FLARE-ON CHALLENGE 9 SOLUTION

BY MUHAMMAD UMAIR (@M_UMAIRX)

Challenge 8: backdoor

Challenge Prompt

I'm such a backdoor, decompile me why don't you…

7-zip password: flare

Solution

The challenge uses the concept of Dynamic Methods in .NET to conceal meaningful code from static analysis and debugging tools like dnSpy. To keep the complexity of the challenge reasonable, no further obfuscations have been applied to the assembly, and by design static method metadata has been left behind to provide for an alternate way of solving the challenge if the player only possesses limited information on .NET internals.

Once the .NET assembly has been de-obfuscated, the underlying challenge is based on a real-world malware that communicates with its C2 (Command & Control) server over DNS protocol. The logic contained constitutes a fully functional backdoor. However, without describing the complete functionality, we will instead be focusing on the parts that are required for acquiring the flag.

One of the primary differences between normal and dynamically generated CIL (Common Intermediate Language), that is relevant for the purposes of this challenge, is that the metadata tokens contained within dynamic CIL are limited to the scope of the method in question and are not valid outside that dynamic method. CIL belonging to normal methods from an assembly on the other hand contains metadata tokens that are valid throughout that assembly and are kept track of by the Metadata Tables. A mapping object array ***m_tokens*** is therefore created for every dynamic method that maps the metadata tokens inside it to the outside context as shown below.

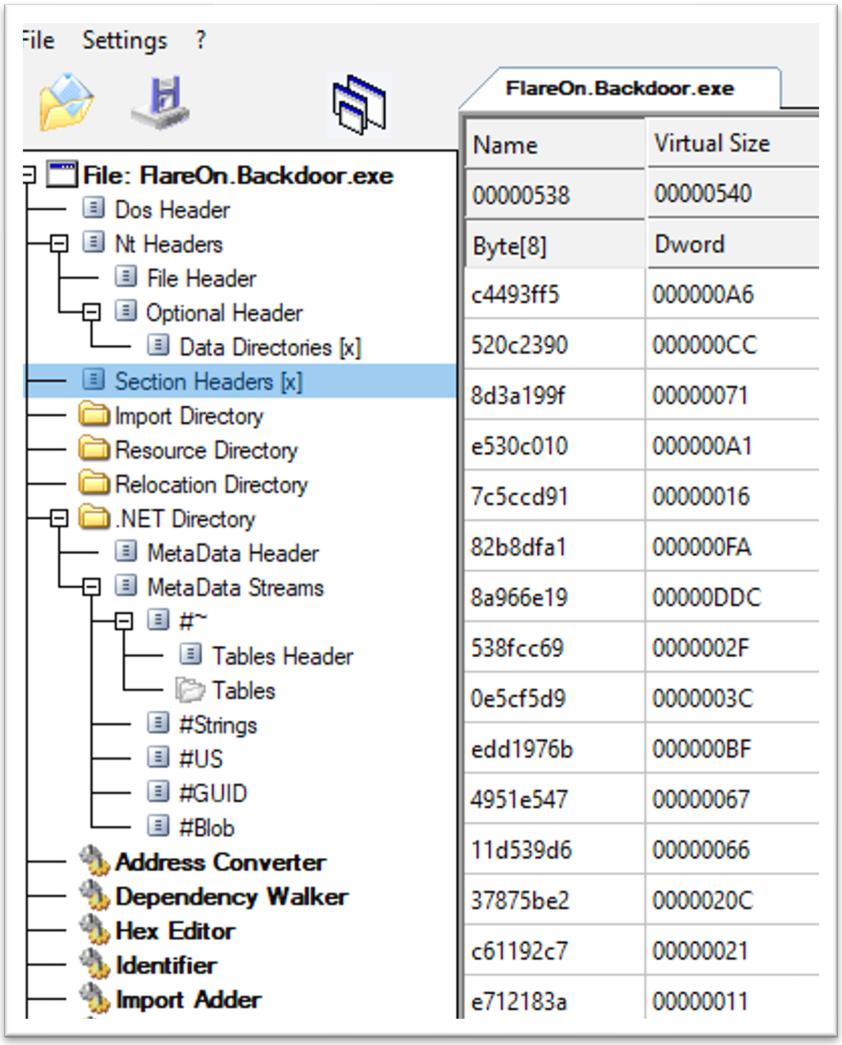
```
0:000> !DumpObj /d 0000000002da63a8
Name:        System.Reflection.Emit.DynamicMethod
MethodTable: 000007fee057c238
EEClass:     000007fee0802840
Size:        112(0x70) bytes
File:  C:\Windows\Microsoft.Net\assembly\GAC_64\mscorlib\v4.0_4.0.0.0__b77a5c561934e089\mscorl
ib.dll
Fields:
              MT    Field   Offset                 Type VT     Attr            Value Name
000007fee05bb038  4001de4       28 ...mit.DynamicILInfo  0 instance 0000000002da64e0
m_DynamicILInfo
…
0:000> !DumpObj /d 0000000002da64e0
Name:        System.Reflection.Emit.DynamicILInfo
MethodTable: 000007fee05bb038
EEClass:     000007fee08109d8
Size:        64(0x40) bytes
File:  C:\Windows\Microsoft.Net\assembly\GAC_64\mscorlib\v4.0_4.0.0.0__b77a5c561934e089\mscorl
ib.dll
Fields:
              MT    Field   Offset                 Type VT     Attr            Value Name
000007fee057c238  4001dd1        8 ...mit.DynamicMethod  0 instance 0000000002da63a8 m_method
000007fee05e7fe0  4001dd2       10 ...Emit.DynamicScope  0 instance 0000000002da6478 m_scope
…
0:000> !DumpObj /d 0000000002da6478
Name:        System.Reflection.Emit.DynamicScope
MethodTable: 000007fee05e7fe0
EEClass:     000007fee0838dc0
Size:        24(0x18) bytes
File:  C:\Windows\Microsoft.Net\assembly\GAC_64\mscorlib\v4.0_4.0.0.0__b77a5c561934e089\mscorl
```

```
ib.dll
Fields:
              MT    Field   Offset                 Type VT     Attr            Value Name
000007fee0575ff0  4001dd8        8 ...bject, mscorlib]]  0 instance 0000000002da6490 m_tokens
```

Figure 1. Displaying m_tokens array in windbg

Initial Observations

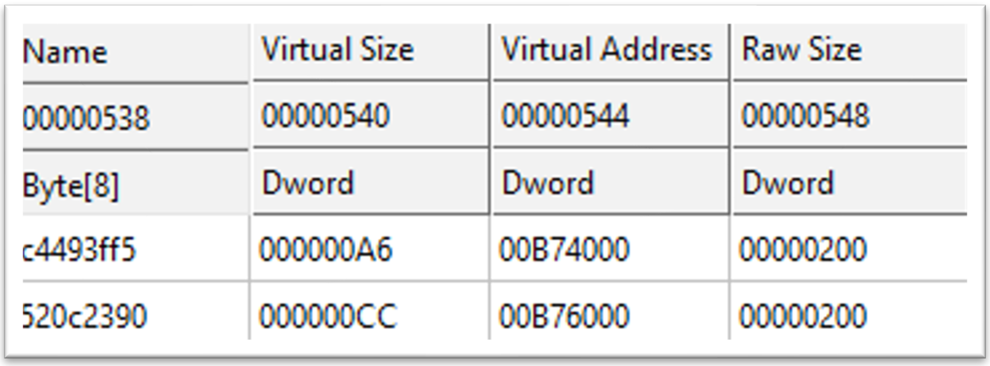
If the binary is opened in CFF explorer and immediately some things stand out. The presence of .NET directory indicates that this is a .NET based assembly. Exploring the section headers as can be seen from Figure 2, there seem to be an abnormally high number of sections.



Name	Virtual Size
00000538	00000540
Byte[8]	Dword
c4493ff5	000000A6
520c2390	000000CC
8d3a199f	00000071
e530c010	000000A1
7c5ccd91	00000016
82b8dfa1	000000FA
8a966e19	00000DDC
538fcc69	0000002F
0e5cf5d9	0000003C
edd1976b	000000BF
4951e547	00000067
11d539d6	00000066
37875be2	0000020C
c61192c7	00000021
e712183a	00000011

Figure 2. Abnormally high number of sections

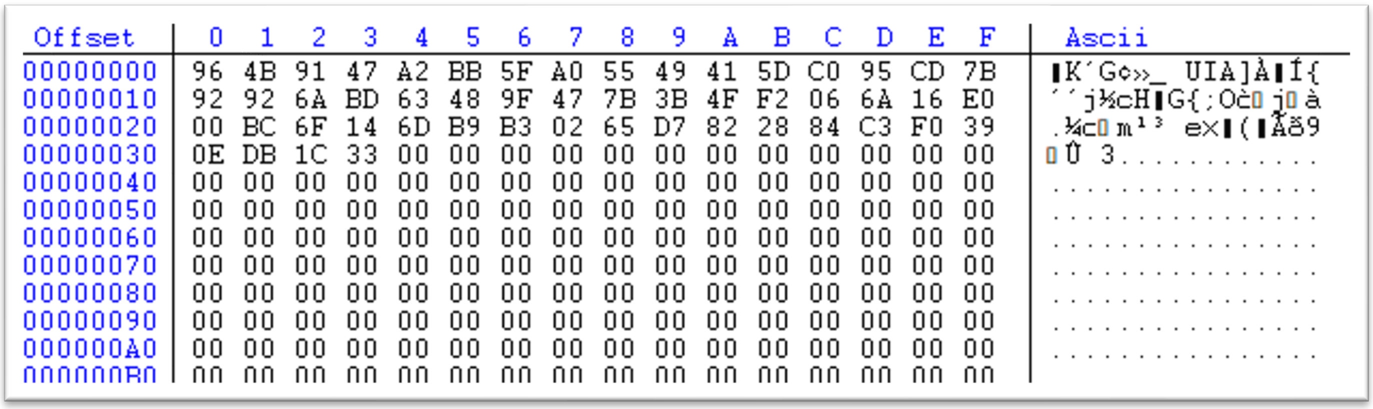
Another curious thing to note about the sections is that the ***Virtual Size*** of most of these is smaller than their ***Raw Size*** as can be seen in Figure 3.



Name	Virtual Size	Virtual Address	Raw Size
00000538	00000540	00000544	00000548
Byte[8]	Dword	Dword	Dword
c4493ff5	000000A6	00B74000	00000200
520c2390	000000CC	00B76000	00000200

Figure 3. Virtual Size smaller than Raw Size

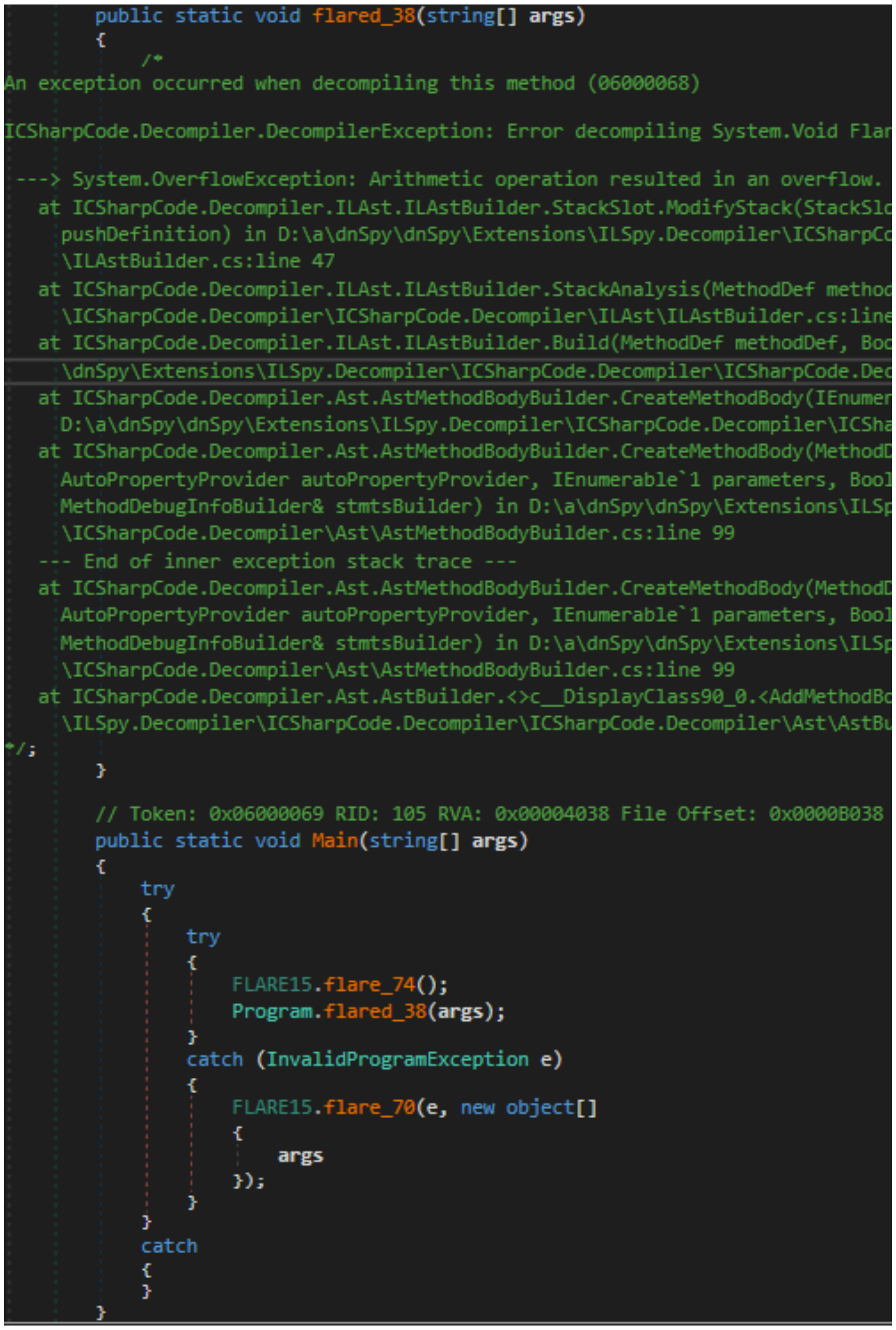
Looking at the hex dumps of these sections, it can be noticed that they appear to contain data up to the ***Virtual Size*** of the section, and the rest is padded with zeros. This is something important to remember. Example in Figure 4.



```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00000000  96 4B 91 47 A2 BB 5F A0 55 49 41 5D C0 95 CD 7B  ▮K´G¢»_ UIA]À▮Í{
00000010  92 92 6A BD 63 48 9F 47 7B 3B 4F F2 06 6A 16 E0  ´´j½cH▮G{;Oò▯j▯à
00000020  00 BC 6F 14 6D B9 B3 02 65 D7 82 28 84 C3 F0 39  .¼o▯m¹³ e×▮(▮Ãð9
00000030  0E DB 1C 33 00 00 00 00 00 00 00 00 00 00 00 00  ▯Û 3............
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 4. Raw Size - Virtual Size padded with zeros

When the assembly is then opened in dnSpy it appears to contain a lot of functions that fail to decompile. In fact, all the functions prefixed with ***flared_*** do not decompile. If the IL of these methods is observed in dnSpy, it will appear to make no sense, hinting at the IL either being encrypted or overwritten by junk data.



