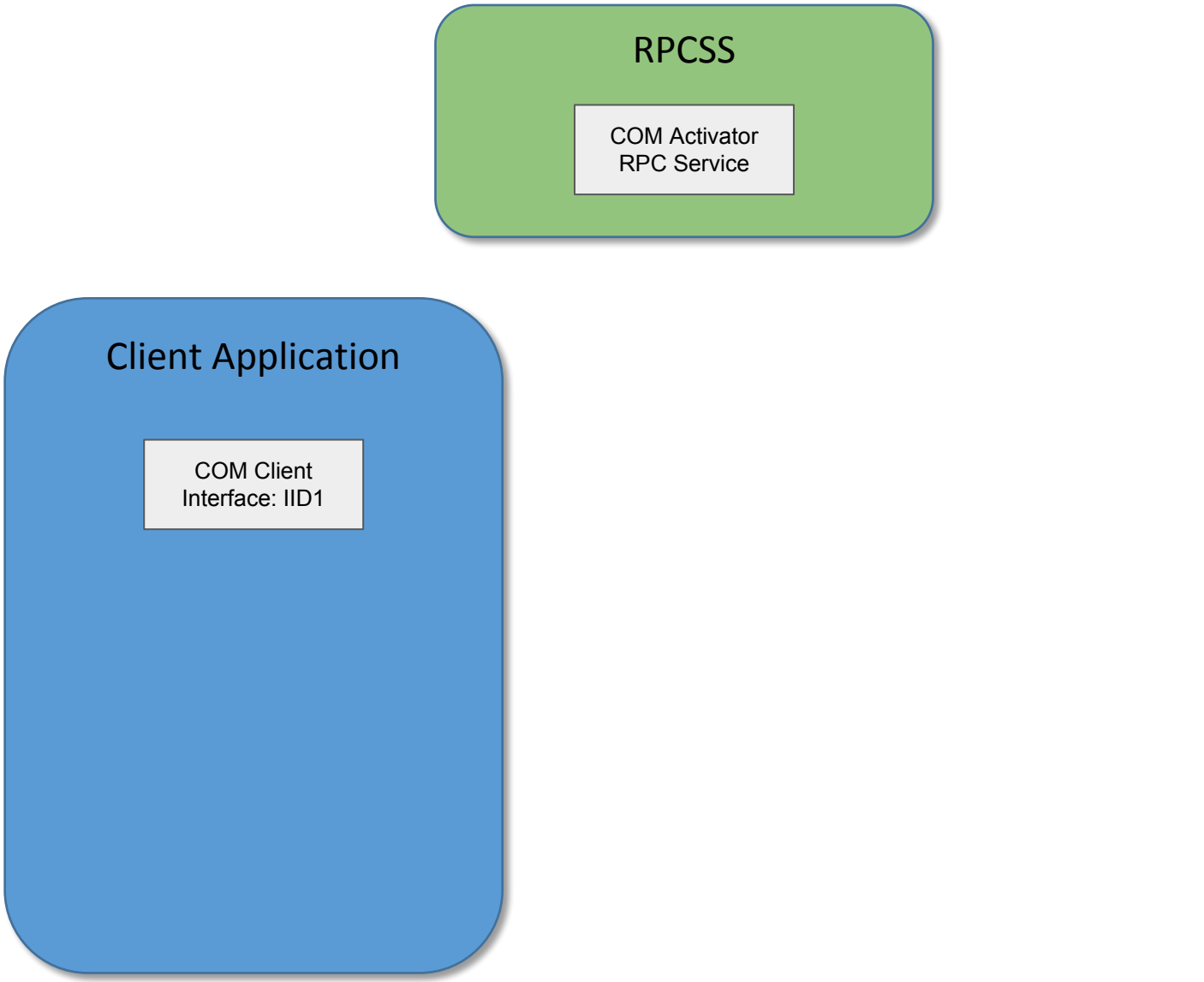
RPCView doesn't seem to work with symbol servers. So need to pull symbols manually. Use symchk from Debugging Tools for Windows, cache in a known directory such as c:\symbols.
symchk /s srv*c:\symbols*https://msdl.microsoft.com/download/symbols c:\windows\system32*.dll

COM Services



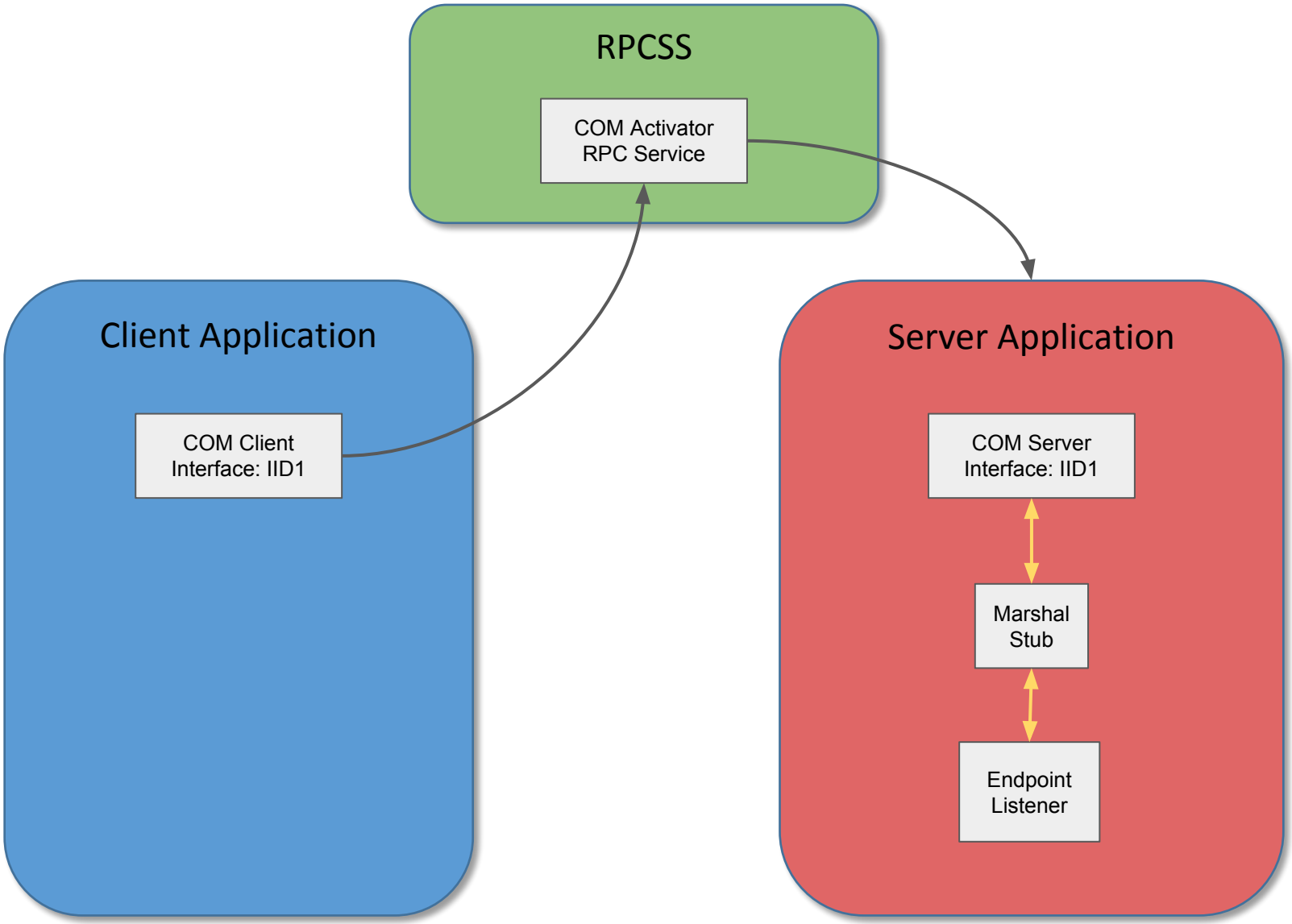
Tools and workbook at: https://goo.gl/8ardeY

