

Forum:[Hack Forums](#)

Thread: [Dynamic API Call With Dynamic Delegates](#)

frontenhack - April 23, 2014, 5:13 am

Hello HF :)

Today i'm gonna show you this code, it automatically generates delegate types with Reflection.Emit.

So
Code public DynamicDelegateApiInvoker()
{
 AssemblyName assembly = new AssemblyName();
 assembly.Version = new Version(1, 0, 0, 0);
 assembly.Name = "ReflectionEmitDelegate";
 assemblyBuilder = AppDomain.CurrentDomain.DefineDynamicAssembly(assembly,
AssemblyBuilderAccess.Run);

}This is the constructor of the invoker class.

It generates a dynamic assembly.

Code public T InvokeApi<T>(string library, string function, object[] parameters)
{
 IntPtr hModule = LoadLibrary(library);
 if (hModule == IntPtr.Zero) return default(T);
 IntPtr hHandle = GetProcAddress(hModule, function);
 if (hHandle == IntPtr.Zero) return default(T);

 Type[] parameterTypes = new Type[parameters.Length];
 for (int i = 0; i < parameters.Length; i++)
 parameterTypes[i] = parameters[i].GetType();This is the start of the code, T is a generic
type defined when calling the method.

The function accept :the library name (Ex. "user32.dll")

the function name (Ex. "MessageBoxA")

the parameters (Ex. new object[] {IntPtr.Zero, "Hello", "Buddy", 0})

Firstly it loads the library and check for errors

Then it gets the function address and check for errors.

after it creates parameter type array for constructing the delegate.

Then it creates the delegate using TypeBuilder.

Example Code :

Code class Program

{
 static void Main(string[] args)
 {
 DynamicDelegateApiInvoker api = new DynamicDelegateApiInvoker();
 api.InvokeApi<int>("user32.dll", "MessageBoxA", new object[] {IntPtr.Zero, "Hello", "Buddy",
0});

```
Process.GetCurrentProcess().WaitForExit();
```

```
}  
}
```

To see the full code go to my blog : frontenhack.me

morphine1337 - April 23, 2014, 8:08 am

Nice, this is pretty good, thank you very much.

iVision - April 23, 2014, 11:25 am

Looks useful I guess for malware people anyway it is not fail-proof: CallingConventions.Standard you don't know the convention, it could be cdecl which usually the case when you are invoking third party libraries however for the WinAPI it is StdCall (Standard). If you choose the wrong convention I think, never tried it this would happen (assuming the no name mangling if this was the case you had to look at the export table to find the name or do the math):

- a. If the caller assumes stdcall but the API is cdecl: it leaves the arguments on the stack
- b. If the caller assumes cdecl but the API is stdcall: since the API already cleaned it parameters, so the calling API will pop off wrong things.

Both messes up the stack :(The rest seems fine I think although from an object oriented perspective it doesn't make sense to have multiple instances of the class? Why isn't it static?

Zettabit - April 23, 2014, 11:33 pm

If you're going to host it off site, at least upload it to a place that doesn't go down for no reason.

Only3 - April 23, 2014, 11:54 pm

You Dont Need Apis

Pure Reflection

```
Code public static object DynamicInvoke(string Dll, string Name, object[] Param, Type Ret)
{
    List<Type> Types = new List<Type>();
    for (int i = 0; i <= Param.Length - 1; i++)
        Types.Add(Param[i].GetType());
    AssemblyBuilder A = AppDomain.CurrentDomain.DefineDynamicAssembly(new
AssemblyName("DI"), AssemblyBuilderAccess.Run);
    ModuleBuilder M = A.DefineDynamicModule("MB");
    MethodBuilder MB = M.DefinePInvokeMethod(Name, Dll, MethodAttributes.Public |
MethodAttributes.Static | MethodAttributes.PinvokeImpl, CallingConventions.Standard, Ret,
Types.ToArray(), CallingConvention.Winapi, CharSet.Ansi);
    MB.SetImplementationFlags(MethodImplAttributes.PreserveSig);
    M.CreateGlobalFunctions();
    return M.GetMethod(Name).Invoke(null, Param);
}
```

frontenhack - April 24, 2014, 12:10 am

(Apr 23, 2014 11:54 PM)Only3 Wrote: [\[CITE\]](#) You Dont Need Apis

Pure Reflection

```
Code public static object DynamicInvoke(string Dll, string Name, object[] Param, Type Ret)
{
    List<Type> Types = new List<Type>();
    for (int i = 0; i <= Param.Length - 1; i++)
```

```
Types.Add(Param[i].GetType());
AssemblyBuilder A = AppDomain.CurrentDomain.DefineDynamicAssembly(new
AssemblyName("DI"), AssemblyBuilderAccess.Run);
ModuleBuilder M = A.DefineDynamicModule("MB");
MethodBuilder MB = M.DefinePInvokeMethod(Name, Dll, MethodAttributes.Public |
MethodAttributes.Static | MethodAttributes.PinvokeImpl, CallingConventions.Standard, Ret,
Types.ToArray(), CallingConvention.Winapi, CharSet.Ansi);
MB.SetImplementationFlags(MethodImplAttributes.PreserveSig);
M.CreateGlobalFunctions();
return M.GetMethod(Name).Invoke(null, Param);
}This was already released, mine was more unique xD
```

Only3 - April 24, 2014, 10:41 am

if this ur point ok but its been posted as well from what i remember

frontenhack - April 26, 2014, 10:07 am

(Apr 24, 2014 10:41 AM)Only3 Wrote: [\[CITE\]](#) if this ur point ok but its been posted as well from what i remember

Dynamic delegates with emit i think no, anyway thanks for posting :)