```
        public static void flared_38(string[] args)
        {
            /*
An exception occurred when decompiling this method (06000068)

ICSharpCode.Decompiler.DecompilerException: Error decompiling System.Void Flar

 ---> System.OverflowException: Arithmetic operation resulted in an overflow.
   at ICSharpCode.Decompiler.ILAst.ILAstBuilder.StackSlot.ModifyStack(StackSlo
     pushDefinition) in D:\a\dnSpy\dnSpy\Extensions\ILSpy.Decompiler\ICSharpCo
     \ILAstBuilder.cs:line 47
   at ICSharpCode.Decompiler.ILAst.ILAstBuilder.StackAnalysis(MethodDef methoc
     \ICSharpCode.Decompiler\ICSharpCode.Decompiler\ILAst\ILAstBuilder.cs:line
   at ICSharpCode.Decompiler.ILAst.ILAstBuilder.Build(MethodDef methodDef, Boc
     \dnSpy\Extensions\ILSpy.Decompiler\ICSharpCode.Decompiler\ICSharpCode.Dec
   at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(IEnumer
     D:\a\dnSpy\dnSpy\Extensions\ILSpy.Decompiler\ICSharpCode.Decompiler\ICSha
   at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(MethodD
     AutoPropertyProvider autoPropertyProvider, IEnumerable`1 parameters, Bool
     MethodDebugInfoBuilder& stmtsBuilder) in D:\a\dnSpy\dnSpy\Extensions\ILSp
     \ICSharpCode.Decompiler\Ast\AstMethodBodyBuilder.cs:line 99
   --- End of inner exception stack trace ---
   at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(MethodD
     AutoPropertyProvider autoPropertyProvider, IEnumerable`1 parameters, Bool
     MethodDebugInfoBuilder& stmtsBuilder) in D:\a\dnSpy\dnSpy\Extensions\ILSp
     \ICSharpCode.Decompiler\Ast\AstMethodBodyBuilder.cs:line 99
   at ICSharpCode.Decompiler.Ast.AstBuilder.<>c__DisplayClass90_0.<AddMethodBc
     \ILSpy.Decompiler\ICSharpCode.Decompiler\ICSharpCode.Decompiler\Ast\AstBu
*/;
        }

        // Token: 0x06000069 RID: 105 RVA: 0x00004038 File Offset: 0x0000B038
        public static void Main(string[] args)
        {
            try
            {
                try
                {
                    FLARE15.flare_74();
                    Program.flared_38(args);
                }
                catch (InvalidProgramException e)
                {
                    FLARE15.flare_70(e, new object[]
                    {
                        args
                    });
                }
            }
            catch
            {
            }
        }
```

Figure 5. Example of a failed method decompilation

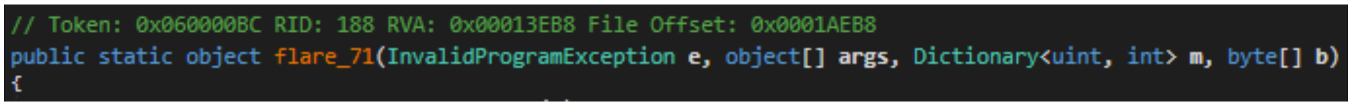
Other than that, there does not seem to be any meaningful method or class names present within the binary. Another interesting observation is that all the ***flared_*** prefixed functions appear to be called in exception-handled methods where the ***InvalidProgramException*** is being caught. Example in Figure 5, which shows the ***Main*** function of the assembly, and the corresponding ***flared_*** method that is failing to decompile.

Finally, also being called from the ***Main*** is a method ***flare_74***, which is initializing some Dictionary, List and ObservableCollection objects.

Analysis Phase 1

The initial path of execution can be debugged within dnSpy. The path flows from the ***Main*** method, initializing some data in ***flare_74***, generating ***InvalidProgramExceptions*** along the way and eventually ending up in ***flare_71***. The logic inside ***flare_71*** eventually leads to the creation and calling of a Dynamic Method, at which point dnSpy loses control and is unable to display any decompilation. Analyzing the logic of the method we can observe the following.

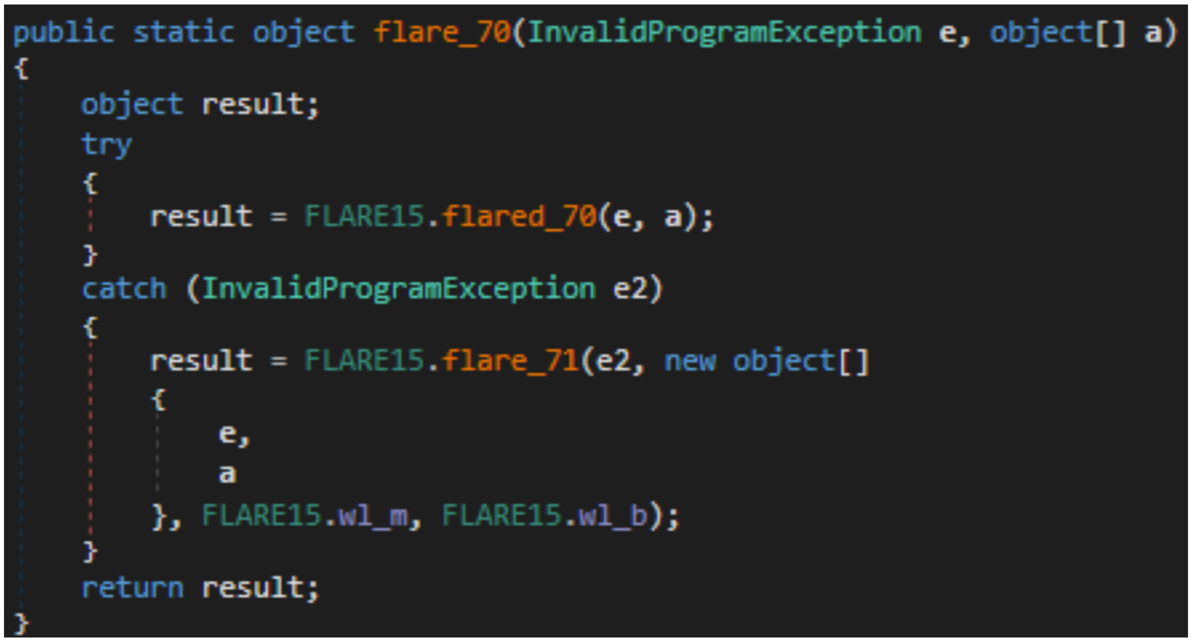
The method takes 4 arguments as can be seen in Figure 6.



```
// Token: 0x060000BC RID: 188 RVA: 0x00013EB8 File Offset: 0x0001AEB8
public static object flare_71(InvalidProgramException e, object[] args, Dictionary<uint, int> m, byte[] b)
{
```

Figure 6. Argument listing of Dynamic Method generator

First 2 arguments contain an exception and an array of objects. These 2 arguments can be traced back to ***flared_70*** which generated an ***InvalidProgramException.*** The exception itself eventually becomes the 1st argument whereas the original arguments that were passed to ***flared_70*** become the object array being passed as the 2nd argument. This can be observed from Figure 7.

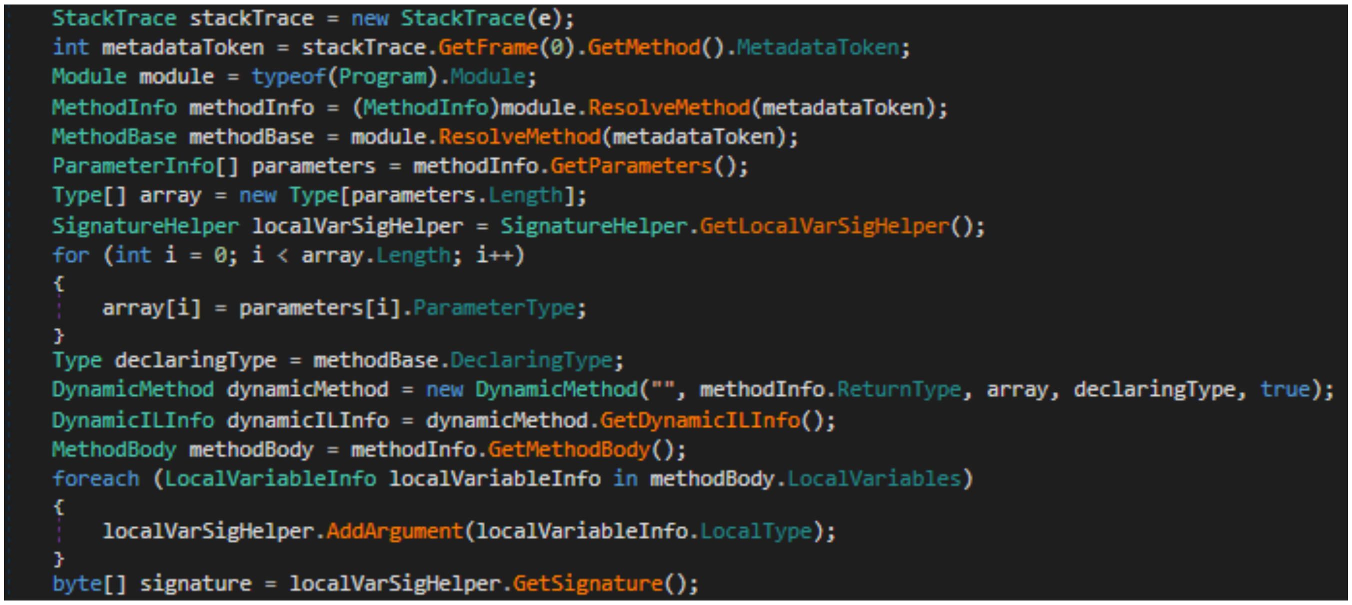


```
public static object flare_70(InvalidProgramException e, object[] a)
{
    object result;
    try
    {
        result = FLARE15.flared_70(e, a);
    }
    catch (InvalidProgramException e2)
    {
        result = FLARE15.flare_71(e2, new object[]
        {
            e,
            a
        }, FLARE15.wl_m, FLARE15.wl_b);
    }
    return result;
}
```

Figure 7. Example of 'flared_' prefixed functions being wrapped in exception handlers

The remaining 2 arguments contain a Dictionary of integers and a byte array respectively.

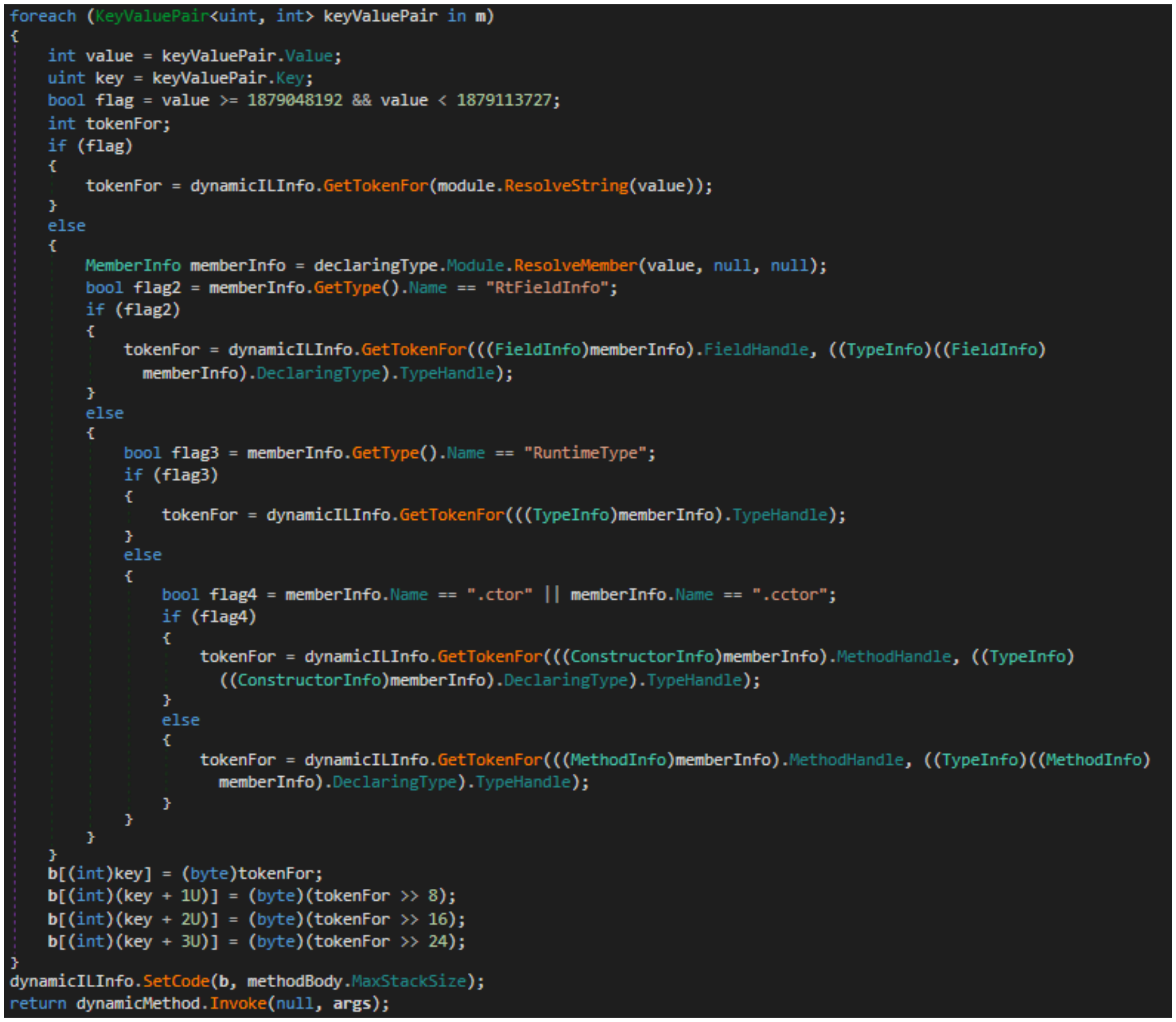
Analyzing the first part of ***flare_71***, the exception (1st argument) is used to get information structures on the method which triggered the exception itself. Information is collected on the parameters, local variables, return type and signature of the exception generating method and it is used to prepare a Dynamic Method object as shown in Figure 8.



```
StackTrace stackTrace = new StackTrace(e);
int metadataToken = stackTrace.GetFrame(0).GetMethod().MetadataToken;
Module module = typeof(Program).Module;
MethodInfo methodInfo = (MethodInfo)module.ResolveMethod(metadataToken);
MethodBase methodBase = module.ResolveMethod(metadataToken);
ParameterInfo[] parameters = methodInfo.GetParameters();
Type[] array = new Type[parameters.Length];
SignatureHelper localVarSigHelper = SignatureHelper.GetLocalVarSigHelper();
for (int i = 0; i < array.Length; i++)
{
    array[i] = parameters[i].ParameterType;
}
Type declaringType = methodBase.DeclaringType;
DynamicMethod dynamicMethod = new DynamicMethod("", methodInfo.ReturnType, array, declaringType, true);
DynamicILInfo dynamicILInfo = dynamicMethod.GetDynamicILInfo();
MethodBody methodBody = methodInfo.GetMethodBody();
foreach (LocalVariableInfo localVariableInfo in methodBody.LocalVariables)
{
    localVarSigHelper.AddArgument(localVariableInfo.LocalType);
}
byte[] signature = localVarSigHelper.GetSignature();
```

Figure 8. Information gathering from exception method

Looking at the processing in the second part of the method, it can be deduced that the 3rd argument contains metadata tokens and their offsets within the method body in the form of a dictionary, whereas the 4th argument contains the IL code of the method which has been stripped of its metadata tokens.

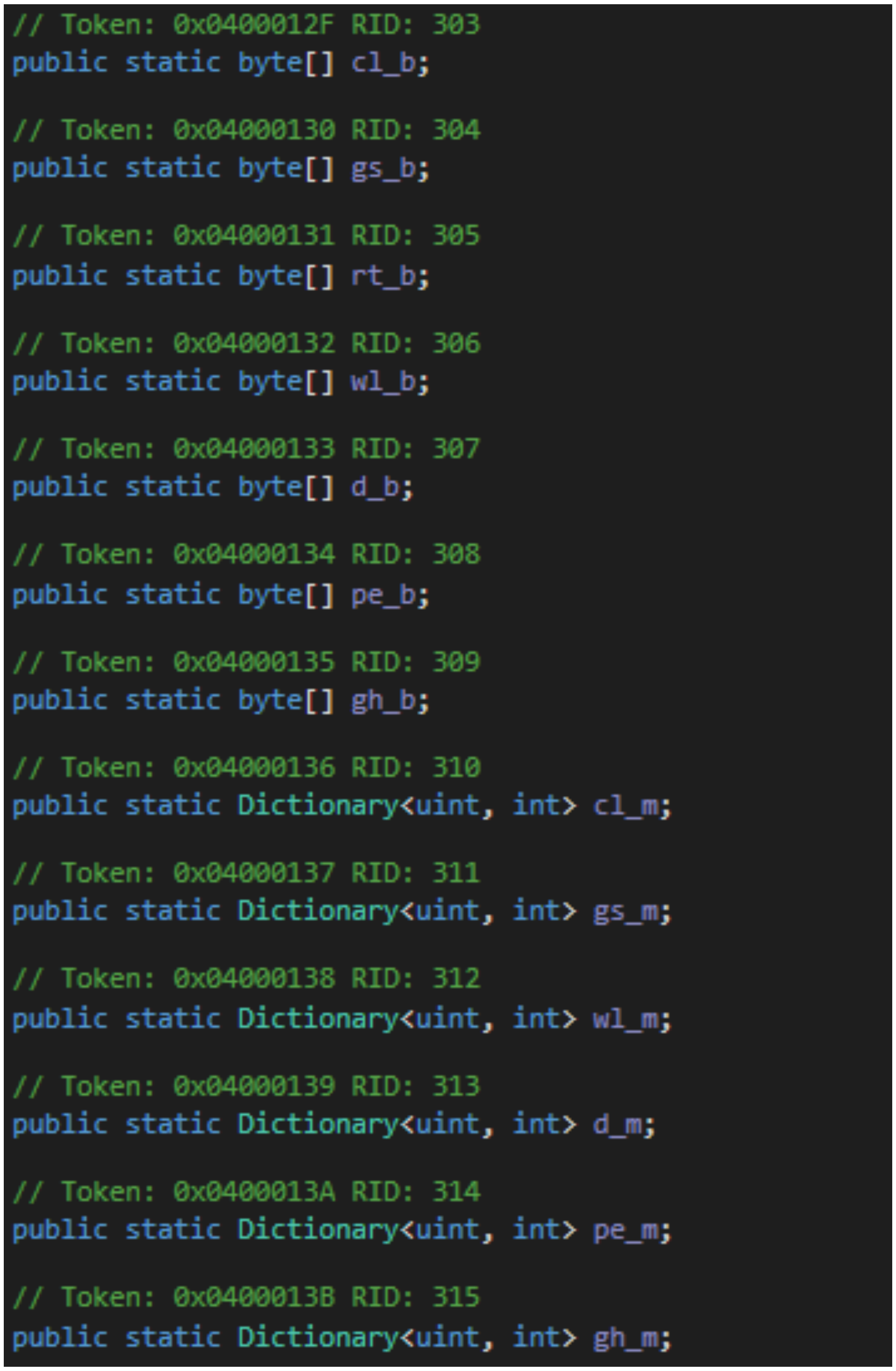


```
foreach (KeyValuePair<uint, int> keyValuePair in m)
{
    int value = keyValuePair.Value;
    uint key = keyValuePair.Key;
    bool flag = value >= 1879048192 && value < 1879113727;
    int tokenFor;
    if (flag)
    {
        tokenFor = dynamicILInfo.GetTokenFor(module.ResolveString(value));
    }
    else
    {
        MemberInfo memberInfo = declaringType.Module.ResolveMember(value, null, null);
        bool flag2 = memberInfo.GetType().Name == "RtFieldInfo";
        if (flag2)
        {
            tokenFor = dynamicILInfo.GetTokenFor(((FieldInfo)memberInfo).FieldHandle, ((TypeInfo)((FieldInfo)
              memberInfo).DeclaringType).TypeHandle);
        }
        else
        {
            bool flag3 = memberInfo.GetType().Name == "RuntimeType";
            if (flag3)
            {
                tokenFor = dynamicILInfo.GetTokenFor(((TypeInfo)memberInfo).TypeHandle);
            }
            else
            {
                bool flag4 = memberInfo.Name == ".ctor" || memberInfo.Name == ".cctor";
                if (flag4)
                {
                    tokenFor = dynamicILInfo.GetTokenFor(((ConstructorInfo)memberInfo).MethodHandle, ((TypeInfo)
                      ((ConstructorInfo)memberInfo).DeclaringType).TypeHandle);
                }
                else
                {
                    tokenFor = dynamicILInfo.GetTokenFor(((MethodInfo)memberInfo).MethodHandle, ((TypeInfo)((MethodInfo)
                      memberInfo).DeclaringType).TypeHandle);
                }
            }
        }
    }
    b[(int)key] = (byte)tokenFor;
    b[(int)(key + 1U)] = (byte)(tokenFor >> 8);
    b[(int)(key + 2U)] = (byte)(tokenFor >> 16);
    b[(int)(key + 3U)] = (byte)(tokenFor >> 24);
}
dynamicILInfo.SetCode(b, methodBody.MaxStackSize);
return dynamicMethod.Invoke(null, args);
```

Figure 9. Dynamic token generation and method invocation

The method then attempts to determine the type of the metadata token and use the ***DynamicILInfo.GetTokenFor*** method to generate a token that will be valid for the created Dynamic Method. This function helps populate the mapping object array explained earlier. The generated tokens are placed back into the IL body and the Dynamic Method is invoked as illustrated in Figure 9.

We can see that the metadata tokens/offsets and IL code that is being passed to this function is coming from data that was initialized in ***flare_74*** from the ***Main*** method. Looking at all the data that gets initialized (Figure 10), it can be deduced that these objects contain data for 7 methods that get converted to Dynamic Methods and invoked at various stages.



