

.NET 高级代码审计（第九课） BinaryFormatter 反序列化漏洞

Ivan1ee@360 天眼云影实验室

2019 年 03 月 01 日

0x00 前言

BinaryFormatter 和 SoapFormatter 两个类之间的区别在于数据流的格式不同，其他的功能上两者差不多，BinaryFormatter 位于命名空间

System.Runtime.Serialization.Formatters.Binary 它是直接用二进制方式把对象进行序列化，优点是速度较快，在不同版本的.NET 平台里都可以兼容。但是使用反序列化不受信任的二进制文件会导致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 BinaryFormatter 序列化

使用 BinaryFormatter 类序列化的过程中，用[Serializable]声明这个类是可以被序列化的，当然有些不想被序列化的元素可以用[NoSerialized]属性来规避。下面通过一个实例来说明问题，首先定义 TestClass 对象

```
[Serializable]
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.Age = 18;
testClass.Name = "Ivanlee";
testClass.Classname = "360";
FileStream stream = new FileStream(@"d:\Binary1.dat", FileMode.Create);
BinaryFormatter bFormat = new BinaryFormatter();
bFormat.Serialize(stream, testClass);
stream.Close();
```

常规下使用 Serialize 得到序列化后的二进制文件内容打开后显示的数据格式如下

启动	Binary.dat	Binary1.dat																					
编辑为: 十六进制 (00)			运行脚本								运行模板												
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF						
0000h:	00	01	00	00	00	FF	FF	FF	FF	01	00	00	00	00	00	00ÿÿÿÿ.....						
0010h:	00	0C	02	00	00	00	3E	57	70	66	41	70	70	31	2C	20>WpfAppl,						
0020h:	56	65	72	73	69	6F	6E	3D	31	2E	30	2E	30	2E	30	2C	Version=1.0.0.0,						
0030h:	20	43	75	6C	74	75	72	65	3D	6E	65	75	74	72	61	6C	Culture=neutral						
0040h:	2C	20	50	75	62	6C	69	63	4B	65	79	54	6F	6B	65	6E	, PublicKeyToken						
0050h:	3D	6E	75	6C	6C	05	01	00	00	00	11	57	70	66	41	70	=null.....WpfAp						
0060h:	70	31	2E	54	65	73	74	43	6C	61	73	73	03	00	00	00	p1.TestClass....						
0070h:	09	63	6C	61	73	73	6E	61	6D	65	04	6E	61	6D	65	03	.classname.name.						
0080h:	61	67	65	01	01	00	08	02	00	00	00	06	03	00	00	00	age.....						
0090h:	03	33	36	30	06	04	00	00	00	07	49	76	61	6E	31	65	360.....Ivanle						
00A0h:	65	12	00	00	00	0B											e.....						

0x02 BinaryFormatter 反序列化

2.1、反序列化用法

反序列过程是将二进制数据转换为对象，通过创建一个新对象的方式调用 Deserialize 多个重载方法实现的，查看定义可以看出和 SoapFormatter 格式化器一样实现了 IRemotingFormatter、IFormatter 接口

```
namespace System.Runtime.Serialization.Formatters.Binary
{
    public sealed class BinaryFormatter : IRemotingFormatter, IFormatter
    {
        public BinaryFormatter();
        public BinaryFormatter(ISurrogateSelector selector, StreamingContext context);

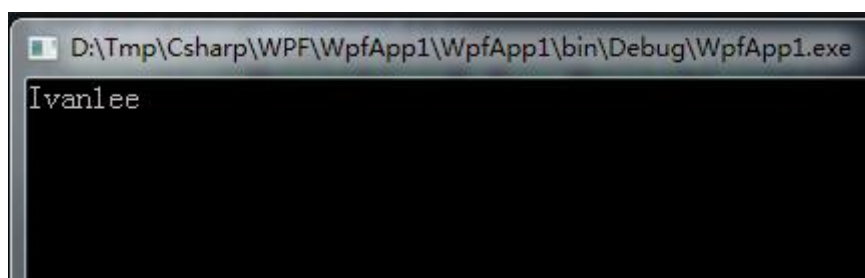
        public FormatterTypeStyle TypeFormat { get; set; }
        public FormatterAssemblyStyle AssemblyFormat { get; set; }
        public TypeFilterLevel FilterLevel { get; set; }
        public ISurrogateSelector SurrogateSelector { get; set; }
        public SerializationBinder Binder { get; set; }
        public StreamingContext Context { get; set; }

        public object Deserialize(Stream serializationStream);
        public object Deserialize(Stream serializationStream, HeaderHandler handler);
        public object DeserializeMethodResponse(Stream serializationStream, HeaderHandler handler, IMethodCallMessage methodCallMessage);
        public void Serialize(Stream serializationStream, object graph);
        public void Serialize(Stream serializationStream, object graph, Header[] headers);
        public object UnsafeDeserialize(Stream serializationStream, HeaderHandler handler);
        public object UnsafeDeserializeMethodResponse(Stream serializationStream, HeaderHandler handler, IMethodCallMessage methodCallMessage);
    }
}
```