COM Activation

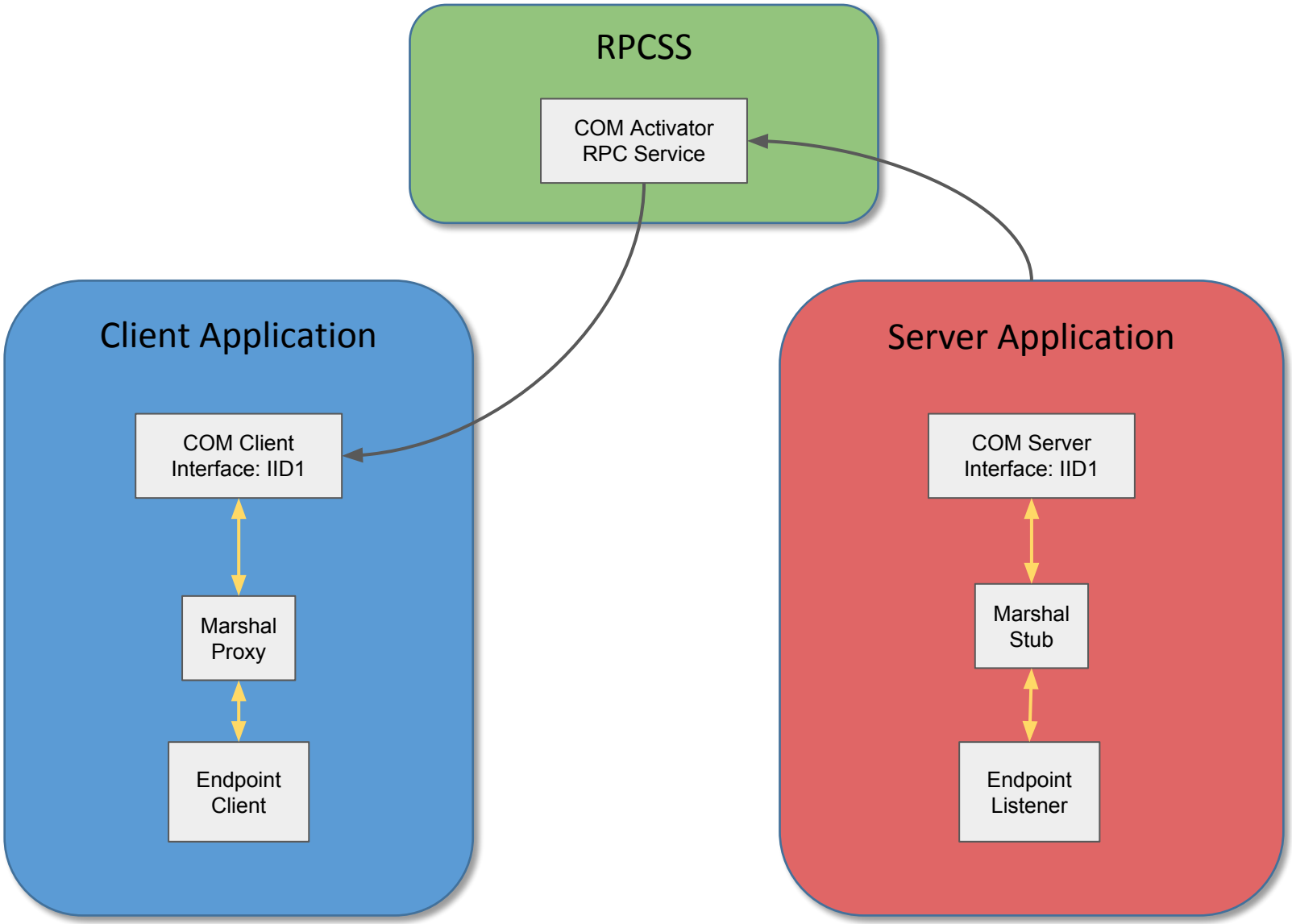


Tools and workbook at: https://goo.gl/8ardeY

COM Activation

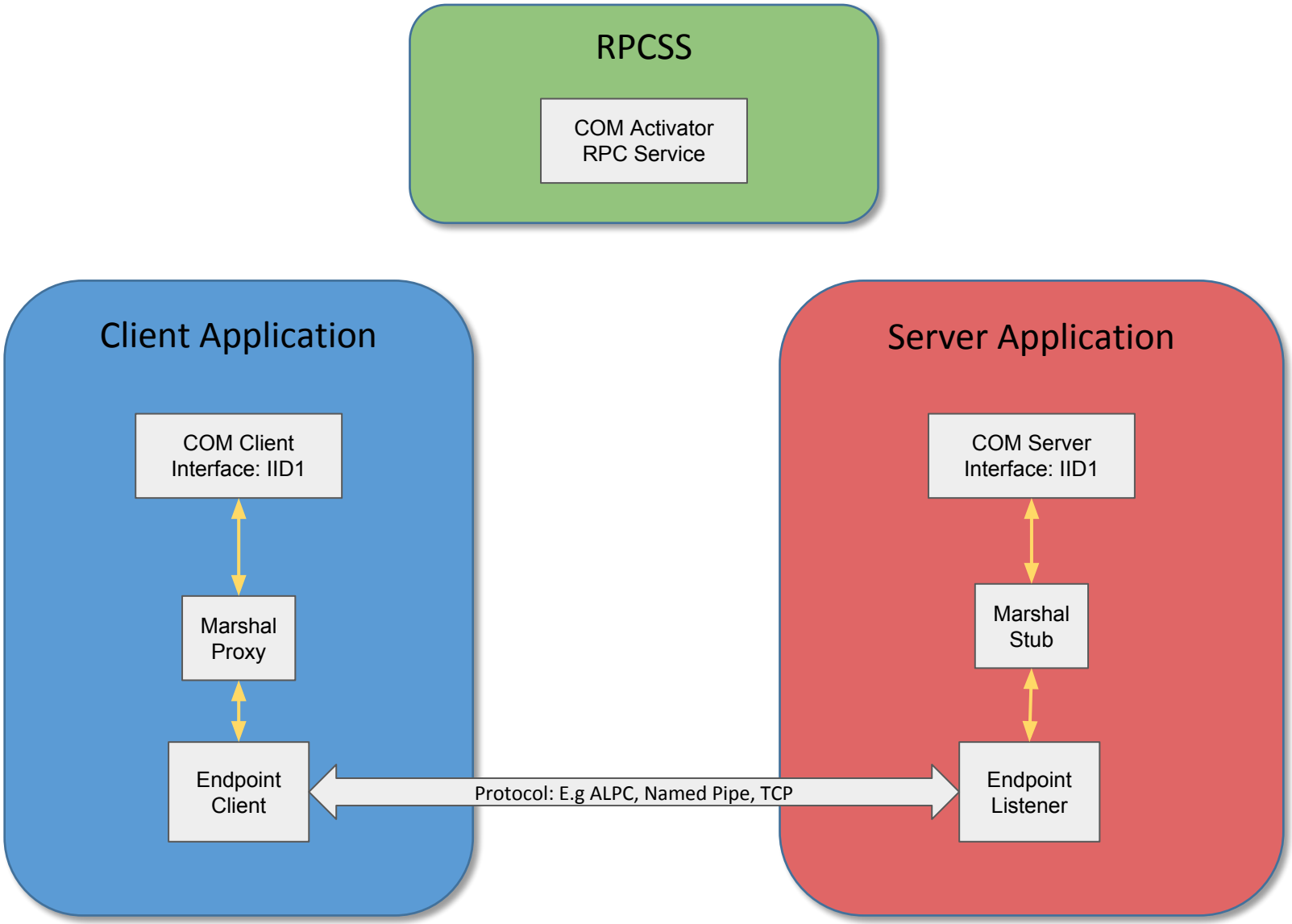


COM Activation



Tools and workbook at: https://goo.gl/8ardeY

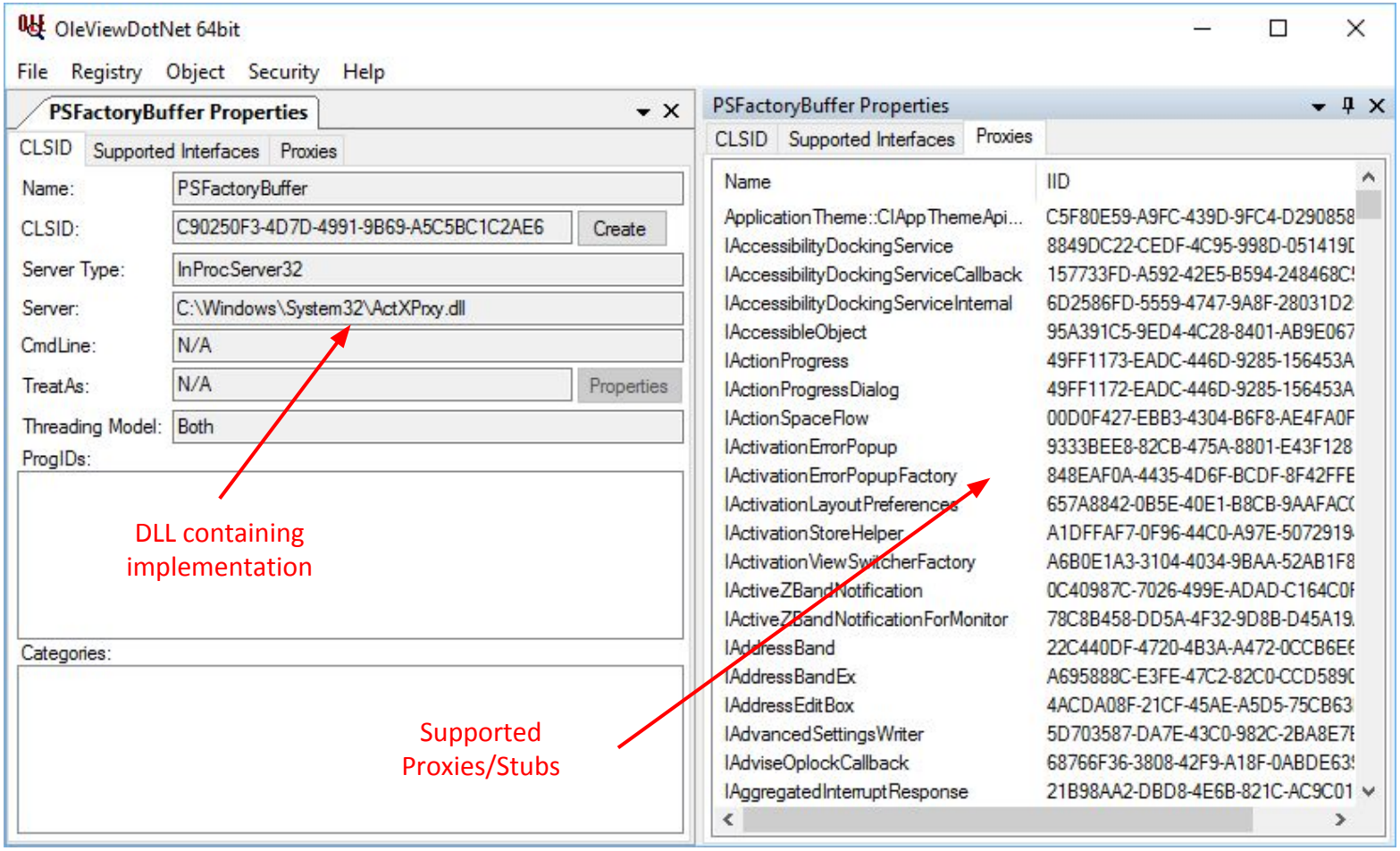