```
// Token: 0x0400012F RID: 303
public static byte[] cl_b;

// Token: 0x04000130 RID: 304
public static byte[] gs_b;

// Token: 0x04000131 RID: 305
public static byte[] rt_b;

// Token: 0x04000132 RID: 306
public static byte[] wl_b;

// Token: 0x04000133 RID: 307
public static byte[] d_b;

// Token: 0x04000134 RID: 308
public static byte[] pe_b;

// Token: 0x04000135 RID: 309
public static byte[] gh_b;

// Token: 0x04000136 RID: 310
public static Dictionary<uint, int> cl_m;

// Token: 0x04000137 RID: 311
public static Dictionary<uint, int> gs_m;

// Token: 0x04000138 RID: 312
public static Dictionary<uint, int> wl_m;

// Token: 0x04000139 RID: 313
public static Dictionary<uint, int> d_m;

// Token: 0x0400013A RID: 314
public static Dictionary<uint, int> pe_m;

// Token: 0x0400013B RID: 315
public static Dictionary<uint, int> gh_m;
```

Figure 10. Static method data

We know that the specifics of the original ***flared_70*** method were being used to populate the Dynamic Method object. Furthermore, we can compare the size of the *corrupted* IL from ***flared_70*** with the IL being passed in as the 4th argument. These serve as a very good indicator that bytes passed in can be fixed to return to their original state and should belong to the original corrupted ***flared_70*** method. The same can be applied to all the 7 methods for which these structures are being referenced. This function metadata was left behind in the challenge intentionally, to provide for an alternate way of solving the binary. There are 2 ways of fixing these 7 methods.

1. Fix the original IL and raw patch bytes back into the methods.
2. Retrieve the generated IL dynamically, convert it back to static and re-write the binary using a library like dnlib, cecil etc.

However, there are a few things to consider before selecting an approach. The initialized structures that we have only correspond to 7 methods, whereas the binary contains 70 methods in total (up to ***flared_70***) that appear to be corrupted. This means that fixing these methods will only fix 1 phase of obfuscation and more processing will likely be required. One of the quirks of using a library like dnlib to re-write an assembly is that, even if the *MetadataFlags.PreserveAll* flag is used, there is no guarantee that the metadata tokens will not be modified. Modification of these metadata tokens will mean that any original tokens that might have been stored away in some form by the challenge for the remaining methods can become invalid. Another albeit less serious issue that would occur is that all the extra sections in the binary will also get discarded, making it less convenient to access those and use for the solution. This case approach (1) would be the most convenient one to take.

The general structure for the assembly can thus be summarized as follows.

- The ***flared_*** methods contain corrupted data and are always called in a proxy/wrapper method.
- The wrapper catches the ***InvalidProgramException*** from the corrupted method and passes the exception information with all the arguments to another set of methods which eventually generate and call a Dynamic Method corresponding to the original ***flared_*** function that was called.
- 1 of those Dynamic Method generation routines is not corrupted and is the one we just analyzed (***flare_71***).

We can copy the initialization data from ***flare_74*** as is and use it to fix the tokens within the IL code and patch the 7 corresponding ***flared_*** methods. The methods can be found and mapped by getting cross references to ***flare_71***.

```
<PASTE INITIALIZATION DATA HERE FROM THE DECOMPILATION OF flare_74>

Dictionary<string, List<object>> method_map =
    new Dictionary<string, List<object>>{
        {"flared_35", new List<object>{pe_b, pe_m}},
        {"flared_47", new List<object>{d_b, d_m}},
        {"flared_66", new List<object>{gh_b, gh_m}},
        {"flared_67", new List<object>{cl_b, cl_m}},
        {"flared_68", new List<object>{rt_b, null}},
        {"flared_69", new List<object>{gs_b, gs_m}},
        {"flared_70", new List<object>{wl_b, wl_m}}};

ModuleDefMD moduleDef = ModuleDefMD.Load(_path);
byte[] targetAssembly = File.ReadAllBytes(_path);
long methodBodyOffset = 0;
int token = 0;
```

```
uint index = 0;

Console.WriteLine("[+] Dynamic to Static conversion phase 1...");

foreach (KeyValuePair<string, List<object>> method_data in method_map) {
  foreach (var type in moduleDef.GetTypes()) {
    foreach (var method in type.Methods) {
      if (method.Name == method_data.Key) {
        Console.WriteLine("[+] Raw Patching Method: " + type.FullName + "." +
                          method.Name);
        if (method_data.Value[1] != null) {
          foreach (KeyValuePair<uint, int> item in (Dictionary<uint, int>)
                       method_data.Value[1]) {
            token = item.Value;
            index = item.Key;

            ((byte[]) method_data.Value[0])[index] = (byte) token;
            ((byte[]) method_data.Value[0])[index + 1] = (byte)(token >> 8);
            ((byte[]) method_data.Value[0])[index + 2] = (byte)(token >> 16);
            ((byte[]) method_data.Value[0])[index + 3] = (byte)(token >> 24);
          }
        }

        methodBodyOffset =
            (long) moduleDef.Metadata.PEImage.ToFileOffset(method.RVA) +
            method.Body.HeaderSize;
        Array.Copy((byte[]) method_data.Value[0], 0, targetAssembly,
                   methodBodyOffset, ((byte[]) method_data.Value[0]).Length);
      }
    }
  }
}
Console.WriteLine("\n[+] Dumping to: " + _dump_path + "\n");
File.WriteAllBytes(_dump_path, targetAssembly);
```

(Complete solution in Appendix B)

Analysis Phase 2

Phase 1 of the assembly has been fixed. We start by analyzing the 7 methods that were just patched. They are described in Table 1.

Method	Description
flared_35	Initializer routine for a PE parser. Reads in a PE path and creates parsing related objects for it.
flared_47	RC4 decryption method.

flared_66	Takes a metadata token of a method as an argument and generates a pseudo unique (in the context of this assembly) hash that represents that method.
flared_67	A dynamic method generator/invoker that is more generic than the one encountered in *flare_71* and incorporates token decryption. The method takes in a byte array containing non-dynamic IL, a token for the corresponding method and an object array containing the arguments meant to be passed to it. The method parses the IL to figure out the location of the metadata tokens contained within it, xor decodes those tokens with the key ***{ 0xbd, 0xa6, 0x98, 0xa2 }***, generates an equivalent token for the Dynamic Method using *DynamicILInfo.GetTokenFor*, prepares the Dynamic Method and invokes it with the supplied arguments.
flared_68	Reads bytes from an array and converts them to a DWORD/Token.
flared_69	Takes in a hash as an argument and searches for a section within the PE the name of which matches with the beginning characters of the hash. This section, if found, is then read, and returned in a byte array.
flared_70	Given an exception, figures out the originating method, creates a hash for that method via *flared_66*, uses that hash to find and read in the corresponding PE section via *flared_69*, RC4 decrypts the section with the key ***{ 0x12, 0x78, 0xab, 0xdf }.*** Passes that decrypted IL and arguments to *flared_67* which subsequently prepares a Dynamic Method and invokes it.

Table 1.Analysis of patched/fixed (core engine) methods

We can deduce from the analysis in Table 1 that these 7 methods form the core engine of the Dynamic Method resolution, creation, and invocation mechanism for the rest of the assembly. Figure 11 shows a flow chart that describes the execution of every single ***flared_*** method from the proxy call to the core engine resolution, IL resolution, decryption and finally Dynamic Method creation and invocation.

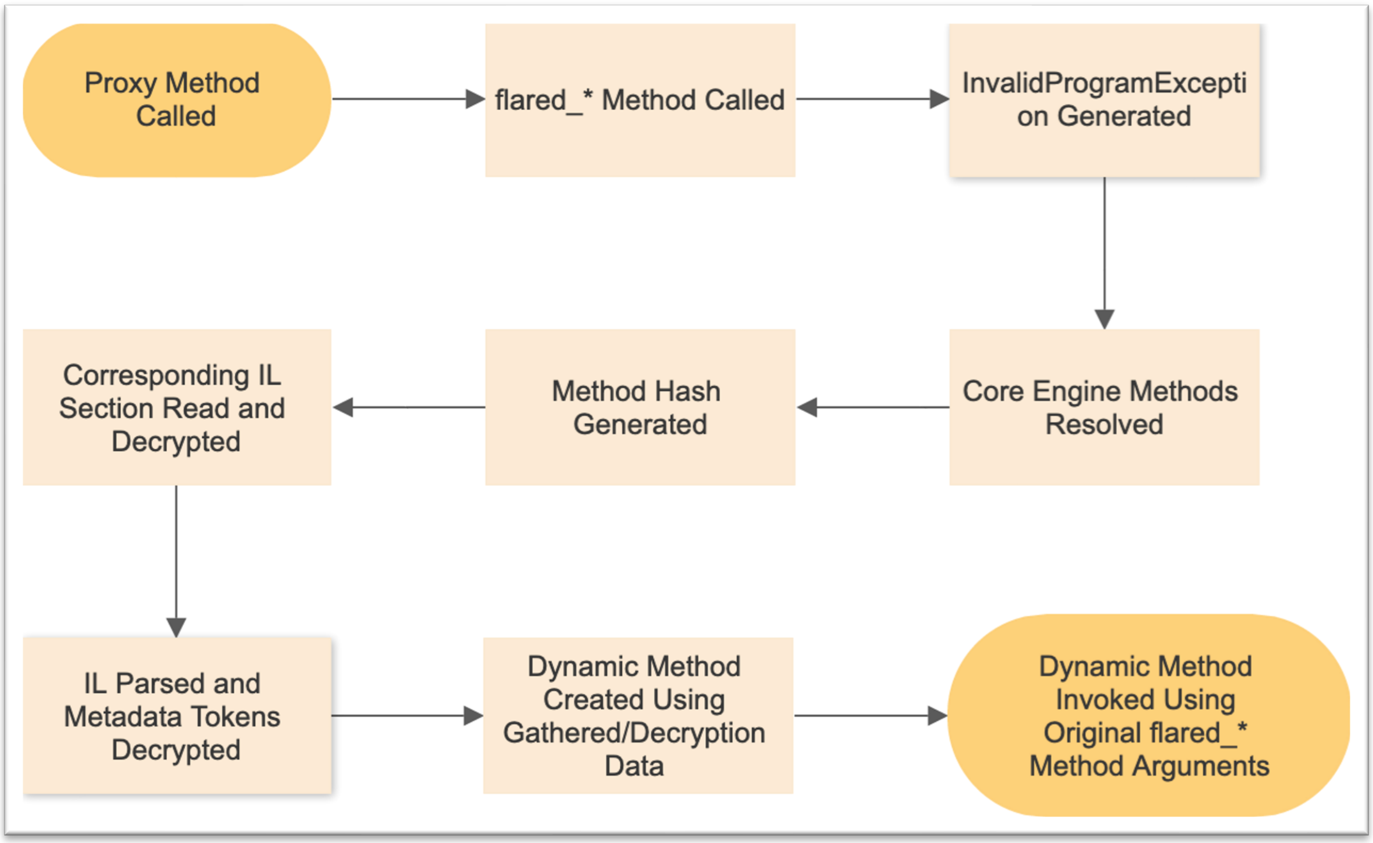

Proxy Method Called
flared_* Method Called
InvalidProgramException Generated
Core Engine Methods Resolved
Method Hash Generated
Corresponding IL Section Read and Decrypted
IL Parsed and Metadata Tokens Decrypted
Dynamic Method Created Using Gathered/Decryption Data
Dynamic Method Invoked Using Original flared_* Method Arguments

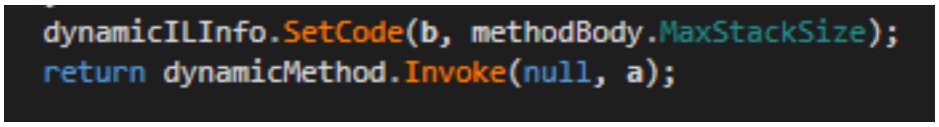
Figure 11. Execution flow of every destroyed (flared_*) method call

The same 2 approaches as stated previously can be used to convert the remaining methods to static. However, for the static approach this time a bit more coding will be required, fortunately a lot of that code can be borrowed directly from the patched methods and adapted. We will be describing both approaches to obtain the final clean assembly.

Conversion using Dynamic Approach

The requirement to be able to convert a Dynamic Method to static is to be able to access the ***DynamicMethod*** object in its finalized state. In common scenarios executing arbitrary methods/constructors via reflection is a very straightforward process and any potential Dynamic Method objects can be obtained from their corresponding delegates. In this case however, the Dynamic Methods are generated at runtime, invoked right after without being assigned to a delegate and then discarded. This makes us require the use of slight trickery to get our desired results.

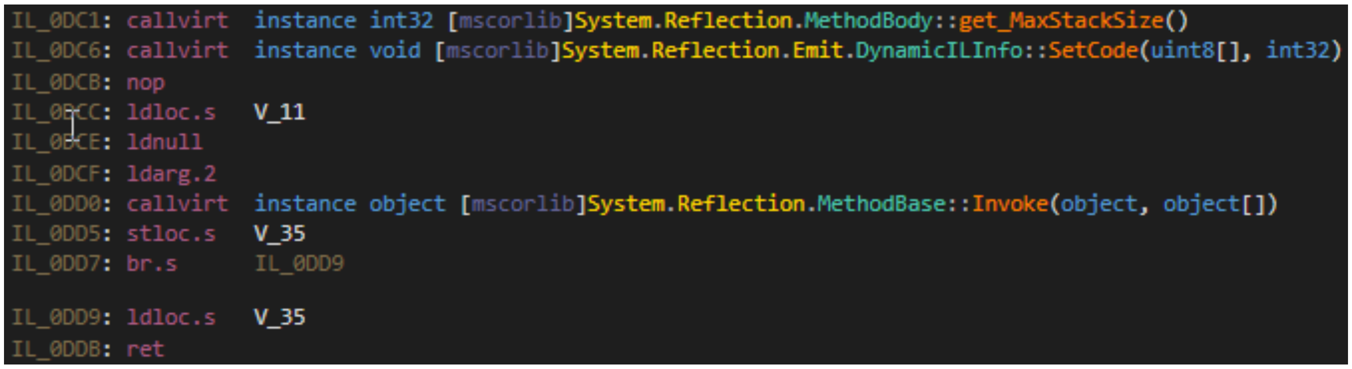
Looking into the ***flared_67*** method which is the primary IL conversion method of the core engine shows that the end of the method the Dynamic Method is being invoked and the result is being returned as can be seen in Figure 12.