我们得到系统提供的四个不同的反序列方法，分别是 Deserialize、DeserializeMethodResponse、UnsafeDeserialize、UnsafeDeserializeMethodResponse。笔者通过创建新对象的方式调用 Deserialize 方法实现的具体实现代码可参考以下

```
FileStream stream2 = new FileStream(@"d:\Binary1.dat", FileMode.Open);
BinaryFormatter bFormat2 = new BinaryFormatter();
var person = bFormat2.Deserialize(stream2);
Console.WriteLine(((TestClass)person).Name);
stream2.Close();
```

反序列化后得到 TestClass 类的成员 Name 的值。



2.2、攻击向量—ActivitySurrogateSelector

由于上一篇中已经介绍了漏洞的原理，所以本篇就不再冗余的叙述，没有看的朋友请参考《.NET 高级代码审计（第八课）SoapFormatter 反序列化漏洞》，两者之间唯一的区别是用了 BinaryFormatter 类序列化数据，同样也是通过重写

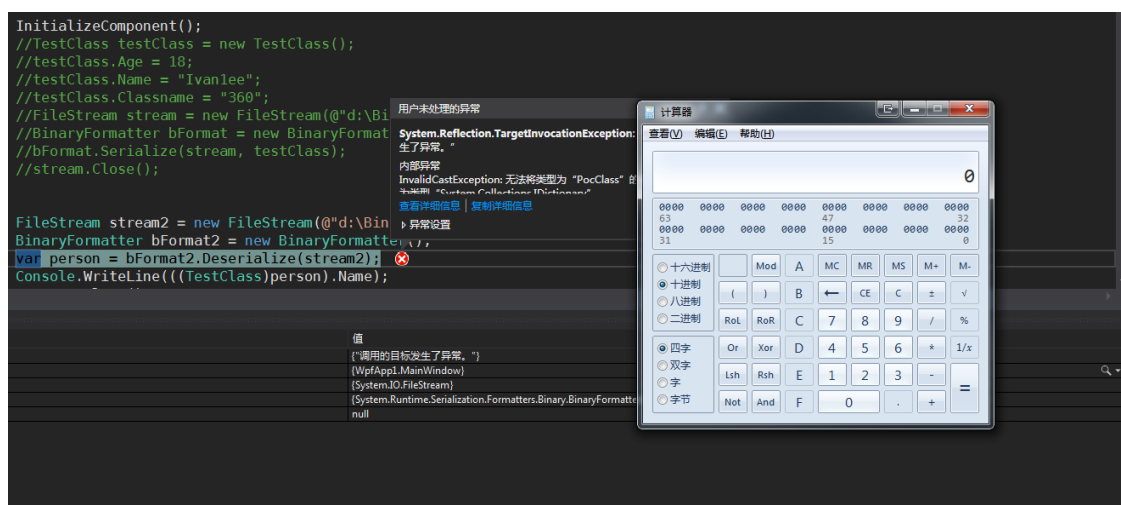
ISerializationSurrogate 调用自定义代码，笔者这里依旧用计算器做演示，生成的二进制文件打开后如下图