COM Activation



Tools and workbook at: https://goo.gl/8ardeY

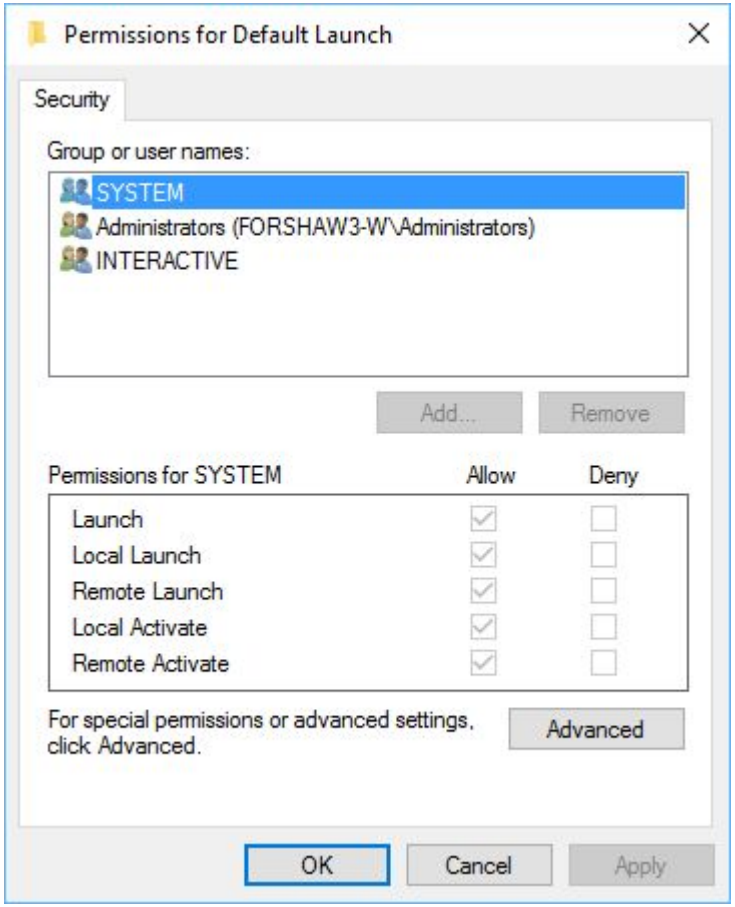
Proxies and Stubs

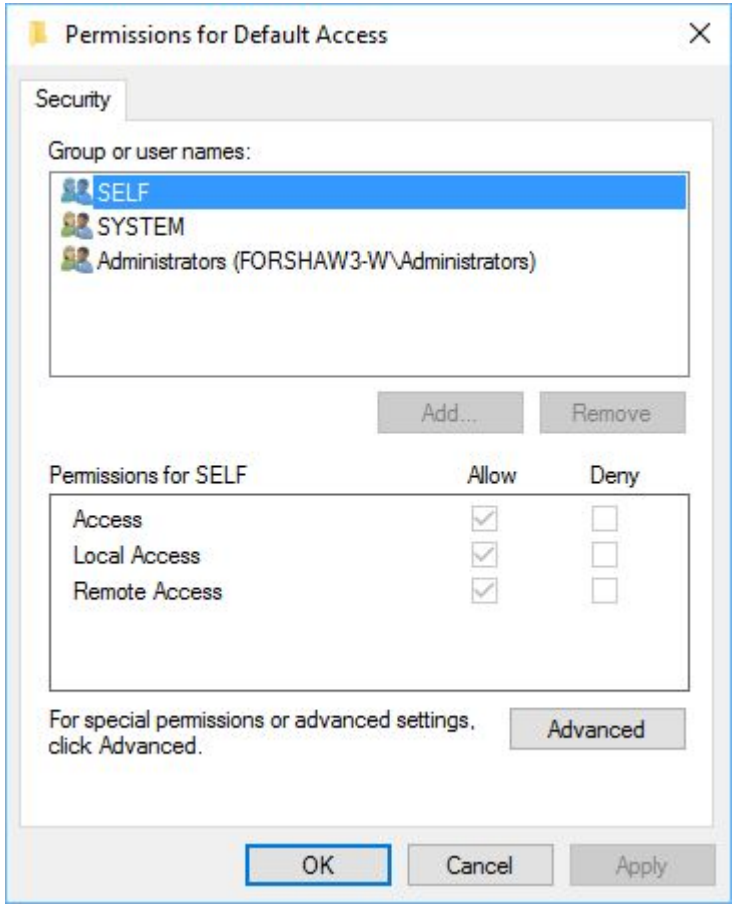
- Like RPC, COM services must define their interface using an IDL file. This must be registered.