```
dynamicILInfo.SetCode(b, methodBody.MaxStackSize);
return dynamicMethod.Invoke(null, a);
```

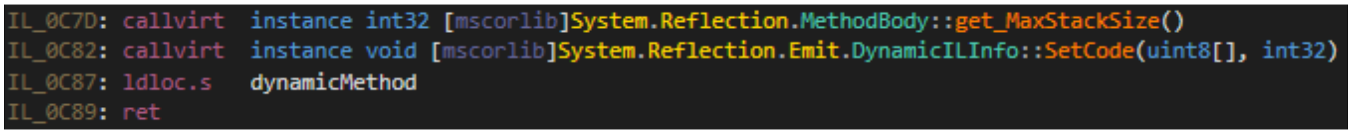
Figure 12. Return sequence of flared_67

We need this method to instead return the ***DynamicMethod*** object WITHOUT invoking it, and then further processing can be done. This seems like a straightforward task, we can just edit the method in dnSpy, remove the *Invoke* call and return the ***DynamicMethod*** object. However, as explained earlier in this document, if use dnlib (which dnSpy is based on) or cecil to rewrite the binary, likely we can end up modifying the metadata tokens and discarding the extra sections making the job much more complex. One solution to avoid this is to use runtime memory patching, which we can achieve via the hot patching library Lib.Harmony. We need to create a patch that will modify the IL as shown in Figure 13 and Figure 14.



```
IL_0DC1: callvirt  instance int32 [mscorlib]System.Reflection.MethodBody::get_MaxStackSize()
IL_0DC6: callvirt  instance void [mscorlib]System.Reflection.Emit.DynamicILInfo::SetCode(uint8[], int32)
IL_0DCB: nop
IL_0DCC: ldloc.s   V_11
IL_0DCE: ldnull
IL_0DCF: ldarg.2
IL_0DD0: callvirt  instance object [mscorlib]System.Reflection.MethodBase::Invoke(object, object[])
IL_0DD5: stloc.s   V_35
IL_0DD7: br.s      IL_0DD9

IL_0DD9: ldloc.s   V_35
IL_0DDB: ret
```

Figure 13. Before Patch



```
IL_0C7D: callvirt  instance int32 [mscorlib]System.Reflection.MethodBody::get_MaxStackSize()
IL_0C82: callvirt  instance void [mscorlib]System.Reflection.Emit.DynamicILInfo::SetCode(uint8[], int32)
IL_0C87: ldloc.s   dynamicMethod
IL_0C89: ret
```

Figure 14. After Patch

This effect can be achieved by replacing the required instructions with NOPs and adding extra NOPs to make sure the size of the method remains the same. Harmony provides Prefix and Postfix methods that allow runtime hooking of assembly methods and argument/result modification. Harmony also provides Transpiler methods, which allow runtime modification of the actual target method code. Following Transpiler patch can be coded to achieve this modification. *(It is of note that Lib.Harmony cannot patch Dynamic IL)*

```
public static IEnumerable<CodeInstruction> Transpiler(
    IEnumerable<CodeInstruction> instructions) {

Console.WriteLine("[+] Invoking Transpiler");
  List<CodeInstruction> ins_list = instructions.ToList();
  int count = ins_list.Count;

  Console.WriteLine("\n[!] Before Patch");
  Console.WriteLine("--------------------------------------------");
  for (int i = 8; i > 0; i--) {
    Console.WriteLine(ins_list [count - i]
                          .ToString());
  }

  Console.WriteLine("-------------------------------------------");

  for (int index = 5; index < 8; index++) {
    ins_list[count - index].opcode = System.Reflection.Emit.OpCodes.Nop;
    ins_list[count - index].operand = null;
  }
  count += 4;

  for (int i = 0; i < 4; i++) {
    ins_list.Insert(count - 5, new CodeInstruction(
                                 System.Reflection.Emit.OpCodes.Nop, null));
  }

  Console.WriteLine("\n[!] After Patch");
  Console.WriteLine("-------------------------------------------");

  for (int i = 12; i > 0; i--) {
    Console.WriteLine(ins_list [count - i]
                          .ToString());
  }
  Console.WriteLine("-------------------------------------------\n");
  instructions = ins_list.AsEnumerable();
  return instructions;
}
```

(Complete solution in Appendix B)

There is another issue, the parent caller methods expect arbitrary types of objects returning from this method call sequence. This is going to cause problems when .NET fails to cast the ***DynamicMethod*** to arbitrary types. We need to code another patch, this time a Postfix that will save the ***DynamicMethod*** object for us and instead return the expected type. This can be done by declaring a static variable to save the ***DynamicMethod*** object and instead replace the return value with the expected type. This Postfix patch can be coded as follows.

```
public static void Postfix(ref Object __result) {

  Console.WriteLine("[+] Invoking Postfix");
  returnedMethod = (DynamicMethod) __result;
  StackTrace stackTrace = new StackTrace();
  string retType =
      ((MethodInfo) stackTrace.GetFrame(5).GetMethod()).ReturnType.Name;
  Console.WriteLine("[!] Return Type: " + retType);

  if (retType == "Int32" || retType == "FLARE06" || retType == "FLARE07" ||
      retType == "FLARE08") {
    __result = 0;
  } else if (retType == "UInt32") {
    __result = (uint) 0;
  } else if (retType == "Char") {
    __result = '0';
  } else if (retType == "Boolean") {
    __result = true;
  } else {
    __result = null;
  }
}
```

(Complete solution in Appendix B)

To get these patches to work, the following needs to be implemented in code.

1. A mapping of ***flared_*** methods to their proxies and types.
2. Creation of a delegate for the retrieved ***DynamicMethod*** object (to finalize the object)
3. Invocation of ***flare_74*** (Initialization method) via reflection to initialize the core engine methods.
4. Conversion of Dynamic IL to Static.
5. Method Body replacement.
6. Assembly rewrite.

For 1. the mapping is easy to create manually, however, we can use the following helper method to speed up the process.

```
ModuleDefMD moduleDef = ModuleDefMD.Load(_path);

foreach (var type in moduleDef.GetTypes()) {
  foreach (var method in type.Methods) {
    if (method.Name.Contains("flared_")) {
      Console.WriteLine("new List<string> { \"\", \"" + method.Name + "\", \"" +
                        type.FullName + "\" },");
    }
  }
}
```

(Complete solution in Appendix B)

The proxy names can be manually filled in afterwards. For 2 we can use a dynamic delegate generation approach to create delegates at runtime without hardcoding. This can be seen in Appendix B. Point 4 requires a long approach where the IL is parsed and the tokens are manually converted to match the outside context, however fortunately for us dnlib provides a convenience method ***DynamicMethodBodyReader*** which automates all of this. For someone interested in the longer method the selected answer of this question does about half of the work: https://stackoverflow.com/questions/4148297/resolving-the-tokens-found-in-the-il-from-a-dynamic-method. Following code shows these steps.

```
public static void FlareOn9_ConvertToStatic_Phase_2_Dynamically(
    string _path, string _dump_path) {
  List<List<string>> funcs = new List<List<string>>(){PASTE MAPPING HERE}

  Assembly asm = Assembly.LoadFrom(_path);
  Module module = asm.GetModules()[0];
  ModuleDefMD moduleDef = ModuleDefMD.Load(_path);
  DelegateTypeFactory df = new DelegateTypeFactory();

  Console.WriteLine(
      "[+] Dynamic to Static conversion phase 2 using dynamic method...");

  Type _flare15 = module.GetType("FlareOn.Backdoor.FLARE15");
  Harmony harmony = new Harmony("flareon");
  MethodBase original_method = AccessTools.Method(_flare15, "flared_67");
  Object dummy = new Object();
  MethodInfo postfix_patch =
      SymbolExtensions.GetMethodInfo(() => Patches.Postfix(ref dummy));
  MethodInfo transpiler_patch =
      SymbolExtensions.GetMethodInfo(() => Patches.Transpiler(null));
  harmony.Patch(original_method, null, new HarmonyMethod(postfix_patch),
              new HarmonyMethod(transpiler_patch));
  MethodInfo methodInfo_74 = _flare15.GetMethod("flare_74");
  Console.WriteLine(
      "[+] Invoking Intialiser Method: FlareOn.Backdoor.FLARE15.flare_74");
  methodInfo_74.Invoke(null, null);

  foreach (List<string> funcs_tuple in funcs) {
    Type _type = module.GetType(funcs_tuple[2]);
    MethodInfo proxy_methodInfo = _type.GetMethod(funcs_tuple[0]);
    MethodInfo target_methodInfo = _type.GetMethod(funcs_tuple[1]);
    Console.WriteLine("[+] Invoking Proxy: " + funcs_tuple[2] + "." +
                      funcs_tuple[0]);
    proxy_methodInfo.Invoke(
        null, new object[proxy_methodInfo.GetParameters().Length]);
    Console.WriteLine("[+] Recovering Method: " + funcs_tuple[2] + "." +
                      funcs_tuple[1]);
    DynamicMethod proxy_dynamicMethod = Patches.returnedMethod;
    Delegate proxy_delegate = proxy_dynamicMethod.CreateDelegate(
        df.CreateDelegateType(proxy_methodInfo));
    DynamicMethodBodyReader dynamicReader =
        new DynamicMethodBodyReader(moduleDef, proxy_delegate.Method);
    dynamicReader.Read();
    MethodDef target_methodDef = dynamicReader.GetMethod();

    foreach (var type in moduleDef.GetTypes()) {
      foreach (var method in type.Methods) {
        if (method.Name == funcs_tuple[1]) {
          Console.WriteLine("[+] Patching Method\n");
          method.FreeMethodBody();
          method.Body = target_methodDef.Body;
        }
      }
    }
  }

  ModuleWriterOptions mwo = new ModuleWriterOptions(moduleDef);
  mwo.MetadataOptions.Flags =
      MetadataFlags.KeepOldMaxStack | MetadataFlags.PreserveAll;
  mwo.Logger = DummyLogger.NoThrowInstance;
  mwo.MetadataOptions.PreserveHeapOrder(moduleDef, true);
  Console.WriteLine("[+] Dumping to: " + _dump_path);
  moduleDef.Write(_dump_path, mwo);
}
```

(Complete solution in Appendix B)

Running this will complete the conversion resulting in a fully fixed assembly.

Conversion using Static Approach

The second method to clean the challenge without delving too much into .NET internals requires the following steps and is based on the analysis described in Table 1. The reason this method is possible is because the metadata of the corrupted methods was left behind in the assembly to provide for the possibility of this alternate solution.

1. Iterate through all methods
2. Generate method hashes replicating the technique implemented in the challenge
3. Retrieve matching sections based on the generated hash
    1. We saw in the initial analysis phase that the ***Virtual Size*** of sections is smaller than the ***Raw Size***. When we read in the sections for static conversion, we will only be reading them up to the ***Virtual Size*** of the section even though the assembly will not be loaded in memory at that point.
4. RC4 decrypt the section using the key ***{ 0x12, 0x78, 0xab, 0xdf }***
5. Parse the IL and fix it by xor decoding the tokens using the key ***{ 0xbd, 0xa6, 0x98, 0xa2 }***
6. Raw patch the fixed method bytes back into the assembly without using any .NET re-writing libraries

In many of these cases parts of the code from decompiled core engine methods can be adapted and reused. We will use dnlib only for (1) and after performing the listed steps will patch back the bytes directly into the binary after making required calculations in a completely raw way without the use of dnlib. Following shows these steps being followed. Like previous code snippets, this does not show the actual implementation details, the complete solution for both Static and Dynamic conversion can be found in Appendix B.

```
public static void FlareOn9_ConvertToStatic_Phase_2_Statically(
    string _path, string _dump_path) {

  ModuleDefMD moduleDef = ModuleDefMD.Load(_path);
  Assembly asm = Assembly.LoadFrom(_path);
  Module module = asm.GetModules()[0];
  byte[] targetAssembly = File.ReadAllBytes(_path);
  byte[] il_data = null;
  long methodBodyOffset = 0;

  Console.WriteLine(
      "[+] Dynamic to Static conversion phase 2 using static method...");
  foreach (var type in moduleDef.GetTypes()) {
    foreach (var method in type.Methods) {
      string hash = GenerateMethodHash(method.MDToken.ToInt32(), module);
      byte[] encrypted_section =
          GetEncryptedSection(hash, moduleDef, targetAssembly);

      if (encrypted_section != null) {
        Console.WriteLine("[+] Raw Patching Method: " + type.FullName + "." +
                          method.Name);
        il_data =
            RC4.Decrypt(new byte[]{0x12, 0x78, 0xab, 0xdf}, encrypted_section);
        il_data = DecryptTokens(il_data, 0xa298a6bd);
        methodBodyOffset =
            (long) moduleDef.Metadata.PEImage.ToFileOffset(method.RVA) +
            method.Body.HeaderSize;
        Array.Copy(il_data, 0, targetAssembly, methodBodyOffset,
                   il_data.Length);
      }
    }
  }

  Console.WriteLine("\n[+] Dumping to: " + _dump_path);
  File.WriteAllBytes(_dump_path, targetAssembly);
}
```

(Complete solution in Appendix B)

Analysis Phase 3

Now that a clean assembly has been obtained, it is only a matter of performing traditional source code analysis. At this point it would be advisable to export the complete decompilation as a Visual Studio project using dnSpy as this would allow easy renaming/refactoring of the code making the static analysis process smoother.

Spending some time reading and understanding the source code shows that the challenge is a Backdoor that communicates with its C2 over DNS. The backdoor consists of a State Machine that has been initialized in the class ***FLARE13***. The states have been enumerated in ***FLARE08*** and analyzing the code will reveal that they map to the descriptions in Table 2.

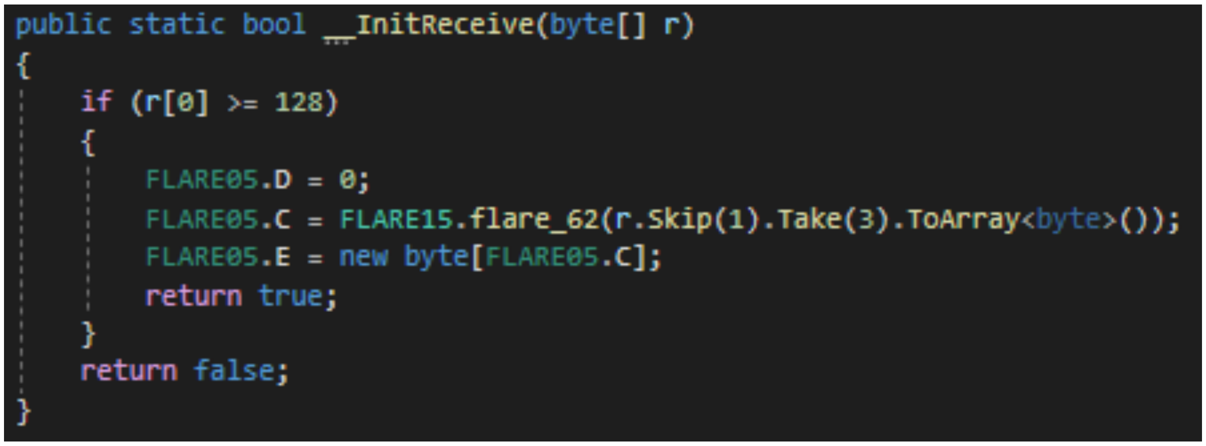
State	Description
A	Begin
B	Sleep
C	Active
D	Receive
E	Execute Command

F	Send
G	Send and Receive
H	Sleep

Table 2.

The backdoor reaches out to its C2 which is ***flare-on.com*** in this case and the state of the backdoor depends on the responses it receives. Upon execution the Backdoor will switch to the *Alive* state where on the first execution it will attempt to retrieve an agent ID from the C2. The initial beacon is based on ***FLARE03.alive_key,*** the details of which are not important. However, the response that is returned from the C2 is used to set ***FLARE03._agent_id***. Any IP can be returned, and the 4th octet of the response IP will be set as the agent ID.

From here on out to send a command to the backdoor, the first octet of the response IP must be greater or equal than 128 as can be seen in Figure 14.



```
public static bool __InitReceive(byte[] r)
{
    if (r[0] >= 128)
    {
        FLARE05.D = 0;
        FLARE05.C = FLARE15.flare_62(r.Skip(1).Take(3).ToArray<byte>());
        FLARE05.E = new byte[FLARE05.C];
        return true;
    }
    return false;
}
```

Figure 14. First octet value check for receiving commands

The full payload (all 4 octets) of the response IP to be able to receive commands will be interpreted as follows.

```
<GREATER THAN 128>.<SIZE OF COMMAND BYTE #3>.<SIZE OF COMMAND BYTE #2>.<SIZE OF COMMAND BYTE
#1>
```

The subsequent IP response will be interpreted as follows.