启动	Binary.dat																								
▼ 编辑为: 十六进制 00		▼ 运行脚本				▼ 运行模板																			
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF								
77A0h:	4A	73	6F	6E	43	6F	6E	76	65	72	74	00	74	65	73	74	JsonConvert.test								
77B0h:	00	47	65	74	49	6E	76	6F	63	61	74	69	6F	6E	4C	69	.GetInvocationLi								
77C0h:	73	74	00	54	6F	4C	69	73	74	00	52	65	65	6E	6C	69	st.ToList.Reenli								
77D0h:	73	74	00	41	70	70	6C	69	63	61	74	69	6F	6E	54	72	st.ApplicationTr								
77E0h:	75	73	74	00	67	65	74	5F	4F	75	74	00	4F	70	65	6E	ust.get_Out.Open								
77F0h:	53	74	61	6E	64	61	72	64	4F	75	74	70	75	74	00	4D	StandardOutput.M								
7800h:	6F	76	65	4E	65	78	74	00	53	79	73	74	65	6D	2E	54	oveNext.System.T								
7810h:	65	78	74	00	52	65	61	64	41	6C	6C	54	65	78	74	00	ext.ReadAllText.								
7820h:	67	65	74	5F	45	72	72	6F	72	54	65	78	74	00	53	74	get_ErrorText.St								
7830h:	72	65	61	6D	69	6E	67	43	6F	6E	74	65	78	74	00	63	reamingContext.c								
7840h:	6F	6E	74	65	78	74	00	76	00	78	00	54	6F	41	72	72	ontext.v.x.ToArr								
7850h:	61	79	00	53	79	73	74	65	6D	2E	53	65	63	75	72	69	ay.System.Securi								
7860h:	74	79	2E	50	6F	6C	69	63	79	00	67	65	74	5F	41	73	ty.Policy.get_As								
7870h:	73	65	6D	62	6C	79	00	67	65	74	5F	50	61	74	68	54	sembly.get_PathT								
7880h:	6F	41	73	73	65	6D	62	6C	79	00	53	65	6C	65	63	74	oAssembly.Select								
7890h:	4D	61	6E	79	00	53	79	73	74	65	6D	2E	52	75	6E	74	Many.System.Runt								
78A0h:	69	6D	65	2E	53	65	72	69	61	6C	69	7A	61	74	69	6F	ime.Serializatio								
78B0h:	6E	2E	46	6F	72	6D	61	74	74	65	72	73	2E	42	69	6E	n.Formatter.Bin								
78C0h:	61	72	79	00	49	44	69	63	74	69	6F	6E	61	72	79	00	ary.IDictionary.								
78D0h:	53	74	72	69	6E	67	44	69	63	74	69	6F	6E	61	72	79	StringDictionary								
78E0h:	00	6F	70	5F	45	71	75	61	6C	69	74	79	00	6F	70	5F	.op_Equality.op_								
78F0h:	49	6E	65	71	75	61	6C	69	74	79	00	53	79	73	74	65	Inequality.Syste								
7900h:	6D	2E	53	65	63	75	72	69	74	79	00	57	69	6E	64	6F	m.Security.Windo								
7910h:	77	73	49	64	65	6E	74	69	74	79	00	49	73	4E	75	6C	wsIdentity.IsNul								
7920h:	6C	4F	72	45	6D	70	74	79	00	00	00	0F	63	00	6D	00	lOrEmpty....c.m.								
7930h:	64	00	2E	00	65	00	78	00	65	00	00	17	2F	00	63	00	d...e.x.e.../.c.								
7940h:	20	00	63	00	61	00	6C	00	63	00	2E	00	65	00	78	00	.b.a.l.c...e.x.								
7950h:	65	00	00	01	00	13	70	00	7C	00	70	00	6C	00	75	00	e....p. .p.l.u.								
7960h:	67	00	69	00	6E	00	3D	00	00	2B	74	00	68	00	65	00	g.i.n.=...+t.h.e.								
7970h:	20	00	70	00	6C	00	75	00	67	00	69	00	6E	00	20	00	.p.l.u.g.i.n. .								
7980h:	74	00	6F	00	20	00	62	00	65	00	20	00	75	00	73	00	t.o..b.e. .u.s.								