Tools and workbook at: https://goo.gl/8ardeY

COM Security



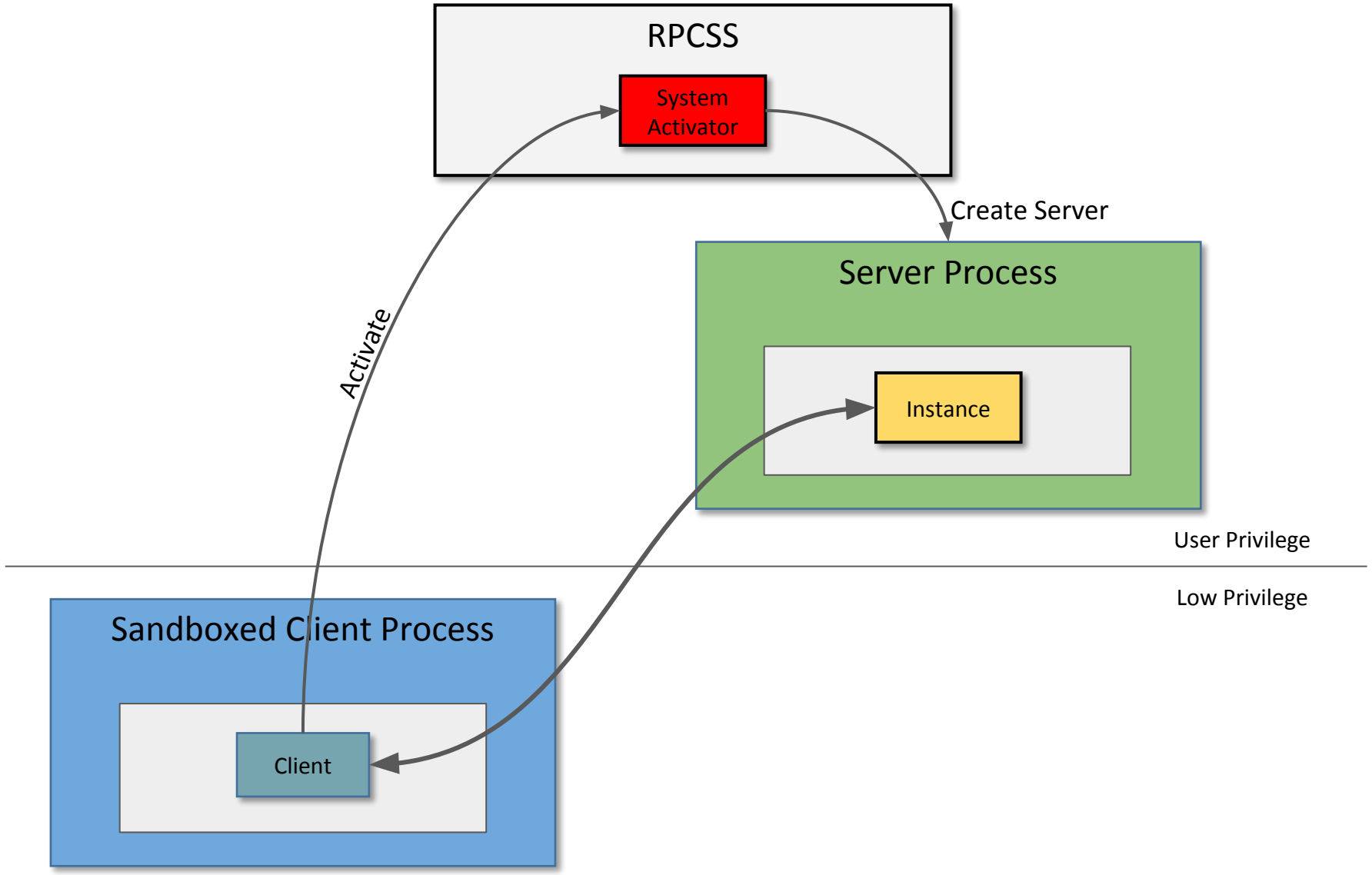


Launch = Create a new instance of the server.
Activate = Create new object on existing server.
Enforced in RPCSS

Access = Call methods on existing objects.
Enforced in Server Process

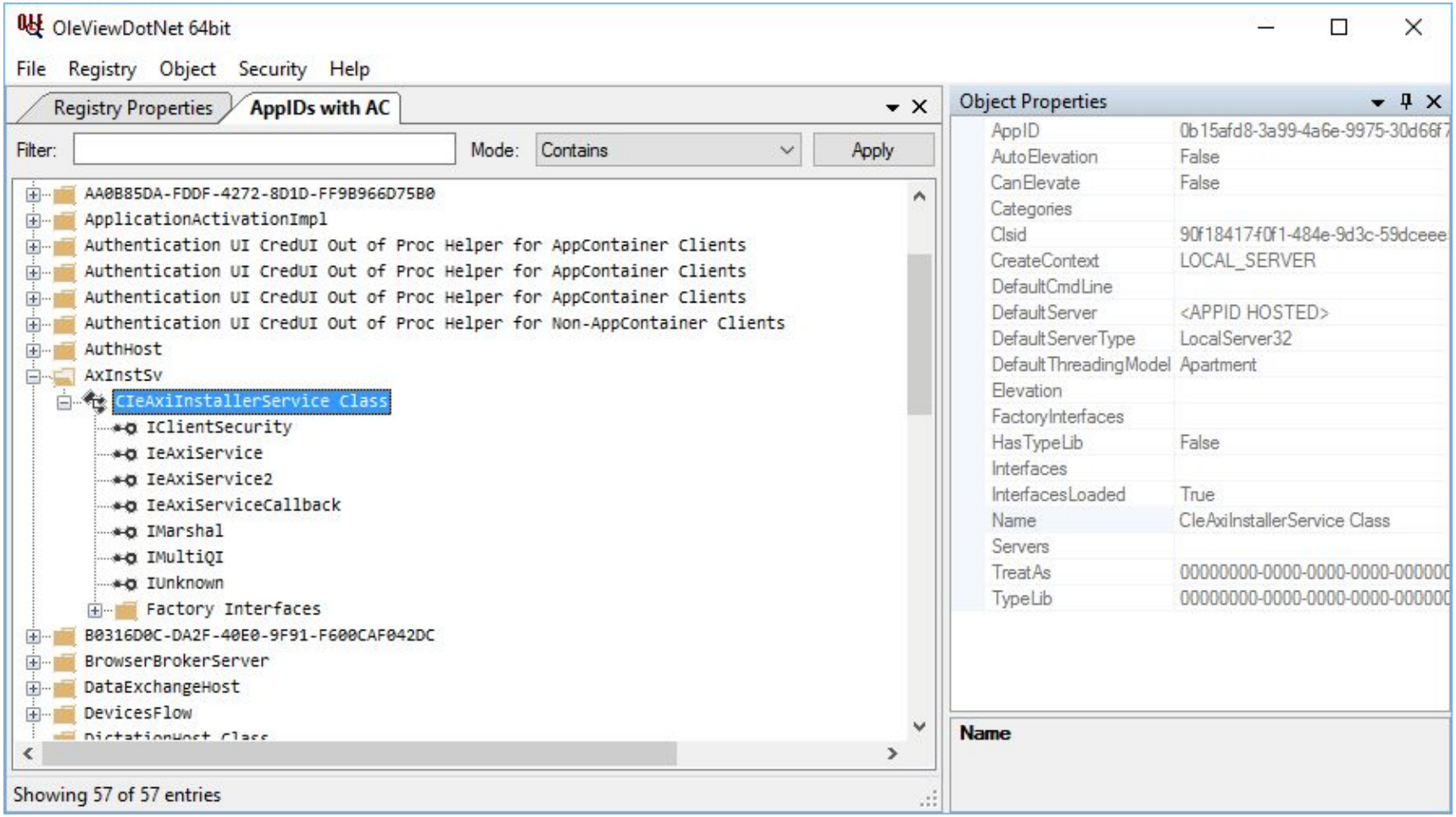
Tools and workbook at: https://goo.gl/8ardeY