```
<TYPE OF CMD>.<CMD BYTE #1>.<CMD BYTE #2>.<CMD BYTE #3>
```

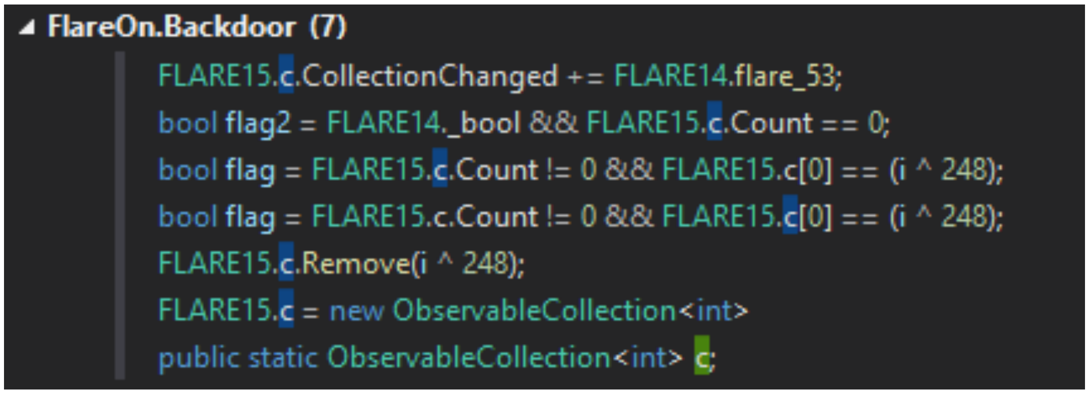
Analysis shows that the command type enums have been defined in ***FLARE06.TT*** and the command handler logic is present inside ***flared_56.*** The descriptions listed in Table 3 can be described following analysis of the command handler.

Enum	Value	Command Type
A	70	Execute arbitrary command
B	71	Decompress and execute arbitrary command
C	43	Execute hardcoded reconnaissance command
D	95	Download and execute
E	96	Download, decompress and execute

Table 3.

Interestingly it can be seen inside the command handler method ***flared_56***, that there exist many hardcoded and in most cases base64 encoded commands. Decoding them will reveal that all these commands perform actions related to system/victim reconnaissance. To execute these commands, the command type of ***FLARE06.TT.C*** (43) must be used.

Another interesting element is ***FLARE15.c*** which gets initialized way back in ***flare_74*** that we had skipped over. This is an ***ObservableCollection*** object which allows event handler callbacks to be added on element addition/removal. Searching for cross references to this object will reveal some interesting details. The references also show ***flare_53*** as the callback routine that gets setup for this object as can be seen in Figure 15.

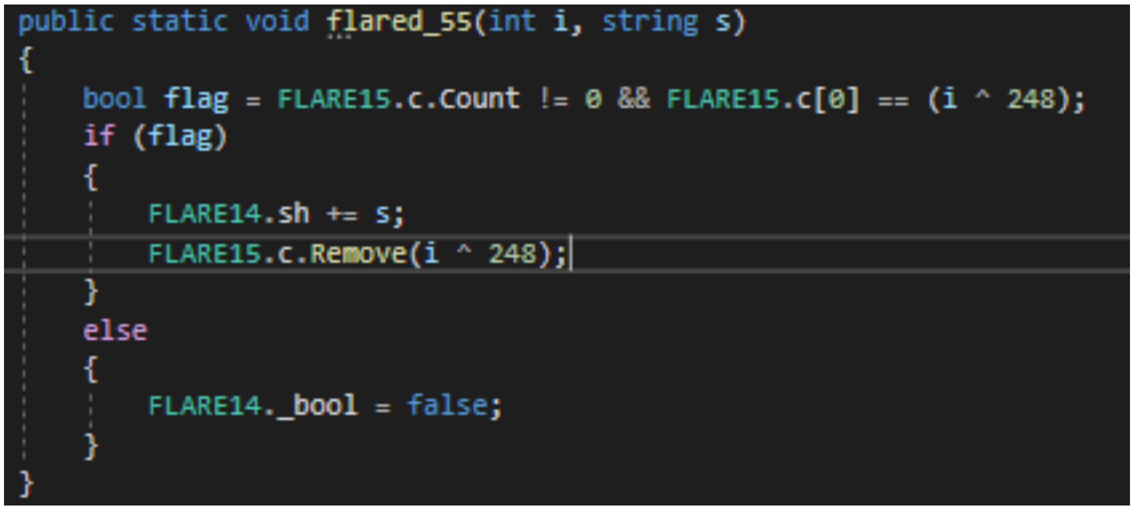


```
◢ FlareOn.Backdoor (7)
    FLARE15.c.CollectionChanged += FLARE14.flare_53;
    bool flag2 = FLARE14._bool && FLARE15.c.Count == 0;
    bool flag = FLARE15.c.Count != 0 && FLARE15.c[0] == (i ^ 248);
    bool flag = FLARE15.c.Count != 0 && FLARE15.c[0] == (i ^ 248);
    FLARE15.c.Remove(i ^ 248);
    FLARE15.c = new ObservableCollection<int>
    public static ObservableCollection<int> c;
```

Figure 15. XRefs to FLARE15.c

With each reconnaissance command, the following actions take place.

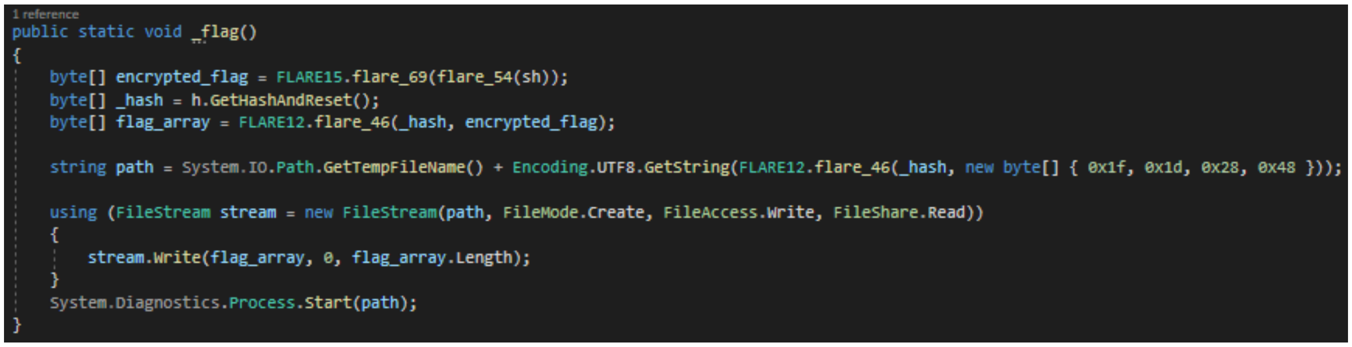
1. In most commands a *SHA256* hash object is added to by the combination of the command text and the result of ***flare_57***
2. Some arbitrary string value is appended to ***FLARE14.sh***
3. The command value is xor-ed with ***0xf8*** as shown in Figure 16 and removed from ***FLARE15.c*** if it exists
4. If the xor-ed command does not exist however, then the flag ***FLARE14._bool*** will be set to *false.*
5. This removal will trigger the callback method ***flare_53***



```
public static void flared_55(int i, string s)
{
    bool flag = FLARE15.c.Count != 0 && FLARE15.c[0] == (i ^ 248);
    if (flag)
    {
        FLARE14.sh += s;
        FLARE15.c.Remove(i ^ 248);
    }
    else
    {
        FLARE14._bool = false;
    }
}
```

Figure 16. Command id removal function

The callback routine checks if ***FLARE14._bool*** is *true*, and element count in ***FLARE15.c*** is zero. If this is the case then a very interesting method ***flare_55*** gets executed. Analyzing this method, it can be deduced that this method decrypts and launches the flag. Figure 17 shows the refactored version of this method.



```
1 reference
public static void _flag()
{
    byte[] encrypted_flag = FLARE15.flare_69(flare_54(sh));
    byte[] _hash = h.GetHashAndReset();
    byte[] flag_array = FLARE12.flare_46(_hash, encrypted_flag);

    string path = System.IO.Path.GetTempFileName() + Encoding.UTF8.GetString(FLARE12.flare_46(_hash, new byte[] { 0x1f, 0x1d, 0x28, 0x48 }));

    using (FileStream stream = new FileStream(path, FileMode.Create, FileAccess.Write, FileShare.Read))
    {
        stream.Write(flag_array, 0, flag_array.Length);
    }
    System.Diagnostics.Process.Start(path);
}
```

Figure 17. Flag decryption method

The analysis of this method shows that the following needs to happen for successful decryption of the flag.

1. All the reconnaissance commands need to be executed in the order specified by the elements in ***FLARE15.c*** after they have been xor-ed with ***0xf8***.
2. If even 1 out of sequence recon command is received, the flag will not decrypt, and the binary will have to be relaunched.
3. The commands when received in sequence will generate the string ***FLARE14.sh*** which will be reversed, and its first 8 characters will be used to identify the section that contains the encrypted flag. This will also generate a *SHA256* hash based on the combination of the command texts and ***flare_57*** return value. This hash will be used as the RC4 decryption key for the flag and the extension of the flag.
4. If the generated hash is incorrect, the flag will fail to decrypt.

Flag Decryption using Dynamic Approach

We will create a simple DNS server and specify a sequence of hardcoded IPs that will execute the reconnaissance commands in the correct order, thereby decrypting and launching the flag. Following are the requirements to achieve this.

1. The first response IP will be utilized in setting the agent ID
2. The command type should be 43 ***(FLARE06.TT.C***/recon***)***
3. The sequence of commands should be as follows (***FLARE15.c*** xor decoded with ***0xf8***)
    - 2,10,8,19,11,1,15,13,22,16,5,12,21,3,18,17,20,14,9,7,4
4. Set the VM DNS server IP to *127.0.0.1*

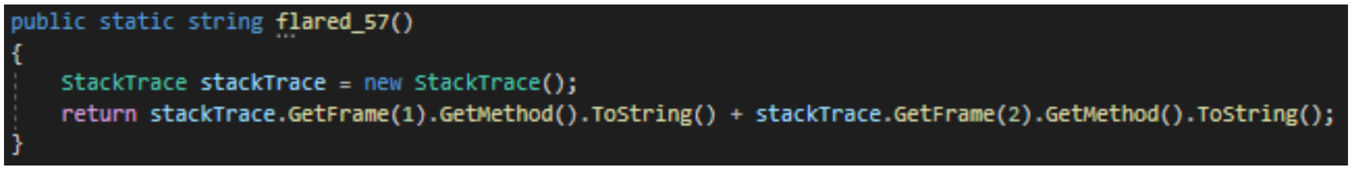
The following DNS server script will feed the IP responses required for the flag to be decrypted and launched.

```
import socket


class DNSQuery:
    # ref: https://code.activestate.com/recipes/491264/
    def __init__(self, data):
        self.data = data
        self.dominio = ''
        self.DnsType = ''

        hdns = data[-4:-2].encode("hex")
        if hdns == "0001":
            self.DnsType = 'A'
        elif hdns == "000f":
            self.DnsType = 'MX'
        elif hdns == "0002":
            self.DnsType = 'NS'
        elif hdns == "0010":
            self.DnsType = "TXT"
        else:
            self.DnsType = "Unknown"

        tipo = (ord(data[2]) >> 3) & 15
        if tipo == 0:
            ini = 12
            lon = ord(data[ini])
            while lon != 0:
                self.dominio += data[ini + 1:ini + lon + 1] + '.'
                ini += lon + 1
                lon = ord(data[ini])

    def response(self, ip):
        packet = ''
        if self.dominio:
            packet += self.data[:2] + "\x81\x80"
            packet += self.data[4:6] + self.data[4:6] + '\x00\x00\x00\x00'
            packet += self.data[12:]
            packet += '\xc0\x0c'
            packet += '\x00\x01\x00\x01\x00\x00\x00\x3c\x00\x04'
            packet += str.join('', map(lambda x: chr(int(x)), ip.split('.')))
        return packet


if __name__ == '__main__':
    ip_payload = ['192.168.1.203', # <agent_id>
                  '129.0.0.2',      # <command size>
                  '43.50.0.0',      # <type 43/recon command id>
                  '129.0.0.3',      # repeat for the full payload sequence
                  '43.49.48.0',
                  '129.0.0.2',
                  '43.56.0.0',
                  '129.0.0.3',
                  '43.49.57.0',
                  '129.0.0.3',
                  '43.49.49.0',
                  '129.0.0.2',
                  '43.49.0.0',
                  '129.0.0.3',
                  '43.49.53.0',
                  '129.0.0.3',
                  '43.49.51.0',
                  '129.0.0.3',
                  '43.50.50.0',
                  '129.0.0.3',
                  '43.49.54.0',
                  '129.0.0.2',
                  '43.53.0.0',
                  '129.0.0.3',
                  '43.49.50.0',
                  '129.0.0.3',
                  '43.50.49.0',
                  '129.0.0.2',
                  '43.51.0.0',
                  '129.0.0.3',
                  '43.49.56.0',
                  '129.0.0.3',
                  '43.49.55.0',
                  '129.0.0.3',
                  '43.50.48.0',
                  '129.0.0.3',
                  '43.49.52.0',
                  '129.0.0.2',
                  '43.57.0.0',
                  '129.0.0.2',
                  '43.55.0.0',
                  '129.0.0.2',
                  '43.52.0.0']

    ip_payload_count = len(ip_payload)
    index = -1
    prev_dom = ''

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind(('', 53))

    try:
        while index < ip_payload_count:
            try:
                data, addr = sock.recvfrom(1024)
                p = DNSQuery(data)

                if p.dominio.endswith('flare-on.com.'):
                    if prev_dom != p.dominio:
                        print('[%d/%d] prev: %s' % (index + 1, ip_payload_count, prev_dom))
                        print('[%d/%d] curr: %s' % (index + 1, ip_payload_count, p.dominio))
                        index += 1
                        prev_dom = p.dominio

                    _ip = ip_payload[index % ip_payload_count]

                    sock.sendto(p.response(_ip), addr)
                    print('[%d/%d] query: %s -> %s -> %s -> %s' % (
                        index + 1, ip_payload_count, addr[0], p.DnsType, p.dominio, _ip))
            except socket.error:
                pass

        sock.close()
    except KeyboardInterrupt:
        sock.close()
```

This script however will not work with the cleaned binary and will only work with the original challenge and will take around ~20 minutes to finish depending on the system. The processing is slow due to constant Dynamic IL generation. While the script will be very fast with the fully cleaned binary, the decrypted flag will be incorrect, the reason for which will be discussed in the following section of the solution.

Flag Decryption using Static Approach

We already have a solution for the challenge, however there is an alternate way to decrypt the flag statically and almost instantly without having to wait for ~20 minutes. We know all the ingredients that need to be factored in to get the correct hash to decrypt the flag except for one. The final *SHA256* hash is the result of the correct sequence of commands prepended with a string that is returned from ***flare_57***. This a proxy for the method shown in Figure 18.



```
public static string flared_57()
{
    StackTrace stackTrace = new StackTrace();
    return stackTrace.GetFrame(1).GetMethod().ToString() + stackTrace.GetFrame(2).GetMethod().ToString();
}
```

Figure 18. Strings from stack trace that factor into the flag decryption key

The method is simple enough and is attempting to retrieve the prototypes for 2 parent methods in its call stack. However, the result of this method call will be different depending on whether the parent function (command handler routine in this case) was being called statically or via a Dynamic Method. The correct string will only be returned if the parent method has been executed dynamically. Otherwise, we will end up with a corrupted flag, which is what happens if you run the DNS script with the final cleaned binary which has zero dynamic methods, and thus the incorrect stack traces.

We can either set a breakpoint on this inside Windbg using SOS.dll, or since we already have the solution code we can add a small function to it, invoke the proxy method (***flare_57***) via reflection and get the correct string quickly. Following code will achieve this for us.

```
public static void FlareOn9_GetStackFactor(string _path) {

  Assembly asm = Assembly.LoadFrom(_path);
  Module module = asm.GetModules()[0];

  Type _flare15 = module.GetType("FlareOn.Backdoor.FLARE15");
  MethodInfo methodInfo_74 = _flare15.GetMethod("flare_74");
  Console.WriteLine(
      "[+] Invoking Intialiser Method: FlareOn.Backdoor.FLARE15.flare_74");
  methodInfo_74.Invoke(null, null);

  Type _flare14 = module.GetType("FlareOn.Backdoor.FLARE14");
  MethodInfo methodInfo_57 = _flare14.GetMethod("flare_57");
  Console.WriteLine("\n[+] Stack Factor: " +
                    (string) methodInfo_57.Invoke(null, null));
}
```

(Complete solution in Appendix B)

The string returning from this is as follows.