按照惯例用 BinaryFormatter 类的 Deserialize 方法反序列化

```
FileStream stream2 = new FileStream(@"d:\Binary.dat", FileMode.Open);
BinaryFormatter bFormat2 = new BinaryFormatter();
var person = bFormat2.Deserialize(stream2);
Console.WriteLine(((TestClass)person).Name);
stream2.Close();
```

计算器弹出，但同时也抛出了异常，这在 WEB 服务情况下会返回 500 错误。

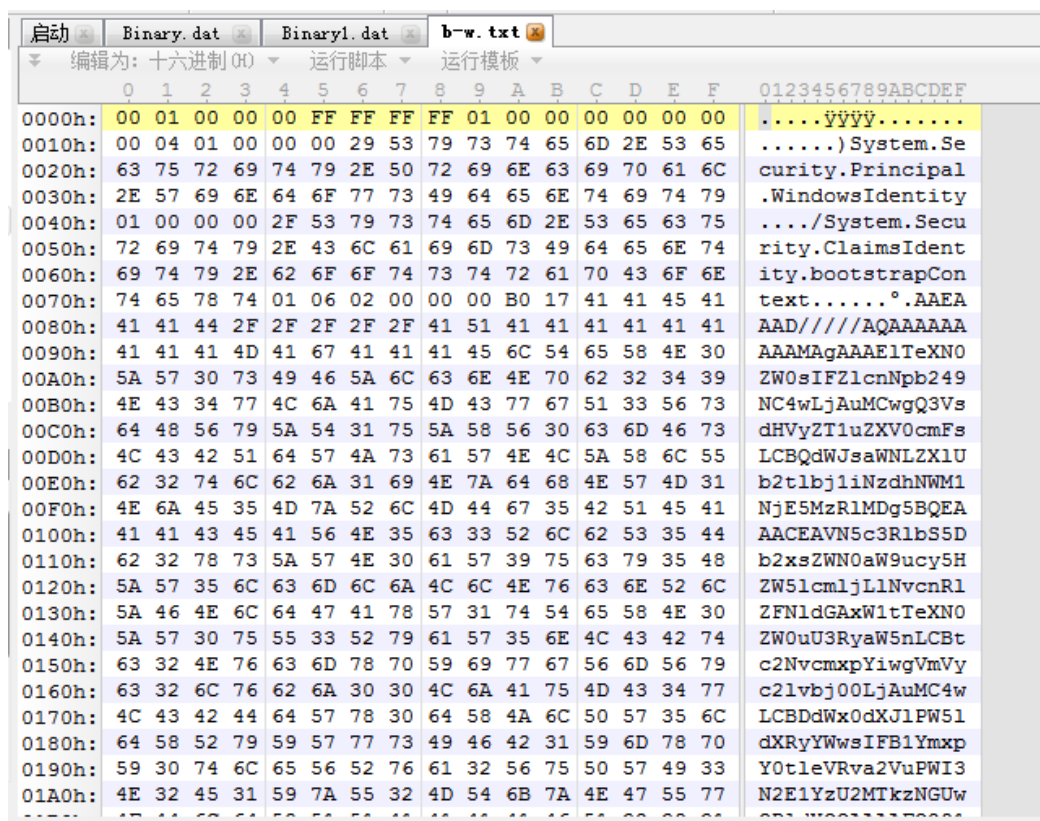


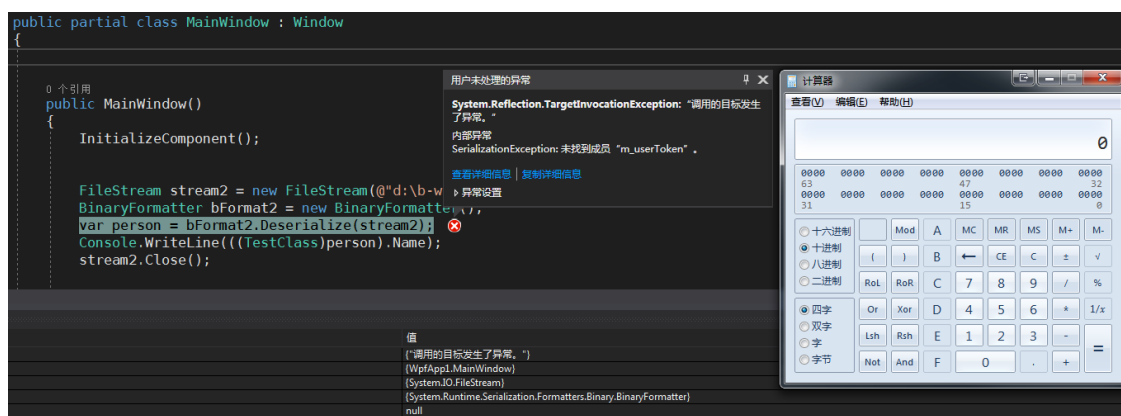
2.3、攻击向量—WindowsIdentity

有关 WindowsIdentity 原理没有看的朋友请参考《.NET 高级代码审计（第二课）

Json.Net 反序列化漏洞》，因为 WindowsIdentity 最终是解析 Base64 编码后的数

据，所以这里将 Serializer 后的二进制文件反序列化后弹出计算器





0x03 代码审计视角

3.1、UnsafeDeserialize

从代码审计的角度找到漏洞的 EntryPoint，相比 Deserialize，UnsafeDeserialize 提供了更好的性能，这个方法需要传入两个必选参数，第二个参数可以为 null，这种方式不算很常见的，需要了解一下，下面是不安全的代码：

```
public static object UnsafeDeserializeData(string path)
{
    FileStream fs = File.Open(path, FileMode.Open);
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    var objects = binaryFormatter.UnsafeDeserialize(fs, null);
    fs.Close();
    return objects;
}
```

攻击者只需要控制传入字符串参数 path 便可轻松实现反序列化漏洞攻击。

3.2、UnsafeDeserializeMethodResponse

相比 DeserializeMethodResponse，UnsafeDeserializeMethodResponse 性能上更加出色，这个方法需要传入三个必选参数，第二和第三个参数都可为 null，这种方式也不算很常见，只需要了解一下，下面是不安全的代码：


```
public static object UnsafeDeserializeMethodResponseData(string path)