RunAs Interactive User



Tools and workbook at: https://goo.gl/8ardeY

DEMO 4: Inspecting Accessible COM Services



Tools and workbook at: https://goo.gl/8ardeY

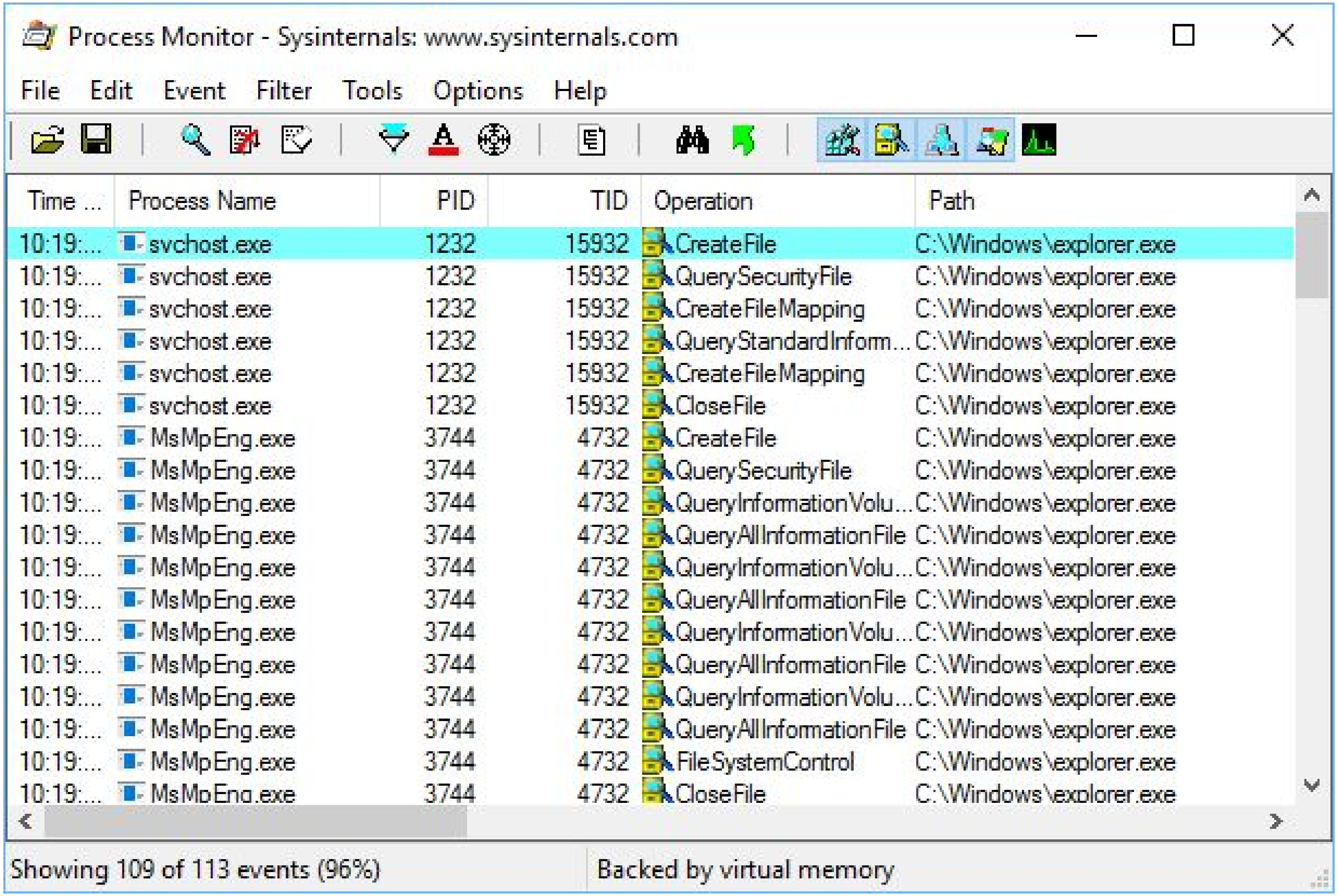
Bug Classes and Exploitation

Example Vulnerabilities

- Toolkit comes with some example vulnerabilities that we can exploit.
  - *LogicalEoPWorkshopDriver* - Simple kernel driver containing a number of logical vulnerabilities
  - *RpcServer* - Simple RPC server containing a number of logical vulnerabilities
  - *COMServer* - Simple COM server in .NET to demonstrate COM based vulnerabilities, is also a client to exploit the an IStorage vulnerability.
- Exploitation Tools
  - *DemoClient* - Simple interface to “exploit” the majority of vulnerabilities
  - *ExploitDotNetDCOMSerialization* - Tool we’ll use to exploit the COM Server via .NET DCOM

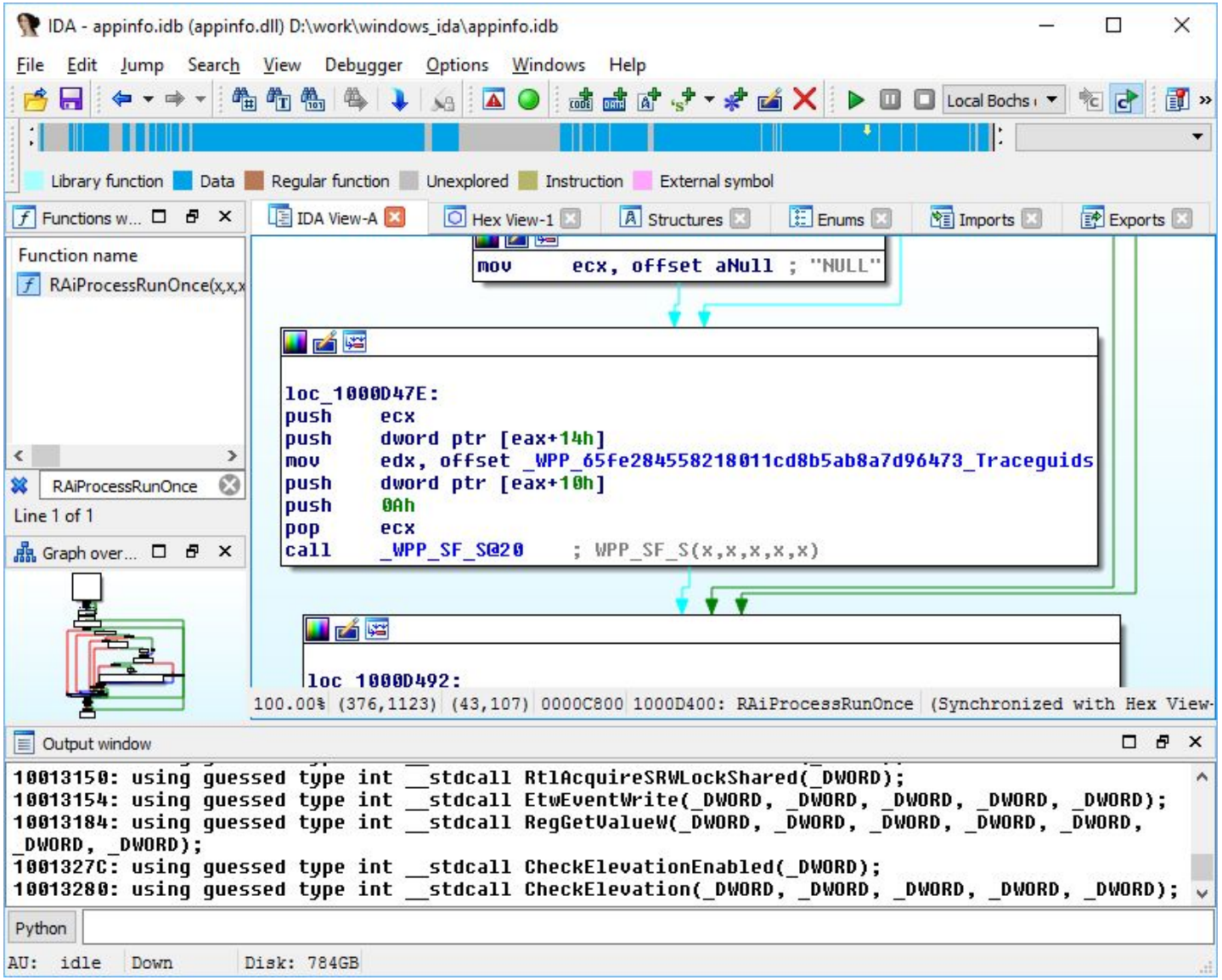
Tools and workbook at: https://goo.gl/8ardeY