```
System.Object InvokeMethod(System.Object, System.Object[], System.Signature,
Boolean)System.Object Invoke(System.Object, System.Reflection.BindingFlags,
System.Reflection.Binder, System.Object[], System.Globalization.CultureInfo)
```

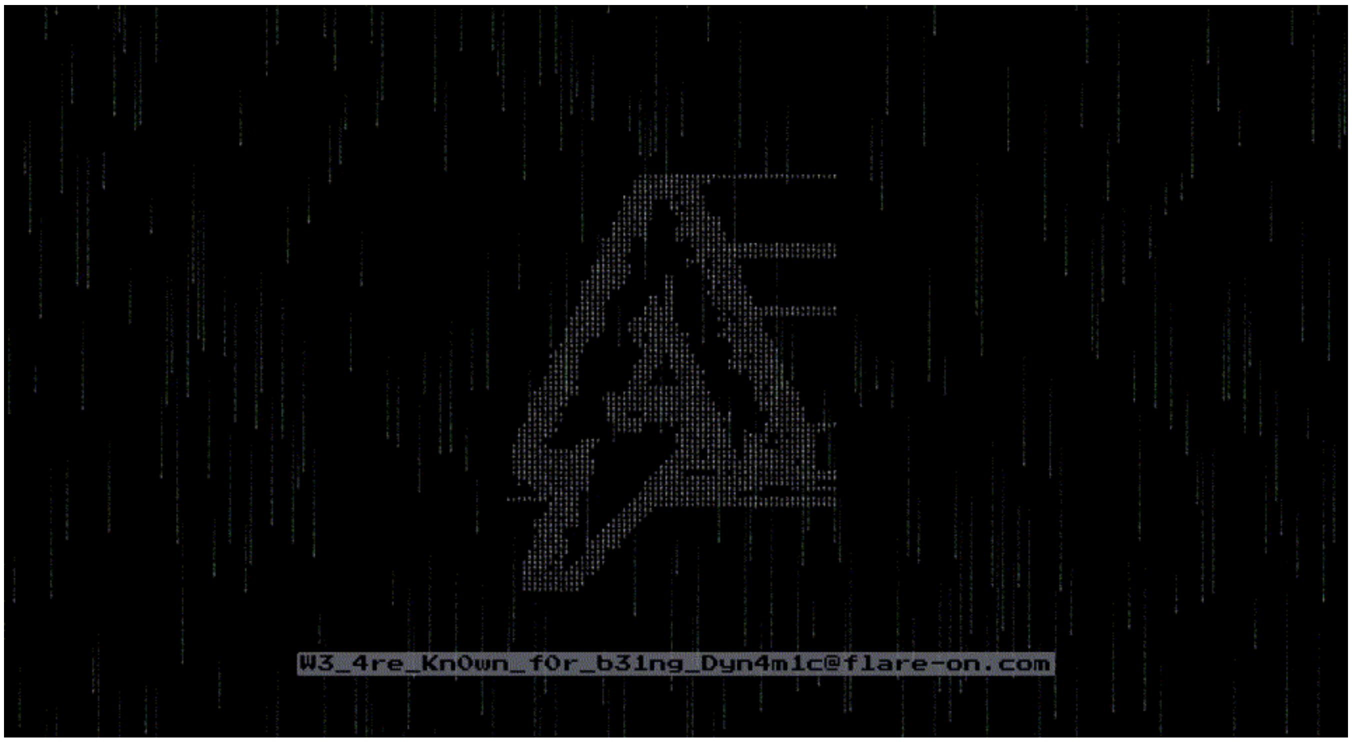
Another thing to be mindful of is that while all 21 reconnaissance commands are required to be executed in the correct sequence for dynamic flag decryption, the *SHA256* hash is generated factoring in only the first 20 commands from the sequence. Command '4' does not call any hashing methods. Based on this information we can create the following python script to statically decrypt the flag.

```
import os
import lief
import hashlib
import webbrowser
from arc4 import ARC4

if __name__ == '__main__':
    pe_path = <CHALLENGE PATH>
stack_factor = b 'System.Object InvokeMethod(System.Object, System.Object[], System.Signature,
'\
b 'Boolean)System.Object Invoke(System.Object, System.Reflection.BindingFlags, '\
b 'System.Reflection.Binder, System.Object[], System.Globalization.CultureInfo)'
flag_hash = 'd7df382b7146818c2e197e38709e87bfb5708e61139392e43c99743a79b2bea5' [::-1]
encrypted_flag_extension = bytes([0x1f, 0x1d, 0x28, 0x48])
hash_data_dict = {
    1: b
'RwBlAHQALQBOAGUAdABJAFAAQQBkAGQAcgBlAHMAcwAgAC0AQQBkAGQAcgBlAHMAcwBGAGEAbQBpAGwAeQAgAEkAUAB2A
DQAIAB8ACAAUwBlAGwAZQBjAHQALQBPAGIAagBlAGMAdAAgAEkAUABBAGQAZAByAGUAcwBzAA==',
    2: b
'RwBlAHQALQBOAGUAdABOAGUAaQBnAGgAYgBvAHIAIAAtAEEAZABkAHIAZQBzAHMARgBhAG0AaQBsAHkAIABJAFAAdgA0A
CAAfAAgAFMAZQBsAGUAYwB0AC0ATwBiAGoAZQBjAHQAIAAiAEkAUABBAEQARAByAGUAcwBzACIA',
    3: b 'whoami',
    5: b 'net user',
    7: b
'RwBlAHQALQBDAGgAaQBsAGQASQB0AGUAbQAgAC0AUABhAHQAaAAgACIAQwA6AFwAUAByAG8AZwByAGEAbQAgAEYAaQBsA
GUAcwAiACAAfAAgAFMAZQBsAGUAYwB0AC0ATwBiAGoAZQBjAHQAIABOAGEAbQBlAA==',
    8: b
'RwBlAHQALQBDAGgAaQBsAGQASQB0AGUAbQAgAC0AUABhAHQAaAAgACcAQwA6AFwAUAByAG8AZwByAGEAbQAgAEYAaQBsA
GUAcwAgACgAeAA4ADYAKQAnACAAfAAgAFMAZQBsAGUAYwB0AC0ATwBiAGoAZQBjAHQAIABOAGEAbQBlAA==',
    9: b
'RwBlAHQALQBDAGgAaQBsAGQASQB0AGUAbQAgAC0AUABhAHQAaAAgACcAQwA6ACcAIAB8ACAAUwBlAGwAZQBjAHQALQBPA
GIAagBlAGMAdAAgAE4AYQBtAGUA',
    10: b 'hostname',
    11: b
'RwBlAHQALQBOAGUAdABUAEMAUABDAG8AbgBuAGUAYwB0AGkAbwBuACAAfAAgAFcAaABlAHIAZQAtAE8AYgBqAGUAYwB0A
CAAewAkAF8ALgBTAHQAYQB0AGUAIAAtAGUAcQAgACIARQBzAHQAYQBiAGwAaQBzAGgAZQBkACIAfQAgAHwAIABTAGUAbAB
lAGMAdAAtAE8AYgBqAGUAYwB0ACAAIgBMAG8AYwBhAGwAQQBkAGQAcgBlAHMAcwAiACwAIAAiAEwAbwBjAGEAbABQAG8Ac
gB0ACIALAAgACIAUgBlAG0AbwB0AGUAQQBkAGQAcgBlAHMAcwAiACwAIAAiAFIAZQBtAG8AdABlAFAAbwByAHQAIgA=',
    12: b
'JAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANAAuADUAMAAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpA
CAAdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsADsAJAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANAA
uADUAMQAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpACAAdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsADsAJAAoAHAAa
QBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANgA1AC4ANgA1ACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvAGkAIAB0AHQ
AbAApACAALQBlAHEAIAAkAG4AdQBsAGwAOwAkACgAcABpAG4AZwAgAC0AbgAgADEAIAAxADAALgA2ADUALgA1ADMALgA1A
DMAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8AaQAgAHQAdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA7ACQAKABwAGkAbgB
nACAALQBuACAAMQAgADEAMAAuADYANQAuADIAMQAuADIAMAAwACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvAGkAIAB0AHQAb
AApACAALQBlAHEAIAAkAG4AdQBsAGwA',
    13: b
'bnNsb29rdXAgZmxhcmUtb24uY29tIHwgZmluZHN0ciAvaSBBZGRyZXNzO25zbG9va3VwIHdlYm1haWwuZmxhcmUtb24uY
29tIHwgZmluZHN0ciAvaSBBZGRyZXNz',
    14: b
'JAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4AMQAwAC4AMgAxAC4AMgAwADEAIAB8ACAAZgBpAG4AZABzAHQAcgAgA
C8AaQAgAHQAdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA7ACQAKABwAGkAbgBnACAALQBuACAAMQAgADEAMAAuADEAMAA
uADEAOQAuADIAMAAxACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvAGkAIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwAO
wAkACgAcABpAG4AZwAgAC0AbgAgADEAIAAxADAALgAxADAALgAxADkALgAyADAAMgAgAHwAIABmAGkAbgBkAHMAdAByACA
ALwBpACAAdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsADsAJAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4AMQAwA
C4AMgA0AC4AMgAwADAAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8AaQAgAHQAdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA
=',
    15: b
'JAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4AMQAwAC4AMQAwAC4ANAAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpA
CAAdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsADsAJAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4AMQAwAC4ANQA
wAC4AMQAwACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvAGkAIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwAOwAkACgAc
ABpAG4AZwAgAC0AbgAgADEAIAAxADAALgAxADAALgAyADIALgA1ADAAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8AaQAgAHQ
AdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA7ACQAKABwAGkAbgBnACAALQBuACAAMQAgADEAMAAuADEAMAAuADQANQAuA
DEAOQAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpACAAdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsAA==',
    16: b
'JAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANQAxAC4AMQAxACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvA
GkAIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwAOwAkACgAcABpAG4AZwAgAC0AbgAgADEAIAAxADAALgA2ADUALgA
2AC4AMQAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpACAAdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsADsAJAAoAHAAa
QBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANQAyAC4AMgAwADAAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8AaQAgAHQ
AdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA7ACQAKABwAGkAbgBnACAALQBuACAAMQAgADEAMAAuADYANQAuADYALgAzA
CAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvAGkAIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwA',
    17: b
'JAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANAA1AC4AMQA4ACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvA
GkAIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwAOwAkACgAcABpAG4AZwAgAC0AbgAgADEAIAAxADAALgA2ADUALgA
yADgALgA0ADEAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8AaQAgAHQAdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA7ACQAK
ABwAGkAbgBnACAALQBuACAAMQAgADEAMAAuADYANQAuADMANgAuADEAMwAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpACA
AdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsADsAJAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4ANgA1AC4ANQAxA
C4AMQAwACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvAGkAIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwA',
    18: b
'JAAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4AMQAwAC4AMgAyAC4ANAAyACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvA
GkAIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwAOwAkACgAcABpAG4AZwAgAC0AbgAgADEAIAAxADAALgAxADAALgA
yADMALgAyADAAMAAgAHwAIABmAGkAbgBkAHMAdAByACAALwBpACAAdAB0AGwAKQAgAC0AZQBxACAAJABuAHUAbABsADsAJ
AAoAHAAaQBuAGcAIAAtAG4AIAAxACAAMQAwAC4AMQAwAC4ANAA1AC4AMQA5ACAAfAAgAGYAaQBuAGQAcwB0AHIAIAAvAGk
AIAB0AHQAbAApACAALQBlAHEAIAAkAG4AdQBsAGwAOwAkACgAcABpAG4AZwAgAC0AbgAgADEAIAAxADAALgAxADAALgAxA
DkALgA1ADAAIAB8ACAAZgBpAG4AZABzAHQAcgAgAC8AaQAgAHQAdABsACkAIAAtAGUAcQAgACQAbgB1AGwAbAA=',
    19: b
'JChwaW5nIC1uIDEgMTAuNjUuNDUuMyB8IGZpbmRzdHIgL2kgdHRsKSAtZXEgJG51bGw7JChwaW5nIC1uIDEgMTAuNjUuN
C41MiB8IGZpbmRzdHIgL2kgdHRsKSAtZXEgJG51bGw7JChwaW5nIC1uIDEgMTAuNjUuMzEuMTU1IHwgZmluZHN0ciAvaSB
0dGwpIC1lcSAkbnVsbDskKHBpbmcgLW4gMSBmbGFyZS1vbi5jb20gfCBmaW5kc3RyIC9pIHR0bCkgLWVxICRudWxs',
    20: b
'RwBlAHQALQBOAGUAdABJAFAAQwBvAG4AZgBpAGcAdQByAGEAdABpAG8AbgAgAHwAIABGAG8AcgBlAGEAYwBoACAASQBQA
HYANABEAGUAZgBhAHUAbAB0AEcAYQB0AGUAdwBhAHkAIAB8ACAAUwBlAGwAZQBjAHQALQBPAGIAagBlAGMAdAAgAE4AZQB
4AHQASABvAHAA',
    21: b
'RwBlAHQALQBEAG4AcwBDAGwAaQBlAG4AdABTAGUAcgB2AGUAcgBBAGQAZAByAGUAcwBzACAALQBBAGQAZAByAGUAcwBzA
EYAYQBtAGkAbAB5ACAASQBQAHYANAAgAHwAIABTAGUAbABlAGMAdAAtAE8AYgBqAGUAYwB0ACAAUwBFAFIAVgBFAFIAQQB
kAGQAcgBlAHMAcwBlAHMA',
    22: b 'systeminfo | findstr /i "Domain"'
}
sequence = [0xfa, 0xf2, 0xf0, 0xeb, 0xf3, 0xf9, 0xf7, 0xf5, 0xee, 0xe8, 0xfd, 0xf4, 0xed,
0xfb, 0xea, 0xe9, 0xec,
    0xf6, 0xf1, 0xff]
skip = [3, 5, 10, 22]
encrypted_flag = None

pe = lief.PE.parse(pe_path)
sha256 = hashlib.new('sha256')

for section in pe.sections:
    if flag_hash.startswith(section.name):
    encrypted_flag = bytes(section.content)
```

```
for item in sequence:
    key = item ^ 0xf8

if key in skip:
    sha256.update(stack_factor + hash_data_dict[key])
else :
    sha256.update(stack_factor + b 'powershell -exec bypass -enc "' + hash_data_dict[key] + b
'"')

decrypted_flag = ARC4(sha256.digest()).decrypt(encrypted_flag)
flag_file_name = 'flag' + ARC4(sha256.digest()).decrypt(encrypted_flag_extension).decode("utf-
8")

with open(flag_file_name, 'wb') as outfile:
    outfile.write(decrypted_flag)

webbrowser.open('file://' + os.path.join(os.getcwd(), flag_file_name))
```

Flag

The final decrypted flag is an animated GIF. If the flag was decrypted dynamically the challenge assembly will pop it up on the screen with the system default program once the correct payload has been processed.



Appendix A.

Flag	W3_4re_Kn0wn_f0r_b31ng_Dyn4m1c@flare-on.com
Flag Decryption Key (RC4)	0x94, 0x4c, 0xee, 0x4d, 0x42, 0x58, 0x3a, 0x53, 0xe8, 0x1a, 0x7e, 0xa5, 0xc9, 0xdc, 0x2b, 0xb6, 0xb9, 0x01, 0x21, 0x3a, 0x0e, 0xb3, 0x28, 0x6c, 0xa6, 0x9d, 0x3f, 0x01, 0xef, 0x84, 0xac, 0xbb
Section Decryption Key (RC4)	0x12, 0x78, 0xab, 0xdf
Token Decryption Key (xor)	0xbd, 0xa6, 0x98, 0xa2
Concatenated Hash for Flag Section Identification	5aeb2b97a34799c34e29393116e8075bfb78e90783e791e2c8186417b283fd7d
Tools Used	dnSpy, dnlib, Lib.Harmony, Visual Studio, Python, CFF Explorer

Fun fact: The hash for flag section identification is SHA256("W3_4re_Kn0wn_f0r_b31ng_Dyn4m1c@flare-on.com")

Appendix B.

Complete solution source code for dynamic conversion method, static conversion method and acquisition of stack factor that is required for static flag decryption.

Program.cs

```
using System;
using HarmonyLib;
using System.Reflection;
using System.Reflection.Emit;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Security.Cryptography;
using dnlib.DotNet;
using dnlib.DotNet.Emit;
using dnlib.DotNet.Writer;

namespace FlareOn9_DynamicToStatic {
class Program {
  static void Main(string[] args) {
    string _path;
    string _dump_path_phase_1;
    string _dump_path_phase_2;

    if (args.Length != 2) {
      PrintHelp();
      Environment.Exit(1);
    }

    _path = args[0];
    _dump_path_phase_1 =
        _path.Remove(_path.Length - 4, 4) + "_phase_1_cleaned.exe";
    _dump_path_phase_2 =
        _path.Remove(_path.Length - 4, 4) + "_phase_2_cleaned.exe";

    if (args[1] == "dynamic") {
      FlareOn9_ConvertToStatic_Phase_1(_path, _dump_path_phase_1);
      FlareOn9_ConvertToStatic_Phase_2_Dynamically(_dump_path_phase_1,
                                                   _dump_path_phase_2);
    } else if (args[1] == "static") {
      FlareOn9_ConvertToStatic_Phase_1(_path, _dump_path_phase_1);
      FlareOn9_ConvertToStatic_Phase_2_Statically(_dump_path_phase_1,
                                                  _dump_path_phase_2);
    } else if (args[1] == "getstackfactor") {
      FlareOn9_GetStackFactor(_path);
    } else {
      PrintHelp();
    }
  }
  public static void PrintHelp() {
    Console.WriteLine(
        "Run as: FlareOn9_DynamicToStatic.exe <binary path> <action>\n\nActions:\n- dynamic:
conversion using dynamic method\n- static: conversion using static method\n- getstackfactor:
get stack factor needed for static flag decryption");
  }

  public static void FlareOn9_ConvertToStatic_Phase_1(string _path,
                                                      string _dump_path) {

    <PASTE INITIALIZATION DATA HERE FROM THE DECOMPILATION OF flare_74>

    Dictionary<string, List<object>> method_map =
        new Dictionary<string, List<object>>{
            {"flared_35", new List<object>{pe_b, pe_m}},
            {"flared_47", new List<object>{d_b, d_m}},
            {"flared_66", new List<object>{gh_b, gh_m}},
            {"flared_67", new List<object>{cl_b, cl_m}},
            {"flared_68", new List<object>{rt_b, null}},
            {"flared_69", new List<object>{gs_b, gs_m}},
            {"flared_70", new List<object>{wl_b, wl_m}}};

    ModuleDefMD moduleDef = ModuleDefMD.Load(_path);
    byte[] targetAssembly = File.ReadAllBytes(_path);
    long methodBodyOffset = 0;
    int token = 0;
    uint index = 0;

    Console.WriteLine("[+] Dynamic to Static conversion phase 1...");

    foreach (KeyValuePair<string, List<object>> method_data in method_map) {
      foreach (var type in moduleDef.GetTypes()) {
        foreach (var method in type.Methods) {
          if (method.Name == method_data.Key) {
            Console.WriteLine("[+] Raw Patching Method: " + type.FullName +
                              "." + method.Name);
            if (method_data.Value[1] != null) {
              foreach (KeyValuePair<uint, int> item in (Dictionary<uint, int>)
                           method_data.Value[1]) {
                token = item.Value;
                index = item.Key;