{
    FileStream fs = File.Open(path, FileMode.Open);
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    var objects = binaryFormatter.UnsafeDeserializeMethodResponse(fs, null, null);
    fs.Close();
    return objects;
}
```

3.3、Deserialize

Deserialize 方法很常见，开发者通常用这个方法反序列化，此方法有两个重载，下面是不安全的代码

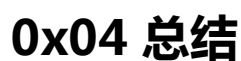
```
public static object DeserializeData(string path)
{
    FileStream fs = File.Open(path, FileMode.Open);
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    var objects = binaryFormatter.Deserialize(fs);
    fs.Close();
    return objects;
}
```

3.4、DeserializeMethodResponse

相比 Deserialize，DeserializeMethodResponse 可对远程方法响应提供的 Stream 流进行反序列化，这个方法需要传入三个必选参数，第二和第三个参数都可为 null，这种方式也不算很常见，只需要了解一下，下面是不安全的代码：

```
public static object DeserializeMethodResponseData(string path)
{
    FileStream fs = File.Open(path, FileMode.Open);
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    var objects = binaryFormatter.DeserializeMethodResponse(fs, null, null);
    fs.Close();
    return objects;
}
```

最后用这个方法弹出计算器，附上动图



实际开发中 BinaryFormatter 类从 .NET Framework 2.0 开始，官方推荐使用 BinaryFormatter 来替代 SoapFormatter，特点是 BinaryFormatter 能更好的支持泛型等数据，而在反序列化二进制文件时要注意数据本身的安全性，否则就会产生反序列化漏洞。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。