HInvoke and avoiding PInvoke

Published: 2022-08-10 · Archived: 2026-04-29 02:08:25 UTC

A very minimalistic approach of calling .net runtime functions or accessing properties using only hashes as identifiers. It does not leave any strings or import references since we dynamically resolve the required member from the mscorlib assembly on runtime.

How does HInvoke work?

Its fairly simple, iterate trough all Types of mscorlib and hash their names using some hashing function. Upon finding the matching type continue by iterating trough all its methods or properties and do the same hashing routine as before. Finish by either invoking the resolved method and if applicable return its returnvalue or return the value of the resolved property. This whole process has one fairly big limitation it can only find methods that have a unique name, as the current implementation is fairly lazy and does not take parameter count or types into account.

```
    public static T InvokeMethod<T>(uint classID, uint methodID, object[]? args = null)
    {
        // Get the System assembly and go trough all its types hash their name
        // and find the hash that matches the supplied one
        var typeDef = typeof(void).Assembly.GetTypes()
            .FirstOrDefault(type => GetHash(type.FullName!) == classID);

        // Use the type and go trough its methods hash their name
        // and find the hash that matches the supplied one
        var methodInfo = typeDef.GetRuntimeMethods()
            .FirstOrDefault(method => GetHash(method.Name) == methodID);

        // Invoke the resolved method with the supplied args
        if (methodInfo != null)
            return (T) methodInfo.Invoke(null, args);

        return default!;
    }
```

Calls using HInvoke look like this

```
            if (HInvoke.GetPropertyValue<bool>(1577037771, 179842977)) // System.Diagnostics.Debugger.IsAttac
                HInvoke.InvokeMethod(1174404872, 2029614223, new object[] {0}); // System.Environment.Exit(0)
```

The HInvoke call requires the two before mentioned hashes, and additionally parameters for the method being called. The example is a common anti debug measure in .net obfuscators, only that this version does not expose the actual call on first glance. It checks the value of `Debugger.IsAttached` in case its true it calls `Environment.Exit` with parameter 0, closing the program.

So in short: We can call every uniquely named method from the .net runtime using only 2 hashes.

Avoiding Pinvoke

Another idea I got while browsing trough the internal parts of the managed .net runtime. There is a class called `Microsoft.Win32.Win32Native` which contains you guessed it managed wrappers for native functions. Since Microsoft already so kindly provides these wrappers it would be a waste to not use them.

There were 2 functions that I found especially interesting: `GetModuleHandle` and `GetProcAddress`. By invoking them we can without any usage of PInvoke in our binary get the address of any unmanaged function. Also by using the delegate pointer type (`delegate*`) we can easily invoke the resolved unmanaged functions.

```
            var module =
                HInvoke.InvokeMethod<IntPtr>(13239936, 811580934,
                    new object[] {"kernel32.dll"}); // Microsoft.Win32.Win32Native.GetModuleHandle
            var address =
                HInvoke.InvokeMethod<IntPtr>(13239936, 1721745356,
                    new object[] {module, "IsDebuggerPresent"}); // Microsoft.Win32.Win32Native.GetProcAddres

            if (((delegate* unmanaged[Stdcall]<bool>) address)())
                Console.WriteLine("Hey meanie I said no debugging :c");
```

The example shows a combination of using the Win32Native class and HInvoke to resolve the address of `kernel32!IsDebuggerPresent`. After it casts a delegate pointer with the unmanaged attribute, the calling convention and the returntype on the resolved address. Then calls it.

You can find the full example code here

This is a rather short post but hopefully interesting to some. For feedback or questions contact me on Twitter or Discord.
