

Sabotage Sec



Background

This is not a Windows COM 101, readers are expected to have a decent understanding of COM and CLR hosting internals

Dotnet unamanged-api is not a stranger to Offssec tool developers, it enables developers to tinker with managed processes and the CLR itself from within a native process(say c/c++). You can see the whole list of supported interfaces listed categorically [here](#). Frequently I go through each category in the list, in the hope of finding any interesting methods exposed by COM interfaces which can be abused, especially those Hosting and debugging interfaces.

long story short, this post is about struggle i faced to solve a specific problem, it turned out to be a rabbit hole. I spend hours and hours googling and reading very old COM headers just to hit another brick wall. I hope this short post helps someone struggling to make COM work 😊

In the middle of cooking up a POC to do some shady dotnet stuff which involved AppDomains and arbitrary assembly loading, I use [ICorPublish](#) interface, which is a debug related COM interface for dotnet, to fetch AppDomain related information from managed processes. when working with ICorPublish you will get to see following interesting interfaces often – [ICorPublishProcess](#), [ICorPublishAppDomain](#) . The ICorPublishAppDomain has two methods GetID and GetName that fetch AppDomain ID and name of each AppDomain present in a managed process. The need for finding AppDomain related information arise when building tools, so this is one way of doing it. **SO OKAY WHAT IS THE ISSUE HERE? The issue is that you cannot fully trust apis like CreateInstance, CoCreateInstance and CoCreateInstanceEx. If the instance (COM object) returned by these apis is not compatible with CLR loaded in a target dotnet application, then behavior cannot be predicted, oftentimes it leads to failure when invoking methods through such instances. What is the reason behind this? They tend to load wrong version of mscordbi.dll from Microsoft.NET directory.**

This post is about how to fix such issue.. When I was playing around with ICorPublish interface for first time, I had to clue about this issue, so I am going to use this interface as a medium to address the issue, If this is your first time hearing it, I hope you find this post useful.

The problem

It is trivial to call CoCreateInstance (or CreateInstance) to instantiate COM class, as shown in the image below, ICorPublish is instantiated and made available to user through the variable pub. This is what we do normally right? RIGHT.. RIIGGHHHT?.

```
HRESULT hr;
ULONG32 puId = 0;
ICorPublish* pub;
CoInitialize(0);
hr = CoCreateInstance(__uuidof(CorpubPublish), 0, CLSCTX_INPROC_SERVER, __uuidof(pub), (void**)&pub);
```

Managed process enumeration

The ICorPublish interface exposes a method called [EnumProcess](#), this can be used to enumerate managed processes running on the system. It returns an instance of [ICorPublishProcessEnum](#), consider this as a list of information regarding each managed process enumerated where each process is represented by [ICorPublishProcess](#) interface. This is accessible to user through the variable pEnum.

```

ICorPublishProcessEnum* pEnum;
hr = pub->EnumProcesses(COR_PUB_MANAGEDONLY, &pEnum);
if (SUCCEEDED(hr))
{
}

```

Lets see how above code is going to behave at runtime! As shown in the image below, check the value of the variable pEnum – its all NULL. This is a clear indicator something went wrong behind the scene especially when the return value hr has S_OK status which in COM's terms all good. This is weird!

Name	Value	Type
CorpubPublish::EnumProcesses returned	S_OK	HRESULT
hr	S_OK	HRESULT
pEnum	0x0000027b5535e650 {m_pFirst=0x0000000000000000 <NULL> m_pCurrent=0x0000000000000000 ...	ICorPublishProcessEnum * {m...
pub	0x0000027b553598d0 {m_hPSAPI.dll={...} m_fpGetModuleFileNameEx=psapi.dll!0x00007ffdb32105... ICorPublish * {mscorlib.dll!Co...	ICorPublish * {mscorlib.dll!Co...

Lets test another method of ICorPublish interface – GetProcess.

Information about a specific process

The GetProcess method exposed by ICorPublish interface is similar to teh EnumProcess method seen earlier but difference is that it is used to fetch the information of a particular managed process via its PID. The variable proc stores the process information.

```

ICorPublishProcess* proc = nullptr;
hr = pub->GetProcess(pid, &proc);

if (SUCCEEDED(hr))
{
}

```

At runtime above code generates following error shown below. It says one or more arguments are invalid, also check value of variable proc which is null. The arguments fed into api are all valid then what does this error tell you? Sometimes errors like this will question your sanity.. LOL

hr	E_INVALIDARG One or more arguments are invalid.
pEnumDomains	0x0000000000000000 <NULL>
pid	0x0000e77c
proc	0x0000000000000000 <NULL>
pub	0x000001cd69f1abd0 {m_hPSAPI.dll={...} m_fpGetMod
puld	0x00000000

The Fix

My stupid ass spent almost a day on above issues, after hours of googling I accidentally landed on a random ass a Processhacker plugin [source](#). **Absolutely a blessing to find below comment!!** Talk about the luck, the plugin uses ICorPublish interface. LOL

```
    CLR_V4 = TRUE,
}

// Using CoCreateInstance always seems to create a v2-compatible class, but not a v4-compatible one.
// For v4 we have to manually load the correct version of mscoredbi.dll.
if (clrV4)
```

As I mentioned in the beginning, if COM object is incompatible with CLR version, COM method fails or worse behave indifferently leaving no signs of any issues, which is the case with EnumProcess method as seen above.