Dynamic Analysis with Process Monitor



Tools and workbook at: https://goo.gl/8ardeY

Reverse Engineering with IDA Pro



Tools and workbook at: https://goo.gl/8ardeY

Path Canonicalization

```
bool TestLoadLibrary(const wchar_t* name)
{
  wstring full_path = L"C:\\Windows\\" + name;
  HMODULE hModule = LoadLibrary(full_path.c_str());
  if (hModule != nullptr)
  {
    printf("Loaded module: %p\n", hModule);
    FreeLibrary(hModule);
    return true;
  }
  return false;
}
```

No verification on name

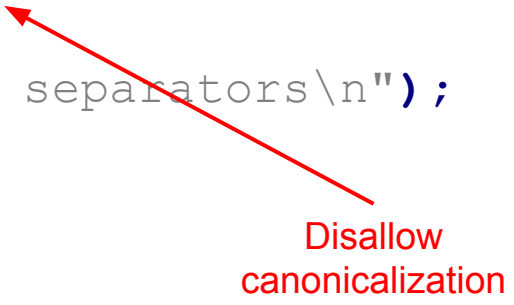
Tools and workbook at: https://goo.gl/8ardeY

DEMO 5 : Exploiting Path Canonicalization

Tools and workbook at: https://goo.gl/8ardeY

Insecure Path Usage

```
bool TestLoadLibraryCanonical(const wchar_t* name)
{
  if (wcschr(name, '\\') || wcschr(name, '/'))
  {
    printf("Error, name contains path separators\n");
    return false;
  }
  return TestLoadLibrary(name);
}
```

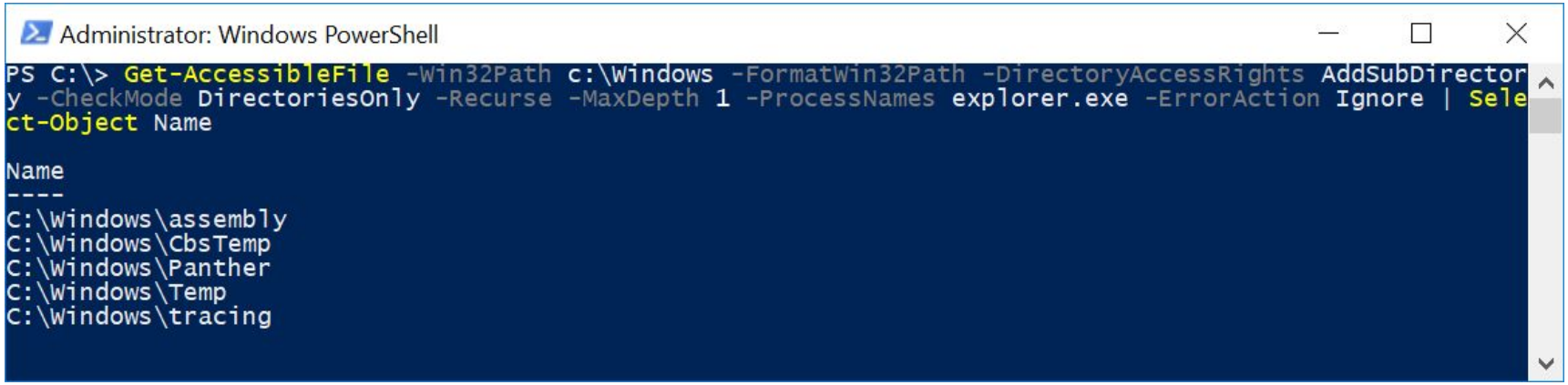


- All paths will be c:\windows\name
- We can't write to c:\Windows, or can we?

Tools and workbook at: https://goo.gl/8ardeY

NTFS Alternate Data Streams (ADS)

- ADS allows you to create substreams on NTFS files by separating using the ':' separtor
  - E.g. abc:xyz is stream named 'xyz' on the existing file 'abc'
- Also works for directories as if we've got *AddSubDirectory* access



C:\Windows\Tracing we can write an ADS to!

Tools and workbook at: https://goo.gl/8ardeY

DEMO 6: Exploiting Named Streams

Tools and workbook at: https://goo.gl/8ardeY

File Time-of-check Time-of-use

```
bool TestLoadLibraryTocTou(const wchar_t* lib_path) {
  if (VerifyEmbeddedSignature(lib_path)) {
    HMODULE hModule = LoadLibrary(lib_path);
    if (hModule != nullptr) {
      printf("Loaded module: %p\n", hModule);
      FreeLibrary(hModule);
      return true;
    }
  }
  return false;
}
```

Verifies that the DLL is signed

Load the library path is signed

Exploiting TOCTOU

- A few different ways to exploit:
  - Race condition between check time and library loading
    - We could rewrite the file in between the check and the load.
  - Exploit differing path parsing behaviours between functions
    - LoadLibrary will search the PATH for a filename which isn't an absolute path
    - Accessing a file takes path verbatim, LoadLibrary plays some games with extensions

lpFileName [in]
...
If the string specifies a module name without a path and the file name extension is omitted, the function appends the default library extension .dll to the module name.
To prevent the function from appending .dll to the module name, include a trailing point character (.) in the module name string.

c:\abc becomes c:\abc.dll when loaded

Tools and workbook at: https://goo.gl/8ardeY

DEMO 7 : TOCTOU on Name

Tools and workbook at: https://goo.gl/8ardeY

File Time-of-check Time-of-use

```
bool TestLoadLibraryTocTouHardened(const wchar_t* lib_path)
{
  LPWSTR ext = PathFindExtensionW(lib_path);
  if (ext == nullptr || _wcsicmp(ext, L".dll") != 0)
    return false;

  HANDLE handle = CreateFile(lib_path, ...);
  if (!CheckFileIsInSystem(handle)) {
    return false;
  }
  return LoadLibrary(lib_path);
}
```