                ((byte[]) method_data.Value[0])[index] = (byte) token;
                ((byte[]) method_data.Value[0])[index + 1] = (byte)(token >> 8);
                ((byte[]) method_data.Value[0])[index + 2] =
                    (byte)(token >> 16);
                ((byte[]) method_data.Value[0])[index + 3] =
                    (byte)(token >> 24);
              }
            }

            methodBodyOffset =
                (long) moduleDef.Metadata.PEImage.ToFileOffset(method.RVA) +
                method.Body.HeaderSize;
            Array.Copy((byte[]) method_data.Value[0], 0, targetAssembly,
                       methodBodyOffset,
                       ((byte[]) method_data.Value[0]).Length);
          }
        }
      }
    }
    Console.WriteLine("\n[+] Dumping to: " + _dump_path + "\n");
    File.WriteAllBytes(_dump_path, targetAssembly);
  }

  public static void FlareOn9_ConvertToStatic_Phase_2_Dynamically(
      string _path, string _dump_path) {
    List<List<string>> funcs = new List<List<string>>(){
        new List<string>{"flare_01", "flared_00", "FlareOn.Backdoor.FLARE01"},
        new List<string>{"flare_02", "flared_01", "FlareOn.Backdoor.FLARE01"},
        new List<string>{"flare_03", "flared_02", "FlareOn.Backdoor.FLARE02"},
        new List<string>{"flare_04", "flared_03", "FlareOn.Backdoor.FLARE02"},
        new List<string>{"flare_05", "flared_04", "FlareOn.Backdoor.FLARE02"},
        new List<string>{"flare_06", "flared_05", "FlareOn.Backdoor.FLARE02"},
        new List<string>{"flare_07", "flared_06", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_08", "flared_07", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_09", "flared_08", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_10", "flared_09", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_11", "flared_10", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_12", "flared_11", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_13", "flared_12", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_14", "flared_13", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_15", "flared_14", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_16", "flared_15", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_17", "flared_16", "FlareOn.Backdoor.FLARE03"},
        new List<string>{"flare_18", "flared_17", "FlareOn.Backdoor.FLARE04"},
        new List<string>{"flare_19", "flared_18", "FlareOn.Backdoor.FLARE04"},
        new List<string>{"flare_19", "flared_19", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_20", "flared_20", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_21", "flared_21", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_22", "flared_22", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_23", "flared_23", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_24", "flared_24", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_25", "flared_25", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_26", "flared_26", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_27", "flared_27", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_28", "flared_28", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_29", "flared_29", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_30", "flared_30", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_31", "flared_31", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_32", "flared_32", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_33", "flared_33", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_34", "flared_34", "FlareOn.Backdoor.FLARE05"},
        new List<string>{"flare_38", "flared_36", "FlareOn.Backdoor.FLARE09"},
        new List<string>{"flare_39", "flared_37", "FlareOn.Backdoor.FLARE09"},
        new List<string>{"Main", "flared_38", "FlareOn.Backdoor.Program"},
        new List<string>{"flare_72", "flared_39", "FlareOn.Backdoor.Program"},
        new List<string>{"flare_73", "flared_40", "FlareOn.Backdoor.Program"},
        new List<string>{"flare_40", "flared_41", "FlareOn.Backdoor.FLARE10"},
        new List<string>{"flare_41", "flared_42", "FlareOn.Backdoor.FLARE11"},
        new List<string>{"flare_42", "flared_43", "FlareOn.Backdoor.FLARE11"},
        new List<string>{"flare_43", "flared_44", "FlareOn.Backdoor.FLARE11"},
        new List<string>{"flare_44", "flared_45", "FlareOn.Backdoor.FLARE11"},
        new List<string>{"flare_45", "flared_46", "FlareOn.Backdoor.FLARE11"},
        new List<string>{"flare_48", "flared_48", "FlareOn.Backdoor.FLARE13"},
        new List<string>{"flare_49", "flared_49", "FlareOn.Backdoor.FLARE13"},
        new List<string>{"flare_50", "flared_50", "FlareOn.Backdoor.FLARE13"},
        new List<string>{"flare_51", "flared_51", "FlareOn.Backdoor.FLARE14"},
        new List<string>{"flare_53", "flared_52", "FlareOn.Backdoor.FLARE14"},
        new List<string>{"flare_54", "flared_53", "FlareOn.Backdoor.FLARE14"},
        new List<string>{"flare_55", "flared_54", "FlareOn.Backdoor.FLARE14"},
        new List<string>{"flare_56", "flared_55", "FlareOn.Backdoor.FLARE14"},
        new List<string>{"flare_52", "flared_56", "FlareOn.Backdoor.FLARE14"},
        new List<string>{"flare_57", "flared_57", "FlareOn.Backdoor.FLARE14"},
        new List<string>{"flare_58", "flared_58", "FlareOn.Backdoor.FLARE15"},
        new List<string>{"flare_59", "flared_59", "FlareOn.Backdoor.FLARE15"},
        new List<string>{"flare_60", "flared_60", "FlareOn.Backdoor.FLARE15"},
        new List<string>{"flare_61", "flared_61", "FlareOn.Backdoor.FLARE15"},
        new List<string>{"flare_62", "flared_62", "FlareOn.Backdoor.FLARE15"},
        new List<string>{"flare_63", "flared_63", "FlareOn.Backdoor.FLARE15"},
        new List<string>{"flare_64", "flared_64", "FlareOn.Backdoor.FLARE15"},
        new List<string>{"flare_65", "flared_65", "FlareOn.Backdoor.FLARE15"}};

    Assembly asm = Assembly.LoadFrom(_path);
    Module module = asm.GetModules()[0];
    ModuleDefMD moduleDef = ModuleDefMD.Load(_path);
    DelegateTypeFactory df = new DelegateTypeFactory();

    Console.WriteLine(
        "[+] Dynamic to Static conversion phase 2 using dynamic method...");

    Type _flare15 = module.GetType("FlareOn.Backdoor.FLARE15");
    Harmony harmony = new Harmony("flareon");
    MethodBase original_method = AccessTools.Method(_flare15, "flared_67");
    Object dummy = new Object();
    MethodInfo postfix_patch =
        SymbolExtensions.GetMethodInfo(() => Patches.Postfix(ref dummy));
    MethodInfo transpiler_patch =
        SymbolExtensions.GetMethodInfo(() => Patches.Transpiler(null));
    harmony.Patch(original_method, null, new HarmonyMethod(postfix_patch),
                  new HarmonyMethod(transpiler_patch));
    MethodInfo methodInfo_74 = _flare15.GetMethod("flare_74");
    Console.WriteLine(
        "[+] Invoking Initializer Method: FlareOn.Backdoor.FLARE15.flare_74");
    methodInfo_74.Invoke(null, null);  // Invoke Initialization method

    foreach (List<string> funcs_tuple in funcs) {
      Type _type = module.GetType(funcs_tuple[2]);
      MethodInfo proxy_methodInfo = _type.GetMethod(funcs_tuple[0]);
      MethodInfo target_methodInfo = _type.GetMethod(funcs_tuple[1]);
      Console.WriteLine("[+] Invoking Proxy: " + funcs_tuple[2] + "." +
                        funcs_tuple[0]);
      proxy_methodInfo.Invoke(
          null, new object[proxy_methodInfo.GetParameters().Length]);
      Console.WriteLine("[+] Recovering Method: " + funcs_tuple[2] + "." +
                        funcs_tuple[1]);
      DynamicMethod proxy_dynamicMethod = Patches.returnedMethod;
      Delegate proxy_delegate = proxy_dynamicMethod.CreateDelegate(
          df.CreateDelegateType(proxy_methodInfo));
      DynamicMethodBodyReader dynamicReader =
          new DynamicMethodBodyReader(moduleDef, proxy_delegate.Method);
      dynamicReader.Read();
      MethodDef target_methodDef = dynamicReader.GetMethod();

      foreach (var type in moduleDef.GetTypes()) {
        foreach (var method in type.Methods) {
          if (method.Name == funcs_tuple[1]) {
            Console.WriteLine("[+] Patching Method\n");
            method.FreeMethodBody();
            method.Body = target_methodDef.Body;
          }}}}

    ModuleWriterOptions mwo = new ModuleWriterOptions(moduleDef);
    mwo.MetadataOptions.Flags =
        MetadataFlags.KeepOldMaxStack | MetadataFlags.PreserveAll;
    mwo.Logger = DummyLogger.NoThrowInstance;
    mwo.MetadataOptions.PreserveHeapOrder(moduleDef, true);
    Console.WriteLine("[+] Dumping to: " + _dump_path);
    moduleDef.Write(_dump_path, mwo);
  }

  public static void FlareOn9_ConvertToStatic_Phase_2_Statically(
      string _path, string _dump_path) {
    ModuleDefMD moduleDef = ModuleDefMD.Load(_path);
    Assembly asm = Assembly.LoadFrom(_path);
    Module module = asm.GetModules()[0];
    byte[] targetAssembly = File.ReadAllBytes(_path);
    byte[] il_data = null;
    long methodBodyOffset = 0;

    Console.WriteLine(
        "[+] Dynamic to Static conversion phase 2 using static method...");

    foreach (var type in moduleDef.GetTypes()) {
      foreach (var method in type.Methods) {
        string hash = GenerateMethodHash(method.MDToken.ToInt32(), module);
        byte[] encrypted_section =
            GetEncryptedSection(hash, moduleDef, targetAssembly);

        if (encrypted_section != null) {
          Console.WriteLine("[+] Raw Patching Method: " + type.FullName + "." +
                          method.Name);
          il_data = RC4.Decrypt(new byte[]{0x12, 0x78, 0xab, 0xdf},
                              encrypted_section);
          il_data = DecryptTokens(il_data, 0xa298a6bd);
          methodBodyOffset =
              (long) moduleDef.Metadata.PEImage.ToFileOffset(method.RVA) +
              method.Body.HeaderSize;
          Array.Copy(il_data, 0, targetAssembly, methodBodyOffset,
                     il_data.Length);
        }}}

    Console.WriteLine("\n[+] Dumping to: " + _dump_path);
    File.WriteAllBytes(_dump_path, targetAssembly);
  }
  public static void FlareOn9_GetStackFactor(string _path) {
    Assembly asm = Assembly.LoadFrom(_path);
    Module module = asm.GetModules()[0];

    Type _flare15 = module.GetType("FlareOn.Backdoor.FLARE15");
    MethodInfo methodInfo_74 = _flare15.GetMethod("flare_74");
    Console.WriteLine(
        "[+] Invoking Intializer Method: FlareOn.Backdoor.FLARE15.flare_74");
    methodInfo_74.Invoke(null, null);

    Type _flare14 = module.GetType("FlareOn.Backdoor.FLARE14");
    MethodInfo methodInfo_57 = _flare14.GetMethod("flare_57");
    Console.WriteLine("\n[+] Stack Factor: " +
                      (string) methodInfo_57.Invoke(null, null));
  }

  public static byte[] DecryptTokens(byte[] il_data, uint xor_key) {
    uint code = 0;
    uint ctoken = 0;
    Dictionary<uint, Enums.OpType> graph = new Dictionary<uint, Enums.OpType>(){
        {0x0058, Enums.OpType.NONE},
        {0x00d6, Enums.OpType.NONE},
        {0x00d7, Enums.OpType.NONE},
        {0x005f, Enums.OpType.NONE},
        {0xfe00, Enums.OpType.NONE},
        {0x003b, Enums.OpType.DWORD_OFFSET},
        {0x002e, Enums.OpType.BYTE_OFFSET},
        {0x003c, Enums.OpType.DWORD_OFFSET},
        {0x002f, Enums.OpType.BYTE_OFFSET},
        {0x0041, Enums.OpType.DWORD_OFFSET},
        {0x0034, Enums.OpType.BYTE_OFFSET},
        {0x003d, Enums.OpType.DWORD_OFFSET},
        {0x0030, Enums.OpType.BYTE_OFFSET},
        {0x0042, Enums.OpType.DWORD_OFFSET},
        {0x0035, Enums.OpType.BYTE_OFFSET},
        {0x003e, Enums.OpType.DWORD_OFFSET},
        {0x0031, Enums.OpType.BYTE_OFFSET},
        {0x0043, Enums.OpType.DWORD_OFFSET},
        {0x0036, Enums.OpType.BYTE_OFFSET},
        {0x003f, Enums.OpType.DWORD_OFFSET},
        {0x0032, Enums.OpType.BYTE_OFFSET},
        {0x0044, Enums.OpType.DWORD_OFFSET},
        {0x0037, Enums.OpType.BYTE_OFFSET},
        {0x0040, Enums.OpType.DWORD_OFFSET},
        {0x0033, Enums.OpType.BYTE_OFFSET},
        {0x008c, Enums.OpType.TOKEN},
        {0x0038, Enums.OpType.DWORD_OFFSET},
        {0x002b, Enums.OpType.BYTE_OFFSET},
        {0x0001, Enums.OpType.NONE},
        {0x0039, Enums.OpType.DWORD_OFFSET},
        {0x002c, Enums.OpType.BYTE_OFFSET},
        {0x003a, Enums.OpType.DWORD_OFFSET},
        {0x002d, Enums.OpType.BYTE_OFFSET},
        {0x0028, Enums.OpType.TOKEN},
        {0x0029, Enums.OpType.TOKEN},
        {0x006f, Enums.OpType.TOKEN},
        {0x0074, Enums.OpType.TOKEN},
        {0xfe01, Enums.OpType.NONE},
        {0xfe02, Enums.OpType.NONE},
        {0xfe03, Enums.OpType.NONE},
        {0x00c3, Enums.OpType.NONE},
        {0xfe04, Enums.OpType.NONE},
        {0xfe05, Enums.OpType.NONE},
        {0xfe16, Enums.OpType.TOKEN},
        {0x00d3, Enums.OpType.NONE},
        {0x0067, Enums.OpType.NONE},
        {0x0068, Enums.OpType.NONE},
        {0x0069, Enums.OpType.NONE},
        {0x006a, Enums.OpType.NONE},
        {0x00d4, Enums.OpType.NONE},
        {0x008a, Enums.OpType.NONE},
        {0x00b3, Enums.OpType.NONE},
        {0x0082, Enums.OpType.NONE},
        {0x00b5, Enums.OpType.NONE},
        {0x0083, Enums.OpType.NONE},
        {0x00b7, Enums.OpType.NONE},
        {0x0084, Enums.OpType.NONE},
        {0x00b9, Enums.OpType.NONE},
        {0x0085, Enums.OpType.NONE},
        {0x00d5, Enums.OpType.NONE},
        {0x008b, Enums.OpType.NONE},
        {0x00b4, Enums.OpType.NONE},
        {0x0086, Enums.OpType.NONE},
        {0x00b6, Enums.OpType.NONE},
        {0x0087, Enums.OpType.NONE},
        {0x00b8, Enums.OpType.NONE},
        {0x0088, Enums.OpType.NONE},
        {0x00ba, Enums.OpType.NONE},
        {0x0089, Enums.OpType.NONE},
        {0x0076, Enums.OpType.NONE},
        {0x006b, Enums.OpType.NONE},
        {0x006c, Enums.OpType.NONE},
        {0x00e0, Enums.OpType.NONE},
        {0x00d2, Enums.OpType.NONE},
        {0x00d1, Enums.OpType.NONE},
        {0x006d, Enums.OpType.NONE},
        {0x006e, Enums.OpType.NONE},
        {0xfe17, Enums.OpType.NONE},
        {0x0070, Enums.OpType.TOKEN},
        {0x005b, Enums.OpType.NONE},
        {0x005c, Enums.OpType.NONE},
        {0x0025, Enums.OpType.NONE},
        {0xfe11, Enums.OpType.NONE},
        {0x00dc, Enums.OpType.NONE},
        {0xfe18, Enums.OpType.NONE},
        {0xfe15, Enums.OpType.TOKEN},
        {0x0075, Enums.OpType.TOKEN},
        {0x0027, Enums.OpType.TOKEN},
        {0xfe09, Enums.OpType.WORD_CONST},
        {0x0002, Enums.OpType.NONE},
        {0x0003, Enums.OpType.NONE},
        {0x0004, Enums.OpType.NONE},
        {0x0005, Enums.OpType.NONE},
        {0x000e, Enums.OpType.BYTE_CONST},
        {0xfe0a, Enums.OpType.WORD_CONST},
        {0x000f, Enums.OpType.BYTE_CONST},
        {0x0020, Enums.OpType.DWORD_CONST},
        {0x0016, Enums.OpType.NONE},
        {0x0017, Enums.OpType.NONE},
        {0x0018, Enums.OpType.NONE},
        {0x0019, Enums.OpType.NONE},
        {0x001a, Enums.OpType.NONE},
        {0x001b, Enums.OpType.NONE},
        {0x001c, Enums.OpType.NONE},
        {0x001d, Enums.OpType.NONE},
        {0x001e, Enums.OpType.NONE},
        {0x0015, Enums.OpType.NONE},
        {0x001f, Enums.OpType.BYTE_CONST},
        {0x0021, Enums.OpType.QWORD_CONST},
        {0x0022, Enums.OpType.DWORD_CONST},
        {0x0023, Enums.OpType.QWORD_CONST},
        {0x00a3, Enums.OpType.TOKEN},
        {0x0097, Enums.OpType.NONE},
        {0x0090, Enums.OpType.NONE},
        {0x0092, Enums.OpType.NONE},
        {0x0094, Enums.OpType.NONE},
        {0x0096, Enums.OpType.NONE},
        {0x0098, Enums.OpType.NONE},
        {0x0099, Enums.OpType.NONE},
        {0x009a, Enums.OpType.NONE},
        {0x0091, Enums.OpType.NONE},
        {0x0093, Enums.OpType.NONE},
        {0x0095, Enums.OpType.NONE},
        {0x008f, Enums.OpType.TOKEN},
        {0x007b, Enums.OpType.TOKEN},
        {0x007c, Enums.OpType.TOKEN},
        {0xfe06, Enums.OpType.TOKEN},
        {0x004d, Enums.OpType.NONE},
        {0x0046, Enums.OpType.NONE},
        {0x0048, Enums.OpType.NONE},
        {0x004a, Enums.OpType.NONE},
        {0x004c, Enums.OpType.NONE},
        {0x004e, Enums.OpType.NONE},
        {0x004f, Enums.OpType.NONE},
        {0x0050, Enums.OpType.NONE},
        {0x0047, Enums.OpType.NONE},
        {0x0049, Enums.OpType.NONE},
        {0x004b, Enums.OpType.NONE},
        {0x008e, Enums.OpType.NONE},
        {0xfe0c, Enums.OpType.WORD_CONST},
        {0x0006, Enums.OpType.NONE},
        {0x0007, Enums.OpType.NONE},
        {0x0008, Enums.OpType.NONE},
        {0x0009, Enums.OpType.NONE},
        {0x0011, Enums.OpType.BYTE_CONST},
        {0xfe0d, Enums.OpType.WORD_CONST},
        {0x0012, Enums.OpType.BYTE_CONST},
        {0x0014, Enums.OpType.NONE},
        {0x0071, Enums.OpType.TOKEN},
        {0x007e, Enums.OpType.TOKEN},
        {0x007f, Enums.OpType.TOKEN},
        {0x0072, Enums.OpType.TOKEN},
        {0x00d0, Enums.OpType.TOKEN},
        {0xfe07, Enums.OpType.TOKEN},
        {0x00dd, Enums.OpType.DWORD_OFFSET},
        {0x00de, Enums.OpType.BYTE_OFFSET},
        {0xfe0f, Enums.OpType.NONE},
        {0x00c6, Enums.OpType.TOKEN},
        {0x005a, Enums.OpType.NONE},
        {0x00d8, Enums.OpType.NONE},
        {0x00d9, Enums.OpType.NONE},
        {0x0065, Enums.OpType.NONE},
        {0x008d, Enums.OpType.TOKEN},
        {0x0073, Enums.OpType.TOKEN},
        {0xfe19, Enums.OpType.BYTE_CONST},
        {0x0000, Enums.OpType.NONE},
        {0x0066, Enums.OpType.NONE},
        {0x0060, Enums.OpType.NONE},
        {0x0026, Enums.OpType.NONE},
        {0x00fe, Enums.OpType.NONE},
        {0x00fd, Enums.OpType.NONE},
        {0x00fc, Enums.OpType.NONE},
        {0x00fb, Enums.OpType.NONE},
        {0x00fa, Enums.OpType.NONE},
        {0x00f9, Enums.OpType.NONE},
        {0x00f8, Enums.OpType.NONE},
        {0x00ff, Enums.OpType.NONE},
        {0xfe1e, Enums.OpType.NONE},
        {0xfe1d, Enums.OpType.NONE},
        {0x00c2, Enums.OpType.TOKEN},
        {0x005d, Enums.OpType.NONE},
        {0x005e, Enums.OpType.NONE},
        {0x002a, Enums.OpType.NONE},
        {0xfe1a, Enums.OpType.NONE},
        {0x0062, Enums.OpType.NONE},
        {0x0063, Enums.OpType.NONE},
        {0x0064, Enums.OpType.NONE},
        {0xfe1c, Enums.OpType.TOKEN},
        {0xfe0b, Enums.OpType.WORD_CONST},
        {0x0010, Enums.OpType.BYTE_CONST},
        {0x00a4, Enums.OpType.TOKEN},
        {0x009b, Enums.OpType.NONE},
        {0x009c, Enums.OpType.NONE},
        {0x009d, Enums.OpType.NONE},
        {0x009e, Enums.OpType.NONE},
        {0x009f, Enums.OpType.NONE},
        {0x00a0, Enums.OpType.NONE},
        {0x00a1, Enums.OpType.NONE},
        {0x00a2, Enums.OpType.NONE},
        {0x007d, Enums.OpType.TOKEN},
        {0x00df, Enums.OpType.NONE},
        {0x0052, Enums.OpType.NONE},
        {0x0053, Enums.OpType.NONE},
        {0x0054, Enums.OpType.NONE},
        {0x0055, Enums.OpType.NONE},
        {0x0056, Enums.OpType.NONE},
        {0x0057, Enums.OpType.NONE},
        {0x0051, Enums.OpType.NONE},
        {0xfe0e, Enums.OpType.WORD_CONST},
        {0x000a, Enums.OpType.NONE},
        {0x000b, Enums.OpType.NONE},
        {0x000c, Enums.OpType.NONE},
        {0x000d, Enums.OpType.NONE},
        {0x0013, Enums.OpType.BYTE_CONST},
        {0x0081, Enums.OpType.TOKEN},
        {0x0080, Enums.OpType.TOKEN},
        {0x0059, Enums.OpType.NONE},
        {0x00da, Enums.OpType.NONE},
        {0x00db, Enums.OpType.NONE},
        {0x0045, Enums.OpType.SWITCH},
        {0xfe14, Enums.OpType.NONE},
        {0x007a, Enums.OpType.NONE},
        {0xfe12, Enums.OpType.BYTE_CONST},
        {0x0079, Enums.OpType.TOKEN},
        {0x00a5, Enums.OpType.TOKEN},
        {0xfe13, Enums.OpType.NONE},
        {0x0061, Enums.OpType.NONE}};

