.Net Reflection without System.Reflection.Assembly

This is a quick blog to cover an alternative technique to load a .Net Assembly without having to call the suspicious Assembly.LoadFile() or Assembly.Load() Functions.

Not too long ago I released a tool called RunDotNetDll32 to make it easier to execute methods from .Net DLLs without going through the process of loading them and executing them in PowerShell. It can be downloaded here. However, under the hood the application is still called System.Reflection.Assembly.LoadFile(). Below is a C# example of this process.

```
String path = ".\WheresMyImplant.dll";
String nameSpaceName = "WheresMyImplant";
String className = "Implant";
String method = "RunPowerShell";
String arguments = "$env:logonserver";
Assembly assembly = Assembly.LoadFile(Path.GetFullPath(path));
Type type = assembly.GetType(namespaceName + "." + className);
MethodInfo methodInfo = type.GetMethod(method);
Console.WriteLine((String)methodInfo.Invoke(null, arguments));
```

While this was functional, it was also not ideal if you're trying to avoid calling suspicious methods or APIs.

Enter type loading

There are a multitude of ways to call reflection, one of which is the System.Type.GetType() method. This particular method requires the DLL to be in one of two places: the same directory as the executing assembly or in the GAC. Using this as a search path, the Assembly's Fully Qualified Name can be passed as a parameter and be automatically loaded.

```
String path = ".\WheresMyImplant.dll";
String nameSpaceName = "WheresMyImplant";
String className = "Implant";
String method = "RunPowerShell";
String arguments = "$env:logonserver";
AssemblyName assemblyName =
AssemblyName.GetAssemblyName(Path.GetFullPath(path));
String fullClassName = String.Format("{0}.{1}", namespaceName, className);
Type type = Type.GetType(String.Format("{0}, {1}", fullClassName,
assemblyName.FullName));
MethodInfo methodInfo = type.GetMethod(method);
Console.WriteLine((String)methodInfo.Invoke(null, arguments));
```

There is nothing particularly special about System.Reflection.AssemblyName in this context, but it does save us some effort by automatically parsing the Assembly's Fully Qualified Name.

In PowerShell, the previous code could be equivalently executed with the following script:

```
$Path = ".\WheresMyImplant.dll";
$NameSpaceName = "WheresMyImplant";
$ClassName = "Implant";
$Method = "RunPowerShell";
$Arguments = "`$env:logonserver";
$Full_Path = [System.IO.Path]::GetFullPath($Full_Path);
$AssemblyName = [System.Reflection.AssemblyName]::GetAssemblyName($Path)
$Full_Class_Name = "$NameSpaceName.$ClassName"
$Type_Name = "$Full_Class_Name, $($AssemblyName.FullName)"
$Type = [System.Type]::GetType($Type_Name)
$MethodInfo = $Type.GetMethod($Method)
$MethodInfo.Invoke($null, $Arguments)
```

Again, note that the DLL being reflectively loaded needs to exists in the same directory as the executing assembly or in the GAC.

RunDotNetDll32 – Now with less Assembly.LoadFile()

Given this the option to run a method using Type.GetType() has been added along with other flags, doing away with the previously used positional parameters.

Using LoadFile

```
rundotnetdll32.exe WheresMyImplant.dll,WheresMyImplant,Implant,RunPowerShell
$env:LogonServer
----------
Namespace: WheresMyImplant
Class: Implant
Method: RunPowerShell
Arguments: $env:LogonServer
----------
\\TestServer2016
```

Using GetType

```
rundotnetdll32 .exe -t
WheresMyImplant.dll,WheresMyImplant,Implant,RunPowerShell $env:LogonServer
----------
Namespace: WheresMyImplant
Class: Implant
Method: RunPowerShell
Arguments: $env:LogonServer
----------
\\TestServer2016
```

Other RunDotNetDll32 Changes

Along with removing the positional parameters and replacing them with flags, the listing functions have been refined.

Namespaces

```
rundotnetdll32.exe -l WheresMyImplant.dll
[N] WheresMyImplant
[N] Empire
[N] Unmanaged.Libraries
[N] Unmanaged.Headers
[N] Org.BouncyCastle.Crypto
[N] Org.BouncyCastle.Utilities
[N] Org.BouncyCastle.Crypto.Digests
```

NameSpace Classes

```
rundotnetdll32.exe -l WheresMyImplant.dll -n Unmanaged.Libraries
[N] Unmanaged.Libraries
   [C] secur32
   [C] wlanapi
   [C] crypt32
   [C] ntdll
   [C] PROCESSINFOCLASS
   [C] _PROCESS_BASIC_INFORMATION
   [C] kernel32
   [C] dbghelp
   [C] _LOADED_IMAGE
   [C] advapi32
   [C] CRED_TYPE
   [C] LOGON_FLAGS
   [C] vaultcli
   [C] wtsapi32
   [C] _WTS_INFO_CLASS
   [C] _WTS_CONNECTSTATE_CLASS
   [C] _WTS_SESSION_INFO
   [C] user32
```

Class Methods

```
rundotnetdll32.exe -l WheresMyImplant.dll -n Unmanaged.Libraries -c kernel32
[N] Unmanaged.Libraries
   [C] kernel32
      [M] SetThreadContext
      [M] VirtualAlloc
      [M] VirtualAllocEx
      [M] VirtualProtect
```

```
      [M] VirtualProtectEx
      [M] VirtualQueryEx
      [M] VirtualQueryEx64
      [M] WaitForSingleObject
      [M] WaitForSingleObjectEx
      [M] WriteProcessMemory
      [M] WriteProcessMemory
      [M] CloseHandle
      [M] CreateProcess
      [M] CreateRemoteThread
      [M] CreateThread
      [M] CreateToolhelp32Snapshot
      [M] GetCurrentThread
      [M] GetCurrentProcess
      [M] GetModuleHandle
      [M] GetNativeSystemInfo
      [M] GetPrivateProfileString
      [M] GetProcAddress
      [M] GetSystemInfo
      [M] GetThreadContext
      [M] IsWow64Process
      [M] Module32First
      [M] Module32Next
      [M] LoadLibrary
      [M] Process32First
      [M] Process32Next
      [M] OpenProcess
      [M] OpenProcessToken
      [M] OpenThread
      [M] OpenThreadToken
      [M] ReadProcessMemory
      [M] ReadProcessMemory64
      [M] ResumeThread
      [M] SetConsoleCtrlHandler
      [M] ToString
      [M] Equals
      [M] GetHashCode
      [M] GetType
```

Method Parameters

```
rundotnetdll32.exe -l WheresMyImplant.dll -n Unmanaged.Libraries -c kernel32 -m
OpenProcess
[N] Unmanaged.Libraries
   [C] kernel32
      [M] OpenProcess
         [P] 0  dwDesiredAccess System.UInt32
         [P] 1  bInheritHandle  System.Boolean
```

```
[P] 2  dwProcessId     System.UInt32
[R] 0  IntPtr          System.IntPtr
```