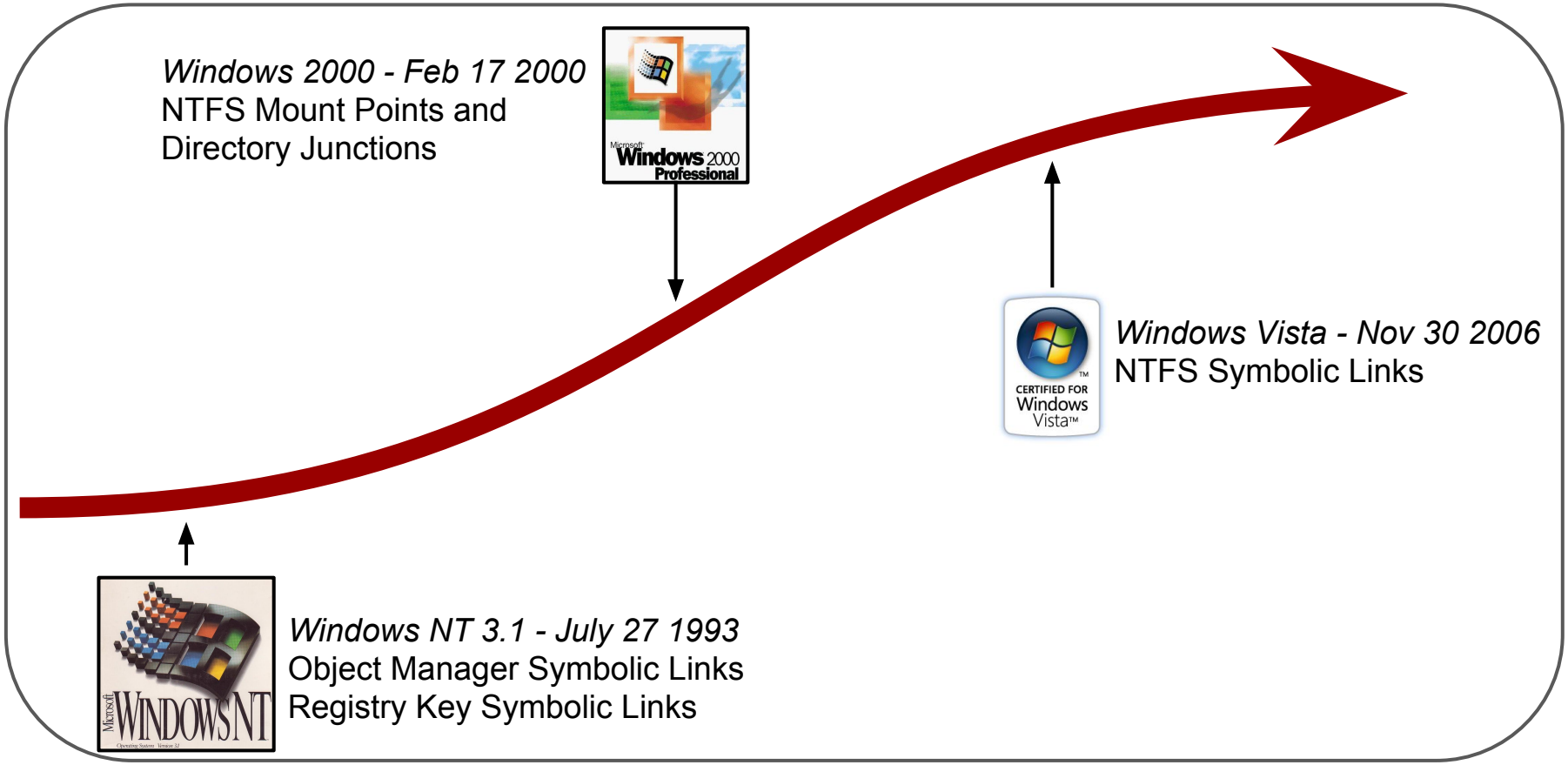
Ensure extension is .DLL

Lock file so can't be written to

Check opened file is in system directory

Tools and workbook at: https://goo.gl/8ardeY

Windows Symbolic Links



Opportunistic Locks (OPLOCK)

- Winning the TOCTOU race means either brute force or finding some way of timing the request.
- We can win the race in many cases using OPLOCKS
  - Locks a file to prevent access, can select Read/Write/Delete or Exclusive
  - Gets a callback when someone else tries to open the file. Closing handle allows that use to continue.

```
DeviceIoControl(g_hFile,
   FSCTL_REQUEST_OPLOCK_LEVEL_1,
   NULL, 0,
   NULL, 0,
   &bytesReturned,
   &g_o);
```

Note: Must use a Level 1 “Exclusive” lock for system files as normal user always gets Read sharing access.

Tools and workbook at: https://goo.gl/8ardeY

DEMO 8 : Symbolic Link TOCTOU

Tools and workbook at: https://goo.gl/8ardeY

Reading Files Under Impersonation

```
bool TestCreateProcess() {
  RpcImpersonateClient();
  WCHAR cmdline[] = L"c:\\windows\\notepad.exe";
  if (CreateProcess(cmdline, cmdline, ...)) {
    return true;
  }
  return false;
}
```

Calling User is Impersonated

Created process uses current process token, not impersonated token

- CreateProcess uses the current process's token by default, not any impersonation token
- However the file is accessed under the identity of the impersonated user
- Can we exploit this?

Current User's DosDevices Directory

- Current user's DosDevices directory is stored in \Sessions\0\DosDevices\X-Y
  - X-Y is the current user's login ID
- This is writable by the current user for obvious reasons
- We can re-direct C: to anywhere we like and get arbitrary process running with the identity of the RPC server

NOTE: This will won't work in a sandbox. It also used to work for DLLs but Microsoft fixed that glitch 😑

Tools and workbook at: https://goo.gl/8ardeY

DEMO 9 : DosDevices Redirect

Insecure Kernel Resource Access

```
NTSTATUS CreateFile(PUNICODE_STRING Path) {
  OBJECT_ATTRIBUTES obj_attr = { 0 };
  HANDLE Handle = NULL;
  ULONG AttributeFlags = OBJ_KERNEL_HANDLE;

  InitializeObjectAttributes(&obj_attr,
         Path, AttributeFlags);

  return ZwCreateFile(&Handle,
                      MAXIMUM_ALLOWED,
                      &obj_attr ...);
}
```

Setting attribute flags

Calling Zw* function, will transition to Kernel previous process mode

- Calls to Zw functions transition to kernel mode (when called from kernel code) which disables all security.
- Should be setting OBJ_FORCE_ACCESS_CHECK flag.

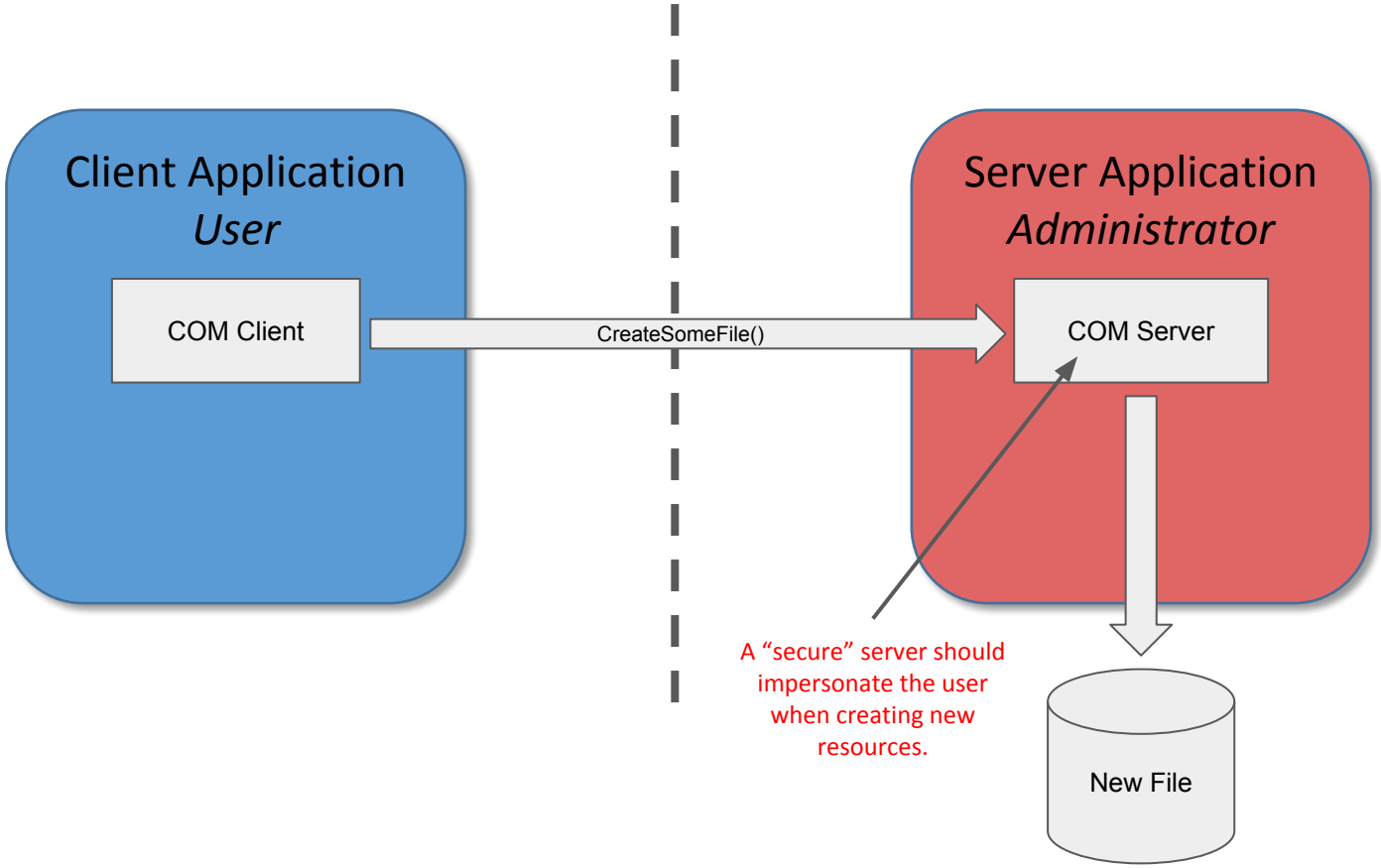
Resource Access

- File/Registry Key/Resource by default created with inherited security descriptor for parent
- However as long as call takes place inside the current process then the OWNER of the file will be the current user
- If inherited descriptor has CREATOR OWNER ACE we get those access rights
- Even if not we're owner so can open for WRITE_DAC access and modify at will