    for (int i = 0; i < il_data.Length;) {
      if (il_data[i] == 0xfe) {
        code = (uint) 0xfe00 + il_data[i + 1];
        i++;
      } else {
        code = il_data[i];
      }

      Enums.OpType t = graph[code];
      i++;

      switch (t) {
        case Enums.OpType.BYTE_CONST:
        case Enums.OpType.BYTE_OFFSET:
          i += 1;
          break;
        case Enums.OpType.DWORD_CONST:
        case Enums.OpType.DWORD_OFFSET:
          i += 4;
          break;
        case Enums.OpType.NONE:
          break;
        case Enums.OpType.QWORD_CONST:
          i += 8;
          break;
        case Enums.OpType.SWITCH:
          i += 4 + (ReadToken(il_data, i) * 4);
          break;
        case Enums.OpType.TOKEN:
          ctoken = (uint) ReadToken(il_data, i);
          ctoken ^= xor_key;

          il_data[i] = (byte) ctoken;
          il_data[i + 1] = (byte)(ctoken >> 8);
          il_data[i + 2] = (byte)(ctoken >> 16);
          il_data[i + 3] = (byte)(ctoken >> 24);
          i += 4;
          break;
        case Enums.OpType.WORD_CONST:
          i += 2;
          break;
      }
    }

    return il_data;
  }

  public static int ReadToken(byte[] blob, int offset) {
    int token = 0;
    token = blob[offset + 3] * 0x1000000;
    token += blob[offset + 2] * 0x10000;
    token += blob[offset + 1] * 0x100;
    token += blob[offset];

    return token;
  }

  public static byte[] GetEncryptedSection(string hash, ModuleDefMD moduleDef,
                                           byte[] assembly) {
    byte[] il_data = null;

    foreach (dnlib.PE.ImageSectionHeader ih in moduleDef.Metadata.PEImage
                 .ImageSectionHeaders) {
      if (hash.StartsWith(ih.DisplayName)) {
        il_data = new byte[ih.VirtualSize];
        Array.Copy(assembly, ih.PointerToRawData, il_data, 0, il_data.Length);
        break;
      }
    }

    return il_data;
  }

  public static string GenerateMethodHash(int token, Module module) {
    MethodInfo methodInfo = null;
    ConstructorInfo constructorInfo = null;
    System.Reflection.MethodBody methodBody = null;
    string vars = "";
    string parms = "";
    byte[] attributes = null;
    byte[] returntype = null;
    byte[] convention = null;

    try {
      methodInfo = (MethodInfo) module.ResolveMethod(token);
      methodBody = methodInfo.GetMethodBody();
      attributes = Encoding.ASCII.GetBytes(methodInfo.Attributes.ToString());
      returntype = Encoding.ASCII.GetBytes(methodInfo.ReturnType.ToString());
      convention =
          Encoding.ASCII.GetBytes(methodInfo.CallingConvention.ToString());

      foreach (ParameterInfo pi in methodInfo.GetParameters()) {
        parms += pi.ParameterType;
      }
    } catch (InvalidCastException e) {
      constructorInfo = (ConstructorInfo) module.ResolveMethod(token);
      methodBody = constructorInfo.GetMethodBody();
```

```
      attributes =
          Encoding.ASCII.GetBytes(constructorInfo.Attributes.ToString());
      returntype = Encoding.ASCII.GetBytes(constructorInfo.Name);
      convention =
          Encoding.ASCII.GetBytes(constructorInfo.CallingConvention.ToString());

      foreach (ParameterInfo pi in constructorInfo.GetParameters()) {
        parms += pi.ParameterType;
      }
    }

    byte[] maxstacksize =
        Encoding.ASCII.GetBytes(methodBody.MaxStackSize.ToString());
    byte[] length = BitConverter.GetBytes(methodBody.GetILAsByteArray().Length);

    foreach (LocalVariableInfo lvi in methodBody.LocalVariables) {
      vars += lvi.LocalType;
    }

    byte[] bvars = Encoding.ASCII.GetBytes(vars);
    byte[] bparms = Encoding.ASCII.GetBytes(parms);

    IncrementalHash hash = IncrementalHash.CreateHash(HashAlgorithmName.SHA256);
    hash.AppendData(length);
    hash.AppendData(attributes);
    hash.AppendData(returntype);
    hash.AppendData(maxstacksize);
    hash.AppendData(bvars);
    hash.AppendData(bparms);
    hash.AppendData(convention);

    byte[] hash_bytes = hash.GetHashAndReset();
    StringBuilder result = new StringBuilder(hash_bytes.Length * 2);

    for (int i = 0; i < hash_bytes.Length; i++) {
      result.Append(hash_bytes [i]
                      .ToString("x2"));
    }

    return result.ToString();
  }
}}
```

Patches.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Reflection.Emit;
using System.Diagnostics;
using HarmonyLib;
```

```
namespace FlareOn9_DynamicToStatic {
class Patches {
  public static void Postfix(ref Object __result) {
    Console.WriteLine("[+] Invoking Postfix");
    returnedMethod = (DynamicMethod) __result;
    StackTrace stackTrace = new StackTrace();
    string retType =
        ((MethodInfo) stackTrace.GetFrame(5).GetMethod()).ReturnType.Name;
    Console.WriteLine("[!] Return Type: " + retType);

    if (retType == "Int32" || retType == "FLARE06" || retType == "FLARE07" ||
        retType == "FLARE08") {
      __result = 0;
    } else if (retType == "UInt32") {
      __result = (uint) 0;
    } else if (retType == "Char") {
      __result = '0';
    } else if (retType == "Boolean") {
      __result = true;
    } else {
      __result = null;
    }
  }
  public static IEnumerable<CodeInstruction> Transpiler(
      IEnumerable<CodeInstruction> instructions) {

    Console.WriteLine("[+] Invoking Transpiler");
    List<CodeInstruction> ins_list = instructions.ToList();
    int count = ins_list.Count;

    Console.WriteLine("\n[!] Before Patch");
    Console.WriteLine("--------------------------------------------");

    for (int i = 8; i > 0; i--) {
      Console.WriteLine(ins_list [count - i]
                        .ToString());
    }

    Console.WriteLine("--------------------------------------------");

    for (int index = 5; index < 8; index++) {
      ins_list[count - index].opcode = System.Reflection.Emit.OpCodes.Nop;
      ins_list[count - index].operand = null;
    }

    count += 4;

    for (int i = 0; i < 4; i++) {
      ins_list.Insert(count - 5, new CodeInstruction(
                               System.Reflection.Emit.OpCodes.Nop, null));
    }

    Console.WriteLine("\n[!] After Patch");
    Console.WriteLine("--------------------------------------------");

    for (int i = 12; i > 0; i--) {
      Console.WriteLine(ins_list [count - i]
                        .ToString());
    }
    Console.WriteLine("--------------------------------------------\n");
    instructions = ins_list.AsEnumerable();

    return instructions;
  }

  public static DynamicMethod returnedMethod;
}}
```

DelegateTypeFactory.cs

```
using System;
using System.Reflection;
using System.Reflection.Emit;
using System.Linq;

namespace FlareOn9_DynamicToStatic {
/// https://stackoverflow.com/questions/9505117/creating-delegates-dynamically-with-
parameter-names
class DelegateTypeFactory {
  private readonly ModuleBuilder m_module;

  public DelegateTypeFactory() {
    var assembly = AssemblyBuilder.DefineDynamicAssembly(
        new AssemblyName("DelegateTypeFactory"),
        AssemblyBuilderAccess.RunAndCollect);
    m_module = assembly.DefineDynamicModule("DelegateTypeFactory");
  }

  public Type CreateDelegateType(MethodInfo method) {
    string nameBase =
        string.Format("{0}{1}", method.DeclaringType.Name, method.Name);
    string name = GetUniqueName(nameBase);

    var typeBuilder =
        m_module.DefineType(name, TypeAttributes.Sealed | TypeAttributes.Public,
                            typeof(MulticastDelegate));

    var constructor = typeBuilder.DefineConstructor(
        MethodAttributes.RTSpecialName | MethodAttributes.HideBySig |
            MethodAttributes.Public,
        CallingConventions.Standard, new[]{typeof(object), typeof(IntPtr)});
    constructor.SetImplementationFlags(MethodImplAttributes.CodeTypeMask);

    var parameters = method.GetParameters();

    var invokeMethod = typeBuilder.DefineMethod(
        "Invoke",
        MethodAttributes.HideBySig | MethodAttributes.Virtual |
            MethodAttributes.Public,
        method.ReturnType, parameters.Select(p => p.ParameterType).ToArray());
    invokeMethod.SetImplementationFlags(MethodImplAttributes.CodeTypeMask);

    for (int i = 0; i < parameters.Length; i++) {
      var parameter = parameters[i];
      invokeMethod.DefineParameter(i + 1, ParameterAttributes.None,
                                    parameter.Name);
    }

    return typeBuilder.CreateType();
  }

  private string GetUniqueName(string nameBase) {
    int number = 2;
    string name = nameBase;
    while (m_module.GetType(name) != null) name = nameBase + number++;
    return name;
  }
}}
```

RC4.cs

```
namespace FlareOn9_DynamicToStatic {
public class RC4 {
  public static byte[] Decrypt(byte[] pwd, byte[] data) {
    int a, i, j, k, tmp;
    int[] key, box;
    byte[] cipher;

    key = new int[256];
    box = new int[256];
    cipher = new byte[data.Length];

    for (i = 0; i < 256; i++) {
      key[i] = pwd[i % pwd.Length];
      box[i] = i;
    }
    for (j = i = 0; i < 256; i++) {
      j = (j + box[i] + key[i]) % 256;
      tmp = box[i];
      box[i] = box[j];
      box[j] = tmp;
    }
    for (a = j = i = 0; i < data.Length; i++) {
      a++;
      a %= 256;
      j += box[a];
      j %= 256;
      tmp = box[a];
      box[a] = box[j];
      box[j] = tmp;
      k = box[((box[a] + box[j]) % 256)];
      cipher[i] = (byte)(data[i] ^ k);
    }
    return cipher;
  }
}}
```

Enums.cs

```
namespace FlareOn9_DynamicToStatic {
class Enums {
  public enum OpType {
    NONE,
    TOKEN,
    BYTE_OFFSET,
    DWORD_OFFSET,
    BYTE_CONST,
    WORD_CONST,
    DWORD_CONST,
    QWORD_CONST,
    SWITCH
  }}}
```

References

Andreas Pardeike. (2022). Lib.Harmony, A library for patching, replacing and decorating .NET and Mono methods during runtime:
https://github.com/pardeike/Harmony

Andreas Pardeike. (2022). Lib.Harmony Transpiler usage:
https://harmony.pardeike.net/articles/patching-transpiler.html

0xd4d. (2022). dnlib .NET module/assembly reader/writer library:
https://github.com/0xd4d/dnlib

0xd4d. (2022). DynamicMethodBodyReader from dnlib:
https://github.com/0xd4d/dnlib/blob/master/src/DotNet/Emit/DynamicMethodBodyReader.cs

Stackoverflow. (2017). Resolution for tokens inside dynamic IL:
https://stackoverflow.com/questions/4148297/resolving-the-tokens-found-in-the-il-from-a-dynamic-method

Stackoverflow. (2012). Dynamic delegate generation:
https://stackoverflow.com/questions/9505117/creating-delegates-dynamically-with-parameter-names

Wikipedia. (2022). List of CIL instructions:
https://en.wikipedia.org/wiki/List_of_CIL_instructions

ActiveState. (2006). Mini Fake DNS Server Recipe:
https://code.activestate.com/recipes/491264/