In the processHacker code there a mention of IClassFactory_CreateInstance, note that it is written in C, when I used the same in C++ code, it gave me an error (error – not defined), I thought its a missing header sighhhhhh.....after a bit of digging, I got the header where it is defined – [Unknown.h](#)

Adding of the header in my project did not solve the issue, so I dived into header file and saw below code. The IClassFactory_CreateInstance is a macro made for C not C++, the COBJMACROS definition confirms it. Nevertheless seeing the definition of IClassFactory_CreateInstance macro below, it is very easy to reproduce the C++ code.

```
496     #ifdef COBJMACROS
497
498
499     #define IClassFactory_QueryInterface(This,riid,ppvObject)    \
500         ( (This)->lpVtbl -> QueryInterface(This,riid,ppvObject) )
501
502     #define IClassFactory_AddRef(This)    \
503         ( (This)->lpVtbl -> AddRef(This) )
504
505     #define IClassFactory_Release(This)    \
506         ( (This)->lpVtbl -> Release(This) )
507
508
509     #define IClassFactory_CreateInstance(This,pUnkOuter,riid,ppvObject)    \
510         ( (This)->lpVtbl -> CreateInstance(This,pUnkOuter,riid,ppvObject) )
511
512     #define IClassFactory_LockServer(This,fLock)    \
513         ( (This)->lpVtbl -> LockServer(This,fLock) )
514
515     #endif /* COBJMACROS */
516
517
518     #endif /* C style interface */
519
```

The solution is :

- Load correct mscordbi.dll
- Use DllGetClassObjectInternal Win32 api to interface pointer to target class, in our case ICorPublish COM class.
- Call CreateInstance on above pointer to properly instantiate the COM class.

i have written C++ code shown below to manually use correct dotnet core library to activate a COM class, the variable Publish has CLR v4 compatible ICorPublish instance.

```
HRESULT result;
ICorPublish* Publish = nullptr;
HMODULE mscordbiDllBase;

mscordbiDllBase = LoadLibrary(L"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\mscorlib.dll");

//Function pointer - typedef HRESULT(__stdcall* dllGetClassObject)(REFCLSID, REFIID, LPVOID*);
dllGetClassObject dw = (dllGetClassObject)GetProcAddress(mscordbiDllBase, "DllGetClassObjectInternal");
if (dw)
{
    IClassFactory* factory;
    result = (HRESULT)dw(CLSID_CorpubPublish_I, IID_IClassFactory, (LPVOID*)&factory);
    if (SUCCEEDED(result))
    {
        factory->CreateInstance(NULL, IID_ICorPublish_I, (void**)&Publish);
        factory->Release();
    }
}
```

Now we can use this object (Publish) to invoke interface methods discussed earlier EnumProcess and GetProcess.

Now when executing the new code, pEnum returned valid data after the call to EnumProcess interface method. Earlier this variable returned null values.

Name	Value
hr	S_OK
pEnum	0x00000229c748fe10 <...
pub	0x00000229c748a670 <...

Same is the case with method GetProcess interface method, the variable proc now returns valid data and hr is S_OK after call to the method.

hr	S_OK
pEnumDomains	0x0000000000000000 <...
pid	0x0000e77c
proc	0x000002cff11c42f0 <...
pub	0x000002cff11ccde0 <...

Just for the sake of it, below code prints the AppDomain friendly name and associated ID.

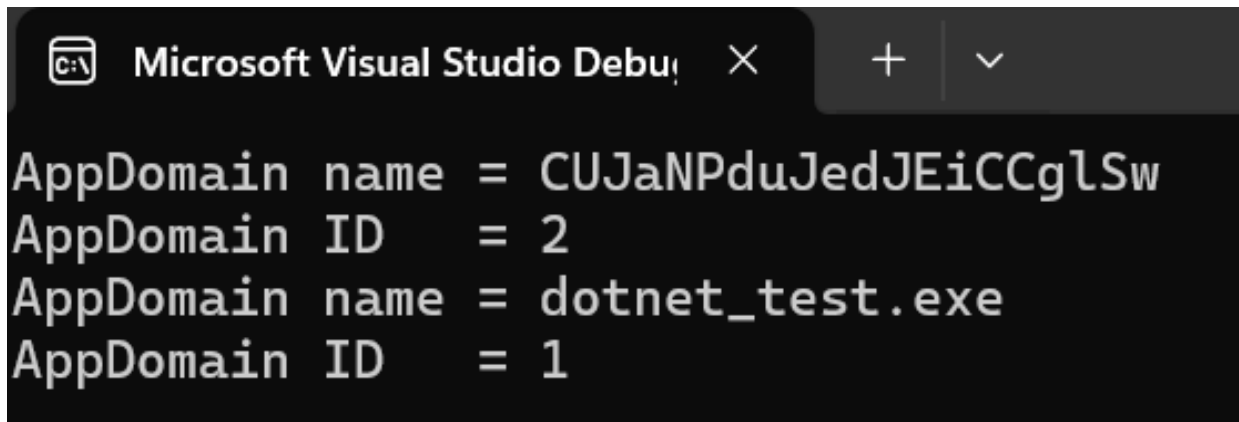
```

appDomains[j]->GetName(NAME_LEN, &size, name);
appDomains[j]->GetID(&puId);

if (size > 0)
{
    wprintf(L"AppDomain name = %s\n", name);
    printf("AppDomain ID    = %d\n", puId);
}

```

The domain CUJaNPduJedJEiCCglSw is created by CreateDomain method exposed by ICorRuntimeHost interface.



```

Microsoft Visual Studio Debug Console
AppDomain name = CUJaNPduJedJEiCCglSw
AppDomain ID    = 2
AppDomain name = dotnet_test.exe
AppDomain ID    = 1

```