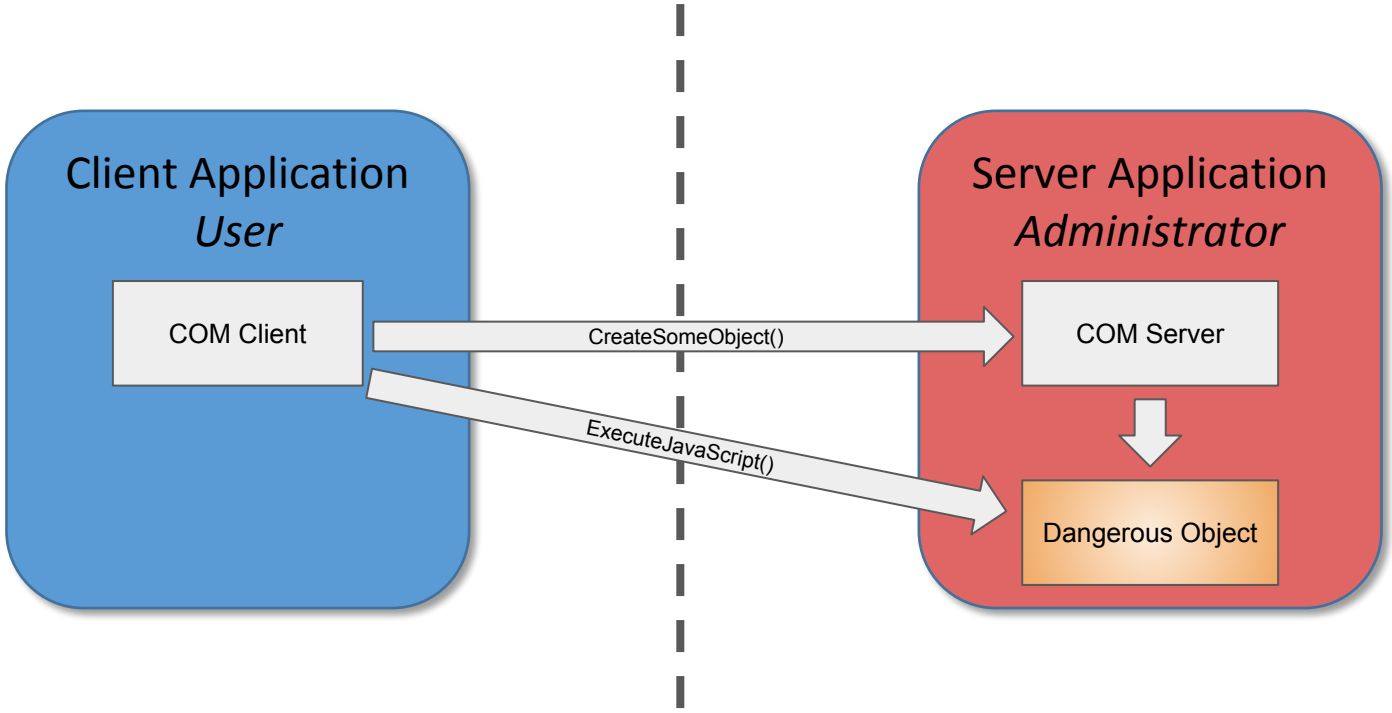
Tools and workbook at: https://goo.gl/8ardeY

DEMO 10 : Privileged Resource Creation

Bound COM Objects



Bound COM Objects



Tools and workbook at: https://goo.gl/8ardeY

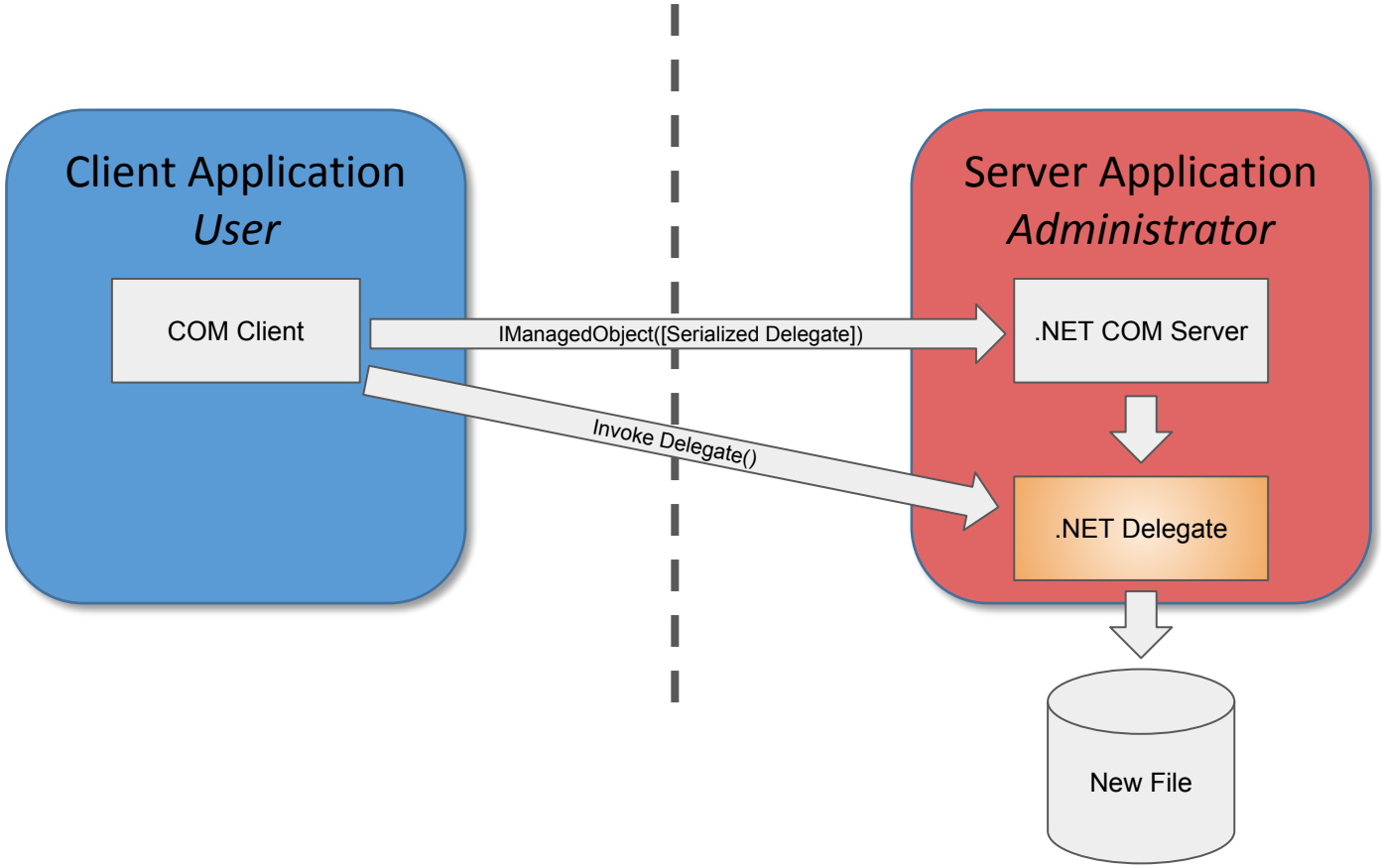
.NET DCOM Services

```
[ComVisible(true),
Guid("801445A7-C5A9-468D-9423-81F9D13FEE9B")]
public class COMService : ICOMInterface {
}

int cookie = reg_services.RegisterTypeForComClients(
    typeof(COMService),
    RegistrationClassContext.LocalServer,
    RegistrationConnectionType.MultipleUse);

Console.ReadLine();
reg_services.UnregisterTypeForComClients(cookie);
```

.NET DCOM Objects (Binary Serialization)



Tools and workbook at: https://goo.gl/8ardeY

DEMO 11 : .NET DCOM Elevation

Tools and workbook at: https://goo.gl/8ardeY

Resources

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

https://github.com/google/symboliclink-testing-tools

https://github.com/tyranid/oleviewdotnet

https://github.com/tyranid/ExploitDotNetDCOM

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